2019 IEEE International Conference on Big Data (Big Data)

Spatiotemporal Real-Time Anomaly Detection for Supercomputing Systems

Qiao Kang*, Ankit Agrawal*, Alok Choudhary*, Alex Sim‖, Kesheng Wu‖, Rajkumar Kettimuthu‡, Peter H. Beckman‡, Zhengchun Liu‡ and Wei-keng Liao*

* *ECE, Northwestern Univeristy*
Evanston, IL, USA
{qiao.kang, ankitag, choudhar, wkliao}@eecs.northwestern.edu

‖ *Lawrence Berkeley National Laboratory*
Berkeley, IL, USA
{asim, kwu}@lbl.gov

‡ *Argonne National Laboratory*
Lemont, IL, USA
{kettimut, beckman, zhengchun.liu}@anl.gov

***Abstract*—The demands of increasingly large scientific application workflows lead to the need for more powerful supercomputers. As the scale of supercomputing systems have grown, the prediction of fault tolerance has become an increasingly critical area of study, since the prediction of system failures can improve performance by saving checkpoints in advance. We propose a real-time failure detection algorithm that adopts an event-based prediction model. The prediction model is a convolutional neural network that utilizes both traditional event attributes and additional spatio-temporal features. We present a case study using our proposed method with six years of reliability, availability, and serviceability event logs recorded by Mira, a Blue Gene/Q supercomputer at Argonne National Laboratory. In the case study, we have shown that our failure prediction model is not limited to predict the occurrence of failures in general. It is capable of accurately detecting specific types of critical failures such as coolant and power problems within reasonable lead time ranges. Our case study shows that the proposed method can achieve a F_1 score of 0.56 for general failures, 0.97 for coolant failures, and 0.86 for power failures.**

***Index Terms*—Blue Gene/Q, system anomaly detection, RAS**

I. Introduction

The increasing data size of scientific application workflows has created a strong demand for more powerful supercomputers. As the scale of supercomputer systems grows, system failures, which can negatively affect system performance [1], become critical. If knowledge about failures can be predicted via heuristics, one can deploy an autonomous system that can schedule resources and actions in order to maximize the overall system performance, as argued in [2]. For example, failure recovery measures such as checkpoint saving can be used to reduce the cost from system failures [3] [4].

In this paper, we focus on failure predictions of IBM Blue Gene (BG) systems, but our method is not limited to these systems. Failure prediction for BG systems faces two significant challenges. First, the number of failure events is rare [5]. A classification problem with a small number of positive classes suffers from a high false-negative ratio. As a result, a model can have high accuracy but low precision and recall. Second, the prediction lead time, defined as the time difference between prediction and the failure, should be reasonably large; yet current lead times of only a few seconds do not give the system enough time to deploy failure prevention measures [6]. A more practical range of lead time is from minutes to hours, as argued in [5] and [6].

For the third-generation BG/Q, these two challenges have become more severe because of the growing reliability, availability, and serviceability (RAS) event log size. According to a case study of Mira logs from 2013 to 2017, the number of fatal events is much smaller than the number of nonfatal events. BG/P systems, on the other hand, did not have a similar ratio of nonfatal and fatal events, according to the case study of the Intrepid logs from 2013 to 2014. Moreover, Blue Gene/Q systems generate RAS events in real-time with high-volume RAS event data. For example, Mira generates 80 times more RAS event logs than did Intrepid from 2013 to 2014. Researchers have focused on failure predictions for the earlier Blue Gene systems using machine learning techniques, but the new challenges brought by the BG/Q system demand new methods that give better prediction accuracy and modeling of lead time.

To meet this demand, we propose a new formulation of features based on spatiotemporal locality assumptions mentioned in [7] and algorithms for failure prediction. We define the relation between system error and events. Then,

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

we apply a temporal clustering algorithm for fatal event records. A cluster represents a system error that should be detected. Later, we propose a feature-matching method for linking nonfatal event clusters and fatal event clusters that are likely caused by the same system error. We construct two types of features from every nonfatal event cluster. A neural network is proposed to model the correlation between the two types of features and the occurrence of system error. We also propose a real-time failure detection algorithm using the trained model.

We present a case study using our proposed method with six years of reliability, availability, and serviceability event logs recorded by Mira, a Blue Gene/Q supercomputer at Argonne National Laboratory. We use the 2013-2017 Mira RAS event data as training data for prediction models. Then, we emulate the real-time prediction of failures for the period of data in 2018 using the trained models. We use accuracy, precision, recall, and the F_1 score as metrics and present results based on a variety of model parameters. In addition to predicting the occurrence of failures in general, we present the use cases of our proposed algorithm for critical types of failures: coolant and power failures.

The rest of this paper is arranged as the following. In section 2, we briefly introduce the BG/Q system and discuss existing literature related to the Blue Gene system anomaly detection. In section 3, we propose our feature extraction techniques, training model, and real-time anomaly detection algorithm. In section 4, we present a case study of five years of Mira RAS event logs using our proposed methods.

II. Background and Related Work

BG/Q is the third generation of the IBM Blue Gene series of supercomputers that can be scaled up to 20 PFs [8]. Details about network and message units are described in [9].

