Exploring the Capabilities of Support Vector Machines in Detecting Silent Data Corruptions

Omer Subasi,[1] Sheng Di,[3] Leonardo Bautista-Gomez,[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
{leonardo.bautista, osman.unsal, jesus.labarta, adrian.cristal}@bsc.es
{sdi1, cappello, pbalapra}@anl.gov

Abstract

As the exascale era approaches, the increasing capacity of high-performance computing (HPC) systems with targeted power and energy budget goals introduces significant challenges in reliability. Silent data corruptions (SDCs), or silent errors, are one of the major sources that corrupt the execution results of HPC applications without being detected.

In this work, we explore a set of novel SDC detectors - by leveraging epsilon-insensitive support vector machine regression - to detect SDCs that occur in HPC applications. The key contributions are threefold. (1) Our exploration takes temporal, spatial, and spatiotemporal features into account and analyzes different detectors based on different features. (2) We provide an in-depth study on the detection ability and performance with different parameters, and we optimize the detection range carefully. (3) Experiments with eight real-world HPC applications show that support-vector-machine-based detectors can achieve detection sensitivity (i.e., recall) up to 99% yet suffer a less than 1% false positive rate for most cases. Our detectors incur low performance overhead, 5% on average, for all benchmarks studied in this work.

Keywords: Silent Data Corruptions, Support Vector Machines, HPC Applications.

1. Introduction

The typical future exascale high-performance computing (HPC) system is expected to have one billion processing elements. This increase in system complexity coupled with the associated thermal and power challenges is expected to increase error rates. Thus, reliability is a serious concern advancing to the exascale era. Silent data corruptions (SDCs) or silent errors are one of the most significant problems stymieing the reliability of HPC applications running on such systems. As opposed to fail-stop errors, silent errors are hazardous because they cannot be detected by the underlying hardware: the application data and results are corrupted without any indication to users. Therefore, effective and efficient detection of SDCs is critical to guarantee the correctness of the HPC application results.

In this work, we explore a set of novel and efficient SDC detectors by leveraging machine learning. In particular, we use a support vector machine (SVM) supervised learning method to detect SDCs. SVMs are effective because their non-linear nature detects complex SDCs. In this work, we undertake a design space exploration of SVM regression, namely spatial, temporal, and spatiotemporal regression. Our strategy focuses on analyzing the different features of each set of observed data, involving the following two critical steps: (1) predicting the values for each data point[1] by using a dynamic ε[2] in Vapnik's loss function [1] and (2) checking the observed value for each data point to see if it falls inside the confidence value range.

To design and implement our data analytic detectors, we have to resolve two significant challenges. On the one hand, designing an effective data prediction algorithm based on SVM is challenging, especially because of the data dynamics. In particular, we observe that impact error bounds correspond to the insensitivity of the loss function for an SVM, and the correspondence is diverse because the impact error bound

[1] The user annotates *state variables* (e.g., density, pressure) such that our detectors check them at each application iteration (Section 3).

[2] This parameter refers to the insensitivity, meaning the amount of deviation tolerated by the SVM during the regression process.

changes dynamically at runtime. On the other hand, devising an appropriate detection range that achieves both a low false positive rate and high recall require a careful trade-off. Moreover, the detection range formulation should be generic enough to fit as many HPC applications as possible.

In this work, we devise a set of novel SDC detectors - with extensive evaluation over five different error distributions and eight real-world HPC applications. Our main contributions are summarized as follows:

- We design dynamic, online, SVM-based SDC detectors for HPC applications. To the best of our knowledge, this is the first machine-learning-based framework leveraging temporal, spatial and spatiotemporal SVM regression to detect SDCs for HPC applications. The predictors incorporate different features while maintaining a dynamic loss function.
- We provide an in-depth study of the detection ability and performance with different parameters, and we optimize the detection range carefully.
- We implement our detectors library supporting a wide range of HPC applications. It can be downloaded from [2].
- We evaluate our detectors using eight real-world HPC applications with five different error distributions and compare our detectors with the state-of-the-art SDC adaptive impact-driven (AID) detector [3] and multivariate interpolation [4]. Experiments show that our detectors can achieve detection sensitivity (i.e., recall) up to 99% yet suffer a less than 1% false positive rate for most cases. Our detectors also incur low performance overhead, 5% on average, for all benchmarks examined in the work.

We proposed a novel SDC detector based on spatial evolution of data in a prior study [5]. This work extends the prior study with temporal and spatiotemporal regression. It analyzes the effect of temporal and spatiotemporal data in detecting SDCs.

The rest of the paper is organized as follows: in Section 2, we present the background for this study. In Section 3, we discuss the design of our detectors in detail. In Section 4, we evaluate our detectors in terms of detection and prediction capability and

performance overheads. In addition, we compare our detectors' performance to that of the AID algorithm and multivariate interpolation. Section 5 describes the state of the art in SDC mitigation research, and Section 6 includes concluding remarks.

2. Background

This section provides an overview of SVMs, which are the core technique used in our solution. We then continue with data prediction types and discuss the impact-driven SDC detection, which is the fundamental detection model we employ. Finally, we close the section with an overview of the AID algorithm and multivariate interpolation because they are the most related to our work.

2.1. SVMs: An Overview

SVMs were originally designed for pattern classification problems by Vapnik and coworkers [1], and they have been widely applied to other fields for function approximation signal processing, regression, and time series prediction [6, 7, 8, 9]. The key feature of SVMs is that they leverage the *structural risk minimization* principle to find a decision function with a good generalization capacity. The solution to a particular problem depends only on a specific subset of the training data points called *support vectors* [1]. Figure 1 shows the difference between SVMs and other linear classifiers. SVMs construct a maximum margin hyperplane, whereas linear classifiers attempt to find some hyperplane. As a result, SVMs are able to reach a unique and global optimal solution as opposed to other linear classifiers.

To handle nonlinearity, a technique called *kernel trick* is applied in SVMs. The points in the input space are mapped to a high-dimensional *feature space* via nonlinear mapping, where they are linearly separable and the optimal hyperplane is constructed in the feature space, as illustrated in Figure 2, where φ refers to some kernel function.

Figure 3 shows the basic architecture of SVMs and how they work[3]. The input $\mathbf{x}$ vector and the support vectors x_i are mapped by Φ into a feature space. Then, the dot products are computed using a proper kernel function. All dot products are then

[3]This example is adopted from Smola and Schölkopf [10].

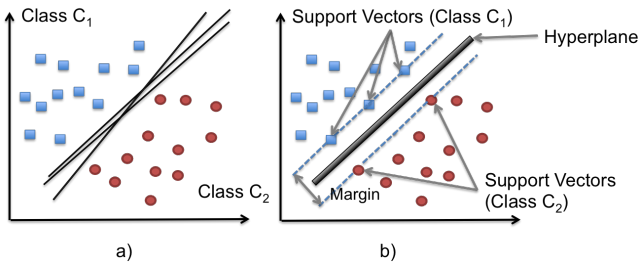


Figure 1: SVM classification compared with other linear classifiers

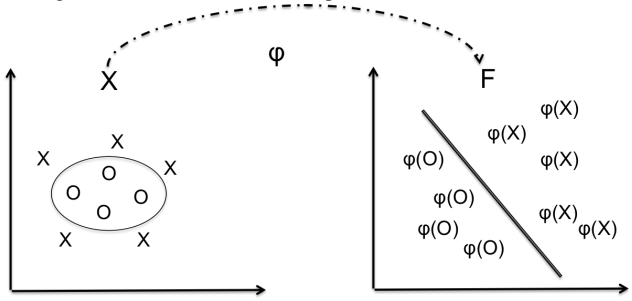


Figure 2: SVM kernel trick to tackle a nonlinear data problem

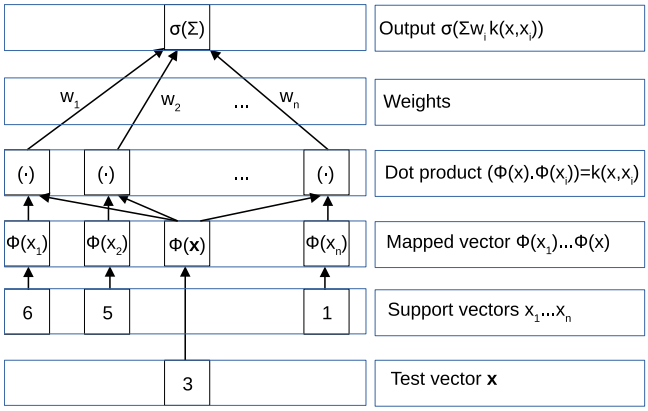


Figure 3: SVM architecture. The input **x** vector and the support vectors x_i (which are digits in this example) are mapped by Φ into a feature space where dot products are computed. Kernel k is used in practice to compute the dot products. The results are linearly combined by weights w_i, which are found by solving a quadratic program. Finally, the sign function $\sigma(\Sigma) = sign(\Sigma)$ is used to classify the input **x** vector.

linearly combined by the weights, which are computed by solving a quadratic program that finds the optimal hyperplane. Finally, the sign of the linear combination (which is computed by the weights found in the previous step) becomes the class of the input vector $\mathbf{x}$.

