Spatial Support Vector Regression to Detect Silent Errors in the Exascale Era

Omer Subasi,[12] Sheng Di,[3] Leonardo Bautista-Gomez,[3] Prasanna Balaprakash,[3]
Osman Unsal,[1] Jesus Labarta,[12] Adrian Cristal,[124] Franck Cappello[3]
[1]Barcelona Supercomputing Center, [2]Universitat Politecnica de Catalunya, Barcelona, Spain
[3]Argonne National Laboratory, Lemont, Illinois, USA
[4]IIIA - Artificial Intelligence Research Institute CSIC - Spanish National Research Council, Spain
{omer.subasi, osman.unsal, jesus.labarta, adrian.cristal}@bsc.es
{sdi1, leobago, 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 low-memory-overhead SDC detector, by leveraging epsilon-insensitive support vector machine regression, to detect SDCs that occur in HPC applications that can be characterized by an impact error bound. The key contributions are three fold. (1) Our design takes spatial features (i.e., neighbouring data values for each data point in a snapshot) into training data, such that little memory overhead (less than 1%) is introduced. (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 our detector can achieve the detection sensitivity (i.e., recall) up to 99% yet suffer a less than 1% of false positive rate for most cases. Our detector incurs low performance overhead, 5% on average, for all benchmarks studied in the paper. Compared with other state-of-the-art techniques, our detector exhibits the best tradeoff considering the detection ability and overheads.

I. 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 in the exascale era. Silent data corruptions (SDCs) or silent errors are one of the most significant problems in terms of 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 in order to guarantee the correctness of the HPC application results.

In this work, we propose a novel and efficient SDC detector leveraging machine learning. In particular, we use Support Vector Machine (SVM) supervised learning method to detect SDCs. SVMs are effective because their non-linear nature detects complex SDCs. To decrease the memory overheads, we employ spatial SVM which uses neighbouring data points instead of a time-series based temporal method with relatively higher memory overhead. We term our implementation spatial support-vector-machine detector (or SSD in short). Our strategy focuses on the analysis of the spatial features, namely the dynamic spatial support vector machine 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 [33] and (2) checking the observed value for each data point to see whether it falls inside the confidential value range. Since the analysis makes use of only the current-time-step data, the detector incurs little memory overhead (less than 1%). *Considering the detection ability, performance (5% on average) and memory overheads, SSD provides the best tradeoff compared with the state of the art techniques.*

Existing research on SDC detection [5, 12, 14] is based mainly on temporal curve-fitting models. In particular, Adaptive Impact-Driven detector (AID) [14] adopts different types of curve fitting. In that work, solid experiments with real-world HPC applications were performed to validate that the *impact error bound* with respect to the runtime value range can be used to characterize the impact of SDCs on the execution results for most applications. Although AID can achieve high recall (ratio of detected SDCs over all SDCs) and low false positive rate[3] on SDC detections, the memory overhead can be considerable in some cases. The reason is that as a temporal scheme, AID maintains 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 [14], the overall memory cost compared with the total application's memory usage can be

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

[2]This parameter refers to the insensitivity, that is, the amount of deviations tolerated by the SVM during the process of regression.

[3]False positive rate is defined as the ratio of the number of time steps with false alarms to the total number of time steps in the execution. The larger false positive rate, the lower execution performance, so false positive rate is expected to be minimized.

more than 50% [14] in some cases.

To design and implement our data analytic detector, 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 a support vector machine, and the correspondence is diverse since the impact error bound 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 careful a tradeoff. Moreover, the detection range formulation should be generic enough to fit as many HPC applications as possible.

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

- We design a dynamic spatial SVM-based SDC detector, namely SSD, for HPC applications. To the best of our knowledge, this is the first machine learning based scheme leveraging spatial SVM to detect SDCs for HPC applications. The predictor can incorporate spatial features while maintaining a dynamic loss function, such that our detector suffers little memory overhead.
- 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 SSD library supporting a wide range of HPC applications. It can be downloaded from [2].
- We evaluate SSD by eight real-world HPC applications and compare it with the state-of-the-art SDC detector AID [14] and multivariate interpolation [3] with five different error distributions. Experiments show that our detector can achieve the detection sensitivity (i.e., recall) up to 99% yet suffer a less than 1% of false positive rate for most cases. Our detector also incurs low performance overhead, 5% on average, for all benchmarks studied in the paper.

The rest of the paper is organized as follows. In Section II we present the background for this study. In Section III we discuss the design of our detector in detail. In Section IV we evaluate our detector in terms of detection and prediction capability, and performance and memory overheads. In addition, we compare the performance of our detector to that of the AID algorithm and multivariate interpolation. In Section V we present the state of the art in SDC mitigation research. In Section VI we provide concluding remarks.