BG/Q systems have two major types of nodes: compute nodes and I/O nodes. Compute nodes are used for running applications, and I/O nodes are used for storing files shipped from computing nodes. The compute nodes are interconnected in a five-dimensional torus topology. Every node has ten bidirectional ports with 2 GB/s bandwidth. In addition to computing and I/O nodes, service nodes are used to connect to every compute node, and every I/O node via JTAG interfaces with 1 Gb Ethernet for monitoring the system [10]. These service nodes report runtime noninvasive RAS events, which are the objectives of this study.

The Blue Gene/Q Reliability, availability, and serviceability (BG/Q RAS) Events Book[11] describes the details about RAS events of the BG/Q system. Every RAS event has many attributes, such as component, category, severity, and time stamp, location.

The followings attributes are of particular interest in this paper.

1) **Component**: the software component detecting and reporting the event, for example, CNK (compute node kernel), MC (machine controller), and MUDM (memory unit)
2) **Category**: the entity that encountered an error, for example, software error, BQC (chip error), Coolant, AC/DC power, and DDR (memory controller)
3) **Severity**: the various levels of severity: INFO – message that highlights the progress of system software; WARN – message that indicates potential harmful situations, such as a software error threshold or failure of a redundant component; and FATAL – the message that indicates severe system errors, which can lead to application fail or abort
4) **Event_time**: timestamp of the event in seconds
5) **Location**: the rack, midplane, node board, and node that an event is reported from

Predicting BG system failures is a concern of the high-performance computing (HPC) community for handling resilience on exascale supercomputers. Failures of high-end supercomputers can disrupt the running of applications. A commonly used strategy is to save checkpoints regularly. Later, if a node fails, the application can restart at another node with the data status at the saved checkpoints [3]. If future failures can be accurately predicted based on historical data, checkpoints can be saved accordingly, instead of regularly. Thus, the efficiency of handling resilience at the exascale can be improved with the knowledge of future failure predictions.

Liang et al. [12] introduced the concept of clustering and compression for RAS event data. Their methods can successfully remove more than 99% of raw RAS events without losing the accuracy for portraying the failures. Subsequently, Liang et al. [13] formulated the prediction of failures in BG/L systems as a problem of nonfatal and fatal RAS event correlation. They also showed the temporal and spatial localities of the events in the system. This assumption has also been validated in another study [7]. More recently, Liang et al. [5] proposed a feature extraction method for predicting failures in BG/L systems. This method, known as the period-based method, is widely used for fatal event prediction. A period-based model is a prediction model that uses a fixed temporal window size for feature extractions. The model used in [5] divides time into fixed-size intervals; features extracted from the intervals are used to predict failures in future intervals.

Following Liang and his colleagues' works, researchers proposed new techniques for predicting failures in BG systems. Gujrati et al. [14] proposed a meta-learning failure predictor for BG/L systems. Zheng et al. [15] proposed preprocessing methods for system RAS events.

The concept of lead time was introduced in [16]. Lead time refers to the time difference of failure and the alarm for that failure raised by a prediction model. Thompson et al. [17] have tested their intention to maximize the lead time for their model. Zheng [6] proposed an approach that

can estimate the lead time using the arithmetic mean of time differences between nonfatal and fatal events.

Event-driven approaches associate a fixed number of adjacent events reported before the occurrence of failures. Yu et al. [18] evaluated the impact of lead time and window size for both period-based and event-driven prediction models using a Bayesian network as a prediction model. Di etal. studied the correlation among fatal events [19] [20]. Later, they presented a similarity-based event filtering analysis for BG/Q system [21]. The event-driven approach differs from the period-based approach in the way that the time window for RAS events features are not fixed ranges. Instead, the number of events is fixed.

Similar to most machine learning problems, the modeling of failure occurrences for the BG/Q system can be divided into two phases: feature extraction and model fitting. In the feature extraction process, the input is a sequence of nonfatal events, and the output is if fatal events will occur in the future. The matching of the nonfatal event features to failure prediction labels is a challenging task [6]. If the nonfatal event features are not associated with the right fatal events, the prediction accuracy can be undesirable, since they do not necessarily correlate with each other. Zheng [6] used a randomized algorithm to correlate the pairs with a high Pearson correlation coefficient. However, their approach is applicable to post analysis of historical log instead of real-time prediction. Furthermore, recent studies have shown that lead time is critical. If the lead time is too small, the system does not have response time for checkpoint saving. On the other hand, if the lead time is too large, it is difficult to predict with high accuracy the timestamp interval that a potential failure can happen.

The most widely used real-time anomaly detection algorithm for BG systems is the period-based approach proposed by [5]. Although not discussed in their paper, the period-based method has the clear advantage of bounding the lead time. Moreover, their method can be readily deployed to control the nodes of real-time systems. Yu et al. [18] concluded that the event-driven feature extraction method constructs better features because the matching of nonfatal events and fatal events can be more accurate. However, real-time event-driven methods have not received sufficient attention because of the difficulty of feature construction. In the post-analysis of historical logs, we can associate fatal events with the last few nonfatal events according to their timestamp, but this binding method cannot be used in real-time systems since the goal is to predict whether those fatal events will occur or not. In this paper, we address the problem of event-driven real-time anomaly detection by proposing a new algorithm.

III. Design

In a supercomputer system, let $E = \{e_i : i \in \mathbb{N}\}$ be independent temporal system errors. E is a hidden sequence that a system-monitoring program aims to predict in advance. Every RAS event is associated with one of these system errors so that we can infer the system errors from the RAS events. Moreover, some system errors are severe errors that can prevent a block from booting. For those severe errors, they report at least one RAS event with FATAL severity.