SVMs can also be used for regression problems, such as ε-insensitive SVM regression. The basic idea involves fitting a function to the training data where deviations below ε are tolerated. Slack variables are often introduced to make the problem tractable or handle noisy/inseparable data [1].

As follows, we list four key properties of SVMs and compare them to other techniques. (1) The first property of a SVM is its *duality property*: the data only appear in dot products both in the decision function and training algorithm. This enables SVMs to operate in higher dimensions without explicit transformation and to use the aforementioned kernel trick to tackle the nonlinearly separable data as opposed to linear techniques. (2) The second property relates to the *kernel trick* technique, where the implicit mapping is taken rather than the explicit computation of the kernel itself. This prevents costly processing of high-dimensional data. (3) The third property of SVMs is the ability to control capacity by *maximizing the margin*. This property mitigates the overfitting problem that exists in other techniques, such as neural networks. (4) The forth property guarantees the convergence of the SVM algorithm. SVMs have the *convexity property*, which makes them solvable in polynomial time. This property allows SVMs to effectively avoid the local minimum solution, guaranteeing the global optimum of the solution.

The reader is advised to refer to [1] and [11] for detailed general discussions about SVMs, and [10] for SVM regression.

2.2. Temporal vs. Spatial Prediction

Data prediction can be categorized into two classes. In temporal prediction, previous time step snapshots of data are used to make a prediction. In contrast, in spatial prediction only the neighboring data points are used to make a prediction. A third class, termed spatiotemporal, forms by considering spatial and temporal prediction together. Figure 4 shows the techniques. Specifically, it depicts the spatial and temporal SVM-

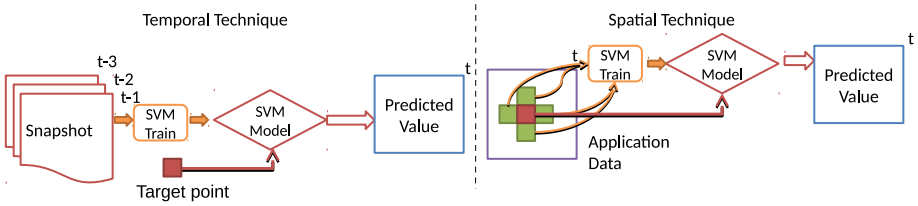


Figure 4: Temporal (temporal SVM predictor) versus spatial prediction (spatial SVM predictor)

based predictors. In the spatial SVM predictor, neighboring points are used to obtain an SVM model, which is then used to predict a value for the target point. In contrast, the temporal SVM predictor leverages application snapshots of previous time steps.

2.3. Impact-Driven SDC Detection

Research by Di and Cappello [3] demonstrates that not all SDCs may impact the application execution results significantly, and the primary focus should be on the influential SDCs. Di and Cappello gave an in-depth analysis of the impact of SDCs on HPC execution results, and revealed that the impact of SDCs can be characterized by an *impact error bound*, which is defined as the maximum ratio of the data value change between adjacent time steps to the global value range size for every data point in a snapshot. As long as the data changes incurred by SDCs are within such an impact error bound, the maximum deviation of the data values (compared to the original fault-free results) will be limited to only 3% of the global value range for most of cases. In particular, the experiments with real-world HPC applications (as shown in [3]) demonstrated that the impact error bound = 0.00078125 (or 0.0001 some times) is enough for detecting SDCs for most applications. In this work, we leverage such an impact error bound to devise our SVM-based detectors.

2.4. AID: Adaptive Impact-Driven SDC Detector

AID [3] is an outstanding SDC detector, which allows different processes to dynamically select the best-fit curve fitting models with minimum prediction errors based on their runtime data. The curve fitting models include last-state, linear, and quadratic. The AID algorithm incorporates types and periodically selects the best curve fitting

with the lowest prediction error, and the selection process is conducted periodically (every 20 iterations as set in the experiments). The detection is performed by maintaining a normal value range that is based on the user-specified impact error bound and the dynamically aggregated value range for data points. If the observed value for any data point falls outside the normal value range, the corresponding time step will be treated as an SDC step and correction operation (such as restarting the application from one previous checkpoint file) will be triggered accordingly. Otherwise, the execution will not be interrupted.

2.5. Multivariate Interpolation

We now examine the multivariate interpolation method proposed by Bautista-Gomez and Cappello [4]. Multivariate interpolation is a mathematical technique used for functions with more than one variable. The interpolation itself can be implemented with different techniques. Bautista-Gomez and Cappello [4] chose linear interpolation for simplicity. For three-dimensional (3D) space, for example, linear interpolation can be performed by

$$f(x, y, z) = f(x_a, y, z) + (f(x_b, y, z) - f(x_a, y, z)) \times \frac{x - x_a}{x_b - x_a}.$$

For any data point in a snapshot, its value will be predicted by using its neighboring points, and the predicted value will also be compared with a normal range for detecting possible anomalies. In [4], the normal range is acquired at the beginning of the execution by estimating the maximum error, which then is used until the end of the execution.

3. Dynamic Online SVM-Based SDC Detectors

In this section, we first present the formalization of our predictors. Then, we discuss our detection range design and detail its implementation.

3.1. Formalization of The SVM Predictors

Let $\{(x_1, y_1), ...(x_n, y_n)\} \subset X \times \Re$ be training data, where X denotes the space of input patterns and $X = \Re^d$ for some dimension d. In our $\varepsilon - SV$ regression [1],

the aim is to approximate a function $f(x)$ such that it deviates at most from targets y_i while being as flat as possible. Therefore, we set $\varepsilon = \theta_I r_i^j$, where θ_I is the impact error bound of the application under consideration and r_i^j is the estimated value range for the ith input pattern in the jth iteration of the application. In our SVM, the target function takes the conventional form

$$f(x) = \langle w, x \rangle + b, w \in X, b \in \Re, \tag{1}$$

where $\langle w, x \rangle$ denotes the dot product in X. Furthermore, to tackle nonlinear regression problem, a mapping Φ, called a *kernel*, is introduced such that the patterns are mapped into some *feature space* F where they are linearly separable:

$$\Phi : X \to F. \tag{2}$$

The SVM-based prediction can be formalized as follows:

$$\text{MINIMIZE} \quad \frac{1}{2} \parallel w \parallel^2 + \gamma \sum_{i=1}^{n} (\kappa_i \xi_i + \kappa_i^* \xi_i^*) \tag{3}$$

subject to

$$y_i - \langle w, \Phi(x) \rangle - b \leq \theta_I r_i^j + \kappa_i \xi_i \tag{4}$$

$$\langle w, \Phi(x) \rangle + b - y_i \leq \theta_I r_i^j + \kappa_i^* \xi_i^* \tag{5}$$

$$\xi, \xi^* \geq 0, \tag{6}$$

where γ is the regularization parameter, κ_i and κ_i^* are criticality coefficients, and ξ_i and ξ_i^* are slack variables. The regularization parameter determines the trade-off between the flatness of f and the amount of deviations larger than $\theta_I r_i^j$ that is tolerated. When provided, the criticality coefficients convey the relative vulnerabilities of variables. The higher the coefficient, the higher the penalty. When not provided, all coefficients are assumed to be one. The slack variables can have various purposes. They can be used to cope with the infeasibility of the optimization. They can also be used for noisy or inseparable data.

