2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Machine Learning Models for GPU Error Prediction in a Large Scale HPC System

Bin Nie*, Ji Xue*, Saurabh Gupta†, Tirthak Patel‡, Christian Engelmann§, Evgenia Smirni*, and Devesh Tiwari‡
* College of William and Mary ({bnie, xuejimic, esmirni}@cs.wm.edu)
† Intel Labs (saurabg@gmail.com)
§ Oak Ridge National Laboratory (engelmannc@ornl.gov)
‡ Northeastern University (patel.ti@husky.neu.edu, tiwari@northeastern.edu)

***Abstract*—GPUs are widely deployed on large-scale HPC systems to provide powerful computational capability for scientific applications from various domains. As those applications are normally long-running, investigating the characteristics of GPU errors becomes imperative for reliability. In this paper, we first study the system conditions that trigger GPU errors using six-month trace data collected from a large-scale, operational HPC system. Then, we use machine learning to predict the occurrence of GPU errors, by taking advantage of temporal and spatial dependencies of the trace data. The resulting machine learning prediction framework is robust and accurate under different workloads.**

I. Introduction

Over the past decade, GPUs have become an integral part of mainstream high performance computing facilities thanks to the fact that they allow to simulate physical phenomena more quickly and accurately (i.e., at a finer granularity) [1–3]. As GPUs are more widely adopted in scale-out computing architectures, GPU soft errors become a critical challenge. Reliable execution of applications can lead to higher productivity and lower I/O overhead. However, understanding the source of GPU soft errors itself is challenging. Past work has shown evidence that indicates a plausible relationship between power/cooling infrastructure and GPU errors, but there exists no clear understanding on the exact conditions that trigger faults [4].

The key to improving GPU reliability is to understand the relationship among GPU soft errors and different factors including power consumption, temperature, and workload behavior. The goal of this paper is to explore the interaction among temperature, power consumption, workload characteristics, and GPU soft errors, and to exploit these interactions toward GPU soft error prediction in a large scale HPC system.

Previous works have investigated the interplay between temperature and device reliability on hard disk drives, solid state drives, and CPUs [5–8]. In contrast, our work focuses on understanding the interplay between workload/temperature/power consumption and GPU soft errors on the Titan, America's fastest supercomputer for open science [9].

In this paper, we discover that workload characteristics, certain GPU cards, temperature and power consumption could be indicative of GPU errors, but it is non-trivial to exploit them for error prediction. Motivated by these observations and challenges, we explore machine-learning-based error prediction models that capture hidden interactions among system and workload properties. Such models are useful in guiding flexible error protection schemes for GPU nodes, e.g., by dynamically turning on/off error protection based on prediction.

One may argue that completely turning off error protection may be too risky. However, it is important to notice that the impact of error-correcting code (ECC) overhead on real-world computational science applications can be as high as 10% on GPUs [10]. In fact, the decreased memory bandwidth caused by ECC overhead can result in larger performance degradation than the decreased fraction of bandwidth itself due to queuing ramifications. In fact, computational scientists already naively turn off ECC for their application runs [11]. In such cases, a prediction model would be useful instead of always turning off ECC.

Acknowledging the necessity of an error predictor, this paper elaborates on the challenges, process, and solutions involved in building effective machine-learning-based prediction models. We show how to systematically select features by categorizing them into spatial and temporal dimensions. We illustrate how to overcome the imbalanced dataset challenge and trade-offs by taking advantage of the inherent features of the dataset. We use the selected features to train various machine learning models, including Logistic Regression (LR), Gradient Boosting Decision Tree (GBDT), Support Vector Machine (SVM), and Neural Network (NN).

Finally, we evaluate the machine learning models via different metrics and under diverse testing scenarios. Our results indicate that the proposed models achieve high prediction quality and are robust. In particular, the GBDT-based prediction achieves an F1 score of 0.81, significantly outperforming other models. The corresponding high recall (i.e., 0.87) and good precision (i.e., 0.76) indicate that the GBDT-based model is conservative in identifying as many SBE cases as possible. This is preferable as the aftermath of missing an SBE occur-

2158-3927/18/$31.00 Â©2018 IEEE
DOI 10.1109/DSN.2018.00022

rence is likely to be more severe than mislabeling a non-SBE occurrence. Our evaluation also uncovers interesting insights from comparison across different models, training/testing data, and feature combinations. We show that the proposed prediction models impose moderate overhead and are practically feasible for GPU soft error prediction.

II. Background

The Titan supercomputer is one of the fastest supercomputers for open science [9]. The basic block is a node, that consists of one AMD Opteron 6274 CPU and one NVIDIA K20X GPU. Four nodes make up one slot. In each slot, there are two high-speed interconnect Gemini routers, each shared by two nodes. The next granularity level is one cage, which is composed of 8 slots. Three cages form one cabinet. There are 200 cabinets, organized as a 25×8 grid, and a total of 18,688 GPUs on the Titan. Major memory structures in K20X GPUs are protected by error-correcting codes (ECC). Device memory, L2 cache, instruction cache, register files, shared memory, and L1 cache are protected by Single Error Correction Double Error Detection ECC, while read-only data cache is parity protected. Non-memory structures such as logic, thread schedulers, instruction dispatch unit, and interconnection network are not protected.

Various types of GPU errors occur on the system including hardware failures. NVIDIA provides a list of XID errors and documents their possible causes [12]. In this study, we focus on GPU soft errors. Particularly, we target single bit errors (SBEs), as they occur most frequently on Titan GPUs. Other errors, i.e., double bit errors (DBEs), are less frequent and statistically unsuitable for prediction. The ECC mechanism is typically turned on to detect SBEs. However, ECC incurs significant overhead to storage and memory bandwidth. If the impact of SBEs could be understood well with predictive capability, turning-off ECC could improve performance and reduce associated overhead.

Our traces contain GPU-error related data from February 2015 to June 2015 (more than 60 million node hours). SBEs are collected via the *nvidia-smi* utility. This utility provides snapshot information, i.e., it does not timestamp individual SBEs, but records SBEs before and after each batch job. This enables to perform data analytics on SBEs, albeit at the granularity of a "batch job". We denote a batch job as a set of applications that are submitted simultaneously by the same user. Applications (also referred as "apruns") can run within a submitted batch job (also referred as "job"). The SBE count is collected at the start and end of a batch job. The tracing framework can identify the node location on which SBEs occur. We collect GPU resource utilization information such as GPU core-hours, maximum memory consumption, and total memory consumption on a per application basis. We collect the temperature and power consumption information in out-of-band manner without instrumenting applications. This information is approximately collected every minute for every node.

We do recognize that this work is subject to assumptions and limitations. This study assumes apruns having the same binary name are of the same type, as user codes and workflow execution practices are unknown to us. Batch jobs may contain multiple apruns, but we cannot tell which apruns encounter SBEs. Therefore, we conservatively assume that SBEs occur in *all* those apruns. Finally, a large-scale HPC facility is often dynamic with respect to software stack changes and operational practices. Hence, correctly including the impact of these factors in the data analysis is onerous. Moreover, soft errors can be trigged by transient bit-flips due to external charge-carrying particles or device failures, i.e., variations, yield, aging effects, or electron migration. We cannot distinguish soft errors by their root causes in this work due to lack of information.