Programs at control nodes can use RAS events that have been observed to predict whether any fatal RAS events will occur in the future. To be more specific, let $X = \{x_{t+i} : i \in [0, k]\}$ be a sequence of observed RAS events. Let Y be labels that indicate the occurrence of a system error at a certain time. An association function f can be used to make predictions of future failure occurrences.

The design has three objectives. First, X and Y should refer to the same $e_i \in E$. We use a spatiotemporal locality assumption to achieve this goal. Second, we want to apply the lead time constraint, which means that the time difference between the last timestamp of X and the occurrence of Y should be reasonable. For example, predicting some FATAL event that will happen in the next 60 hours is not useful, since the time range is too large. Similarly, predicting the occurrence of a fatal event that will happen in the next second is also not helpful since the system does not have enough response time. We achieve our second objective by using a time window. All predictions of failures should be within the specified time window. Third, the association function f must support real-time systems. To achieve this objective, we propose an anomaly detection algorithm that can utilize trained prediction models.

A. Identifying Independent Failures

The first question to be answered is the definition of system anomaly sequence E. We apply a temporal clustering algorithm for this purpose. The underlying assumption is that the fatal RAS event indicating the same system error has spatiotemporal locality. This assumption has been argued in [22], [23] and [7] .

For arbitrary RAS event records x_a and x_b, let $\text{Dist}_t : X \times X \to \mathbb{N}$ be the temporal distance function. The function is computed as the difference between the timestamps of x_a and x_b. Let h_t be the temporal distance threshold. For INFO/WARN RAS events and FATAL RAS events separately, we apply the clustering algorithm proposed in [12] to the RAS events with the temporal distance function Dist_t and threshold h_t to obtain temporal clusters T_F. The algorithm can be summarized as follows. For all events sorted in ascending order of their timestamps, if two adjacent events have a temporal distance less than h_t, they are joined into the same cluster. By the spatiotemporal locality assumption, RAS events in the same spatiotemporal cluster are associated with the same system errors.

B. Feature Construction

We build input feature vectors from raw RAS events for predicting occurence of failures. There are two types

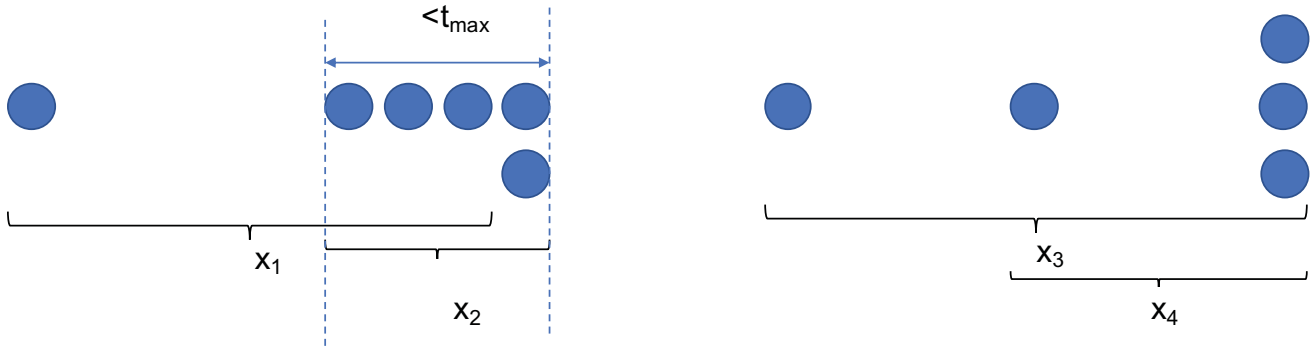


Figure 1: This figure illustrates an example for feature construction as mentioned in Algorithm 2. We set $n_{\text{event}} = 4$. The first feature x_1 consists of the first 4 RAS events. The second feature x_2 consists of 5 RAS events. The condition at Line 7 is triggered for building x_2. After constructing x_2, Line 12 is triggered to jump the starting index of next RAS event to the beginning of x_3.

Algorithm 1: Feature Construction

Data: Ras event sequence $r_1, .., r_n$, minimum number of events for prediction n_{event}, temporal merge threshold $t_{\max}$
Result: An array of feature vectors F.

```
1  F ← ∅
2  start ← 1
3  s ← 0
4  while i ≤ n do
5      if s = n_event then
6          end ← i
7          while end ≤ n or
             r_end.time − r_i.time < t_max do
8              end← end + 1
9          end
10         F add feature vector {r_start, ..., r_end-1}
11         i ← start
12         while i ≤ end and
             r_start.time − r_i.time < t_max do
13             i ← i + 1
14         end
15         start ← i
16         s ← 0
17     else
18         i ← i + 1
19         s ← s + 1
20     end
21 end
```

of variable construction, as mentioned in the background section. One approach is the period-based approach, and the other method is the event-driven approach.

Suppose we have RAS events sequence $x_0, x_1, x_2, ..$ ordered by timestamps. For event-driven approach proposed in [6], there is a parameter n_{event} that indicates a threshold for number of adjacent events for forming a variable. For instance, the j^{th} variable selected is $\{x_i : j \leq i < n_{\text{event}} + j\}$. Therefore, the total number of variables created is equal to the total number of events subtracted by $n_{\text{event}} - 1$. This method works well for BG/L and BG/P systems. However, for BG/Q systems, the number of RAS events is significantly larger than BG/P systems. For example, Mira dataset [24] has one RAS event per 3 seconds on average. Hence the control nodes have to make predictions every 3 seconds. Since adjacent variables only differ by 1 event, adjacent features have small differences given large n_{event}. Moreover, a lot of RAS events in BG/Q systems have precisely the same timestamp, so the variables created depend on the order of RAS events with the same timestamp, which is not well-defined for input variable construction.