II. Background

In this section, we provide an overview of support vector machines (SVMs), which is the core technique to be used in our solution. We then continue with data prediction types. After that, we discuss the impact-driven SDC detection,

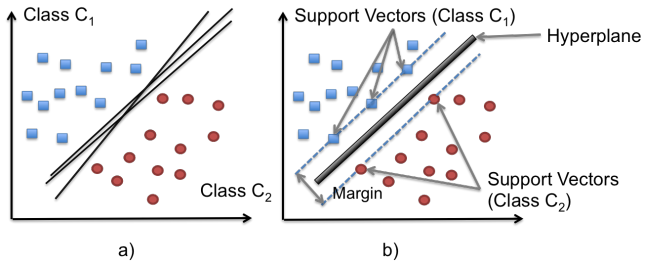


Figure 1. SVM classification compared with other linear classifiers

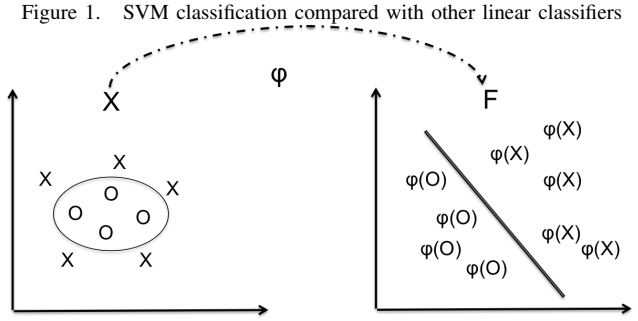


Figure 2. SVM kernel trick to tackle nonlinear data problem

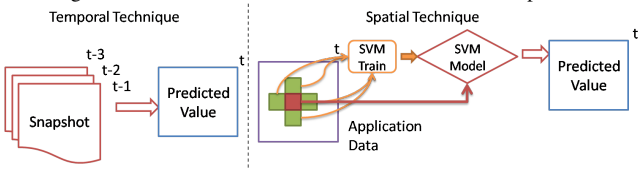


Figure 3. Temporal (e.g. AID) vs. spatial prediction (SVM predictor)

which is the fundamental detection model we will work on. Finally, we close the section with an overview of the AID algorithm and multivariate interpolation, because they are the most related research with our work.

A. Support Vector Machines: An Overview

SVMs were originally designed for pattern classification problems by Vapnik and coworkers [33], and they have been widely applied to other fields for function approximation signal processing, regression and time series prediction [7, 15, 16, 23]. The key feature of SVMs is that they leverage *structural risk minimization* principle to find a decision function with a good generalization capacity. The solution to a particular problem depends only on a subset of the training data points called support vectors [33]. 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.

In order to handle nonlinearity, a technique called *kernel trick* is applied in SVMs. The input space points are mapped to a high-dimensional *feature space* via nonlinear mapping,

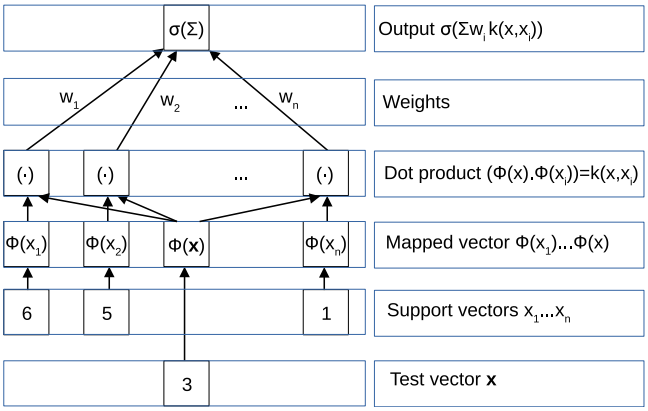


Figure 4. SVM architecture. The input **x** vector and the support vectors x_i (which are digits in this example) are mapped by Φ into a feature space where dot products are computed. Kernel k is used in practice to compute the dot products. The results are linearly combined by weights w_i which are found solving a quadratic program. Finally, the sign function $\sigma(\Sigma) = sign(\Sigma)$ is used to classify the input **x** vector.

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.

SVMs can also be used for regression problems, such as ε-insensitive SVM regression. The basic idea is 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 [33].

Figure 4 shows the basic architecture of SVMs and how they work [4]. The input **x** vector and the support vectors x_i are mapped by Φ into a feature space. Then the dot products are computed by the use of a proper kernel function. All dot products are then 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 **x**.

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 appears in dot products both in the decision function and training algorithm. This enables SVMs to use the aforementioned kernel trick to tackle the nonlinearly separable data as opposed to linear techniques. (2) The second property is related to the *kernel trick* technique where the implicit mapping is taken rather than explicit computation of the kernel itself. This prevents costly processing of high-dimensional data. (3) The third property of SVMs is the 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 convergences of the SVM algorithm. SVMs has the *convexity property* which makes them solvable in polynomial time. This property is able to effectively avoid the local minimum solution, guaranteeing the global optimum of the solution.

The reader is advised to see [33] and [11] for detailed general discussion on SVMs, and Smola and Schölkopf [28] in particular for SVM regression.

[4]This example is adopted from Smola and Schölkopf [28].

B. 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. This causes significant memory cost. In contrast, in spatial prediction only the neighbouring data points are used to make a prediction. Figure 3 shows the techniques. In particular, it depicts our SVM-based predictor. Neighbouring points are used to obtain an SVM model which is then used to predict a value for the target point.