Vapnik's ε-insensitive loss function [1] is

$$|\xi|_\varepsilon := \begin{cases} 0, & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon, & \text{otherwise.} \end{cases} \tag{7}$$

Table 1: Training Set with respect to Regression Type and Size

Size	Spatial	Temporal	Spatiotemporal
1	x_{i-1}	$x_{t_{i-1}}$	-
2	x_{i-1}, x_{i+1}	$x_{t_{i-1}}, x_{t_{i-2}}$	$x_{i-1}, x_{t_{i-1}}$
4	$x_{i-1}, x_{i+1}, x_{i-2}, x_{i+2}$	$x_{t_{i-1}}, x_{t_{i-2}}, x_{t_{i-3}}, x_{t_{i-4}}$	$x_{i-1}, x_{i+1}, x_{t_{i-1}}, x_{t_{i-2}}$

Our key observation is that the impact error bounds correspond to the ε, the insensitivity, in the loss function of an SVM. That is, the impact error bounds specify how much an SVM can tolerate during the process of regression. Hence, the loss function of our SVM predictors is

$$|\xi|_{\theta_I r_i^j} := \begin{cases} 0, & \text{if } |\xi| \leq \theta_I r_i^j \\ |\xi| - \theta_I r_i^j, & \text{otherwise.} \end{cases} \tag{8}$$

Equation (3) presents a convex quadratic problem that is solved by Lagrangian multipliers and Karush-Kuhn-Tucker (KKT) conditions [12] in its dual form. For each *state variable* specified to be protected, Equation (3) is solved at each iteration of the application to make a prediction.

The feature vectors and training set of our predictors depend on the type of regression. If spatial regression is performed, then the training set and feature vector consist of the values of the neighboring data points. If temporal regression is performed, then the training set and feature vector consist of the past data values of the target point. If spatiotemporal regression is performed, then the training set and feature vector consist of both the values of the neighboring data points and the past data values of the target point. Table 1 shows the feature vectors of our detectors, i.e., the data points used in the training set with respect to the regression type and training sizes.

Finally, we examine the admissibility of kernel functions for SVMs and the kernels used in our design. Mercer's condition [13] provides a necessary and sufficient condition for a kernel to be admissible so that the input patterns are mapped to the feature space:

Mercer's Theorem. *Assume $k \in L_\infty(X^2)$ such that the integral operator $T_k : L_2(X) \rightarrow$*

$L_2(X)$

$$T_k f(.) := \int_X k(.,x) f(x) d\mu(x) \tag{9}$$

is positive, where μ denotes a measure on X such that $\mu(X)$ is finite and $supp(\mu) = X$. Let $\psi_j \in L_2(X)$ be the eigenfunction of T_k associated with the eigenvalue $\lambda_j \neq 0$ normalized such that $\| \psi_j \|_{L_2} = 1$, and let $\overline{\psi_j}$ denote its complex conjugate. Then:

1. $(\lambda_j(T))_j \in \ell_1$
2. $k(x, x') = \sum_{j \in N} \lambda_j \overline{\psi_j(x)} \psi_j(x')$ *holds for almost all (x, x'), where the series converges absolutely and uniformly for all (x, x').*

Less formally, the theorem says that if the following holds, then $k(x, x')$ is admissible, meaning it can be written as a dot product in some feature space:

$$\int_{X \times X} k(x, x') f(x) f(x') dx dx' \geqslant 0 \tag{10}$$

for all $f \in L_2(X)$.

The following polynomial and radial basis functions (RBF) are examples for admissible kernels:

$$k(x, x') = (\langle x, x' \rangle + c)^p$$

$$k(x, x') = e^{-\frac{\|x - x'\|^2}{2\sigma^2}}.$$

Even though sigmoid kernels do not satisfy Mercer's conditions, they work well in practice:

$$k(x, x') = \tanh(\vartheta + \nu \langle x, x' \rangle).$$

We explore the effect of different kernels in our design. Specifically, the kernels we study are linear ($p = 1$, $c = 0$), polynomial ($p = 2$, $c = 0$), RBF ($\sigma = 1$), and sigmoid ($\vartheta = 0$, $\nu \langle x, x' \rangle = \langle x, x' \rangle$).

3.2. The Formalization of Detection Range and Algorithm

Our detection model is formalized with the parameters in Table 2. The *detection radius* $\rho(t)$ is defined as

$$\rho(t) := \eta(t) \theta r(t) + feedback(t - 1), \tag{11}$$

Table 2: Detection Model Parameters

Parameter	Description
$X(t)$	Predicted value by the predictor at time t
$V(t)$	Observed value of the data point at time t
$feedback(t-1)$	Max prediction error estimate at time $t-1$
$\eta(t)$	Number of iterations with false positives at time t
θ	Relative impact error bound of the application
$r(t)$	Range of the data point at time t

where $feedback(t-1)$ is the maximum prediction error at time step $t-1$ based on second-order (quadratic) prediction and $r(t) = max(V(t)) - min(V(t))$.

The rationale behind the design is that the detection range is supposed to be enlarged as the application experiences a time step with false positives - in order to minimize false positives. In addition, $\theta r(t)$ is the impact error bound that determines if the SDCs will lead to a significant impact on the execution results. The design of the feedback term *feedback*$(t-1)$ is to adapt to possibly sharp data changes, which will lead to large prediction errors accordingly.

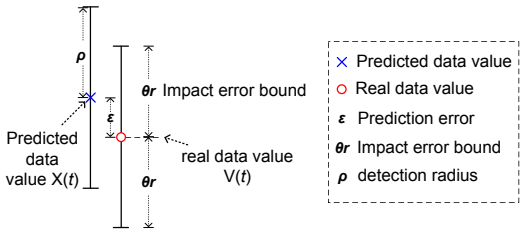


Figure 5: Detection model

Figure 5 illustrates our detection model. At each iteration, our detector checks a data point based on this model. The normal data value range is defined as $[X(t) - \rho(t), X(t) + \rho(t)]$. The detection is performed by checking whether the observed value $V(t)$ falls in this normal range.

Algorithm 1 summarizes our detection algorithm for a single process at each iteration in the execution. For each data value (state variables), the value range is aggregated

Algorithm 1: SVM-Based Detector

Data: Current step t, data value $V(t)$, relative error bound θ

Result: Boolean indicating whether SDC is present

```
1  begin
2      Compute range r
3      isDetected ← false
4      SVM_SetEpsilon(θ, r)
5      SVM_Train()
6      X(t) ← SVM_Predict(t)
7      ρ(t) ← calculateRadius(feedback(t), η(t), θr)
8      isDetected ← checkInRange(ρ(t), X(t), V(t))
9      if (isDetected) then
10         Trigger some operation for data recovery.
11     end
12 end
```

among processes, and the epsilon parameter of the SVM is initialized with the relative error bound θ and value range r (note that the impact error bound is θr). Then, the SVM is trained using data points in the dataset according to the type of spatial, temporal, and spatiotemporal regression. The prediction of the SVM and the computed radius is used to calculate the normal range. The observed value for each data point is checked to determine if it is in the normal range. If not, the current time step will be considered with SDCs.

3.3. Implementation

We implement our detectors following a design based on LibSVM [14]. We integrate our detectors with the Fault Tolerance Interface (FTI) library [15] such that application users are allowed not only to detect the SDCs, but also to correct the errors by checkpoint/restart. Our implementation provides both C and Fortran interfaces so a broad range of HPC applications can use the detector. The library is available for download from [2]. To use our detectors, users must follow four simple steps where they annotate their applications: (1) initialize the detectors by calling SDC_Init(), (2) specify the state variables to protect by calling SDC_Protect(var,ierr), (3) annotate the

execution iterations by calling SDC_Snapshot() in the main loop, and (4) release the memory by calling SDC_Finalize() in the end.

