MACORD: Online Adaptive Machine Learning Framework for Silent Error Detection

Omer Subasi,[1] Sheng Di, [2] Prasanna Balaprakash,[3] Osman Unsal,[2]
Jesus Labarta,[2] Adrian Cristal,[2,4] Sriram Krishnamoorthy,[1] Franck Cappello,[3]
[1]Pacific Northwest National Laboratory, Washington, USA [2]Barcelona Supercomputing Center, Spain
[3]Argonne National Laboratory, Lemont, Illinois, USA
[4]IIIA - Artificial Intelligence Research Institute CSIC - Spanish National Research Council, Spain
{omer.subasi, sriram}@pnnl.gov
{osman.unsal, jesus.labarta, adrian.cristal}@bsc.es
{sdi1, cappello, pbalapra}@anl.gov

***Abstract*—Future high-performance computing (HPC) systems with ever-increasing resource capacity (such as compute cores, memory and storage) may significantly increase the risks on reliability. Silent data corruptions (SDCs) or silent errors are among the major sources that corrupt HPC execution results. Unlike fail-stop errors, SDCs can be harmful and dangerous in that they cannot be detected by hardware. To remedy this, we propose an online *MAchine-learning-based silent data CORruption Detection framework* (abbreviated as *MACORD*) for detecting SDCs in HPC applications. In our study, we comprehensively investigate the prediction ability of a multitude of machine-learning algorithms and enable the detector to automatically select the best-fit algorithms at runtime to adapt to the data dynamics. Because it takes only spatial features (i.e., neighboring data values for each data point in the current time step) into the training data, our learning framework exhibits low memory overhead (less than 1%). Experiments based on real-world scientific applications/benchmarks show that our framework can elevate the detection sensitivity (i.e., recall) up to 99%. Meanwhile the false positive rate is limited to 0.1% in most cases, which is one order of magnitude improvement compared with the latest state-of-the-art spatial technique.**

I. Introduction

High-performance computing (HPC) systems grow very fast with respect to execution scales because of the looming end of Moore's law on integrated circuits, and they are expected to have billions of processing elements in the near future. Error rates dramatically increase because of the growing system complexity that also is coupled with associated thermal and power challenges. Therefore, fault tolerance is becoming an even more important concern for HPC community than ever before. Silent data corruptions (SDCs) or silent errors are among the most significant impacting the fault tolerance of HPC executions. Unlike fail-stop errors, SDCs refer to any unexpected errors that may corrupt the execution results yet cannot be detected by hardware or applications during the execution of applications. Consequently, effective detection of SDCs is critical for guaranteeing the correctness of the HPC application results.

In this work, we propose a novel, efficient SDC detection framework, namely MACORD, which is built upon supervised machine learning algorithms that can be applied online with little memory and execution overhead. The detection involves the following two critical steps: (1) predict the values for each data point using our adaptive learning framework and (2) check the observed value for each data point to see if it falls inside the confidential value range.

MACORD is adaptive, automatic, and extensible in terms of user's demand on detection conditions such as tolerable prediction error. Most importantly, MACORD is an online framework. In the first step toward constructing this framework, we consider 11 state-of-the-art supervised learning algorithms for SDC detection. Then, we conduct a performance analysis of those algorithms. Based on the results, we select five algorithms that could be applied online at runtime (Section III). In the next step, we construct the main algorithm of MACORD which dynamically selects the best of those five algorithms for SDC detection. Unlike our previous work SSD [30] that uses a fixed learning algorithm, our new solution adopts an online, adaptive, extensible framework that can automatically select the best-fit learning algorithm at runtime according to the dynamically observed data. In addition to the learning algorithms explored in this work, MACORD can be extended with more *online-applicable* algorithms, or more detection conditions to adapt to new user demands.

To minimize the memory overheads, MACORD adopts spatial regression that uses neighboring data points instead of a time-series-based prediction method that suffers from relatively high memory overhead. Temporal techniques such as AID [9] maintain typically four recent data values for each data point in the data prediction, which means 400% memory overhead in terms of application state data to protect. As indicated in [9], the overall memory cost compared with the total application's memory usage can be more than 50% [9] in some cases. In absolute terms, our solution incurs less than 1% of memory overhead compared with the application

memory footprint.

Because of our adaptive design, MACORD's detection ability increases significantly. It also benefits from elaborate analysis of the data's spatial features, i.e., smoothness of the neighboring data. Considering the detection ability, performance, and memory overheads, MACORD achieves one order of magnitude less false positive rate compared with the state-of-the-art techniques while achieving low execution/memory overheads (Section IV).

To design and implement our data analytic SDC framework, we must resolve two significant challenges. On one hand, it is fairly non-trivial to design a set of effective, generic data prediction algorithms because of the diverse features of prediction algorithms. In particular, we need to check which supervised learning algorithms can be applied online and what metrics can be used for dynamic selection of the best-fit algorithm. On the other hand, it is challenging to optimize the trade-off between the low false positive rate and high recall because the data are diverse with applications and dynamic over time. In addition, the detection range formulation must be generic enough to fit as many HPC applications as possible. We tackle the first challenge by analyzing the training cost of different algorithms and the performance of algorithms under different metrics. For the latter challenge, we formally and carefully define and test the detection range based on dynamic behavior of data values during application computations.