C. Impact-Driven SDC Detection

Research by Di and Cappello [14] demonstrates that not all SDCs may impact the application execution results significantly, and one should mainly focus 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 [14]) manifested that the impact error bound = 0.00078125 (or 0.0001 some times) is fairly enough for detecting SDCs for most of applications. In this work, we leverage such an impact error bound to devise our SVM-based detector.

D. AID: Adaptive Impact-Driven SDC Detector

AID [14] is an outstanding SDC detector, which allows different processes to dynamically select the best-fit curve fitting models with the minimum prediction errors based on their runtime data. The curve fitting models include last-state, linear, and quadratic curve-fitting models. The AID algorithm incorporates types and periodically selects the best curve fitting that has 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 a SDC step and correction operation (such as restarting application from

one previous checkpoint file) will be triggered accordingly; otherwise, the execution will not be interrupted.

E. Multivariate Interpolation

We now take an overview of the multivariate interpolation method proposed by Bautista-Gomez and Cappello [3]. 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 [3] chose linear interpolation for simplicity. For 3D space, for instance, 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 [3], the normal range is acquired at the beginning of the execution by estimating the maximum error, which then will be used until the end of the execution.

III. Dynamic SVM-Based SDC Detector

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

A. Formalization of The SVM Predictor

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 [33], our 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 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 in order 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

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

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

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

Table I
Training Set w.r.t. Dimension and Size

Size	1D Feature Vectors	2D Feature Vectors
1	x_{i-1}	x_{i-1}
2	x_{i-1}, x_{i+1}	x_{i-1}, x_{i+1}
4	x_{i-1}, x_{i+1}, x_{i-2}, x_{i+2}	x_{i-1}, x_{i+1}, x_{j-1}, x_{j+1}

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 tradeoff between the flatness of f and the amount of deviations larger than $\theta_I r_i^j$ that is tolerated. The criticality coefficients when provided convey the relative vulnerabilities of variables. The higher the coefficient, the higher the penalty is. 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 [33] is

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

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

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

Equation (3) presents a convex quadratic problem that is solved by Lagrangian multipliers and Karush-Kuhn-Tucker (KKT) conditions [19] 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.

Table I shows the feature vectors of our detector, that is, the neighboring data points used in the training set with respect to the different dimension and training sizes.

We close this section by discussing the admissibility of kernel functions for SVMs and the kernels that we use in our design. Mercer's condition provides a necessary and sufficient condition for a kernel to be admissible to be used so that the input patterns are mapped to the feature space:

Mercer 1909. 1: Assume $k \in L_\infty(X^2)$ such that the integral operator $T_k : L_2(X) \to L_2(X)$

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

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 $\parallel \psi_j \parallel_{L_2} = 1$, and let $\overline{\psi_j}$ denote its complex conjugate. Then

Table II
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

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, that is, it can be written as a dot product in some feature space:

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

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 [22], 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 that we study are linear ($p = 1$, $c = 0$), polynomial ($p = 2$, $c = 0$), RBF ($\sigma = 1$), and sigmoid ($\vartheta = 0$, $\nu\langle x,x'\rangle = \langle x,x'\rangle$) kernels.

Remark: Our solution is also applicable to applications that are not characterized by an impact error bound. Since these applications have no specified error bound, there will be no corresponding insensitivity parameter ε for the support vector predictor. To obtain an appropriate insensitivity parameter, cross-validation, such as 10-fold, can be used in the beginning of the application execution. The obtained value can be used for the rest of the execution. Di and Cappello [14] find that only three applications fall into this category. Since these applications are not in the scope of this study, we will report the results for them in future work.

B. The Formalization of Detection Range and Algorithm

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

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

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

Algorithm 1: SVM-Based Detector

```
Data: Current step t, data value V(t), relative error bound θ
Result: Boolean indicating whether SDC is present
begin
    Compute range r /* periodically done */
    isDetected ← false
    SVM_SetEpsilon(θ, r)
    SVM_TrainWithNeighboringPoints()
    X(t) ← SVM_Predict(t)
    ρ(t) ← calculateRadius(feedback(t), η(t), θr)
    isDetected ← checkInRange(ρ(t), X(t), V(t))
    if (isDetected) then
        Trigger some operation for data recovery.
    end
end
```

The rationale behind the above 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 whether the SDCs will lead to a significant impact on the execution results (details can be found in [14]). The design of the feedback term *feedback*$(t-1)$ is to adapt to the possible sharp data changes, which will lead to large prediction errors accordingly.

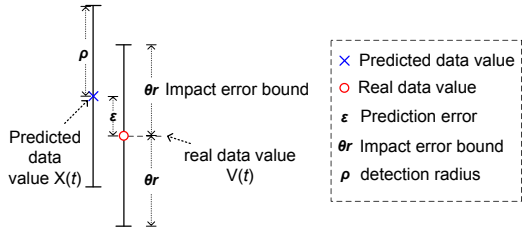


Figure 5. Detection model

Our detection model is illustrated in Figure 5. 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 among processes, and the epsilon parameter of the support vector machine (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 every data point's neighboring data points. The prediction of the SVM and the computed radius are used to calculate the normal range. The observed value for each data point is checked whether it is in the normal range. If not, the current time step will be considered with SDCs.