4. Evaluation

Table 3: Applications Used in Evaluation

Name	Description
Blast2 [16]	Strong shocks and narrow features
SodShock [17]	Sodshock tube for testing compressible code's ability with shocks and contact discontinuities
DMReflection [16]	Double Mach reflection: evolution of an unsteady planar shock on an oblique surface
RHD_Sod [18]	Relativistic Sod Shock-tube: involving the decay of 2D-fluids into 3D-elementary wave structures
RHD_Riemann2D [19]	Relativistic 2D Riemann: exploring interactions of four basic waves consisting of shocks, etc.
BrioWu [20]	Coplanar magneto-hydrodynamic counterpart of hydrodynamic Sod problem
OrszagTang [21]	Simple 2D problem that has become a classic test for magnetohydrodynamics (MHD) codes
Cellular [22]	Burn simulation: cellular nuclear burning problem

This section details the experimental evaluation of our detectors. Our evaluation is twofold. First, we evaluate the false positive rate and detection sensitivity (recall) of our detectors. We additionally evaluate the effect of different parameters on these metrics. Moreover, we compare the detection results of our detectors with that of the AID algorithm [3] and multivariate interpolation [4]. Second, we evaluate the detection overhead of our detectors. We discuss the experimental setup first then present the experimental results.

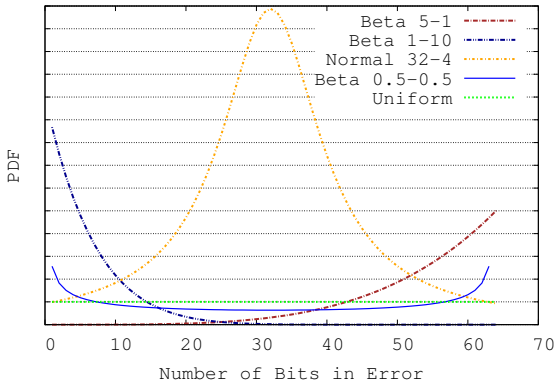


Figure 6: Distributions used in the experimental evaluation

4.1. Experimental Setup

We perform our experiments using the Fusion [23] cluster at Argonne National Laboratory. Table 3 shows the applications employed in our evaluation from the FLASH package [24]. For each application, we protect *state variables*, which are checked at every main iteration of the applications. When assessing detection sensitivity, we use the relative impact error bounds recommended in [3]. In particular, we use 0.0001 for Blast2 and 0.00078125 for the other seven benchmarks. We perform error injection according to the error distribution chosen where injections are performed to the random bit positions of state variables in sensitivity analysis. We do not use any criticality coefficients, meaning we treat all state variables to have the same significance. In fault injection experiments, each single case is repeated 10 times, and the averages are reported.

Because we have no information about how silent errors will exhibit themselves, we use five different error distributions (shown in Figure 6) to cover reasonable scenarios that can occur in the exascale era and to assess our detectors' performance. In Figure 6, the number of bits in the x-axis is 64, and it shows the probability density function (PDF) for a 64-bit word. The exact number of errors injected depends on the distributions and is injected randomly in a word after the number is set.

Beta Distributions. Beta distribution is typically used in control systems and population genetics. This class provides distributions that fit possible scenarios that can occur in the exascale time frame by adjusting shape parameters. Formally the PDF of the beta distribution is defined for $0 \leq x \leq 1$ and shape parameters α and β as

$$f(x|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{12}$$

where Γ is the gamma function that can be viewed as the extension of the factorial function over complex numbers. Formally, it can be defined by the integral (except nonpositive integers)

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx. \tag{13}$$

We use three settings with the beta distributions (Figure 6). Beta distribution with shape parameter $\alpha = 1$ and $\beta = 10$ represent the case where the PDF value decreases

with the number of bits corrupted. This setting represents the case where single-bit errors are more likely than multi-bit errors and the probability of error decreases as the number of bits in error increases. In contrast, $\alpha = 5$ and $\beta = 1$ represent the case where the PDF value increases with the number of bits corrupted. This case can be justified by the postulation that single- or double-bit flip errors are likely to be detected by the underlying hardware ECCs while multi-bit flip errors cannot be detected effectively by hardware ECCs. $\alpha = 0.5$ and $\beta = 0.5$ represent another possible case in which single-bit flip errors or all-bit flip errors are more common than other types of bit flip errors. This case is included to reflect those with erratic behavior.

Normal Distribution. The central limit theorem implies that the number of errors should follow a normal distribution given that the flip event on each bit follows an independent and identical distribution. Therefore, we include normal distribution to account for the case where errors are independent and identically distributed. Formally, the normal distribution is defined with the PDF as follows:

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{14}$$

where μ is the mean and σ is the variance of the distribution. Without loss of generality, μ and σ are set to 32 and 4 respectively in our evaluation.

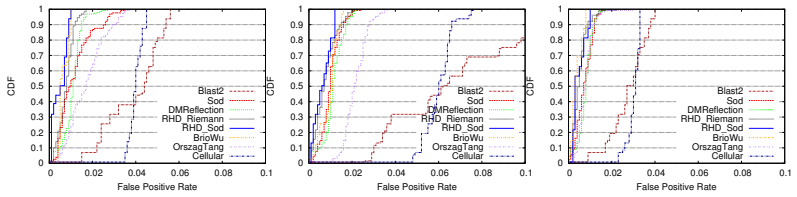
Uniform Distribution. Another possible case is that the number of bits flipped follows a uniform distribution, where the PDF is defined over an interval $[a, b]$ as

$$f(x) = \frac{1}{b-a}. \tag{15}$$

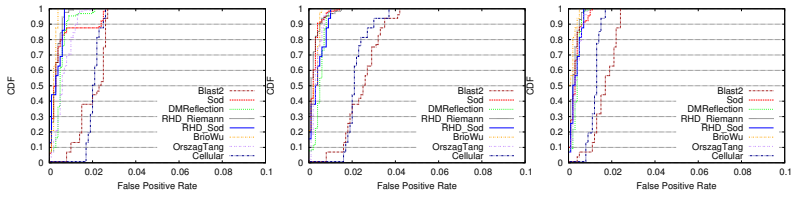
Because we use double precision in our experimentation, the interval in our evaluation is $[1, 64]$. This distribution represents the scenario that on the unprotected hardware, such as logic unit, any number of bit flips can occur. Hence, a uniform distribution on the number of bit flips can be assumed.

4.2. Experimental Results

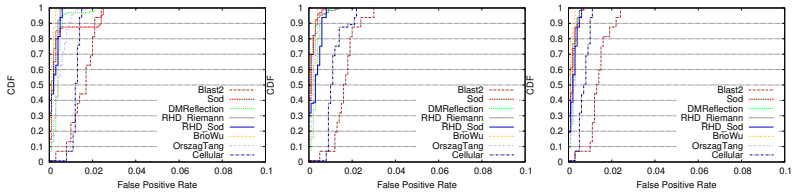
Before presenting the results, we define the relevant concepts: prediction error, false positive rate, and recall. Prediction error is the difference between the predicted value and the observed value. The false positive rate is the ratio of the number of false



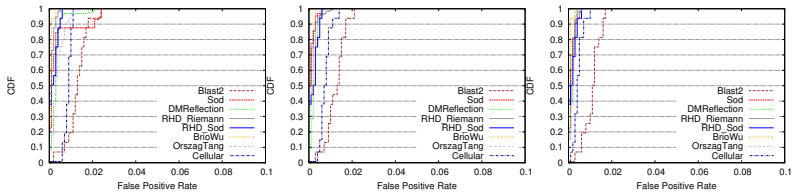
(a) Spatial Beta5-1 FP-r. (b) Temporal Beta5-1 FP (c) Spatiotemporal Beta5-1 FP