In this work, we devise a novel, efficient SDC detection framework with extensive evaluation over four different error distributions and 11 real-world HPC applications from various domains. Our main contributions are summarized as follows:

- We design an online spatial adaptive SDC detection framework for HPC applications. To the best of our knowledge, this is the first learning-based scheme leveraging different supervised learning algorithms to find the best-fit algorithm in detecting SDCs for HPC applications. Our work verifies that incorporating only spatial features using multiple machine learning algorithms can achieve high SDC detection ability.
- We carefully investigate the effect of various detection ranges and different parameters values to optimize the detection ability, based on 11 real-world HPC applications.
- We evaluate MACORD using 11 real-world HPC applications and compare it with the state-of-the-art spatial SDC detector SSD [30] and AID [9] with four error distributions. Experiments show that MACORD can get the detection sensitivity (i.e., recall) up to 99% while suffering only 0.1% of false positive rate in most of cases. Our framework achieves one order of magnitude less false positive rate compared with SSD.

II. Background: Supervised Learning Algorithms

Formally, machine-learning-based modeling can be described as follows: given a set of training points (x_i, y_i), for $x_i \in \mathcal{T} \subset \mathcal{D}$, where $\mathcal{T}$ is a set of training points, $\mathcal{D}$ is the full data set, and $y_i = f(x_i)$ are input and the corresponding output, respectively. Regression modeling attempts to find a surrogate function h for the unknown function f such that the difference between $f(x_i)$ and $h(x_i)$ is minimal for all $x_i \in \mathcal{T}$ (and, ideally, minimal for all $x_i \in \mathcal{D}$).

Here, the linear equation is obtained by minimizing the sum of the squares of the differences between observed values in the training set and the corresponding values provided by the model.

K-nearest-neighbor (K-NN) regression [3] consists of finding K nearest training points in the input space for a given prediction point and returning the mean of the corresponding K outputs as the predicted value. Typically, K is a parameter and Euclidean distance metric is used to determine the nearest neighbors.

Decision tree (DT) regression belongs to the class of recursive partitioning methods [22]. These methods recursively partition the multi-dimensional input space of training points into a number of hyper rectangles such that input configurations with similar outputs fall within the same rectangle. The partition gives rise to a set of if-else rules that can be represented as a decision tree. For each hyper rectangle, a constant value is assigned—typically this is an average over the output values that fall within the given hyper rectangle. Given an unseen input, the algorithm uses the if-else rule to find the leaf and returns the corresponding constant value as the predicted value.

Linear regression (LM) tries to find $h(x)$ by fitting a linear equation to $\mathcal{T}$. It takes a linear functional form $h(x) = c + \sum_{i=1}^{M} \alpha^i \times x^i$, where c is a constant and α^i is the coefficient of the input x^i. Training the model consists in finding appropriate value of (c, α^*) using the method of least-squares [3].

Support vector machines (SVR) [27] are mainly useful for nonlinear regression. They project the input space of the training points into a higher-dimensional feature space and perform linear regression in that space. The main idea is the so-called kernel trick, in which a kernel function is used to perform operations required for regression in the input space, avoiding feature space projection and high dimensional regression, which are computationally expensive. Training consists of solving a quadratic programming problem and using a kernel to compute the influence of neighbor points.

Ada boost regressor (AB) [12] build trees iteratively such that the current tree is fitted to correct the errors of the previous trees. The particularity of ada boosting consists in adaptive weighted training set: after each tree fit, each point in the training set is assigned a weight proportional to the

current training error. At each iteration, a tree is assigned a coefficient such that the training error of the the tree is minimized. Given a new unseen input, each tree predicts a value and weighted average of all predictions gives a final predicted value.

Gaussian process regression (GP) [23] adopts a kernel function to measure the similarity between the given unseen point and performs interpolation using similar observations in the training data. The method assumes that the training points are sampled from a multi-variate Gaussian distribution, and uses the points in the training data to adopt the parameters of the distribution. The advantage of the method stems from the Gaussian distribution defined on the data with which we can estimate confidence intervals for each predicted value.

Stochastic gradient boosting (SGB) [13] is similar to ada boost regressor, in which trees are built sequentially but on a random subset of the training points. The main difference is that the error of the previous tree is used as the negative gradient of loss function being minimized.

Random forest (RF) [4] regression uses a collection of decision trees but for each tree generation, the algorithm takes a subsample of random points from the given training set. Due to the randomness in the sampling, each subsample differs from each other. Given that each individual tree is build on the subsample, it can differ significantly from other trees. For a given unseen input, each tree can make a prediction with respect to its own subsample. The effectiveness of this method comes from the aggregation of predicted output values from different trees.

Extremely randomized trees (ET) [4] is a variant of random forest regression. In this variant, threshold value for the input space partition is computed adaptively to improve the predictive accuracy and to control over-fitting.

Bagging (BR) [4] is another variant of random forest regression. While random forest randomly selects a subset of inputs, bagging regressor considers all the inputs for partitioning the input space.