III. GPU Error Characterization

Soft errors may occur during an application execution on GPUs for multiple reasons, i.e., cosmic ray strikes, voltage fluctuations, elevated temperature, manufacturing defects, and complex workload-hardware interaction. However, pinpointing the root cause of soft errors is challenging and cannot be easily used to predict soft error occurrences. While soft error occurrences have limited predictability, we find that not all soft error occurrences are random. Our results reveal that certain system and workload properties may have hidden correlations with GPU soft errors, albeit such correlations can not be attributed as causations. In particular, we show that certain GPU cards, workload behavior, GPU temperature, and GPU power consumption may have complex interactions with GPU soft error occurrences.

A. SBE Offender Nodes

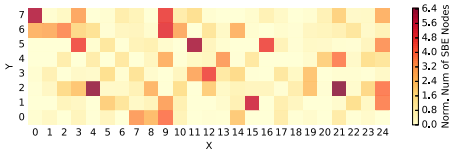


Fig. 1. Non-uniform distribution of GPU error offender nodes at the cabinet level.

We start by investigating how GPU errors are distributed across the entire system. Since the 200 cabinets on the Titan are organized as a 25×8 grid, we present the normalized average value of SBE-affected nodes at the cabinet level in Fig. 1. Clearly, GPU errors are not uniformly distributed. The number of SBE-affected GPU cards are not the majority of all cards in the system either. As shown later in Section VII, exploiting this observation in isolation is not likely to yield good prediction of future SBEs. For example, if we predict that all applications executing on these SBE offender nodes will experience errors, it results in a high false positive rate because SBE offender nodes do not experience errors uniformly over all days either. Actually, 80% of error offender nodes experience a soft error on less than 20% of the total

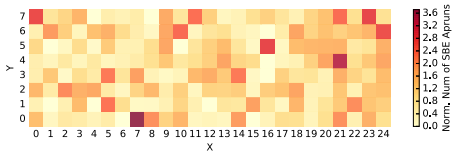
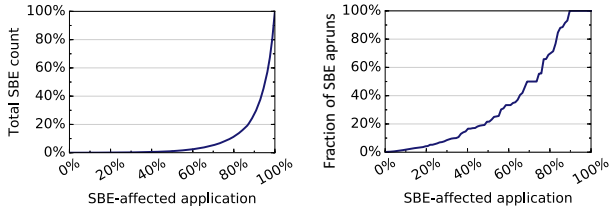


Fig. 2. Non-uniform distribution of SBE-affected application runs at the cabinet level.

days over the trace period. Nevertheless, the non-uniform distribution of soft error offender nodes in the space and time domains open the possibility for learning-based predictions.

B. Application

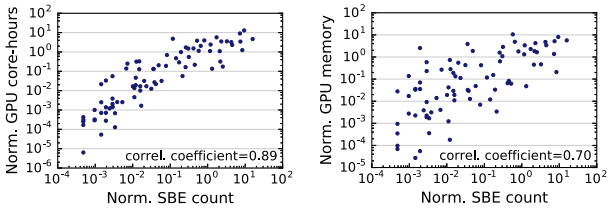
It is also important to analyze the impact of various workloads on GPU soft error occurrences. As a first step, we explore the spatial distribution of SBE-affected applications and observe the non-uniform distribution across the Titan system (see Fig. 2). Next, we look at the severity of SBE-affected applications by analyzing their SBE count (application and SBE correlation is normalized by the GPU core hours, i.e., runtime $\times$ number of nodes). Fig. 3(a) shows that a smaller set of workloads, less than 20% of all applications, experience the majority of errors ($\geq 90\%$). However, Fig. 3(b) shows that even SBE-affected applications do not experience SBEs uniformly across all application runs. The top 20% of the SBE affected workloads experience all their share of soft errors during 60% of their total application executions, while the lower 20% of the SBE affected workloads experience all their share of soft errors during less than 10% of their total application executions. We further investigate the relationship between the severity of SBE-affected application runs and their GPU utilization, i.e., core-hours and memory, see Fig. 4. The high Spearman coefficient values (see inset in each figure for the exact values) indicate that applications with more SBEs tend to utilize more GPU memory and for longer duration.



(a) Total SBE count (b) Fraction of SBE apruns

Fig. 3. Workload and GPU error distribution: a small set of workloads experience most of the soft errors (a), and fraction of executions affected by SBEs for SBE-affected application runs (b).

The above observations imply that application related measurements such as utilization, are good indicators for SBE occurrences. While these observations provide useful ground truth for error prediction, simple prediction strategies based on these findings alone lead to low prediction quality (we address this issue in Section VII).



(a) GPU core-hours (b) GPU memory

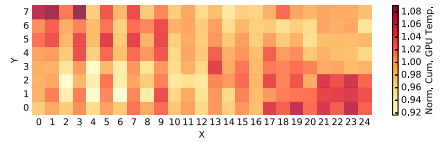
Fig. 4. Scatter plot of SBE count of SBE-affected application runs and their GPU utilization: core-hours (a) and memory (b).

C. Temperature and Power Consumption

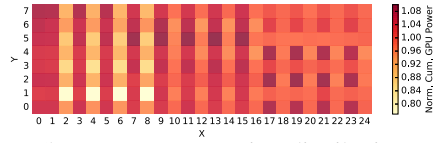
Consistent with previous studies on GPU errors [4, 13–15], we analyze the potential relationship between GPU temperature/power consumption and GPU errors.

1) A bird's eye view

We first explore whether GPU temperature/power consumption correlate with soft error occurrences. Fig. 5 shows the cumulative temperature/power consumption over the entire sampling period of every cabinet in the Titan. We observe that the temperature distribution is non-uniform in space, i.e., cabinets in the upper left corner and lower right corner tend to be hotter than the rest. In contrast, power consumption is more evenly spread, implying that the Titan is intensively utilized both time-wise and space-wise.



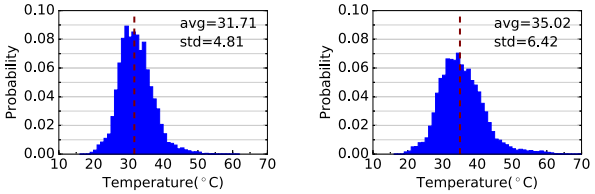
(a) Temperature distribution



(b) Power consumption distribution

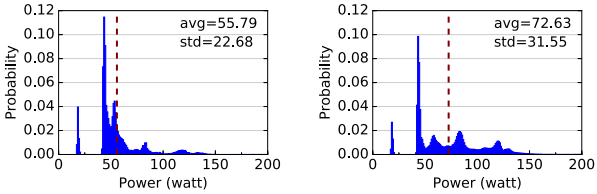
Fig. 5. Distribution of temperature (a) and power consumption (b) accumulative over the whole period at the cabinet level.