We propose an event-driven approach for selecting input variables. Instead of selecting a fixed n_{event} number of RAS events per variable, the new approach sets n_{event} as a minimum threshold. Adjacent events with timestamp differences less than a threshold $t_{\max}$ are merged into the same feature. Therefore, a feature contains at least n_{event} number of RAS events. Algorithm 1 formally describes the proposed feature construction approach. The algorithm iterates through all events with the "while" loop at Line 4. If the number of accumulated events reaches n_{event} at Line 5, the algorithm constructs an input feature. In addition to events with index from variable start to i, Lines 6 to 9 absorb events with timestamps difference from i less than $t_{\max}$. Lines 11 to 14 shift the starting index of the next feature, jumping events with timestamps too close to $x$start. Figure 1 illustrates an example for how RAS events are merged. $t_{\max}$ is equal to 4 in the example. Feature x_1 and x_2 show the case that more than $t_{\max}$ number of events are absorbed into the same input feature by Lines 6 to 9 of Algorithm 1. x_3 shows that the case that the starting indices can have a gap for adjacent input features, which is handled by Lines 11 to 14 of Algorithm 1.

C. Realtime Anomaly Detection

The proposed feature construction method in Section III-B supports real-time anomaly detection.

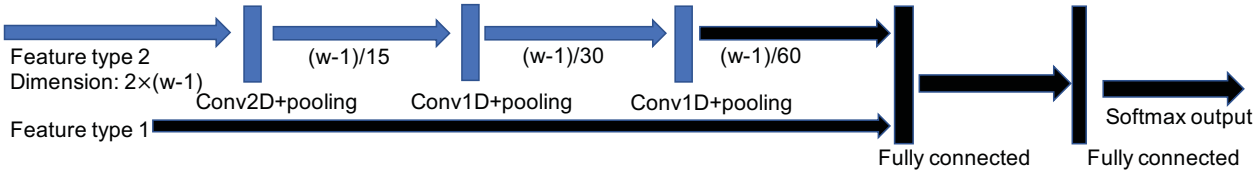


Figure 2: Illustration of convolutional neural network architecture used for training. The 2D convolutional layer has kernel size 2×15 and stride size 2×1. The pooling layer followed by the 2D convolutional layer has kernel size 1×15. The 1D convolutional layers have kernel size 3. The pooling layers followed by the 1D convolutional layers have kernel size 2. Padding is applied to all convolutional+pooling layers. Each of the fully connected layers has 2,048 hidden nodes.

Algorithm 2: Anomaly Detection Algorithm

Data: Maximum lead time threshold t_{lead}, minimum number of events triggering prediction n_{event}, prediction model $\hat{f}$

Result: Continously report if a failure will happen within the lead time range.

```
1  s ← 0
2  F ← Empty Array
3  while True do
4  |   r ← NextEvent
5  |   s ← s + 1
6  |   if r.time − F.last.time > t_lead ∨ s ≤ n_event
   |    then
7  |   |   Add r to F
8  |   else
9  |   |   Report f̂(F)
10 |   |   Clear F
11 |   |   Add r to F
12 |   end
13 end
```

For every input feature vector, we match it with the nearest independent failure with a larger starting timestamp. The difference between the starting timestamp of X and the starting timestamp of its matched independent failure is defined as the lead time. We can define a lead time threshold t_{lead}. If an input feature vector has lead time greater than t_{lead}, we label it with STATUS_SAFE. Otherwise, we label the input feature with STATUS_FATAL. This threshold is also denoted as maximum lead time.

Having constructed input features vectors for X and labels Y, we propose a deep learning model for emulating f. Figure 2 illustrates the proposed architecture of the deep learning model. There are two kinds of feature vectors. The first type summarizes the statistical distribution for the event attributes. The second type captures the spatiotemporal features for event occurrence. Therefore, the proposed model consists of two parts. The first part is a convolutional neural network that has type 2 features as input. After two steps of convolution and pooling, the dimension is reduced to $1 \times \frac{1}{30}w$. In the second part, the output of the convolutional neural network is joined with type 1 features. The joined vector is fed into a fully connected neural network for softmax classification.

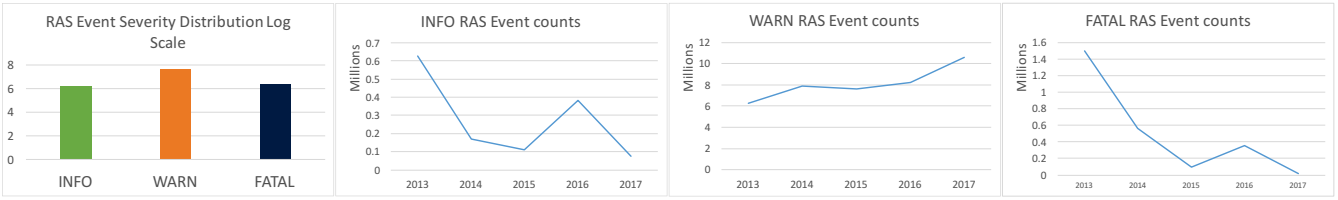
For the first type of feature vector, we reuse two features proposed in [5], namely, the mean of time intervals between adjacent events in X and the time elapsed since the occurrence of last fatal event. In addition, we propose the following features. Every event in X has component and category attributes in terms of strings. We compute the joint probability of the component and category pair in X. The joint probability distribution models the type of system error. Moreover, we count the number of distinct spatial locations for events in X. There are four distinct spatial levels: rack, midplane, node board, and node. This feature indicates whether X contains events from a wide range of locations or not.