Symbolic regression (SYM) [17] deterministically generates white-box models for a given regression problem while minimizing training error and model complexity. It uses a well-known machine learning technique, pathwise regularized learning, to prune a set of candidate basis functions down to a set of compact models.

III. Designing the MACORD Framework

In this section, we present the design of our automatic learning framework MACORD. We first present the formalization of our detection range. Then, we discuss the design of our main algorithm and detail our implementation.

A. Formalizing the Detection Model

Our detection model is formalized with the parameters in Table I. The *detection radius* $\rho(t)$ for a state variable is given by

$$\rho(t) := (\eta(t) + 1)(\theta r(t) + \xi(t-1)), \quad (1)$$

where $\xi(t-1)$ is the maximum prediction error at time step $t-1$ and $r(t)$ is the range of the observed values.

Table I
Detection Parameters

Parameter	Description
$X(t)$	Value predicted at time t
$V(t)$	A data point's observed value at time t
$\xi(t-1)$	Maximum prediction error estimate at time $t-1$
$\eta(t)$	Number of iterations with false positives at time t
θ	Application's relative impact error bound
$r(t)$	Range of a data point at time t

This formula is designed such that, when a false positive is encountered, the detection range should be enlarged to minimize subsequent false positives. The impact error bound signifies the error amount that can be safely ignored. The additive term $\xi(t-1)$ accounts for the evolution of application data values. The detector checks each data point at each application iteration. The normal value range is defined as $[X(t) - \rho(t), X(t) + \rho(t)]$. An error is detected when the observed value falls inside the normal range.

B. Overall Detection Algorithm

To design our main algorithm, we need to select the learning algorithms that can be applied online. We adopt machine learning techniques and also explore symbolic regression [17] as a subset of generic programming. We perform a cost analysis on a wide range of algorithms to determine which ones can be applied online. Figure 1 shows our analysis. The y-axis is in log scale, and shows average training cost in seconds of the corresponding algorithm. From the results, we can observe that K-NN, DT, LM, SVM, and AB can be adopted for online SDC detection because their training costs are significantly lower than other learning algorithms. Therefore, we have selected those five algorithms to be included in MACORD as subdetectors. Considering the other analyzed learning algorithms, the higher training cost of SGB, RF, ET, and BR can be attributed to the ensemble nature of these methods, where several trees are combined to improve generalizability and robustness at the expense of training time. Although AB is an ensemble approach, the low training cost stems from the simple reweighing mechanism to correct the errors. The higher cost of GP is due to the expensive inverse operation on the covariance matrix existing in its algorithmic structure. The cost of SYM is the highest because it tries to find a set of white box models when performing a single regression operation. Although we analyze 11 state-of-the-art learning algorithms, we note that our framework is extensible because any new *online-applicable* algorithm can be directly added to it. (with or without removing another algorithm).

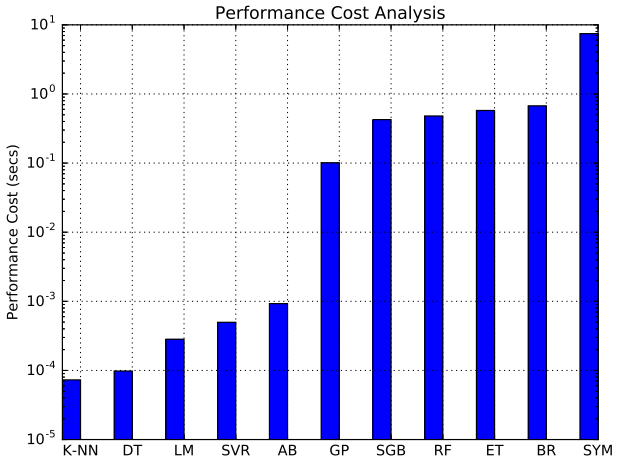


Figure 1. Performance cost analysis for different learning algorithms: K-nearest neighbor (K-NN), decision tree (DT), linear regression (LM), support vector regression (SVR), ada boosting (AB), Gaussian processing (GP), stochastic gradient boosting (SGB), random forest (RF), extremely randomized trees (ET), bagging regressor (BR), and symbolic regression (SYM).

After determining the set of subdetectors (or candidate learning algorithms), we are ready to design the adaptive SDC detection algorithm. During the execution of a parallel application with multiple running processes (or ranks), our detection function is called after the computational phase of the application in each iteration. Algorithm 1 summarizes the detection algorithm for a single process at each iteration. At the beginning, the value range of each state variable is aggregated among processes to compute the impact error bound (see lines 2-3 in Algorithm 1). Then, the candidate learning algorithm with the smallest absolute prediction error, i.e., $|V(t) - X(t)|$, is selected as the best-fit detection algorithm for a period of iterations (line 7 in Algorithm 1). After, the selected algorithm is trained based on the neighboring points of each of the target data points (line 9). Such spatial training leads to a model of the computation, with which the detector can predict a value for each data point (line 11). The prediction values and computed radius are used to calculate the normal range. Finally, the observed value for each data point is checked if it is in the normal range (line 12-16). If not, the current time step will be considered with SDCs.