Next, we compare the non-uniform temperature distribution with the SBE-affected nodes distribution (Fig. 1) by calculating Spearman correlation coefficient at the node level. The low value (0.07) implies that the accumulative temperature distribution is not related to the SBE offender nodes distribution in space. The same observation is reached when comparing the temperature distribution and the SBE-affected application distribution (the Spearman correlation coefficient is only 0.15). Similar analysis is conducted for power consumption, which also shows weak correlation between power consumption and SBE-affected nodes or SBE-affected application runs. In summary, the effect of temperature on SBEs may not be entirely



(a) SBE-free period (b) SBE-affected period

Fig. 6. Temperature distribution of SBE offender nodes during SBE-free periods (a) and SBE-affected periods (b). Vertical lines represent mean values.



(a) SBE-free period (b) SBE-affected period

Fig. 7. Power consumption distribution of SBE offender nodes during SBE-free periods (a) and SBE-affected periods (b). Vertical lines represent mean values.

captured by SBE offender nodes or workload characteristics only.

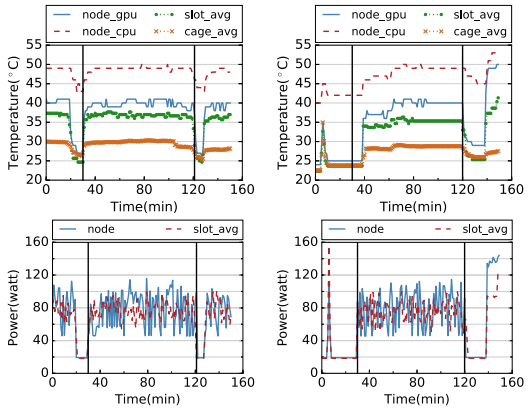
2) Considering the time dimension

We turn the focus to SBE offender nodes and temperature characteristics across time. We divide the time dimension in two parts: (1) the time during which a soft error occurs (*SBE-affected period*) and (2) the time during which no soft error occurs (*SBE-free period*). Fig. 6 shows the empirical temperature distribution of SBE offender nodes during these two periods. The distribution for GPU power consumption is presented in Fig. 7.

We observe that the SBE offender nodes are relatively hotter during the SBE-affected period by more than 3°C on average, compared to SBE-free period (Fig. 6(a) vs. 6(b)). The SBE offender nodes also consume relatively higher power during the SBE-affected period by more than 15 watts on average, compared to the SBE-free period (Fig. 7(a) vs. 7(b)). Note that higher power consumption likely contributes to increased temperature. However, due to varying cooling efficiency and workload characteristics, temperature elevation may be caused by other factors too. The above observation implies that SBEs are more likely to happen during periods of elevated temperature. Our measured data do not conclusively indicate that SBEs definitely occur above a certain threshold of temperature/power consumption. Sometimes even during the SBE-free period, temperature can be significantly high (see Fig. 6(a)), making the SBE occurrence prediction non-trivial. Nevertheless, these observations are encouraging as they demonstrate a relationship between temperature/power consumption and SBE occurrences, which can be potentially used for prediction.

3) Considering the space dimension

Besides the time domain, it is natural to also explore whether similarities exist across space. In fact, our measured data indicate that GPU power consumption and temperature profile can change for the same workload across runs, possibly due to effects from neighboring nodes (i.e., spatial effects). We first investigate how the temperature profile changes when the same workload is executed repeatedly on the same node. Intuitively, one does not expect the temperature profile to change. To test this, we select a computational chemistry application that is executed multiple times on the same node at different times. Fig. 8 shows the temperature and power profiles of GPU during two different runs on different days, but on the same node to avoid location specific power/cooling side-effects. We plot the average temperature and power values for all other nodes in the same slot or cage, as well as the temperature profile of the CPU in the same target node. For the power profile, we do not have the ability to measure CPU power consumption out-of-the-band. We include the 30 min time window before and after the application run to evaluate the results in context.



(a) Temperature and power from first run (b) Temperature and power from second run

Fig. 8. Effect of neighboring components on temperature/power of an application over two runs on the same node overtime. Vertical solid lines represent the start and end of the aprun execution.

From Fig. 8, we observe that the temperature profile changes from one run to another and that it is not necessarily correlated to fluctuations in the power profile. The result indicates that changes in the temperature/power consumption of neighboring nodes and the CPU in the same target node may contribute to the variation in the temperature profile of the target node. Other factors such as change in power/cooling efficiency in the spatial region may also contribute to variation in the temperature profile, although these factors are hard to detect and quantify. Motivated by the above evidence, we argue that temperature and power consumption from neighboring nodes in the same slot, as well as the temperature of the CPU on the

same node, may also help with SBE occurrence prediction. Still, it is non-trivial to understand whether or how much the behavior of neighboring nodes can actually improve the error prediction capabilities. In Section VII, we will quantify the impact of these features on prediction effectiveness.

In summary, our analysis reveals that certain GPU cards, workload, and GPU temperature/power consumption could have predictive or associative capabilities with GPU errors, but it is non-trivial to exploit them for error prediction. In Section VII, we demonstrate that simple schemes based on these observations lead to poor prediction effectiveness, while machine-learning-based approaches capture hidden interactions and significantly outperform simplistic predictions.

IV. Overview of the Methodology

In the previous section, we illustrate that GPU errors are potentially correlated with different system and workload characteristics. Formally, we are interested in finding a mathematical function that maps these properties (features) to the probability of GPU error occurrence. If we express system and workload dependent properties as features $x_0, x_1, x_2, \ldots, x_n$, there exists a function F_{pred}, such that the probability of GPU error occurrence during program execution is expressed by:

$$Prob_{err} = F_{pred}(x_0, x_1, x_2, \ldots, x_n)\ . \quad (1)$$

Note that, many such functions can exist with varying accuracy-levels because the probability of GPU error occurrence during program execution may not always be dependent on the value of different features only. It is possible that a mathematical function can not fully capture the behavior because of inherent randomness involved with soft error occurrences. Therefore, the goal is to "learn" a classification function, F_{pred}, that provides high accuracy based on the available features. Given this, we take the following steps:

Step 1: Feature selection and engineering. We select a set of features as input to the desired function. We elaborate the process, challenges, and solutions involved in selecting a useful set of features (Section V).

Step 2: Function discovery. We discuss how to learn the desired classification function in a generic yet meaningful way. We provide details on the challenges in learning the classification function (Section VI).

Step 3: Analysis of the learned function. We investigate the usefulness of the learned function and analyze the function to assess if it can provide meaningful results under different circumstances (Section VII).

We emphasize that these steps are means to show that such a problem can be solved with reasonable accuracy and under practical constraints. The rest of the paper demonstrates the execution of these steps. We note that deep learning approaches are not covered in this work due to their typical high overhead and limited suitability to the nature of our problem.