C. Implementation

We implement our detector following our design based on LibSVM [8]. We integrate our detector with the FTI library [4] 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 such that a broad range of HPC applications can utilize our detector. The library is available to download from [2]. To use our detector, users need to follow four simple steps where they annotate their applications: (1) initialize the detector 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.

IV. Evaluation

In this section we detail the experimental evaluation of our detector. Our evaluation is two fold. First, we evaluate the false positive rate (FP-rate) and detection sensitivity (recall) of our detector. We additionally evaluate the effect of different parameters on these metrics. Moreover, we compare the detection results of our detector with that of the AID algorithm [14] and multivariate interpolation [3]. Second, we evaluate the detection overhead of our detector. We discuss the experimental setup first and then present the experimental results.

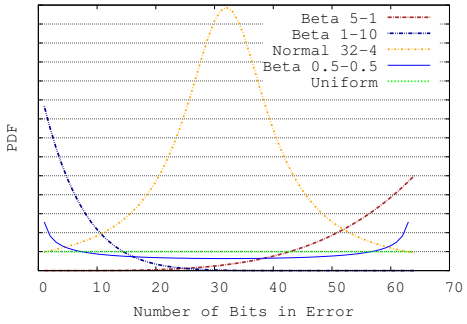


Figure 6. Distributions used in the experimental evaluation

A. Experimental Setup

We perform our experiments using the Fusion [1] cluster at Argonne National Laboratory. Table III shows the applications that we use in our evaluation from the FLASH package [18]. For each application we protect *state variables*, which are checked at every main iteration of the applications. When assessing the detection sensitivity, we use the relative impact error bounds recommended in [14]. 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; that is, we treat all state variables to have the same significance. In fault injection experiments, each single case is repeated 10× and the averages are reported.

Since we have no information about how silent errors will exhibit themselves, we use five different error distributions (as shown in Figure 6) to cover reasonable scenarios that can occur in the exascale era and to assess the performance of our detector.

Beta Distributions: Beta distribution is typically used in control systems and population genetics. This class of distributions provides distributions that fit possible scenarios that can occur in the exascale timeframe by adjusting shape parameters. Formally the probability density function (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 which 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, as shown in Figure 6. Beta distribution with shape parameter $\alpha = 1$ and $\beta = 10$ represent the case where the probability density function (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-bitflip or double-bitflip errors are likely to be detected by the underlying hardware ECCs while multi-bitflip errors cannot be detected effectively by hardware ECCs. $\alpha = 0.5$ and $\beta = 0.5$ represent another possible case in which single-bitflip errors or all-bitflip errors are more common than the other types of bitflip errors. This case is included to reflect the cases with erratic behavior.

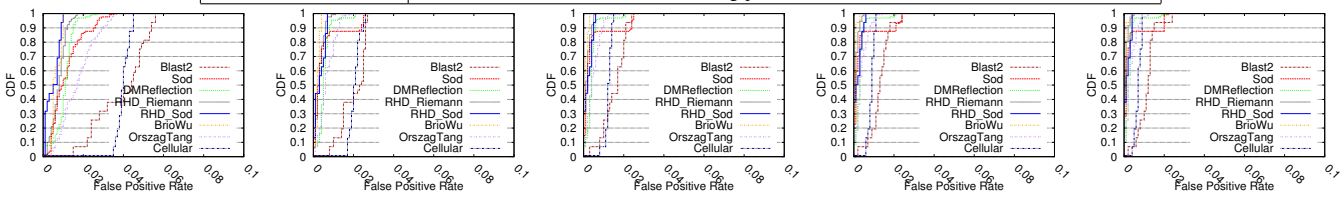
Normal Distribution: The central limit theorem implies that the number of errors should follow a normal distribution given that the flip event on each bit follows an independent and identical distribution. Therefore we include normal distribution to account for the case where errors are independent and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

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

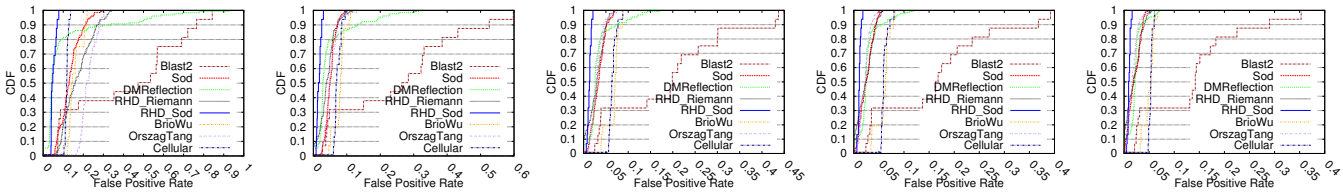
where μ is the mean and σ is the variance of the distribution. Without loss of generality, μ and σ are set to 32 and 4 respectively in our evaluation.