Takeaway-1: *Only a subset of learning algorithms can be utilized online for SDC detection. Most of the algorithms are too costly to be applied online.*

IV. Evaluation

Our MACORD framework evaluation is threefold. First, we evaluate the false positive rate (FP-rate) and detection sensitivity (recall) of MACORD. Moreover, we compare the detection results of MACORD with those of the SSD algorithm [30] and AID [9]. Second, we evaluate the performance of subdetectors in detail. Third, we evaluate the computation and memory overheads of MACORD in comparison with SSD. We first discuss the experimental setup and then present the experimental results.

Algorithm 1: Online Adaptive Learning Framework

```
   Data: Current step t, data value V(t), relative error bound θ
   Data: Algorithm set AS={a_1, a_2, ..., a_n}
   Result: Boolean indicating whether SDC is present
1  begin
2      Compute range r(t) /* periodically done */
3      ρ(t) ← calculateRadius(ξ(t − 1), η(t), θr(t))
4      isDetected ← false
5      counter++
6      if (counter mod period == 0) then
7          chosen_alg ← SelectAlgorithmBasedOnAMetric(AS)
8      end
9      chosen_alg.TrainWithNeighboringPoints()
10     for (each data point) do
11         X(t) ← chosen_alg.Predict(t)
12         isDetected ← checkInRange(ρ(t), X(t), V(t))
13         if (isDetected) then
14             Trigger some operation for data recovery
15             break
16         end
17     end
18 end
```

Table II
Applications Used in Evaluation

Name	Package	Name	Package
Blast2 [7]	Flash	SodShock [28]	Flash
DMReflection [7]	Flash	RHD_Sod [16]	Flash
RHD_Riemann2D [24]	Flash	Vortex [19]	Nek5k
BrioWu [5]	Flash	OrszagTang [20]	Flash
MHD [11]	Nek5k	Cellular [36]	Flash
HeatDistribution [21]	customized		

A. Experimental Setup

We perform our experiments on the Marenostrum Supercomputer [1] hosted at Barcelona Supercomputing Center. Table II shows the applications used in our evaluation from the FLASH package [14], the Nek5k package [18], and a customized stencil application (heat distribution) [21]. For each application, we protect *state variables*, which are checked at every main iteration of the applications. Experiments are performed over 1000 iterations. When assessing the detection sensitivity, we use the relative impact error bounds recommended in [9]. In particular, we set θ to 0.0001 for Blast2, and 0.00078125 for the other benchmarks, because this setting can guarantee that the maximum deviation propagated based on the data sets with undetected errors is always under 5% of the global value range for the whole remaining execution period. We inject errors according to the distribution that the data of state variables would be corrupted with random bit positions based on our sensitivity analysis. In fault injection experiments, each single case is repeated 10×, and we compute and report the average results. We have experimented with 1, 2, and 4 neighbors

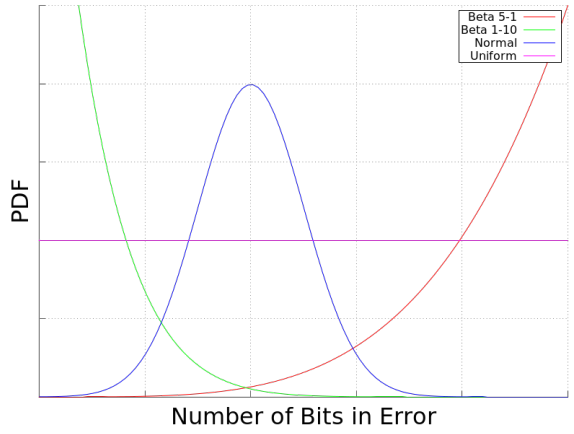


Figure 2. Distributions used in the experimental evaluation

as training sets. We report 2-neighbor results. The other results are similar. For each application, all state variables are protected in our experiments.

As we have no information about how silent errors will exhibit themselves, we use four different error distributions (shown in Figure 2) to cover reasonable scenarios that may occur in the future HPC systems and to assess our framework's performance. These are two beta, normal and uniform distributions.

B. Experimental Results

1) False Positive Rate and Sensitivity: Figure 3 shows the cumulative distribution functions (CDFs) of the false positive rates (FP-rate) unde MACORD, as well as AID and SSD. The results are collected by running applications on 128 processes. The FP-rate is defined as the number of false positive iterations (iterations with at least one false positive detected) over the total number of iterations. The FP-rate is especially useful in assessing a detector's precision. As shown in the figure, MACORD not only outperforms SSD but also achieves one order of magnitude less false positive rates in most cases. The main reason MACORD outperforms SSD is twofold: (1) more precise data prediction (to be shown later) and (2) more accurate detection range estimated. Moreover, in particular, MACORD achieves an FP-rate close to 0.1% on average. MACORD performs a little better than AID, which incurs additional memory penalty due to taking and maintaining snapshots.