V. Feature Selection

Determining an effective set of features to learn the desired function is challenging. First, measuring and collecting plausible features correlated with GPU errors is not always possible. For example, the memory access pattern could be associated with SBEs. However, the overhead to collect this information in a production system with dynamically changing workloads is cost prohibitive. Second, selecting features from what can be measured and collected is taxing. One can conservatively collect data from all instrumentation sources, but it may result in excessive storage and processing overhead without clear understanding if they are indeed related to the final outcome. Consequently, feature selection is a critical aspect toward learning the desired function. We refer to the process of transforming the selected features into quantifiable and meaningful representation as feature engineering.

These challenges are addressed by following the observations discussed in Section III. We identify the features that have correlations with GPU soft errors and organize them into time and space dimensions. The key premise is that soft errors are not an outcome that can be predicted by observing the instantaneous values of features. Therefore, it is important to include both temporal and spatial dimensions. Next, we list different features and their corresponding quantifiable representation.

A. Temporal Features

Application: As discussed in Section III-B, some applications experience higher number of soft errors than others, indicating that application-specific features could be useful toward soft error prediction. We use application-specific features that can be obtained in non-intrusive manner, including the application binary name, total execution time (from past runs), and GPU resource utilization. GPU resource utilization includes the aggregate GPU core time, aggregate GPU memory, and maximum GPU memory. To capture the temporal behavior, we also use the application name that ran before this execution to account for post-effects of an application run.

Temperature/power consumption: We have shown evidence that temperature may be correlated with soft error occurrences (Section III-C2). However, capturing this complex correlation is non-trivial. We propose the following four temperature features to capture temporal aspects. First, we use the mean and standard deviation of the temperature during the current application run as two input features. In addition, to capture the dynamic behavior during a run, we use the mean and standard deviation of the *difference* between two consecutive temperature measurements as two additional input features. The above four features do not account for recent historical temperature behavior. To address this, we use temperature characteristics before the execution of a current application on the node. Specifically, we use the mean and standard deviation of the temperature series and the mean and standard deviation of the *difference* between two consecutive temperature measurements on the same node before the execution of the current application. We consider four time windows: 5min, 15min, 30min, and 60min prior to the start of the current execution to calculate the aforementioned four

temperature features. Similarly, we apply the above described metrics for GPU power consumption.

B. Spatial Features

Node location: Our characterization results indicate that error offender nodes are not uniformly distributed in space, and some error offenders experience SBEs repeatedly (Section III-A). Therefore, node location is used as a feature to capture node-specific and location-specific correlations.

Temperature/power consumption: In Section III-C3, we show the prediction capabilities with temperature and power consumption on neighboring nodes. Similar to the representations of temperature and power consumption used in the temporal feature set, we leverage the mean and standard deviation of temperature and power consumption, as well as the mean and standard deviation of the difference between two consecutive measurements for (1) the temperature of the CPU on the same node and (2) the temperature and power consumption of the GPU nodes in the same slot, as parts of the spatial feature set.

SBE history: We include the error frequency in order to capture non-uniform temporal distribution of SBEs (Section III). Specifically, we use the total error count over the preceding day i.e., in the past 24 hours, at the node-level and for the whole machine as features to capture the spatial behavior of error occurrence. We refer to this information as SBE rate history at the local (node) and global (whole machine) level. We also include the SBE rate in the past 24 hours of the given application and the nodes allocated to it as additional history features.

VI. Machine Learning Framework and Model

In this section, we focus on the discovery of the function that captures the relationship between input features and GPU soft error occurrences. To this end, we use several widely-used machine learning models including Logistic Regression (LR), Gradient Boosting Decision Tree (GBDT), Support Vector Machine (SVM), and Neural Network (NN). Our goal is to understand how the classification function can be learned effectively via carefully choosing a combination of features and an appropriate learning model, as well as what insights can be learned from evaluating such models.

A. Overview

The first step of the machine learning framework requires building the training dataset by collecting input features. In our case, we periodically collect information on input features for jobs running on the Titan. As a second step, this training dataset is used to build the machine learning model. The chosen model outputs the desired classification function that can be used for GPU soft error prediction. The desired classification function is a two-class classifier (i.e., whether an error occurs or not during the target program execution), and is dependent on the training dataset and the selected model. Building the training dataset and estimating the classification function is an iterative process that aims to refine the learned classification function as time passes. Here, the model construction is relatively less frequent (i.e., once every two weeks). The final step is to feed the features of the target program into the models to predict error occurrence.

Some input features for the target application run can be collected prior to execution (e.g., machine-level error rate, node specific characteristics), while certain program specific features such as GPU power and temperature profiles can not always be known a priori. We experiment with two approaches and achieve similar results. In the first approach, the prediction can be done at the end of the application execution, and a possible re-execution may be required depending on the program's resilience needs. In this case, all input features are known correctly. The second approach is that certain input features are learned using statistical models and are fed into the learned function. Note that, this approach can not guarantee that all input feature values are 100% accurate. Fortunately, HPC workloads are fairly repetitive. It is possible to effectively learn and accurately predict program specific features, i.e., their temperature and power profile, by leveraging time-series prediction tools, e.g., [16].

B. Challenge: Imbalanced Dataset

It is desired to select the training and testing data so that they cover a wide variety of workload and system properties, and are also representative of a real-world scenario. In our approach, any workload execution that uses GPU resources is a qualified sample. This ensures that our dataset corresponds to different kinds of workloads distributed over both time and space dimensions. However, this data collection approach results in a challenging problem: a highly imbalanced dataset. The problem stems for the fact that only a limited number ($\leq 2\%$ in our case) of application runs encounter SBEs. This makes the size of majority class (SBE-free samples) much larger than that of the minority class, which is our focus.

Mitigating the imbalanced dataset challenge usually has two solutions. The first one is over-sampling the minority class, i.e., by generating synthetic samples [17, 18]. The other solution is to under-sample the majority class, i.e., by randomly choosing a subset of samples [19] or control under-sampling via clustering algorithms such as k-means [20]. Note that, none of the above methods takes the inherent dataset features into consideration. In the following section, we propose a two-stage method, which first leverages the dataset characteristics to mitigate the challenge of the imbalanced dataset and then apply machine learning models to predict SBE occurrences.

C. Two-Stage Machine Learning Models

1) Leveraging dataset characteristics

In Section III, we observe that a small fraction of GPU nodes and workloads are responsible for a large number of SBEs. It is intuitive to think that previous SBE-affected nodes/workloads may continue seeing SBEs while those SBE-free nodes/workloads are likely to remain in "safe status" in the future. Accordingly, we consider three basic schemes: *Basic A* predicts that any application run involving a SBE

offender node will result in a SBE-affected run. *Basic B* predicts that previously SBE-affected applications will result in future SBE-affected runs. *Basic C* predicts that top SBE-affected applications will result in a future SBE-affected run. Top SBE affected applications are defined as the top 20% applications that encounter SBEs in the training phase in terms of their total number of SBEs.