Table III
APPLICATIONS USED IN EVALUATION

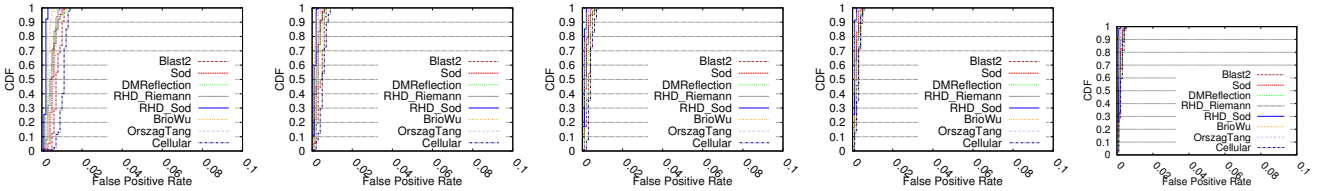
Name	Description
Blast2 [10]	Strong shocks and narrow features
SodShock [29]	Sodshock tube for testing compressible code's ability with shocks & contact discontinuities
DMReflection [10]	Double Mach reflection: an evolution of an unsteady planar shock on an oblique surface
RHD_Sod [20]	Relativistic Sod Shock-tube: involving the decay of 2-fluids into 3-elementary wave structures
RHD_Riemann2D [25]	Relativistic 2D Riemann: exploring interactions of four basic waves consisting of shocks, etc.
BrioWu [6]	Coplanar magneto-hydrodynamic counterpart of hydrodynamic Sod problem
OrszagTang [21]	Simple 2D problem that has become a classic test for MHD codes
Cellular [31]	Burn simulation: cellular nuclear burning problem



(a) SSD Beta 5-1 FP-r. (b) SSD Beta 1-10 FP-r. (c) SSD Normal 32-4 FP-r. (d) SSD Beta 0.5-0.5 FP-r. (e) SSD Uniform FP-r.

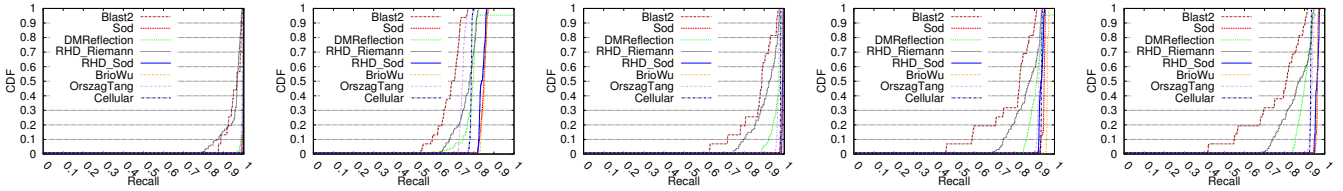


(f) Multiv. Beta 5-1 FP-r. (g) Multiv. Beta1-10 FP-r. (h) Multiv. Normal 32-4 FP-r. (i) Multiv. Beta 0.5-0.5 FP-r. (j) Multiv. Uniform FP-r.

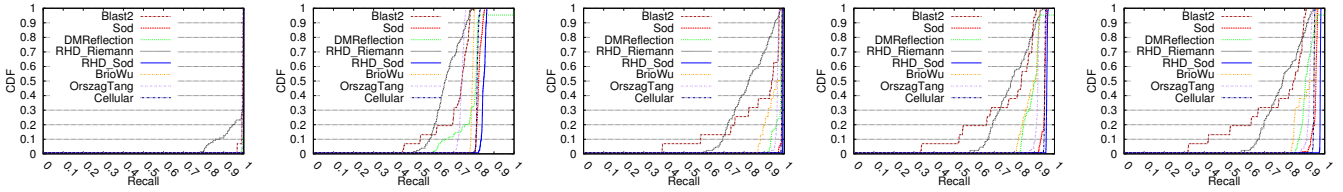


(k) AID Beta 5-1 FP-r. (l) AID Beta 1-10 FP-r. (m) AID Normal 32-4 FP-r. (n) AID Beta 0.5-0.5 FP-r. (o) AID Uniform FP-r.

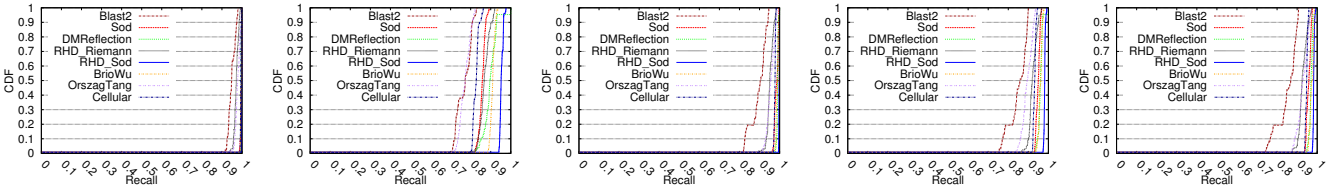
Figure 7. False positive rate results



(a) SSD Beta 5-1 Recall (b) SSD Beta 1-10 Re. (c) SSD Normal 32-4 Re. (d) SSD Beta 0.5-0.5 Re. (e) SSD Uniform Re.