For the second type of feature vector, we record the spatiotemporal difference between adjacent pairs of events in X. We record a vector of size $2 \times |X|$: one dimension is for temporal differences, and the other dimension is for spatial differences. The temporal difference is defined as the difference in timestamps of two events. The spatial difference is defined as the difference in spatial level. If two RAS events are on different racks, they have spatial difference 4. Otherwise, if they are on different midplanes, they have spatial difference 3. Otherwise, if they are on different node boards, they have spatial difference 2. If they are on the same node board, but different node, they have spatial difference 1. If the two events are on the same node, they have spatial difference 0. This type of feature vector explains spatiotemporal variations within X.

Algorithm 2 illustrates our proposed anomaly detection algorithm. A while loop at Line 3 keeps receiving RAS events reported from the control node. The algorithm dynamically constructs features in realtime using the same feature construction strategy described in Algorithm 1. After a feature vector is constructed, the algorithm report if there are any anomalies using regression function $\hat{f}$.

IV. Experimental Results

We use the Mira dataset [24] to evaluate the proposed algorithms. This dataset contains RAS event logs from 2013 to 2018. Figure 3a summarizes the number of events by severity. Most events have WARN-level severity. Events with INFO severity in Figures 3b, 3c, and 3d indicate the number of events across all years by severity. We can



a b c d

Figure 3: (a) Log scale (based 10) number of events for 2013–2018. (b) Number of RAS events with INFO severity across all five years. (c) Number of RAS events with WARN severity across all five years. (d) Number of RAS events with FATAL severity across all five years.

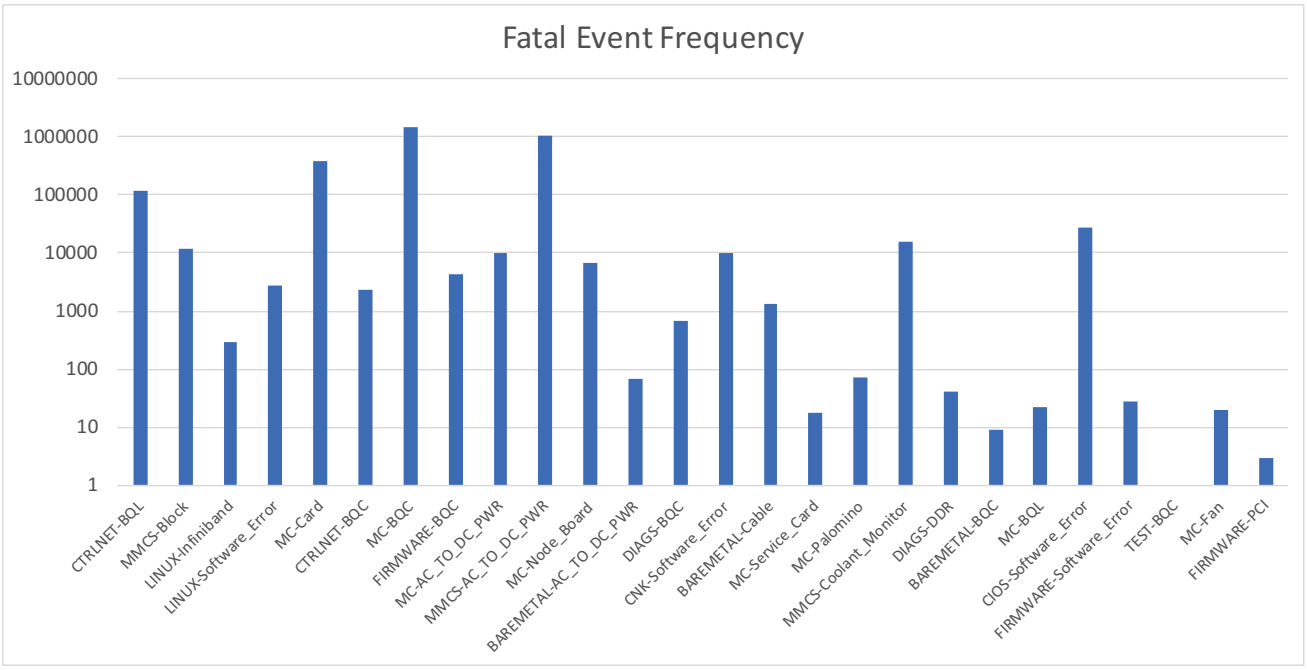


Figure 4: The frequency of all fatal events in logarithm (10) scale.

observe that there are far more RAS WARN events than events with INFO and FATAL severity. Almost every 3 seconds, a WARN event will be reported.