Table I presents the prediction effectiveness of the above three basic schemes, compared to a trivial random classifier that assumes the probability of encountering SBEs is 0.5. Precision is defined as the percentage of correct predictions in all predictions:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}, \quad (2)$$

while recall reveals the ratio of identified samples to the ground truth, expressed by the following formula:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}. \quad (3)$$

The random classifier achieves a mere 0.5 recall. Due to the high imbalance between the two classes, the random classifier achieves very low precision for the SBE class prediction. *Basic A* significantly outperforms the random classifier and the other two basic schemes, achieving a high SBE prediction recall (0.94), albeit at fairly low precision (0.40). This indicates that the scheme *Basic A* could capture most SBE cases but still over-predicts the SBE class, implying that this scheme alone is insufficient for robust prediction.

TABLE I
PRECISION AND RECALL FOR BASIC SCHEMES.

	SBE Sample		Non-SBE Sample	
Scheme	**Precision**	**Recall**	**Precision**	**Recall**
Random	0.02	0.50	0.98	0.50
Basic A	0.40	0.94	0.99	0.98
Basic B	0.02	0.69	0.98	0.24
Basic C	0.00	0.06	0.98	0.76

2) TwoStage method

Inspired by *Basic A*, which achieves a reasonable prediction quality, we derive a *TwoStage* method. This method leverages the inherent temporal dependency of our dataset and takes advantage of the power of machine learning techniques. Unlike *Basic A*, the *TwoStage* method is able to accurately predict the samples from SBE offenders, instead of blindly assuming them to always encounter SBEs in the future. During training, we train the model solely on samples from SBE offender nodes. The prediction flow is presented in Fig. 9. At the first stage, samples are checked to see if they come from SBE or non-SBE offender nodes. They are passed to the second stage only if they come from SBE offender nodes. The advantages of this method are three-fold: (1) the number of SBE offender nodes is much smaller than the number of non-SBE offender nodes. Therefore, this step automatically reduces

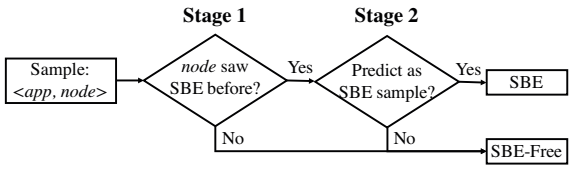


Fig. 9. *TwoStage* method: prediction flow.

the training data size, resulting in less training overhead (both in terms of time and storage). (2) As discussed previously, the relationship between SBEs and different features is complex. By focusing on SBE offender nodes only, we avoid the noise and interference from error-free samples. (3) Most importantly, this approach solves the problem of data imbalance. Now, after the first stage, the ratio between SBE-free samples and SBE-affected ones is roughly $2:1$ (Consider that the original ratio is almost $50:1$). The downside is that this method always misses SBE occurrences on previously error-free nodes. Fortunately, on the Titan, such probability is low and frequent periodic training of the model resolves this issue. Section VII shows that *TwoStage* introduces low overhead and can be trained periodically to provide high prediction quality.

D. Machine Learning Model Selection

We select four widely used machine learning models that provide a wide variety of trade-offs and advantages. **Logistic Regression (LR)** is a simple and fast model for understanding the influence of several independent variables but limited by the linear function between inputs and outputs. **Gradient Boosting Decision Tree (GBDT)** is a boosting-based model that is essentially an ensemble of weak models, that is effective in tackling the variance-bias problem, but is computationally expensive. **Support Vector Machine (SVM)** is designed to solve this problem by performing non-linear classification using a kernel. **Artificial Neural Networks (NN)** are inspired by biological neural networks and are composed of many interconnected neurons. The weights associated with the neurons are used to approximate non-linear functions of the input. Neural networks capture the complex pattern between features and targets.

In the evaluation section (Section VII), we incorporate the aforementioned models to the *TwoStage* method and compare their effectiveness.

VII. EVALUATION AND ANALYSIS

Before discussing the prediction results, we describe the data used for model training and testing, as well as the evaluation metrics.

A. Data Description and Evaluation Metrics

We collect all the features discussed in Section V over the entire sampling period (from January to June, 2015) for both SBE-affected and SBE-free periods. We divide this dataset into three pairs of training and testing sub-datasets based on the time dimension. In each sub-dataset, the training dataset consists of 3.5-month samples, and the samples in the following

two weeks are used for testing. Each sample is identified as the pair of the application name and the node ID. For example, our first training dataset (i.e., DS1) corresponds to 6.7 thousand application executions, with roughly 5 million samples. Note that each application run may produce multiple number of samples depending on the number of nodes allocated during the execution. For determining the length of the training and testing datasets, we follow the rule-of-thumb ratio of the testing data size to the training data size ($20\% - 25\%$) [21]. We also ensure that the three testing datasets cover diverse workloads and have different compositions of samples.

In order to meaningfully evaluate the results, it is important to choose the most appropriate metric. Accuracy is a simple and widely used metric to assess the effectiveness of predictions. However, it is misleading for evaluating imbalanced datasets. In our testing datasets, around 98% of the application executions (samples) fall into the majority class (i.e., non-SBE class). For example, a naive method, such as always predicting each sample as non-SBE case, will lead to an accuracy of 98%. Other commonly used metrics include precision and recall, see Section VI-C1. The main goal of any prediction mechanism is to improve precision without sacrificing recall. However, precision and recall sometimes can be conflicting, as while increasing the true positives, the false positives may also increase [22]. Consequently, we use the *F1 Score* [23], the harmonic mean of precision and recall (see Eq. 4),

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

as the evaluation metric to capture such trade-off between prediction and recall. In general, higher F1 score indicates better prediction quality.

B. Machine Learning Model Comparison

As stated in Section VI-D, we apply four machine learning models (i.e., LR, GBDT, SVM, and NN) on the second stage of the *TwoStage* method. Here, we discuss which machine learning model works most efficiently.

1) Accuracy and robustness comparison

Across machine learning models: Choosing an effective model is one of the key challenges. Fig. 10 reveals the F1 score of SBE class using the first dataset (DS1) for the four machine learning models. Note that the result of SBE-free class is not shown here (also in later evaluation parts) because all models are able to achieve high prediction quality for the SBE-free cases (i.e., the majority class) due to the highly imbalanced nature of our testing samples. We notice that applying machine learning models always significantly surpasses the *Basic A* scheme, with at least 0.1 improvement for the F1 score. Applying GBDT model achieves the highest F1 score (0.81), outperforming the least effective one (LR) by 0.14. To investigate why GBDT works better than the other models, we also look at the precision and recall values. We find that all four models are able to achieve a similar precision values (around 0.8), but GBDT is able to achieve a much higher recall value (0.87) than the other three models (around 0.6). High recall value implies that the boosting nature of GBDT enables it to identify more SBE samples, while similar precision across four different learning models indicates that GBDT also conservatively predicts SBE occurrences as the other three models. This result suggests that GBDT achieves the most accurate prediction of SBE occurrences among the four machine learning models.

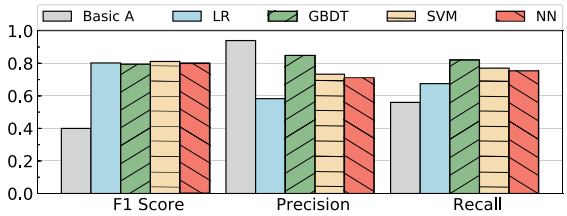


Fig. 10. Comparison of SBE occurrence prediction across different models for DS1.