(f) Multiv. Beta 5-1 Re. (g) Multiv. Beta 1-10 Re. (h) Multiv. Normal 32-4 Re. (i) Multiv. Beta 0.5-0.5 Re. (j) Multiv. Uniform Re.



(k) AID Beta 5-1 Re. (l) AID Beta 1-10 Re. (m) AID Normal 32-4 Re. (n) AID Beta 0.5-0.5 Re. (o) AID Uniform Re.

Figure 8. Recall results

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

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

Since 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 one can assume a uniform distribution on the number of bit flips.

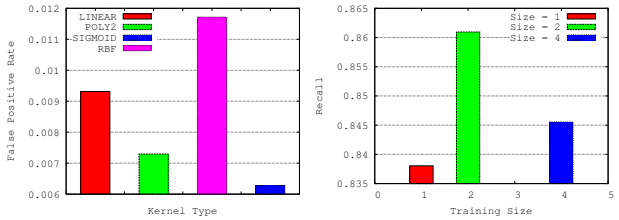
B. Experimental Results

1) False Positive Rate and Sensitivity: Figure 7 shows the cumulative distribution functions (CDFs) of the false positive rate (FP-rate) under our solution (SSD), multivariate interpolation, and AID respectively. The results are collected by running applications on 128 processes. The FP-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 FP-rate is very useful in assessing the precision of a detector. As shown in the figure, SSD outperforms multivariate interpolation and achieves false positive rates close to those of AID. In particular, except for the beta 5-1 distribution, SSD achieves an FP-rate less than 1%. The beta 5-1 distribution is to stress our detector; and even under stress, SSD achieves a 1.7% FP-rate on average. Multivariate interpolation performs poorly especially because of the over-large detection range. Although we improved on the detection range presented in [3], it still exhibits a 4-17% FP-rate on average.

Figure 8 presents the CDFs of the detection sensitivity (recall) for the benchmarks. Recall is defined as the fraction of the true positives that are detected over all SDCs experienced/injected. SSD achieves over 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 SSD achieves 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 SSD outperforms multivariate interpolation is two fold: (1) more precise data prediction (to be shown later) and (2) more accurate detection range estimated.

In our evaluation, we also evaluate four different kernels: linear, polynomial with degree 2, radial basis, and sigmoid functions for our SVM-based SDC detector. 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 since it often achieves relatively high recall). However, this is not the case for the FP-rate. Sigmoid and polynomial kernels consistently lead to the lower FP-rate. We suspect the reason is that data evolves nonhomogeneously among neighbors. Figure 9(a) shows the effect of the kernel type on the FP-rate (representative figure).

When we evaluate the effect of the training size on the FP-rate and recall, we cannot infer any relationship between FP-rate and training size based on the experimental data. With recall, we find an almost universal correlation. Figure 9(b) illustrates the effect of the training size on the recall (representative figure). We see that when only one neighbor is used in the training set, the recall is lowest. Recall is the highest when two neighbors are used in the training set. Two neighbors seem to be optimal; providing enough information while causing relatively low noise.

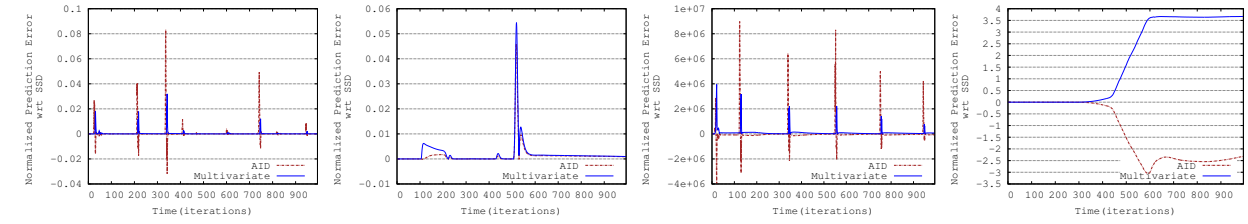


(a) Kernel Effect (b) Training Size Effect

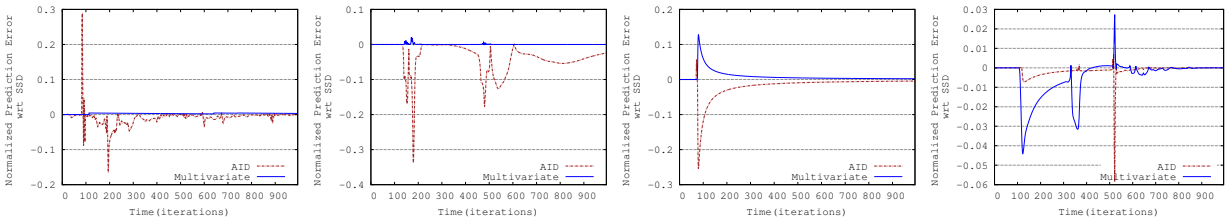
Figure 9. Effect of kernel type and training size

2) Prediction Errors and Range Evolution: Figure 10 shows the normalized prediction error of multivariate interpolation and AID in comparison to our solution SSD. Specifically, brown dotted curve refers to the difference of the prediction error between AID and SSD (negative value means AID leads to smaller prediction errors than SSD and vice versa); blue solid curve refers to the difference of prediction errors between multivariate interpolation and SSD (negative value means multivariate interpolation leads to smaller prediction errors and vice versa). We include the comparison for one state variable and omit others for brevity. We see that the behaviour of detectors changes across benchmarks. No detector always outperforms others on prediction errors (yet AID and SSD outperforms multivariate interpolation in most of cases). Results show that the deviation of prediction error between AID and SSD is larger than that of SSD and multivariate interpolation. This reason is that the AID predictor is based on the temporal evolution of data rather than spatial evolution.