Our case study is the type of application that adopts a prediction model for real-time anomaly detection along with a single time series. Supercomputing systems are constantly improved. Past errors may not occur in the future anymore. Results for using future data to predict past data do not necessarily reflect the model's efficacy. Thus, we should not split the training and testing data in the way that some training data has larger timestamps than testing data since models trained with such settings are not useful for real-world applications. Consequently, the training part of the data is composed of RAS events from 2013 to 2017. RAS events with timestamps in 2018 are used for evaluation.

Table I summarizes the distribution of RAS events in training and testing data according to severity. While the number of events in 2018 for testing is approximately 20% of the total number of events from 2013 to 2018, we can observe that the distribution of fatal events is not uniform

Table I: Number of events used for training and testing. Events for training part have timestamps from 2013 to 2018. Events for testing part have timestamps in 2018.

Severity	Training	Testing
INFO	16, 725, 291	4, 800, 661
WARN	46, 127, 365	13, 762, 599
FATAL	2, 580, 811	239, 719
Total	65, 433, 467	18, 802, 979

in training and testing data. In 2018, the number of fatal events was only 1.3% of the number of all events in this time range. On the other hand, from 2013 to 2017, the number of fatal events is 3.9% of all events in this time range. Furthermore, if we apply the method proposed in Section III-A for clustering all fatal events with a temporal threshold 1 minute. The total numbers of fatal clusters are 3076 from 2013 to 2017 and 119 in 2018. Therefore, the testing set has much less sum of the positive class than the training set. Since the dataset is highly imbalanced, a random guesser on the testing set cannot achieve an expectation of F_1 score more than 0.05. Thus, the failure detection task is challenging.

These distributions of fatal events create a challenge for

Table II: Case study for the Mira dataset using proposed methods. A prediction model $\hat{f}$ is trained with data from 2013 to 2018 by using features extracted by Algorithm 1. We apply Algorithm 2 with the trained model to predict failures in 2018.

n_{event}	Max lead time	Accuracy	Precision	Recall	F_1
300	1800	99.4%	37.3%	65.9%	47.6%
300	3600	99.3%	43.8%	67.5%	53.8%
300	7200	99.0%	39.4%	67.5%	51.4%
600	1800	99.6%	41.3%	47.1%	44.0%
600	3600	99.5%	40.8%	64.7%	50.0%
600	7200	99.6%	46.4%	63.9%	53.8%
900	1800	99.6%	67.4%	39.5%	49.8%
900	3600	99.4%	50.5%	47.6%	49.0%
900	7200	98.7%	28.7%	51.6%	36.9%
1200	1800	99.6%	56.5%	43.1%	48.9%
1200	3600	99.4%	46.7%	45.7%	46.2%
1200	7200	99.2%	71.1%	45.7%	55.6%

Table III: Case study for the Mira dataset using period-based approach proposed in [5] with training data from 2013-2017. We apply Algorithm 2 with the trained model to predict failures in 2018.

Period	Observation Window	Accuracy	Precision	Recall	F_1
1800	5	95.9%	21.3%	35.8%	26.7%
1800	10	95.5%	13.4%	24.6%	17.4%
3600	5	91.8%	28.0%	36.5%	31.7%
3600	10	91.9%	35.7%	39.2%	37.4%
7200	5	87.9%	37.3%	54.5%	45.4%
7200	10	85.5%	42.1%	43.7%	42.9%

prediction models since models can overfit the training dataset based on the fatal event number. We address the imbalanced data problem by multiplying larger weights to the positive class in the cost function during gradient descent. Thus, false-negative prediction has a higher cost in the training process. Therefore, this method prevents the model from overfitting accuracy by reducing the false-negative predictions. We also applied the receiver operating characteristic (ROC) curve for determining the threshold of classification.

A. General Failure Detection

We present a case study for detecting general occurrence of system errors for the Mira dataset. Failure prediction models are built offline. Offline models do not require frequent updates given abundant historical data. However, a frequent update of the prediction models can incorporate failure types that are unobserved in the past. Nonetheless, in our case study, we assume the model is updated once a year. Algorithm 2 utilizes the prediction model for real-time prediction of unseen data from the future. We present simulation results.

Figure 4 illustrates the frequency of all types of fatal events. We can observe that "MC-BQC" and "MMCS-AC_TO_DC_PWR" have more than one million events. Most failures are related to these two. MC refers to the machine controller running on the service node, and MMCS refers to the control system running on the service node. BQC refers to compute chip error. This failure causes the termination of computing jobs. AC_TO_DC_PWR refers to circuit power issues. It implies that the entire node board is in error. All jobs must be terminated on the node board. Thus it is a more severe type of error.

Firstly, we build models for predicting any types of failures in order to make a direct comparison with the previous study. Later, we build prediction models for "MMCS-AC_TO_DC_PWR" and "MMCS-Coolant," which are detrimental failures that can be the interests of system administrators.

On the testing dataset, let the true positive tp be the number of STATUS_FATAL results that $\hat{f}$ predicts correctly. Let the false-negative fn be the number of STATUS_FATAL results that $\hat{f}$ fails to predict. Let the false-positive fp be the number of false-alarm STATUS_FATAL results that $\hat{f}$ reports incorrectly. Let the true negative tn be the number of STATUS_SAFE results that $\hat{f}$ predicts correctly. We use accuracy, precision, recall, and the F_1 score as the evaluation metrics. Accuracy is the percentage of predictions that $\hat{f}$ is correct on the testing dataset, defined as $\frac{\text{tp+tn}}{\text{tp+tn+fp+fn}}$. Precision is the percentage of STATUS_FATAL results that $\hat{f}$ can retrieve from the testing dataset, defined as $\frac{\text{tp}}{\text{tp+fp}}$. Recall that the percentage of predictions with output STATUS_FATAL that $\hat{f}$ made are correct, $\frac{\text{tp}}{\text{tp+fn}}$. The F_1 score is defined as $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$.