Figure 4 presents the CDFs of the detection sensitivity (recall) for the benchmarks. Recall is defined as the fraction of the true positives detected over all SDCs experienced or injected. Our novel generic algorithm achieves more than 90% and up to 99% recall as SSD with error distributions other than Beta 1-10. This distribution injects errors sparsely. As a result the recall is lower than that of other distributions. As far as recall results are considered, AID, SSD, and MACORD performs similarly.

Takeaway-2: *In terms of false positive rates, MACORD outperforms state-of-the-art techniques. In fact, it decreases one order of magnitude over the spatial state-of-the art algorithm, SSD.*

2) Subdetector Performance and Metric Analysis: Figure 5 shows representative figures (from different distributions and benchmarks) highlighting two important quantitative aspects. First, it shows the percentage of time a subdetector achieves minimum prediction error. Second, when achieving minimum prediction error, it shows the percentage of time that the subdetector was the only detector to achieve the minimum prediction error. Hence, there are two important observations. First, all algorithms achieve the uniquely minimum prediction error in some cases. Second, the subdetectors more or less contribute to the main algorithm computation and are being selected by MACORD.

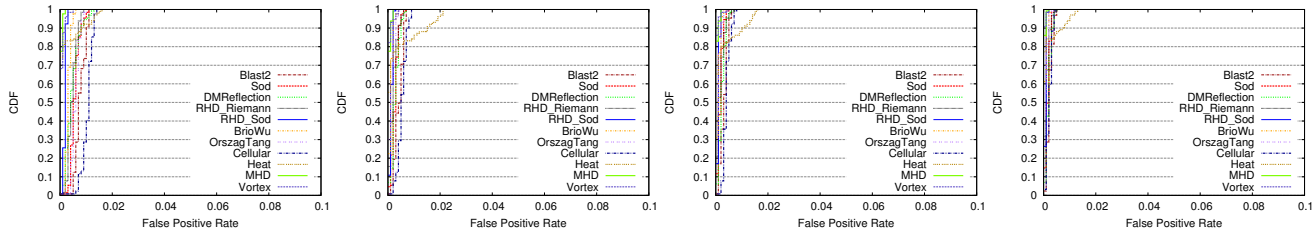
The dynamic selection of subdetector algorithms in MACORD is performed based on prediction error. However we evaluate different metrics. These include the well-known assessment metrics, such as root mean square error, mean absolute error, and root mean square error devision. We evaluate the metrics in both error-free executions and faulty executions with both single- and multi-bit injections. However, we did not observe any tangible effect of the metric in either execution types. Thus, our conclusion is that different metrics do not seem to effect MACORD's performance.

Takeaway-3: *Experimental evaluation shows that different learning algorithms perform differently across diverse distributions and benchmarks.*

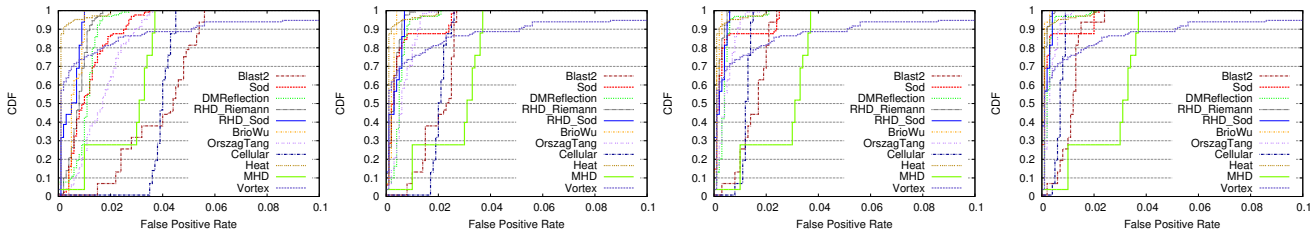
Takeaway-4: *Experimental results show that MACORD has better prediction capability and accuracy than SSD. Combined with tuned definition of the detection range, MACORD performs better than SSD as a result.*

3) Computation Overheads: It has been reported that SSD [30] has 5% computation overhead on average with 1,024 cores. A single SSD run takes about 5×10^{-4} seconds on average. For MACORD, a single run of support vector regression takes about 5×10^{-4}, boosting takes about 9×10^{-4}, nearest neighbor takes about 7×10^{-5}, decision tree takes about 9×10^{-5}, and linear regression takes about 2×10^{-4} seconds on average. This makes a total about less than 16×10^{-4} seconds and average of 3×10^{-4} seconds, which is less than a single run of SSD. Because we periodically select the best-fit algorithm every 10 iterations, per iteration it amortizes about $1{,}5\times10^{-4}$ seconds. That is, the computation overhead of MACORD is very close to that of SSD.