Across different datasets: We have shown that applying GBDT yields to the best prediction result for the first dataset. Here we validate whether GBDT works best for other datasets (i.e., DS2 and DS3). Note that these testing and training datasets are disjoint and the machine learning models are trained independently for each dataset. Table II summarizes the F1 scores of applying different models on the other two datasets. The table shows that applying machine learning models almost always leads to improvement in the F1 score, compared with *Basic A*. Secondly, using GBDT results in satisfactory prediction quality (F1 score) across different datasets and significantly outperforms all the other three models. Even for the most tough-to-predict dataset (DS3), applying GBDT within *TwoStage* improves the F1 score to 0.71. The above observations confirm the efficiency and robustness of GBDT.

TABLE II
F1 SCORE FOR SBE OCCURRENCE PREDICTION.

Dataset	Basic A	LR	GBDT	SVM	NN
DS1	0.56	0.67	**0.81**	0.70	0.69
DS2	0.75	0.80	**0.81**	0.79	0.77
DS3	0.55	0.52	**0.71**	0.55	0.51

2) Model overhead comparison

In the previous subsection, we have illustrated that the *TwoStage* method with GBDT is effective and robust. Here, we evaluate its training overhead, especially since the Titan operation would require re-training to occur periodically. The comparison of the training time of the four machine learning models is presented in Table III. Note that all experiments

are conducted on an Intel Xeon server (Intel E5-4627v2) with 512GB RAM. The training time is the longest for SVM and is approximately one hour. This is due to the computationally expensive quadratic RBF kernel used in the SVM model. LR consumes the least amount of time, but it also fails to provide a guaranteed prediction quality (see Fig. 10 and Table II). Considering both prediction quality and overhead, GBDT is superior as it strikes a good balance between these two measures. Note that since the training process can be done offline and periodically (e.g., repeated every two weeks), the relative model re-training time is truly negligible. Overall, GBDT's small training time would allow re-training to happen even several times during the day if needed. In addition, the data movement overhead for storing and preprocessing the data is of the order of minutes.

TABLE III
MEAN TRAINING TIME FOR VARIOUS MODELS.

Model	LR	GBDT	SVM	NN
Mean Time	4.81 s	40.53 s	1.04 h	20.01 min

The aforementioned evidence supports that *TwoStage* with GBDT is practically feasible for error prediction. In the later sections, we show prediction results based on this model only.

C. Feature Analysis

Besides choosing an appropriate machine learning model, the selection of features is another key to achieving high-quality of prediction. In Section V, we illustrate several features from temporal and spatial perspectives, which may contribute to the SBE occurrence prediction. This does not imply that all features are needed for training the most effective model. Nonetheless, it is non-trivial to discover and engineer the feature set resulting in the highest prediction quality. In this section, we explain how to perform the feature discovery process.

The large number of features and complexity of advanced learning models make it challenging to meaningfully understand the impact of each feature. Consequently, we simplify this problem by grouping features into categories (feature groups) and train the machine learning models with each feature group. The goal is to see which feature group contributes most to the prediction quality. We also train one model with *all* features. Fig. 11 shows the effect of different feature groups on the prediction quality, in the form of the percentage improvement for the F1 score comparing to *Basic A*. The labels in the figure legend indicate the corresponding feature groups used in each experiment.

We observe that almost all models trained with any feature group positively contribute to the SBE occurrence prediction, but with different degrees of improvement. Meanwhile, no single feature group is the winner across all datasets. For example, `Hist` is the most effective feature group for DS1, but it negatively impacts prediction quality in DS2. However, in all datasets, using the combination of all features always

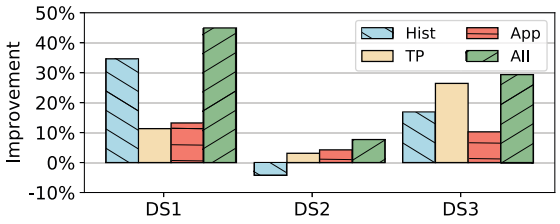


Fig. 11. Effect of different feature groups on F1 score, in terms of the improvement over *Basic A*. **All** means using all features discussed in Section V. **Hist**, **TP**, and **App** correspond to SBE history, temperature/power consumption, and application-related features, respectively.

results in the biggest improvement, implying that all features are valuable and needed for achieving good prediction.

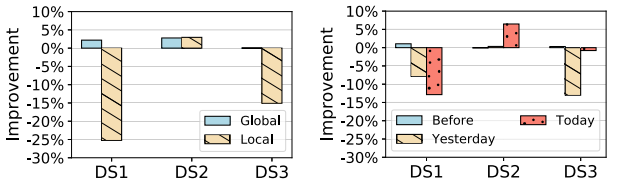
Besides feature grouping, it is also interesting to conduct a deeper and more fine-grained investigation on input features. We start by quantifying the impact of various types of temperature/power consumption features. As stated in Section V, temperature and power consumption features are collected from both temporal and spatial perspectives, on the targeted node and other neighboring nodes in the same slot. Therefore, we conduct experiments with various combination of temperature and power consumption features to see their impact on SBE occurrence prediction, see Table IV. `Cur` refers to using temperature and power consumption data collected only from the targeted node during the application run, together with all other groups of features mentioned in Section V. In addition to the features used in `Cur`, `CurPrev` also leverages temperature and power consumption data prior to the execution of application on the targeted node (in four time windows, up to one hour). Similarly, `CurNei` adds the temperature and power consumption data on neighboring nodes (i.e., in the same slot as the targeted node). `CurPrevNei` leverages all temperature and power consumption features discussed above. Interestingly, we notice that the prediction quality is not significantly affected by the various feature combinations. Looking at F1 score, `CurPrev` and `CurPrevNei` work worse than `Cur`. In contrast, `CurNei` achieves slightly better prediction quality, but it also leverages more features which means it introduces more overhead in terms of data collection and model training. `Cur` exhibits high recall and good precision. Consequently, we select `Cur` as an effective and light-weight representation of temperature and power consumption information for model training.

TABLE IV
EFFECT FROM TEMPORAL AND SPATIAL ASPECTS OF TEMPERATURE AND POWER FEATURES.

Feature Set	Precision	Recall	F1 Score
Cur	0.764	0.865	0.820
CurPrev	0.801	0.830	0.815
CurNei	0.815	0.838	0.826
CurPrevNei	0.807	0.829	0.818

As a next step, we analyze the impact of various types of

history features on the SBE occurrence prediction. Unlike the aforementioned experiments, here we conduct the experiment by *removing* one type of history features and see the decrease in F1 score. First, we compare the effects from global (overall information collected from the whole system) and local (information collected from the targeted node) SBE history on SBE occurrence prediction, see Fig. 12(a). Interestingly, removing global and local history even increases the F1 score in DS2, which is consistent with the observation in Fig. 11, where SBE history features contribute negatively in DS2. However, if we focus on DS1 and DS3, we notice that local history information plays a more important role in prediction, i.e., removing these features leads to 15% to 25% loss in F1 score. The impact of history length on prediction quality is presented in Fig. 12 (b). From this figure, we observe that the importance of SBE history generally increases as it is closer to the current time. Note also that there is no particular length (i.e., today, yesterday, or full history) that is always effective across all datasets. This illustrates the importance of inclusion of all SBE history features.