We present the results detection of all system failures in Table II with different choices of parameters. In general, the accuracy of prediction on the testing dataset is high: more than 98%. High accuracy alone is not sufficient since the dataset is highly imbalanced. Positive class is approximately 5% in the constructed features. Thus, a classifier that always predicts negative class can also achieve 95% accuracy. Therefore, we must present precision and recall results.

One observation is that as we increase n_{event}, the recalls have a decreasing trend, regardless of the maximum lead time for a fatal prediction. Precision, on the other hand, is less stable due to the imbalanced nature of testing data. The fatal clusters that define the binary labels of the classification contain too many different types of fatal events. More than 2,500,000 of fatal events are divided into approximately 3,000 clusters. Thus, it is possible that one cluster contains many independent system errors over an extensive time range. Different types of system error can have very different prediction lead time, which is justified in the next section. However, the occurrence of a fatal cluster is determined by the first independent system error. Therefore, if n_{event} is not large enough, the prediction model may not be able to cover the features for the first system error. Thus, increasing n_{event} improves recalls of the model. However, setting n_{event} to be too large could cause the curse of dimensionality problem. Lots of noise are introduced into the input feature.

We implemented the period-based fatal prediction ap-

proach proposed in [5]. This approach is the most widely used real-time anomaly detection algorithm for BG system, so we make a direct comparison of our method to it. Using the same training and testing dataset, we summarize the results in Table III with different parameters suggested in the paper. The Period column refers to the observation window size in seconds. It defines the maximum lead time of the prediction model. The Observation Windows column refers to the number of observation windows used for building the input features. 5 and 10 are reasonable parameters suggested in the original paper.

With the same maximum lead time, our proposed anomaly detection algorithm achieves higher accuracy, precision, and recall. The proposed algorithm adopts a new event-based feature extraction algorithm. It avoids the association of unrelated input features and prediction according to the spatiotemporal locality assumption. Furthermore, with the help of two different types of features, the model can achieve better results.

B. Critical Failure Detection

As mentioned earlier, "MMCS-AC_TO_DC_PWR" and "MMCS-Coolant" (control system circuit and coolant failures) are the fatal events that report the failure of a large partition of supercomputing systems.

We filter out the fatal events with attributes "MMCS-AC_TO_DC_PWR" and "MMCS-Coolant." Similar to the previous experiments, we use 2013-2017 RAS event data as training data and 2018 RAS event data as testing data. Depending on the different maximum lead time, the distributions of positive and negative classes vary. In our case study, the maximum lead time is from 900 to 7200 seconds. For coolant failures, the number of the positive class in training data is from 2067 to 2148 in training data and from 939 to 947 in testing data. The number of negative class in training data is from 107349 to 107430 in training data and from 18235 to 18243 in testing data. For AC/DC power failures, the number of the positive class in training data is from 1466 to 2388 in training data and from 575 to 650 in testing data. The number of negative class in training data is from 79266 to 80188 in training data and from 20994 to 21069 in testing data. Features for both failures are highly imbalanced. The AC/DC power failure has a smaller proportion of the positive class, so failure detection for it can be more challenging than coolant failure.

Table IV illustrates the precision, recall, and F_1 score of "MMCS-AC_TO_DC_PWR" and "MMCS-Coolant" failures detected with different parameters. We do not present the accuracy results since all of them are extremely high, ranging from 99.8% to 99.9%.

For Coolant failure results presented in Table IV, we can observe the following trends. Setting the maximum lead time 1800 seconds gives the best results for any n_{event}. Given the same lead time 1800 seconds, $n_{\text{event}} = 60$ yields best precision and recall, which is up to 97% F_1 score.

Table IV: Case study for the Mira dataset using proposed methods for coolant and power related events. A prediction model $\hat{f}$ is trained with data from 2013 to 2018 by using features extracted by Algorithm 1. We apply Algorithm 2 with the trained model to predict failures in 2018.

Type	n_{event}	Max lead time	Precision	Recall	F_1
Coolant	15	900	69.7%	88.5%	78.0%
Coolant	15	1800	69.7%	88.5%	78.0%
Coolant	15	3600	69.7%	85.2%	76.7%
Coolant	15	7200	69.4%	80.6%	74.6%
Coolant	60	900	82.4%	100%	90.3%
Coolant	60	1800	94.1%	100%	**97.0**%
Coolant	60	3600	94.1%	91.4%	92.8%
Coolant	60	7200	94.1%	78.0%	85.3%
Coolant	240	900	59.1%	100%	74.3%
Coolant	240	1800	68.2%	88.2%	76.9%
Coolant	240	3600	68.2%	75.0%	71.4%
Coolant	240	7200	68.2%	65.2%	66.7%
AC/DC	15	900	35.2%	62.7%	45.1%
AC/DC	15	1800	42.3%	55.6%	48.1%
AC/DC	15	3600	54.6%	51.7%	53.1%
AC/DC	15	7200	43.5%	46.3%	44.9%
AC/DC	60	900	49.2%	48.9%	49.1%
AC/DC	60	1800	52.6%	50.0%	51.2%
AC/DC	60	3600	71.5%	61.8%	66.3%
AC/DC	60	7200	51.5%	58.2%	54.7%
AC/DC	240	900	65.2%	51.0%	57.3%
AC/DC	240	1800	28.1%	53.3%	36.8%
AC/DC	240	3600	97.5%	77.4%	**86.3**%
AC/DC	240	7200	97.5%	71.83%	82.7%

For the AC to DC power failure results, $n_{\text{event}} = 240$ and maximum lead time of 3600 seconds yield the best precision and recall.