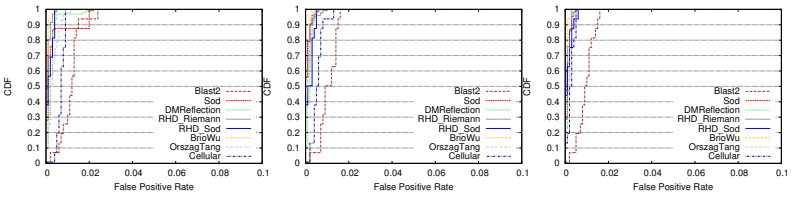
(d) Spatial Beta1-10 FP (e) Temporal Beta1-10 FP (f) Spatiotemporal Beta1-10 FP



(g) Spatial Normal FP (h) Temporal Normal FP (i) Spatiotemporal Normal FP

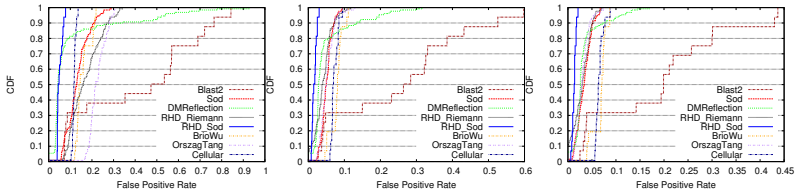


(j) Spatial Beta0.5-0.5 FP (k) Temporal Beta0.5-0.5 FP (l) Spatiotemp Beta0.5-0.5 FP

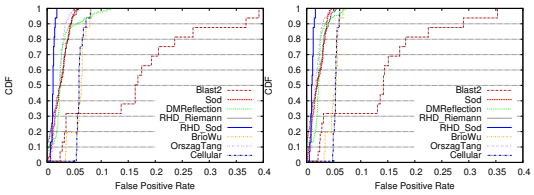


(m) Spatial Uniform FP (n) Temporal Uniform FP (o) Spatiotemporal Uniform FP

Figure 7: False positive rate results for our SVM detectors

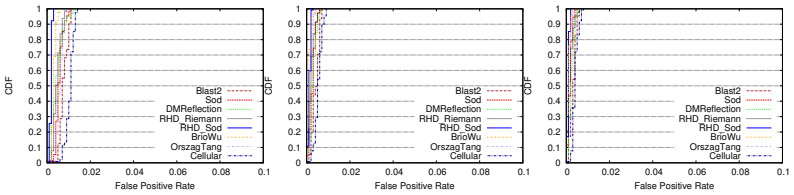


(a) Multiv. Beta 5-1 FP (b) Multiv. Beta 1-10 FP (c) Multiv. Normal 32-4 FP

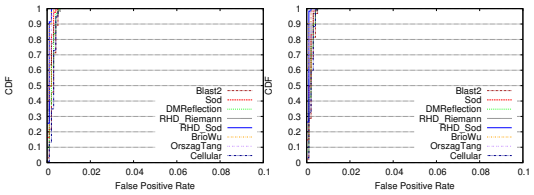


(d) Multiv. Beta 0.5-0.5 FP (e) Multiv. Uniform FP

Figure 8: False positive rate results for multivariate interpolation

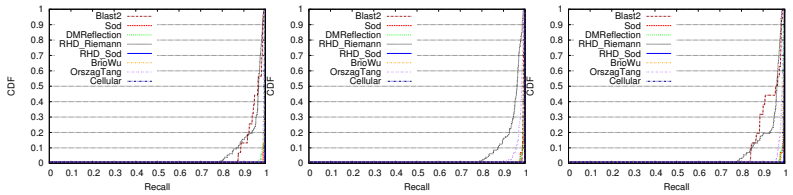


(a) AID Beta 5-1 FP-r. (b) AID Beta 1-10 FP (c) AID Normal 32-4 FP

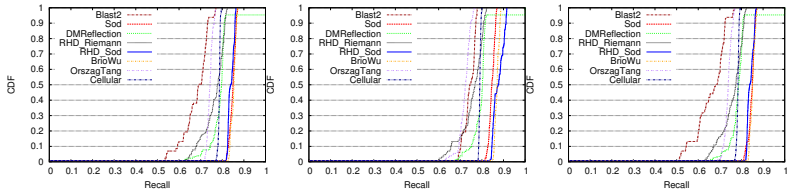


(d) AID Beta 0.5-0.5 FP (e) AID Uniform FP

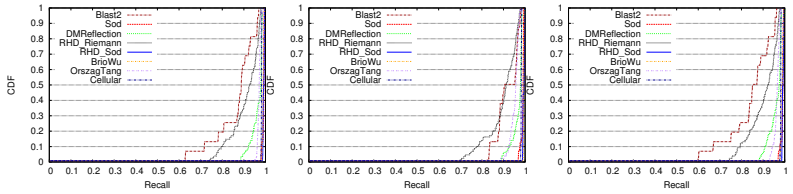
Figure 9: False positive rate results for AID



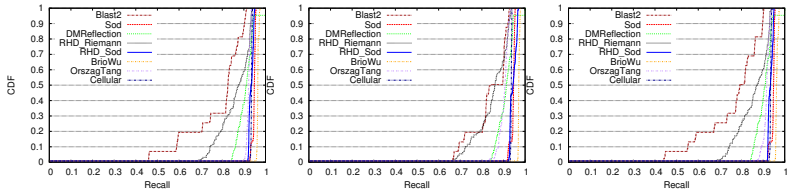
(a) Spatial Beta5-1 Re. (b) Temporal Beta5-1 Re. (c) Spatiotemporal Beta5-1 Re.



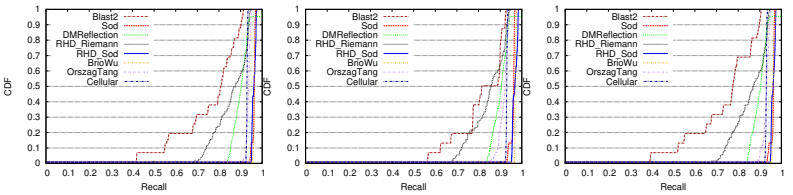
(d) Spatial Beta1-10 Re. (e) Temporal Beta1-10 Re. (f) Spatiotemporal Beta1-10 Re.



(g) Spatial Normal Re. (h) Temporal Normal Re. (i) Spatiotemporal Normal Re.

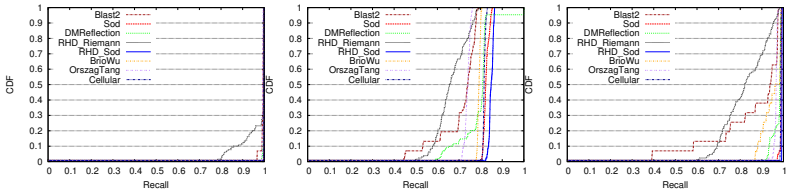


(j) Spatial Beta0.5-0.5 Re. (k) Temporal Beta0.5-0.5 Re.(l) Spatiotemp Beta0.5-0.5 Re.

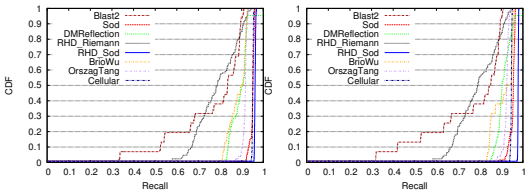


(m) Spatial Uniform Re. (n) Temporal Uniform Re. (o) Spatiotemporal Uniform Re.

Figure 10: Recall results for our SVM detectors

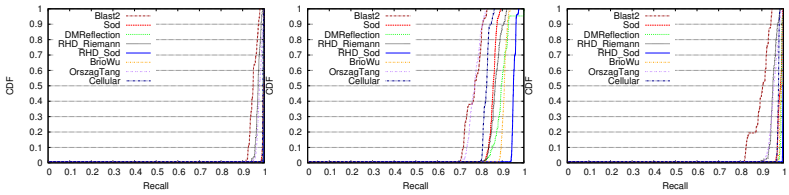


(a) Multiv. Beta 5-1 Re. (b) Multiv. Beta 1-10 Re. (c) Multiv. Normal Re.