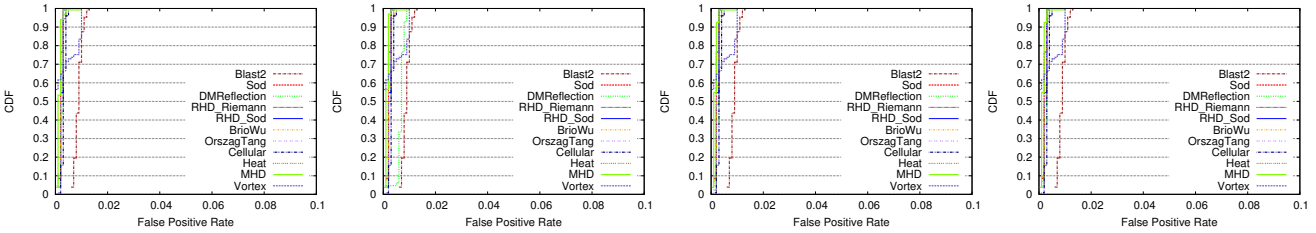
Takeaway-5: *MACORD has low performance (about 5%) and memory (less than 1%) overheads while having a completely user-transparent and fully automatic SDC detection capability, where false positive rate decreases one order of magnitude compared with the state-of-the-art technique SSD.*



(a) AID Beta 5-1 FP-r. (b) AID Beta 1-10 FP-r. (c) AID Normal 32-4 FP-r. (d) AID Uniform FP-r.

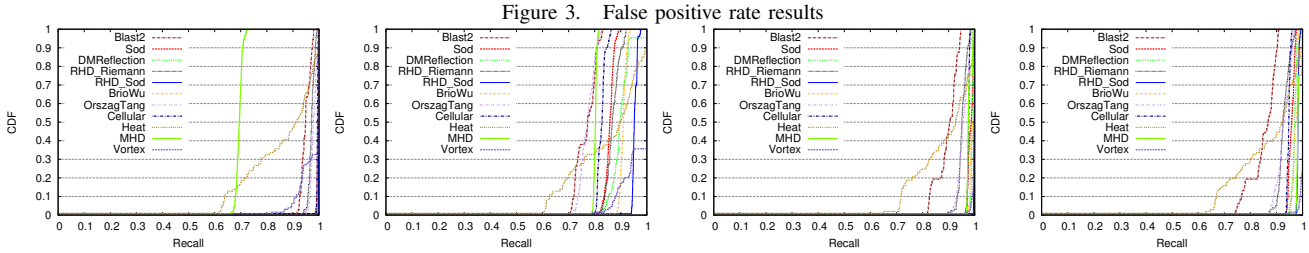


(e) SSD Beta 5-1 FP-r. (f) SSD Beta 1-10 FP-r. (g) SSD Normal 32-4 FP-r. (h) SSD Uniform FP-r.

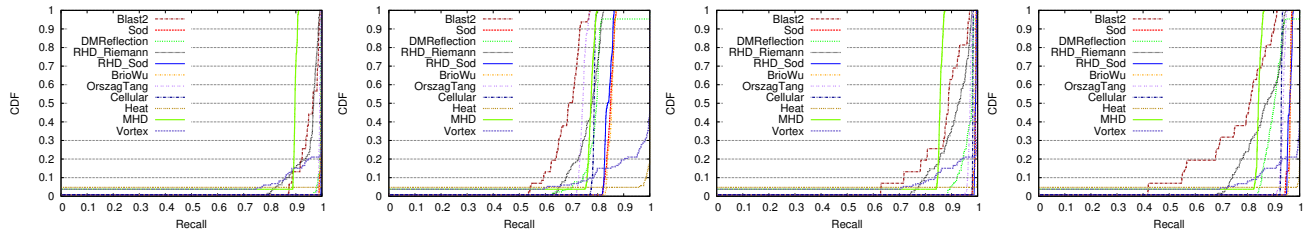


(i) MACORD Beta 5-1 FP-r. (j) MACORD Beta1-10 FP-r. (k) MACORD Normal 32-4 FP-r. (l) MACORD Uniform FP-r.

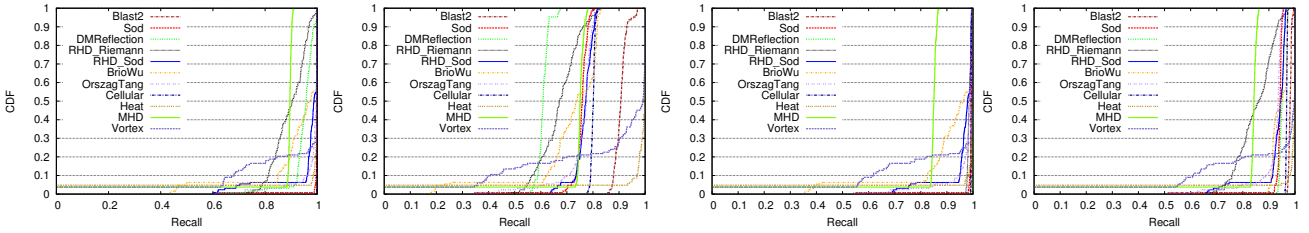
Figure 3. False positive rate results



(a) AID Beta 5-1 Re. (b) AID Beta 1-10 Re. (c) AID Normal 32-4 Re. (d) AID Uniform Re.

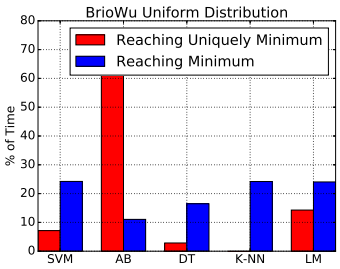


(e) SSD Beta 5-1 Recall (f) SSD Beta 1-10 Re. (g) SSD Normal 32-4 Re. (h) SSD Uniform Re.