(a) Global vs. Local (b) Length of history

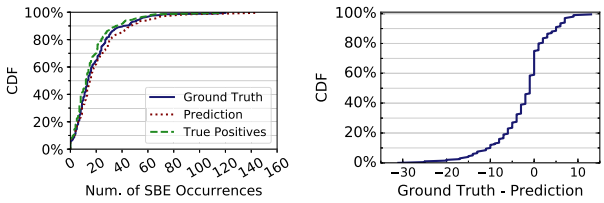
Fig. 12. Decrement on F1 score if removing a certain feature set from the original feature combination: global vs local (a), and different length of SBE history (b).

D. Prediction Analysis

In the previous sections, we have determined that GBDT is the best machine learning model for the *TwoStage* method, and the most effective feature combination for its training. Here, we conduct an evaluation on the prediction quality of this model with the most efficient feature combination as inputs. Due to the space constraints, we illustrate the analysis on the results of using the first dataset only. The quality of prediction for the two other datasets is similar to that of DS1.

1) Spatial robustness

We investigate if *TwoStage* performs well spatially across the entire Titan system. Fig. 13(a) shows the proximity of the cumulative distribution plots of SBE predictions across the entire system for the ground truth, prediction (true positives plus false positives) and true positives. We then present the absolute difference between the number of SBE affected application runs (ground truth) and the prediction for the testing period at the cabinet level, see Fig. 13(b). For over 95% of cabinets, the error difference is relatively small, ranging in $[-15, 13]$. In fact, there are only 3 (out of 200) cabinets where the prediction overestimates SBE affected application runs by more than 25. This is encouraging as thousands of applications are executed over each cabinet. We also perform such analysis at the node level and observe accurate prediction for more than 99% of nodes (result not shown due to space constraint).



(a) Comparison of CDFs (b) Diff. from prediction

Fig. 13. Comparison between SBE occurrence prediction and ground truth at the cabinet level.

We also investigate how the choice of optimal model changes across the various cabinets. We find that *TwoStage* with GBDT remains the close-to-the-best choice among all models for all cabinets. The number of cabinets where this scheme is not the optimal choice is limited across the machine in all three datasets. In fact, we find that even if the prediction model is chosen with the apriori knowledge (oracle) on the optimal model, the overall F1 score improves only by 0.01, 0.02, and 0.001 for the three datasets, respectively. Overall, our results indicate that *TwoStage* with GBDT delivers robust and consistent results across the whole machine and it is not restricted to performing well only in selected sections of the machine.

2) Effect of application runtime

We look into whether the quality of the prediction is significantly impacted by the length of the application execution. In other words, do short-running and long-running applications attain comparable prediction quality? We classify an application as "short-running" if its runtime falls in the bottom 25 percentile range and as "long-running" if its runtime falls in the top 25 percentile range. Table V confirms that both types of application achieve high prediction quality with comparable F1 scores. Moreover, "long-running" applications achieve better prediction quality than "short-running" ones. This is quite favorable since the cost of mislabeling a "long-running" application would be higher, e.g., if re-execution is needed.

TABLE V
SBE OCCURRENCE PREDICTION FOR "SHORT-RUNNING" AND "LONG-RUNNING" APPLICATIONS.

Application	Precision	Recall	F1 Score
All	0.76	0.87	0.81
Short	0.77	0.94	0.84
Long	0.93	0.90	0.92

3) Effect of SBE severity

An error predictor that is able to label more severe application runs (i.e., with a higher number of SBEs) as SBE-affected is desirable. Towards this goal, we first group application runs into four levels of SBE severity (25 percentile per level), i.e., the bottom 25 percentile applications with the least number of SBEs are in level *Light* while the top 25 percentile ones are in level *Extreme*. Table VI presents the percentage of correctly classified SBE-affected runs in each level. Our results indicate that as the number of SBEs increases among application runs in our dataset, the effectiveness of the *TwoStage* method grows. For example, 74% of the application runs in level *Light* are already correctly predicted to be SBE-affected cases. The percentage number increases as the SBE severity level goes higher, becoming 95% for *Extreme* application runs. The results show that *TwoStage* is able to achieve high prediction quality for SBE occurrences, especially for those applications affected by more SBEs.

TABLE VI
PERCENTAGE OF CORRECTLY CLASSIFIED SBE-AFFECTED APPLICATION RUNS IN FOUR SEVERITY LEVELS.

Severity	Light	Moderate	Severe	Extreme
PCT.	74%	88%	93%	95%

VIII. DISCUSSION

Time Series-based Feature Prediction. As stated in Section VI, some input features into the *TwoStage* method cannot be known before the execution of application, such as the temperature and power consumption during the application run. Therefore, we need to leverage time-series prediction tools to forecast those features. Fortunately, there is a rich body of works on time-series prediction. ARMA/ARIMA [24] have been widely used for time series prediction in several systems areas. Tran and Reed [25] use ARIMA to improve block prefetching for scientific applications. Neural networks have been shown effective in capturing temporal and spatial dependencies within time series of data center resource usage [16, 26, 27]. Generally speaking, we can take advantage of these prediction tools to first forecast features based on time series, and then plug them into the *TwoStage* method for SBE occurrence prediction.

Application of SBE Prediction. Intuitively, GPU soft error prediction can work together with system scheduling. For example, based on the prediction result of SBE occurrences, one can dynamically turn on or turn off the ECC protection on targeted nodes and applications for the sake of lower ECC overhead. One may argue that the aftermath of mislabeling a SBE sample can be too much given the fact that no prediction technique can guarantee 100% accuracy. Several prior works indicate that this standpoint is too conservative. First, some hardware errors (i.e., transient bit flips) occurring during application runs can be masked in the final output, meaning that these errors are imperceptible by the end users [28–31]. Moreover, even those corrupted outputs are not always got rejected as long as the severity level of corruption is below a certain user-acceptable threshold [32] . For instance, in the field of approximate computing, users are willing to trade accuracy with better performance [33–37]. Similarly, it is desirable under certain situations to sacrifice accuracy for lower reliability overhead. In fact, due to the prohibitively high error protection overhead, computational scientists may opt to naively off error protection for their application runs [11]. The proposed error predictor allows to strike a balance between performance, overhead, and reliability.