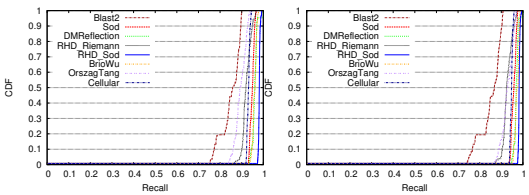


(d) Multiv. Beta 0.5-0.5 Re. (e) Multiv. Uniform Re.

Figure 11: Recall results for multivariate interpolation



(a) AID Beta 5-1 Re. (b) AID Beta 1-10 Re. (c) AID Normal Re.



(d) AID Beta 0.5-0.5 Re. (e) AID Uniform Re.

Figure 12: Recall results for AID

positive iterations (iterations in which at least one false positive occurs) to all iterations. Recall is the number of injected errors over the number of detected errors.

4.2.1. False Positive Rate and Sensitivity

Figures 7 8, and 9 show the cumulative distribution functions (CDFs) of the false positive rate (depicted as FP-rate) under our spatial, temporal, and spatiotemporal detectors; multivariate interpolation; and AID. The results are collected by running applications on 128 processes. We report results where the training size is 2 (the effect of the training size will be discussed later). The false positive rate is defined as the number of false positive iterations (iterations that have at least one false positive detected) over the total number of iterations. The false positive rate is quite useful in assessing the precision of a detector. As shown in the figures, our detectors outperform multivariate interpolation and achieves false positive rates close to those of AID. In particular, except for the Beta 5-1 distribution, our detectors achieve a false positive rate of less than 1%. The Beta 5-1 distribution is to stress our detectors, and, even under stress, our detectors achieve less than 2% false positive rate on average. Multivariate interpolation performs poorly especially because of the overly large detection range. Although we improved on the detection range presented in [4], it still exhibits a 4-17% false positive rate on average. Among our detectors, the spatiotemporal detector outperforms the temporal and spatial detectors because the prediction is done based on both time and space information, which closely reflects the actual computation.

We propose using the spatial or spatiotemporal algorithm. The spatial algorithm has 0% memory cost and is on par with AID in terms of detector performance. Combined with 5% performance overhead on average, our detector is lightweight and almost as efficient as AID. On the other hand, AID incurs up to 52% memory overhead due to the need for retaining the past values. AID will be prohibitive for many applications, whereas our solution does not have high memory or performance costs.

The spatiotemporal algorithm is our best-performing algorithm. Although it incurs some memory penalty due to the single past snapshot, it is still more memory efficient than AID, which requires up to four past snapshots.

Figures 10 11, and 12 present the CDFs of the detection sensitivity (recall) for the

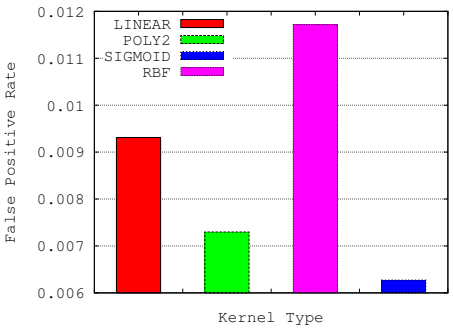
benchmarks for all detectors. The results are with 128 processes and training sets of size 2. Recall is defined as the fraction of the true positives detected over all SDCs experienced/injected. Our spatial, temporal, and spatiotemporal detectors achieve greater than 90% and up to 99% recall as AID with error distributions other than Beta 1-10. This distribution injects errors sparsely. As a result, the recall is lower than that of other distributions. With this distribution, AID achieves 85% recall, and our detectors achieve 79% on average. Multivariate interpolation achieves 77%-99% with Beta 5-1 and Beta 1-10 at the low and high end, respectively. The key reason that our detectors outperform multivariate interpolation is twofold: (1) more precise data prediction and (2) more accurate detection range estimated. Among our spatial, temporal, and spatiotemporal detectors, the recall performance is close to each other.

In our evaluation, we also study four different kernels: linear, polynomial with degree 2, radial basis, and sigmoid functions for our SVM-based SDC detectors. According to the results, no correlation exists between recall and the kernel type. Across applications kernels can incur relatively high or low recall (we still recommend RBF as it often achieves relatively high recall). However, this is not the case for the false positive rate. Sigmoid and polynomial kernels consistently lead to the lower false positive rate. We suspect the reason is that data evolve nonhomogeneously among neighbors. Figure 13(a) shows the effect of the kernel type on the false positive rate (representative figure).

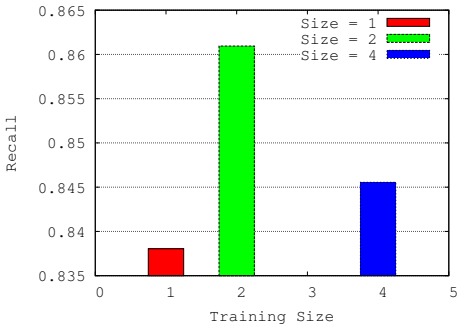
When we evaluate the effect of the training size on the false positive rate and recall, we cannot infer any relationship between false positive rate and training size based on the experimental data. With recall, we find an almost universal correlation. Figure 13(b) illustrates the effect of the training size on the recall (representative figure). Notably, when only one data point is used in the training set, the recall is lowest. Recall is highest when two data point are used in the training set. Two data points seem to be optimal, providing enough information while causing relatively low noise.

4.2.2. Prediction Errors

Figure 14 shows the normalized prediction error of multivariate interpolation and AID in comparison to our solution. Because the prediction errors of spatial, temporal,



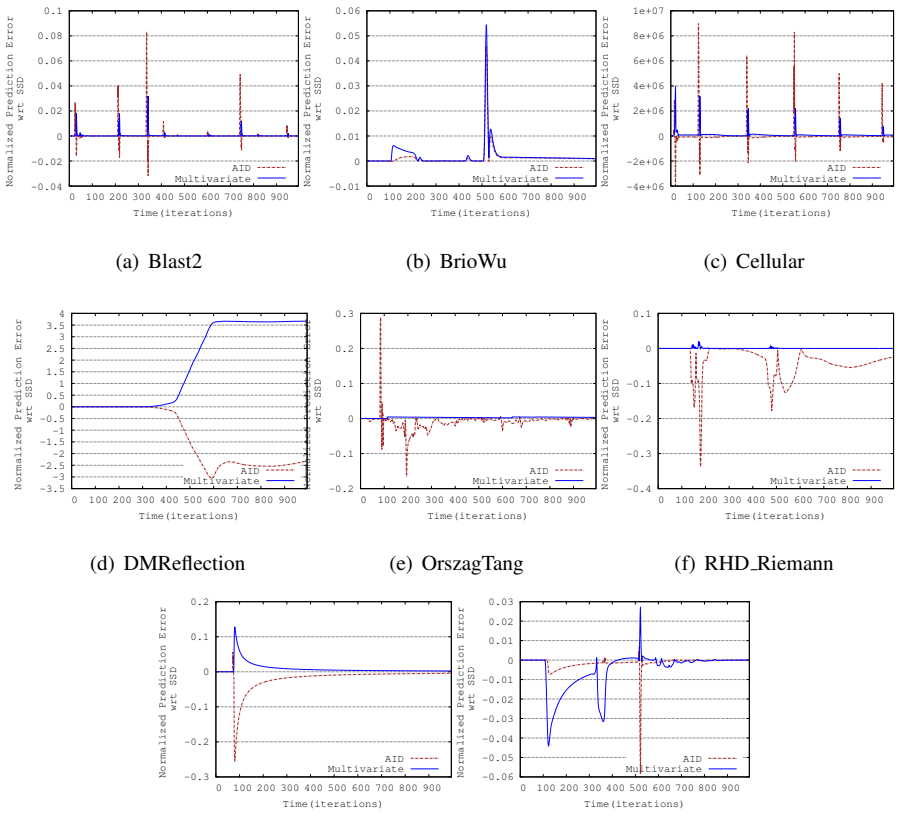
(a) Kernel Effect



(b) Training Size Effect

Figure 13: Effect of kernel type and training size

and spatiotemporal detectors are close, we choose and show the prediction error of the spatial detector. Specifically, the brown dotted curve refers to the difference of the prediction error between AID and our detectors (negative value means AID leads to smaller prediction errors than our detectors). Meanwhile, the blue solid curve refers to the difference of prediction errors between multivariate interpolation and our detectors (negative value means multivariate interpolation leads to smaller prediction errors). We include the comparison for one state variable and omit others for brevity. We see that detectors' behavior changes across benchmarks. No detector always outperforms others on prediction errors. Yet, in most cases, AID and our detectors outperform multivariate interpolation. Results show that the deviation of prediction error between AID and our detectors is larger than that of our detectors and multivariate interpolation.