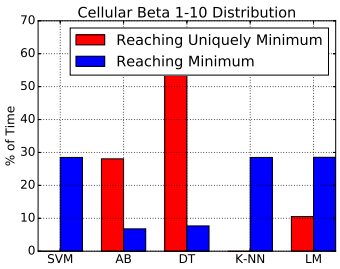


(i) MACORD Beta 5-1 Re. (j) MACORD Beta 1-10 Re. (k) MACORD Normal 32-4 Re. (l) MACORD Uniform Re.

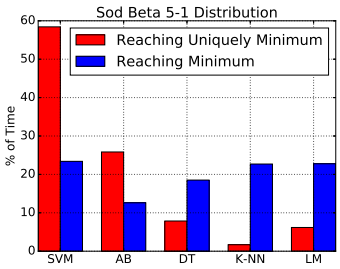
Figure 4. Recall results



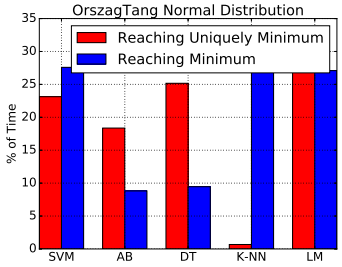
(a) AB outperforms others wrt. unique min.



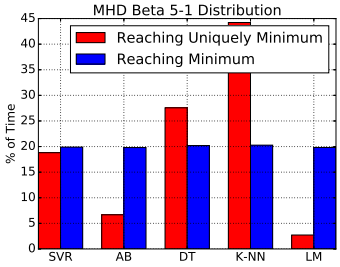
(b) DT outperforms others wrt. unique min.



(c) SVM outperforms others wrt. unique min.



(d) LM outperforms others wrt. unique min.



(e) K-NN outperforms others wrt. unique min.

Figure 5. Subdetector performance

V. Related Work

Approaches to SDC detection can typically be classified as being runtime analysis based, replication-based, or application/algorithm-specific.

Soft error detection based on runtime data analysis has gained momentum in the HPC community. Studies [2, 8, 15] investigate and compare different prediction methods to detect and mitigate SDCs. They convert the problem of detecting SDC into the next-step prediction problem. Runtime data analysis approaches can be categorized into temporal and spatial techniques. The main drawbacks of temporal data analytics are the memory overhead and the computation cost of maintaining snapshot data. As a spatial technique, SSD [30], while having low computation overhead, incurs low memory cost. However, as experimental evaluation showed, our novel framework outperforms SSD. Runtime data analysis approaches can also be categorized into online and offline techniques. Sirius [35] is a neural network based offline SDC detection tool. The training is performed offline and, thus, Sirius fundamentally differs from online techniques.

Replication-based techniques [10] are typically used when a high degree of reliability is required, Different levels of computation redundancy can be utilized to detect and, possibly, correct SDCs. The main disadvantage of computation redundancy or replication is that it incurs prohibitive energy overheads. Selective redundancy schemes [29, 31, 32] has been designed to reduce these overheads.

Specialized detectors have been developed for specific classes of algorithms [6, 26, 33, 37] or programs [25, 34]. While effective, these techniques are not as generally applicable as those presented in this paper.

VI. Conclusion

In this work, we propose MACORD, a novel extensible framework for SDC detection based on online spatial learning algorithms. MACORD automatically selects the best-fit detector algorithms at runtime to adapt to the data dynamics. MACORD only relies on the snapshot data at the current time step, i.e., a spatial technique. Consequently our solution suffers from little memory overhead (less than 1%) and low performance overhead (on par with SSD, which has a performance overhead of 5% on average.). Experiments with 11 real-world HPC applications show that for most of the failure distributions and applications, sensitivity is highup to 99%. Meanwhile, the false positive rate is low and on the order of 0.1% in most cases, which is one order of magnitude improvement over state-of-the-art spatial detectors. In the future, we plan to extend MACORD with techniques such as Kernel Ridge regression, an *online-applicable* technique, to improve its detection ability and to analyze the improvement and the incurred performance overheads.

Acknowledgments

We thank Leonardo Bautista-Gomez for his helpful feedback and discussions. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number 66905, program manager Lucy Nowell. Pacific Northwest National Laboratory is operated by Battelle for DOE under Contract DE-AC05-76RL01830 and the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

REFERENCES

[1] Marenostrum III: http://www.bsc.es/marenostrum-support-services/mn3.

[2] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. Lightweight silent data corruption detection based on runtime data analysis for HPC applications. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 275–278, New York, NY, USA, 2015.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[5] M. Brio and C. Wu. An upwind differencing scheme for the equations of ideal magnetohydrodynamics. *Journal of Computational Physics*, 75(2):400 – 422, 1988.

[6] E. Ciocca, I. Koren, Z. Koren, C. M. Krishna, and D. S. Katz. Application-level fault tolerance in the orbital thermal imaging spectrometer. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, PRDC '04, pages 43–48, Washington, DC, USA, 2004.