IX. RELATED WORK

Characterizing system failures in HPC systems has been an important topic for decades [5, 38–40]. Oliner et al. [41] analyze logs collected from five HPC systems. Researchers have also looked specifically into DRAMs and HDDs and demonstrate pitfalls in error studies and their impact on system reliability assessment [5–7, 38, 42]. Unfortunately, there are relatively limited studies on GPU reliability of large scale systems. One reason is that the GPU architecture is relatively recently deployed in large-scale HPC systems, comparing to other components such as disks and CPUs. Martino et al. [43] investigate GPU errors in Blue Waters at the National Center for Supercomputing Applications, while recent efforts present GPU error characterization for the Titan supercomputer [4, 14]. Those studies point to spatial and temporal locality, resource utilization, workload type, error frequency, and correlation with jobs for various types of GPU errors. None of the above works look into the complex interplay of temperature, power consumption, and GPU SBEs. Close to the work presented in this paper, the impact of temperature and power consumption on GPU soft errors is examined in [15] and a neural-network-based model is proposed to predict the occurrences. In contrast to [15], we use a host of features from both the temporal and spatial perspectives and evaluate their effectiveness across multiple machine learning models.

CONCLUSION

In this paper, we analyzed large amounts of measured system related data to understand the characteristics of temperature, power, workload type, and SBE distribution across space and time. We propose several machine learning-based models that use workload and system features as input for GPU soft-error prediction. We examined their effectiveness under various scenarios and in multiple aspects including its accuracy, robustness, overhead, and model interpretations.

Acknowledgment We thank reviewers for their constructive feedback. The work was supported by in part through NSF grants CCF-1649087, CCF-1717532, Northeastern University, and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Lucy Nowell. This work also used in part the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE under contract number DE-AC05-00OR22725.

REFERENCES

[1] J. S. Vetter, R. Glassbrook *et al.*, "Keeneland: Bringing heterogeneous GPU computing to the computational science community," *Computing in Science & Engineering*, vol. 13, no. 5, pp. 90–95, 2011.

[2] D. Kothe and R. Kendall, "Computational science requirements for leadership computing," *Oak Ridge National Laboratory, Technical Report*, 2007.

[3] C. L. Mendes, B. Bode *et al.*, "Deploying a large petascale system: The blue waters experience," *Procedia Computer Science*, vol. 29, pp. 198–209, 2014.

[4] D. Tiwari, S. Gupta *et al.*, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 331–342.

[5] V. Sridharan, J. Stearley *et al.*, "Feng shui of supercomputer memory positional effects in DRAM and SRAM faults," in *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*. IEEE, 2013, pp. 1–11.

[6] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, no. 1, 2007, pp. 1–16.

[7] L. N. Bairavasundaram, A. C. Arpaci-Dusseau *et al.*, "Characteristics, impact, and tolerance of partial disk failures," Ph.D. dissertation, University of Wisconsin–Madison, 2008.

[8] N. El-Sayed, I. A. Stefanovici *et al.*, "Temperature management in data centers: Why some (might) like it hot," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 163–174, 2012.

[9] "Top500 list," https://www.top500.org/list/2016/06/, 2016.

[10] R. M. Betz, N. A. DeBardeleben *et al.*, "An investigation of the effects of hard and soft errors on graphics processing unit-accelerated molecular dynamics simulations," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 13, pp. 2134–2140, 2014.

[11] A. W. Gotz, M. J. Williamson *et al.*, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. generalized born," *Journal of Chemical Theory and Computation*, vol. 8, no. 5, pp. 1542–1555, 2012.

[12] "Understanding XID errors," http://docs.nvidia.com/deploy/xid-errors/index.html, 2015.

[13] S. Gupta, D. Tiwari *et al.*, "Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems," in *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 2015, pp. 37–44.

[14] B. Nie, D. Tiwari *et al.*, "A large-scale study of soft-errors on GPUs in the field," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 519–530.

[15] B. Nie, J. Xue *et al.*, "Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2017 IEEE 25th International Symposium on*. IEEE, 2017, pp. 22–31.

[16] J. Xue, F. Yan *et al.*, "PRACTISE: Robust prediction of data center time series," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 126–134.

[17] F. Provost, "Machine learning from imbalanced data sets 101," in *Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets*, 2000, pp. 1–3.

[18] N. V. Chawla, K. W. Bowyer *et al.*, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[19] R. Sipos, D. Fradkin *et al.*, "Log-based predictive maintenance," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2014, pp. 1867–1876.

[20] M. M. Botezatu, I. Giurgiu *et al.*, "Predicting disk replacement towards reliable data centers," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 39–48.

[21] I. Guyon, "A scaling law for the validation-set training-set size ratio," *AT&T Bell Laboratories*, pp. 1–11, 1997.

[22] N. V. Chawla, "Data mining for imbalanced datasets: An overview," in *Data Mining and Knowledge Discovery Handbook*. Springer, 2009, pp. 875–886.

[23] D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, 2011.

[24] G. E. Box, G. M. Jenkins *et al.*, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[25] N. Tran and D. A. Reed, "Automatic ARIMA time series modeling for adaptive I/O prefetching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 4, pp. 362–377, 2004.

[26] J. Xue, B. Nie *et al.*, "Fill-in the gaps: Spatial-temporal models for missing data," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–9.

[27] J. Xue, R. Birke *et al.*, "Spatial-temporal prediction models for active ticket managing in data centers," *IEEE Trans. Network and Service Management*, vol. 15, no. 1, pp. 39–52, 2018. [Online]. Available: https://doi.org/10.1109/TNSM.2018.2794409

[28] K. S. Yim, C. Pham *et al.*, "Hauberk: Lightweight silent data corruption error detector for GPGPU," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 287–300.

[29] B. Fang, K. Pattabiraman *et al.*, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 221–230.

[30] S. K. S. Hari, T. Tsai *et al.*, "SASSIFI: Evaluating resilience of GPU applications," in *Proceedings of the Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2015.

[31] G. Li, K. Pattabiraman *et al.*, "Understanding error propagation in GPGPU applications," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 240–251.

[32] R. Venkatagiri, A. Mahmoud *et al.*, "Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–14.

[33] S. Mitra and Y. Hayashi, "Bioinformatics with soft computing," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 5, pp. 616–635, 2006.

[34] R. K. Jena, M. M. Aqel *et al.*, "Soft computing methodologies in bioinformatics," *European Journal of Scientific Research*, vol. 26, no. 2, pp. 189–203, 2009.

[35] H.-L. Truong and T. Fahringer, "Soft computing approach to performance analysis of parallel and distributed programs," *Euro-Par 2005 Parallel Processing*, pp. 622–622, 2005.

[36] S. Mitra, S. K. Pal *et al.*, "Data mining in soft computing framework: a survey," *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 3–14, 2002.

[37] J. Meng, S. Chakradhar *et al.*, "Best-effort parallel execution framework for recognition and mining applications," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–12.

[38] A. A. Hwang, I. A. Stefanovici *et al.*, "Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 111–122.

[39] B. Schroeder, R. Lagisetty *et al.*, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016.*, A. D. Brown and F. I. Popovici, Eds. USENIX Association, 2016, pp. 67–80. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/schroeder

[40] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[41] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE, 2007, pp. 575–584.

[42] B. Schroeder, E. Pinheiro *et al.*, "DRAM errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 193–204.

[43] C. Di Martino, Z. Kalbarczyk *et al.*, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 610–621.