(a) Blast2 (b) BrioWu (c) Cellular

(d) DMReflection (e) OrszagTang (f) RHD_Riemann

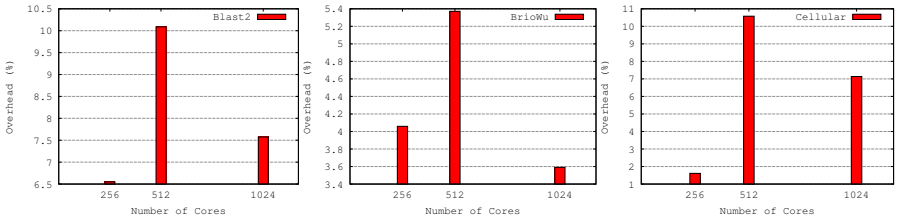
(g) RHD_Sod (h) Sod

Figure 14: Prediction error comparison

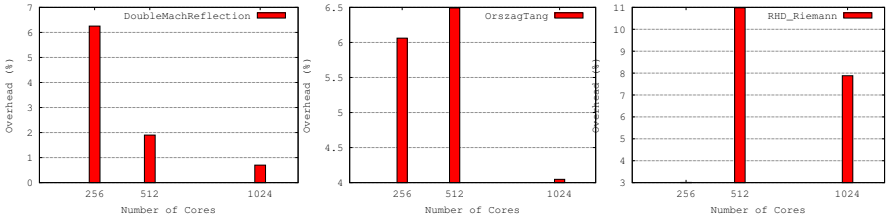
This reason is that the AID predictor only is based on the temporal evolution of data.

4.2.3. Computation Overheads

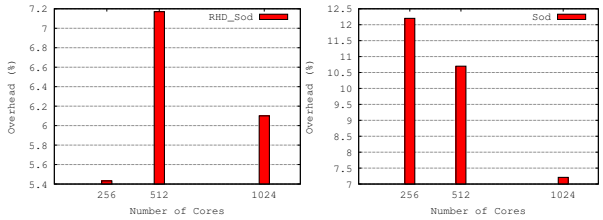
We now present the computation time overheads of our detectors. As the performance overhead does not vary much among different types of regression, we choose and show the performance overheads of the spatial detector. We report the averages over all processes. Figure 15 shows the computation time overheads (in percentages) with 256, 512, and 1,024 cores. From 512 to 1,024 cores, we see a decreasing trend in overheads. When 1,024 cores are utilized, all overheads are less than 8% and are 5% on average. From the results, we see that our solution is both lightweight and efficient.



(a) Blast2 (b) BrioWu (c) Cellular



(d) DMReflection (e) OrszagTang (f) RHD_Riemann



(g) RHD_Sod (h) Sod

Figure 15: Computation time overheads

From the figures, we can see that 512 core execution incurs the highest overhead in some cases. We suspect that the execution overhead may be unstable to a certain extent because it is related to the efficiency of the context switch in time slices and memory management.

4.2.4. Detailed Discussion

Support Vectors as Nonparametric Methods. As opposed to Gaussian processes, SVMs are parametric methods whose parameters are usually optimized through Bayesian techniques or cross validation. However, because the ε corresponds to the impact error bound and we choose not to perform any cross validation for the remaining

parameters in our case, such as regularization parameter γ or kernel parameter σ (both are set to one), to be efficient, support vector regression has essentially become a non-parametric method achieving good performance. On mission-critical situations, some computation cost can be sacrificed, and cross validation can be performed for the remaining parameters. We will investigate parameter optimization as future work.

Case with Sigmoid Kernels. As discussed by Schölkopf [25], choosing the appropriate capacity control is more important than selecting the type of kernels used in support vector learning. However, the performance of sigmoid kernels cannot be overlooked. Experimental data show that when sigmoid kernels are used, the maximum prediction error (less variance) is lower relative to that of the other kernels. Consequently, the false positive rate is relatively lower.

5. Related Work

Research on SDC mitigation can be categorized mainly into three different categories: runtime analysis techniques, replication of computation, and algorithm-based fault tolerance (ABFT) techniques.

Runtime data analysis recently has gained attention in the HPC community. Studies [26, 27, 28] investigate and compare different prediction methods, such as linear curve fitting or autoregressive moving average (ARMA) models, to detect SDCs. They convert the problem of detecting SDC into a next-step prediction problem. Sharma et al. [29] use temporal features of data (in addition to spatial features) and provide a tailored SDC detector for stencil applications, where they use SVMs as a linear function approximator. As a spatial technique, spatial SVM detector (SSD) [30] incurs low memory cost while having low computation overhead. The Sirius [31] is a neural-network-based offline SDC detection tool. Training is performed offline and thus Sirius fundamentally differs from online techniques. Offline techniques are limited by the coverage of training datasets. On the memory side, Subasi et al. [32] propose a hardware-accelerated cyclic redundancy checks (CRCs)-based mechanism for SDCs occurring in memory of HPC applications.

Replication-based schemes [33] can be deployed for mission-critical situations. In

such contexts, double or triple redundancy of computation is performed to detect SDCs by comparing the results of replica computations. The inherent drawback of the replication is its high power/energy cost. For example, with double redundancy, the cost is 100%. Partial replication [34] has been proposed to decrease costs while providing the required level of reliability. Although partial replication is promising, it may not be applicable for certain HPC systems, mainly because errors may not be reproducible for some systems, such as heterogeneous systems.

ABFT [35, 36, 37] techniques are tailored solutions to specific numerical algorithms. As a result, they are usually efficient. However, they fundamentally lack the ability to apply to algorithms other than the specific numerical or algebraic kernel they are designed for.

Fail-stop errors are out of the scope of this study. This type of error usually is mitigated by checkpoint/restart. For instance, FTI [38] is a scalable checkpoint/restart scheme that offers multi-level checkpointing. Moreover, there are techniques specific to programming models, such as [39], [40], [41], and [42], which target task-based computations.

6. Conclusion

In this work, we propose a set of novel lightweight SDC detectors based on online support vector regression. Our detectors are built on spatial, temporal, and spatiotemporal training sets. We have analyzed the capability of our detectors compared with state-of-the-art detectors and note our detectors perform on par with them. In addition, experimental evaluation shows that our detectors incur low performance overhead (5% on average.). Moreover, experiments with eight real-world HPC applications show that for most of the failure distributions and applications, detection sensitivity is high, up to 99%, and the false positive rate is low, less than 1% - except being under stress. Finally, our implementation supports a diverse range of HPC applications in both Fortran or C.

Acknowledgements

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. In addition, this material is based upon work supported by the National Science Foundation under Grant No. 1619253, and also by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Lucy Nowell, under contract number DE-AC02-06CH11357 (DOE Catalog project) and in part by the European Union FEDER funds under contract TIN2015-65316-P.

References

[1] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, 1995.

[2] SDC detection framework and library. [online]. available at : https://collab.cels.anl.gov/display/esr/aid.

[3] S. Di, F. Cappello, Adaptive impact-driven detection of silent data corruption for HPC applications, IEEE Transactions on Parallel and Distributed Systems doi:10.1109/TPDS.2016.2517639.

[4] L. Bautista-Gomez, F. Cappello, Detecting and correcting data corruption in stencil applications through multivariate interpolation, in: 2015 IEEE International Conference on Cluster Computing (CLUSTER), 2015, pp. 595–602.