[7] P. Colella and P. R. Woodward. The piecewise parabolic method (ppm) for gas-dynamical simulations. *Journal of Computational Physics*, 54(1):174 – 201, 1984.

[8] S. Di, E. Berrocal, and F. Cappello. An efficient silent data corruption detection method with error-feedback control and even sampling for HPC applications. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 271–280, 2015.

[9] S. Di and F. Cappello. Adaptive impact-driven detection of silent data corruption for HPC applications. *IEEE Transactions on Parallel and Distributed Systems*, 12/2015 2015.

[10] 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 *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 78:1–78:12, CA, USA, 2012.

[11] P. Fisher. Nek5000 user guide, [online] Available at http://www.mcs.anl.gov/˜fischer/nek5000/examples.pdf.

[12] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, Aug. 1997.

[13] J. H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38(4):367–378, 2002.

[14] 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*, 131:273–334, Nov. 2000.

[15] L. A. B. Gomez and F. Cappello. Detecting and correcting data corruption in stencil applications through multivariate interpolation. In *2015 IEEE International Conference on Cluster Computing*, pages 595–602, Sept 2015.

[16] J. M. Martí and E. Müller. Numerical hydrodynamics in special relativity. *Living Reviews in Relativity*, 6(7), 2003.

[17] T. McConaghy. FFX: Fast, scalable, deterministic symbolic regression technology. In R. Riolo, E. Vladislavleva, and H. J. Moore, editors, *Genetic Programming Theory and Practice IX*, pages 235–260, New York, NY, 2011. Springer New York.

[18] Nek5000 project. https://nek5000.mcs.anl.gov.

[19] P. O'Rourke and M. Sahota. A variable explicit/implicit numerical method for calculating advection on unstructured meshes. *J. Comput. Phys.*, 143(2):312–345, July 1998.

[20] S. A. Orszag and C.-M. Tang. Small-scale structure of two-dimensional magnetohydrodynamic turbulence. *Journal of Fluid Mechanics*, 90:129–143, 1 1979.

[21] N. Ozisik. Finite difference methods in heat transfer. 1994.

[22] J. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221 – 234, 1987.

[23] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[24] C. W. Schulz-Rinne, J. P. Collins, and H. M. Glaz. Numerical solution of the Riemann problem for two-dimensional gas dynamics. *SIAM Journal on Scientific Computing*, 14(6):1394–1414, 1993.

[25] V. C. Sharma, G. Gopalakrishnan, and S. Krishnamoorthy. PRESAGE: protecting structured address generation against soft errors. In *23rd IEEE International Conference on High Performance Computing, HiPC 2016, Hyderabad, India, December 19-22, 2016*, pages 252–261. IEEE Computer Society, 2016.

[26] J. Sloan, R. Kumar, and G. Bronevetsky. Algorithmic approaches to low overhead fault detection for sparse linear algebra. In *Proceedings of the 2012 42Nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '12, pages 1–12, Washington, DC, USA, 2012.

[27] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[28] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1 – 31, 1978.

[29] O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal. Programmer-directed partial redundancy for resilient HPC. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, pages 47:1–47:2, New York, NY, USA, 2015.

[30] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. S. Ünsal, J. Labarta, A. Cristal, and F. Cappello. Spatial support vector regression to detect silent errors in the exascale era. In *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2016, Cartagena, Colombia, May 16-19, 2016*, pages 413–424, 2016.

[31] O. Subasi, G. Yalcin, F. Zyulkyarov, O. S. Unsal, and J. Labarta. A runtime heuristic to selectively replicate tasks for application-specific reliability targets. In *2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016*, pages 498–505, 2016.

[32] O. Subasi, G. Yalcin, F. Zyulkyarov, O. S. Unsal, and J. Labarta. Designing and modelling selective replication for fault-tolerant HPC applications. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017*, pages 452–457, 2017.

[33] D. Tao, S. L. Song, S. Krishnamoorthy, P. Wu, X. Liang, E. Z. Zhang, D. J. Kerbyson, and Z. Chen. New-sum: A novel online ABFT scheme for general iterative methods. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2016, Kyoto, Japan, May 31 - June 04, 2016*, pages 43–55, 2016.

[34] S. Tavarageri, S. Krishnamoorthy, and P. Sadayappan. Compiler-assisted detection of transient memory errors. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 204–215, 2014.

[35] T. E. Thomas, A. J. Bhattad, S. Mitra, and S. Bagchi. Sirius: Neural network based probabilistic assertions for detecting silent data corruption in parallel programs. In *35th Symposium on Reliable Distributed Systems (SRDS)*, 2016.

[36] F. X. Timmes, M. Zingale, K. Olson, B. Fryxell, P. Ricker, A. C. Calder, L. J. Dursi, H. Tufo, P. MacNeice, J. W. Truran, and R. Rosner. On the cellular structure of carbon detonations. *The Astrophysical Journal*, 543(2):938, 2000.

[37] M. Turmon, R. Granat, D. Katz, and J. Lou. Tests and tolerances for high-performance software-implemehted fault detection. *IEEE Transactions on Computers*, 52(5):579–591, May 2003.