In addition to the prediction errors, we also note the significance of the detection range (i.e., normal value range) in terms of the performance of a detector. We present how the detection range of SSD evolves over time. This behaviour is important and should be examined since detection range shows the adaptability of the detector to the data changes over time. Figure 11 shows the detection range evolution over the executions of the benchmarks. We plot the data of a representative process. We see that in all applications the detection range is dynamic as a result of the periodic aggregation of the value ranges and the computation of maximum prediction error estimate. When data changes

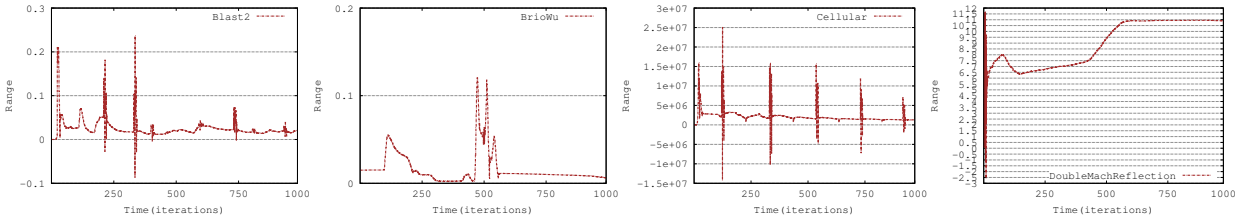


(a) Blast2 (b) BrioWu (c) Cellular (d) DMReflection

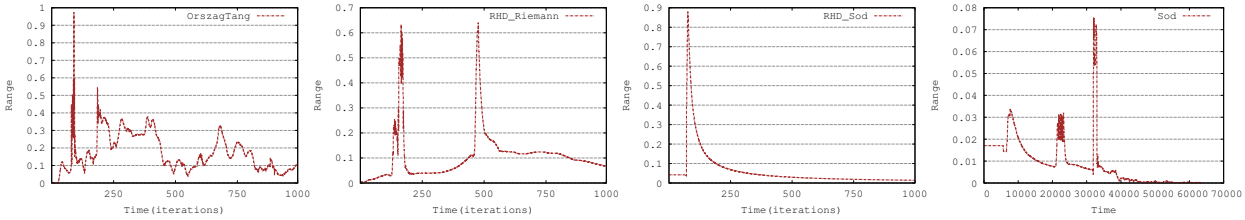


(e) OrszagTang (f) RHD_Riemann (g) RHD_Sod (h) Sod

Figure 10. Prediction error comparison



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



(e) OrszagTang (f) RHD_Riemann (g) RHD_Sod (h) Sod

Figure 11. Detection range evolution

rapidly, so does the detection range.

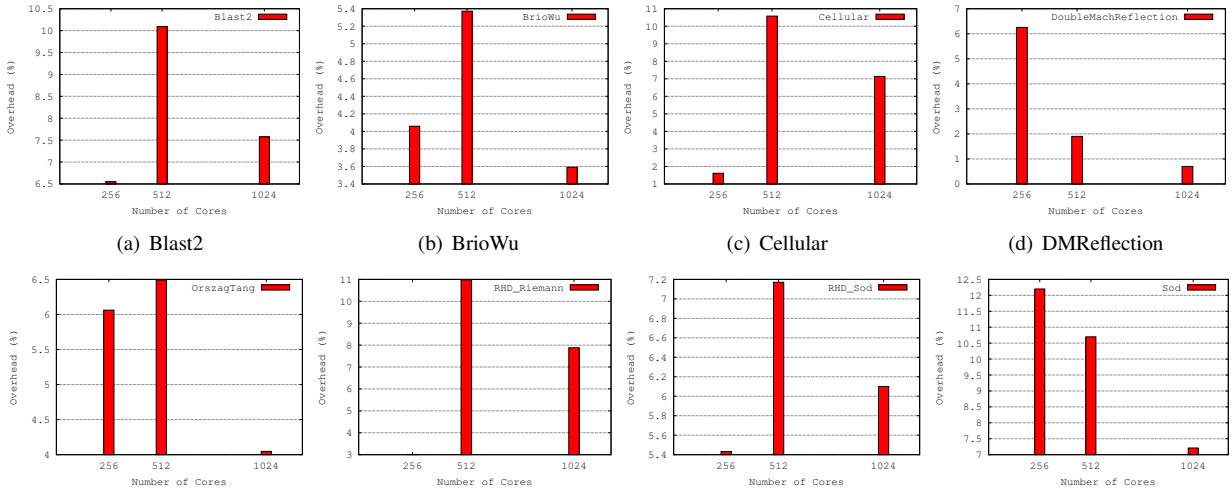
3) Computation Overheads: We now present the computation time overheads of our detector SSD. We report the averages over all processes. Figure 12 shows the computation time overheads (in percentage) with 256, 512, and 1,024 cores. From 512 cores to 1,024 cores we see a decreasing trend in overheads. When 1,024 cores are utilized, all overheads are less than 8% and are 5% on average. From the results, we see that SSD is both lightweight and efficient.