[5] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. S. Ünsal, J. Labarta, A. Cristal, 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, 2016, pp. 413–424. doi:10.1109/CCGrid.2016.33.
URL https://doi.org/10.1109/CCGrid.2016.33

[6] L. Cao, F. Tay, Support vector machine with adaptive parameters in financial time series forecasting, IEEE Transactions on Neural Networks 14 (6) (2003) 1506–1518. doi:10.1109/TNN.2003.820556.

[7] T. Farooq, A. Guergachi, S. Krishnan, Chaotic time series prediction using knowledge based green's kernel and least-squares support vector machines, in: 2007 IEEE International Conference on Systems, Man and Cybernetics, 2007, pp. 373–378. doi:10.1109/ICSMC.2007.4414023.

[8] T. Raicharoen, C. Lursinsap, P. Sanguanbhokai, Application of critical support vector machine to time series prediction, in: Proceedings of the 2003 International Symposium on Circuits and Systems, Vol. 5, 2003, pp. V–741–V–744 vol.5. doi:10.1109/ISCAS.2003.1206419.

[9] Y. Fan, P. Li, Z. Song, Dynamic least squares support vector machine, in: The Sixth World Congress on Intelligent Control and Automation, Vol. 1, 2006, pp. 4886–4889. doi:10.1109/WCICA.2006.1713313.

[10] A. J. Smola, B. Schölkopf, A tutorial on support vector regression, Statistics and Computing 14 (3) (2004) 199–222. doi:10.1023/B:STCO.0000035301.49549.88.
URL http://dx.doi.org/10.1023/B:STCO.0000035301.49549.88

[11] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297. doi:10.1007/BF00994018.
URL http://dx.doi.org/10.1007/BF00994018

[12] H. W. Kuhn, A. W. Tucker, Nonlinear programming, in: Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1951, pp. 481–492.

[13] Z. Ovari, Kernels, eigenvalues and support vector machines (2000).

[14] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27.

[15] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, S. Matsuoka, FTI: High performance fault tolerance interface for hybrid systems, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, New York, NY, USA, 2011, pp. 32:1–32:32. doi:10.1145/2063384.2063427.
URL http://doi.acm.org/10.1145/2063384.2063427

[16] P. Colella, P. R. Woodward, The piecewise parabolic method (ppm) for gas-dynamical simulations, Journal of Computational Physics 54 (1) (1984) 174–201.
URL http://www.sciencedirect.com/science/article/pii/0021999184901438

[17] G. A. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, Journal of Computational Physics 27 (1) (1978) 1–31.
URL http://www.sciencedirect.com/science/article/pii/0021999178900232

[18] J. M. Martí, E. Müller, Numerical hydrodynamics in special relativity, Vol. 6, 2003.

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

[20] M. Brio, C. Wu, An upwind differencing scheme for the equations of ideal magnetohydrodynamics, Journal of Computational Physics 75 (2) (1988) 400–422.
URL http://www.sciencedirect.com/science/article/pii/0021999188901209

[21] S. A. Orszag, C.-M. Tang, Small-scale structure of two-dimensional magnetohydrodynamic turbulence, Journal of Fluid Mechanics 90 (1979) 129–143.
URL http://journals.cambridge.org/article_S002211207900210X

[22] F. X. Timmes, M. Zingale, K. Olson, B. Fryxell, P. Ricker, A. C. Calder, L. J. Dursi, H. Tufo, P. MacNeice, J. W. Truran, R. Rosner, On the cellular structure of carbon detonations, The Astrophysical Journal 543 (2) (2000) 938.
URL http://stacks.iop.org/0004-637X/543/i=2/a=938

[23] Fusion cluster. [online]. available at : http://www.lcrc.anl.gov/.

[24] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, H. Tufo, FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes, apjs 131 (2000) 273–334. doi:10.1086/317361.

[25] B. Schölkopf, Support vector learning (1997).

[26] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, 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, New York, NY, USA, 2015, pp. 275–278. doi:10.1145/2749246.2749253.
URL http://doi.acm.org/10.1145/2749246.2749253

[27] L. A. B. Gomez, F. Cappello, Detecting and correcting data corruption in stencil applications through multivariate interpolation, in: 2015 IEEE International Conference on Cluster Computing, 2015, pp. 595–602. doi:10.1109/CLUSTER.2015.108.

[28] S. Di, E. Berrocal, 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, 2015, pp. 271–280. doi:10.1109/CCGrid.2015.17.

[29] V. Sharma, G. Gopalakrishnan, G. Bronevetsky, Detecting soft errors in stencil based computations, in: The 11th Workshop on Silicon Errors in Logic - System Effects, 2015.

[30] O. Subasi, S. Di, L. Bautista-Gomez, P. Balaprakash, O. S. Ünsal, J. Labarta, A. Cristal, 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, 2016, pp. 413–424. doi:10.1109/CCGrid.2016.33.
URL http://dx.doi.org/10.1109/CCGrid.2016.33

[31] T. E. Thomas, A. J. Bhattad, S. Mitra, 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.

[32] O. Subasi, O. S. Ünsal, J. Labarta, G. Yalcin, A. Cristal, Crc-based memory reliability for task-parallel HPC applications, in: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS, Chicago, IL, USA, May 23-27, 2016, 2016, pp. 1101–1112. doi:10.1109/IPDPS.2016.70.
URL http://dx.doi.org/10.1109/IPDPS.2016.70

[33] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, 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, CA, USA, 2012, pp. 78:1–78:12.
URL http://dl.acm.org/citation.cfm?id=2388996.2389102

[34] O. Subasi, J. Arias, O. Unsal, J. Labarta, A. Cristal, Programmer-directed partial redundancy for resilient HPC, in: Proceedings of the 12th ACM International Conference on Computing Frontiers, CF '15, New York, NY, USA, 2015, pp. 47:1–47:2. doi:10.1145/2742854.2742903.
URL http://doi.acm.org/10.1145/2742854.2742903

[35] M. Turmon, R. Granat, D. Katz, J. Lou, Tests and tolerances for high-performance software-implemehted fault detection, IEEE Transactions on Computers 52 (5) (2003) 579–591. doi:10.1109/TC.2003.1197125.

[36] E. Ciocca, I. Koren, Z. Koren, C. M. Krishna, 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, Washington, DC, USA, 2004, pp. 43–48.
URL http://dl.acm.org/citation.cfm?id=977407.978747

[37] J. Sloan, R. Kumar, 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, Washington, DC, USA, 2012, pp. 1–12.
URL http://dl.acm.org/citation.cfm?id=2354410.2355166

[38] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, S. Matsuoka, FTI: high performance fault tolerance interface for hybrid systems, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC, 2011, pp. 32:1–32:32. doi:10.1145/2063384.2063427.
URL http://doi.acm.org/10.1145/2063384.2063427

[39] O. Subasi, J. Arias, O. Unsal, J. Labarta, A. Cristal, Nanocheckpoints: A task-based asynchronous dataflow framework for efficient and scalable checkpoint/restart, in: 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2015, pp. 99–102.

[40] O. Subasi, F. Zyulkyarov, O. S. Unsal, J. Labarta, Marriage between coordinated and uncoordinated checkpointing for the exascale era, in: 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, New York, NY, USA, August 24-26, 2015, 2015, pp. 470–478. doi:10.1109/HPCC-CSS-ICESS.2015.150.
URL http://dx.doi.org/10.1109/HPCC-CSS-ICESS.2015.150

[41] T. V. Martsinkevich, O. Subasi, O. S. Unsal, F. Cappello, J. Labarta, Fault-tolerant protocol for hybrid task-parallel message-passing applications, in: IEEE International Conference on Cluster Computing, CLUSTER, 2015, pp. 563–570.

[42] O. Subasi, T. Martsinkevich, F. Zyulkyarov, O. Unsal, J. Labarta, F. Cappello, Unified fault-tolerance framework for hybrid task-parallel message-passing applications, The International Journal of High Performance Computing Applications 0 (0) (2016) 1094342016669416.
URL http://dx.doi.org/10.1177/1094342016669416