The best parameter settings for coolant and AC to DC power suggested that the number of events and lead time we should choose for different types of failures can be different. If the number of events we use for making a prediction is too large for coolant failure (240 events), we could introduce too much noise to the input feature, which could cause overfitting problems during the training, given the fact that the number of coolant events is very small. AC to DC power failure, on the other hand, has 100 times more events than the coolant failure. Thus, it is preferable to use a larger size of input feature. The best choice of lead time depends on the system characteristics. Broad lead time can increase the false-positive rate because the number of the positive class is forced to be reduced. Choosing a narrow lead time range can increase the false-negative rate since the scope of the negative class is enlarged.

In this section, we have shown that it is possible to build models for specific types of failures. Compared with results in Table II, the best results in Table IV are much better. Mentioned earlier, the traditional assumption for spatiotemporal locality of a system error may not be applicable to BG/Q data. Multiple independent system errors can occur in a single time range since the total number of components in the whole system is enormous. Thus, the detection occurrence of specific types of failure can be a good future direction.

V. Summary

In this paper, we have proposed an algorithm for detecting system anomalies in the BG/Q system. The problem is challenging because system failures are rare. The algorithm can construct features based on a spatiotemporal locality assumption and make a prediction using our customized convolutional neural network in real-time. Experimental results have shown that the proposed approach has better prediction accuracy compared with the traditional period-based method in a previous study with controllable lead time. We have found that our proposed approach is applicable for detecting specific critical types of failures.

There are opportunities to improve the failure prediction model better. The current prediction model is a binary classifier. It can be generalized to a multi-class classifier. In other words, instead of the predicting occurrence of a failure within a fixed lead time, the model can output probabilities of failures within a different length of lead time ranges in real-time.

Acknowledgments

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357, DE-AC02-05CH11231, DE-SC0014330 and DE-SC0019358. The RAS event data we used in this paper was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This research also used resources of the National Energy Research Scientific Computing Center.

References

[1] 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.

[2] R. Kettimuthu, Z. Liu, I. T. Foster, P. H. Beckman, A. Sim, K. Wu, W.-k. Liao, Q. Kang, A. Agrawal, and A. N. Choudhary, "Towards autonomic science infrastructure: Architecture, limitations, and open issues.," in *AI-Science@ HPDC*, pp. 2–1, 2018.

[3] "Resilience at exascale." http://snir.cs.illinois.edu/PDF/UWM-resilience.pdf, 2012.

[4] K. Tang, D. Tiwari, S. Gupta, P. Huang, Q. Lu, C. Engelmann, and X. He, "Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 311–322, IEEE, 2016.

[5] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pp. 583–588, IEEE, 2007.

[6] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for blue gene/p," in *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on*, pp. 15–22, IEEE, 2010.

[7] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems," *Journal of Parallel and Distributed Computing*, vol. 69, no. 7, pp. 652–665, 2009.

[8] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, *et al.*, "The ibm blue gene/q compute chip," *Ieee Micro*, vol. 32, no. 2, pp. 48–60, 2012.

[9] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The ibm blue gene/q interconnection network and message unit," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pp. 1–10, IEEE, 2011.

[10] P. Wautelet, M. Boiarciuc, J. Dupays, S. Giuliani, M. Guarrasi, G. Muscianisi, and M. Cytowski, "Best practice guide—blue gene/q v1. 1.1," *PRACE-Partnership for Advanced Computing in Europe*, 2014.

[11] "Bluegene/q ras events." https://reports.alcf.anl.gov/data/datadictionary/RasEventBook.html, 2014.

[12] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a bluegene/l prototype," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pp. 476–485, IEEE, 2005.

[13] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pp. 425–434, IEEE, 2006.

[14] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A meta-learning failure predictor for blue gene/l systems," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, pp. 40–40, IEEE, 2007.

[15] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pp. 572–577, IEEE, 2009.

[16] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pp. 161–174, IEEE, 2007.

[17] J. Thompson, D. W. Dreisigmeyer, T. Jones, M. Kirby, and J. Ladd, "Accurate fault prediction of bluegene/p ras logs via geometric reduction," in *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on*, pp. 8–14, IEEE, 2010.

[18] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for blue gene/p: Period-based vs event-driven," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pp. 259–264, IEEE, 2011.

[19] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello, "Exploring properties and correlations of fatal events in a large-scale hpc system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 361–374, 2018.

[20] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of hpc log events," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 442–451, IEEE, 2017.

[21] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello, "Characterizing and understanding hpc job failures over the 2k-day life of ibm bluegene/q system," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 473–484, IEEE, 2019.

[22] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: Long-term measurement," *Analysis, and Implications. In SC*, 2017.

[23] L. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir, "Reducing waste in extreme scale systems through introspective analysis," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 212–221, IEEE, 2016.

[24] "Aclf mira." https://reports.alcf.anl.gov/data/mira.html, 2017.