4) Detailed Discussion: **Support vectors as nonparametric method:** As opposed to Gaussian processes, support vector machines are parametric methods whose parameters are usually optimized through Bayesian techniques or cross-validation. In our case, however since the ε corresponds to the impact error bound and we choose not perform any cross validation for the remaining parameters, such as regularization parameter γ or kernel parameter σ (both are set to one), to be efficient, support vector regression has essentially become a nonparametric 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 Scholkopf [24], choosing the appropriate capacity control is more important than choosing the type of kernels used in support vector learning. However, the performance of sigmoid kernels cannot be overlooked. The experimental data shows that when they 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.

V. Related Work

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



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

(e) OrszagTang (f) RHD_Riemann (g) RHD_Sod (h) Sod

Figure 12. Computation time overheads

runtime analysis technique.

ABFT [9, 27, 32] techniques are tailored solutions to specific numerical algorithms. As a result, they are usually efficient. However, they fundamentally lack the ability to be applicable to other algorithms than the specific numerical or algebraic kernel they are designed for.

Replication-based schemes [17] can be deployed for mission-critical situations. In such contexts, double or triple redundancy of computation is performed in order to detect SDCs by comparing the results of replica computations. The inherent drawback of the replication is its high cost; for instance, with double redundancy, the cost is 100%. Partial replication [30] has been proposed to decrease costs while providing 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.

Runtime data analysis recently has gained attention in the HPC community. Studies [5, 12, 13] investigate and compare different prediction methods such as linear curve fitting or ARMA models, to detect SDCs. They convert the problem of detecting SDC into next-step prediction problem. Sharma et al. [26] utilize temporal features of data (in addition to spatial features) and provide a tailored SDC detector for stencil applications where they use support vector machines as a linear function approximator. The main drawbacks of temporal data analytics are the memory overhead and the computation cost of maintaining snapshot data. In contrast, as a spatial technique, SSD incurs low memory cost while having low computation overhead.

VI. Conclusion

In this work, we propose a novel lightweight SDC detector based on dynamic spatial support vector regression. Since our solution only relies on the snapshot data at current time step (i.e., a spatial technique), our detector suffers from little memory overhead (less than 1%) and low performance overhead (5% on average.). Experiments with eight real-world HPC applications show that for most of the failure distributions and applications detection sensitivity is high and up to 99% and the false positive rate is low and less than 1% except being under stress. Our implementation supports wide range of HPC applications in both Fortran or C.

VII. Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357, by FI-DGR 2013 scholarship, by HiPEAC PhD Collaboration Grant, the European Community's Seventh Framework Programme [FP7/2007-2013] under the Mont-blanc 2 Project (www.montblanc-project.eu), grant agreement no. 610402, and TIN2015-65316-P.

References

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

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

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

[4] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and 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, pages 32:1–32:32, New York, NY, USA, 2011.

[5] 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.

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

[7] L. Cao and F. Tay. Support vector machine with adaptive parameters in financial time series forecasting. *Neural Networks, IEEE Transactions on*, 14(6):1506–1518, Nov 2003.

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

[9] 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.

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

[11] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[12] S. Di, E. Berrocal, L. Bautista-Gomez, K. Heisey, R. Guptal, and F. Cappello. Toward effective detection of silent data corruptions for hpc applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2014.

[13] 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.

[14] S. Di and F. Cappello. Adaptive impact-driven detection of silent data corruption for hpc applications. In *www.mcs.anl.gov/publication/adaptive-impact-driven-detection-silent-data-corruption-hpc-applications*, 2015.

[15] Y. Fan, P. Li, and Z. Song. Dynamic least squares support vector machine. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 1, pages 4886–4889, 2006.

[16] T. Farooq, A. Guergachi, and S. Krishnan. Chaotic time series prediction using knowledge based green's kernel and least-squares support vector machines. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 373–378, Oct 2007.

[17] 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.

[18] 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. *apjs*, 131:273–334, Nov. 2000.

[19] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481 – 492, 1951.

[20] J. M. Mart and E. Mller. Numerical hydrodynamics in special relativity. *Living Reviews in Relativity*, 6(7), 2003.

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

[22] Z. Ovari. Kernels, eigenvalues and support vector machines, 2000.

[23] T. Raicharoen, C. Lursinsap, and P. Sanguanbhokai. Application of critical support vector machine to time series prediction. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 5, pages V–741–V–744 vol.5, May 2003.

[24] B. Scholkopf. Support vector learning, 1997.

[25] 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.

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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