Failure Prediction in IBM BlueGene/L Event Logs

Conference Paper · November 2007

DOI: 10.1109/ICDM.2007.46 · Source: IEEE Xplore

Yanyong Zhang
Rutgers, The State University of New Jersey
152 PUBLICATIONS **7,123** CITATIONS
SEE PROFILE

Hui Xiong
Southwest University, China
395 PUBLICATIONS **12,444** CITATIONS
SEE PROFILE

Ramendra Sahoo
Teradata
49 PUBLICATIONS **2,047** CITATIONS
SEE PROFILE

Seventh IEEE International Conference on Data Mining

Failure Prediction in IBM BlueGene/L Event Logs

Yinglung Liang, Yanyong Zhang
ECE Department, Rutgers University
{ylliang, yyzhang}@ece.rutgers.edu

Hui Xiong
MSIS Department, Rutgers University
hxiong@andromeda.rutgers.edu

Ramendra Sahoo
IBM TJ Watson Research Center
rsahoo@us.ibm.com

Abstract

Frequent failures are becoming a serious concern to the community of high-end computing, especially when the applications and the underlying systems rapidly grow in size and complexity. In order to develop effective fault-tolerant strategies, there is a critical need to predict failure events. To this end, we have collected detailed event logs from IBM BlueGene/L, which has 128K processors, and is currently the fastest supercomputer in the world. In this study, we first show how the event records can be converted into a data set that is appropriate for running classification techniques. Then we apply classifiers on the data, including RIPPER (a rule-based classifier), Support Vector Machines (SVMs), a traditional Nearest Neighbor method, and a customized Nearest Neighbor method. We show that the customized nearest neighbor approach can outperform RIPPER and SVMs in terms of both coverage and precision. The results suggest that the customized nearest neighbor approach can be used to alleviate the impact of failures.

1 Introduction

The large processing and storage demands of meta-scale scientific and engineering applications have led to the development and deployment of IBM BlueGene/L. As applications and the underlying platforms scale to this level, failures are becoming a serious concern [4, 10], as they can have severe impact on the system performance and operational costs. However, preventing failures from occurring is very challenging, if at all possible. Our earlier study [12] pointed out that the capability of predicting the time/location of the next failure, though not perfect, can considerably boost the benefits of runtime techniques such as job checkpointing or scheduling [12, 6].

Failure prediction is a challenging problem, mainly due to the fact that the number of fatal events (in a level of hundreds) is far exceeded by the number of nonfatal events (in a level of millions). In the field of data mining and machine learning, this problem is known as rare class analysis problem, which is believed to be challenging. Another important obstacle is the lack of real-world data and consequently the lack of the thorough understanding of their properties. Despite these challenges, there have been several attempts [11, 8, 9] to derive realistic failure prediction models. While these studies have demonstrated reasonable prediction accuracy, they failed to prove that their prediction methodology is of much practical use. For instance, some focus on long term prediction based on seasonal system behavior, without pinpointing the occurrence times of the failures [11]. As another example, some studies tried to predict whether there will be a failure in the next 30 seconds [9], and within such a short notice, no remedial measures can be taken, especially for large-scale systems. To address these issues, in this study, we derive our prediction models from the failure logs collected from IBM BlueGene/L over a period of 142 days. To emphasize both prediction accuracy and prediction utility, we partition the time into fixed windows (each window is a few hours), and attempt to predict whether there will be a fatal event in every window based on the event patterns in the preceding windows.

Our prediction effort consists of two main parts. First, we need to extract a set of features that can accurately capture the characteristics of failures. After establishing features, we exploit four classifiers including RIPPER (a rule-based classifier), Support Vector Machines (SVMs), a traditional Nearest Neighbor, and a customized Nearest Neighbor for predicting failure events. Our evaluation results show that the customized nearest neighbor predictor can substantially outperform them in both coverage and precision. We also discuss the feasibility of employ the nearest neighbor prediction to improve the runtime fault-tolerance of IBM BlueGene/L.

The rest of the paper is organized as follows. In Section 2, we provide details on the BlueGene/L failure data sets. Our overall prediction methodology and feature selection algorithm are presented in Section 3. Next, we discuss the details of the three classifiers in Section 4, and the detailed evaluation results in Section 5. Finally, we present the concluding remarks in Section 7.

2 Overview of BlueGene/L and the RAS Event Logs

BlueGene/L has 128K PowerPC 440 700MHz processors, which are organized into 64 racks. Each rack consists of 2 midplanes, and a midplane (with 1024 processors) is the granularity of job allocation. A midplane contains 16 node cards (which houses the compute chips), 4 I/O cards (which houses the I/O chips), and 24 midplane switches (through which different midplanes connect). RAS events are logged through the Machine Monitoring and Control System (CMCS), and finally stored in a DB2 database engine. The logging granularity is less than 1 millisecond. More detailed descriptions of the BlueGene/L hardware and the logging mechanism can be found in [4].

Every RAS event that is recorded by IBM BlueGene/L CMCS has the following relevant attributes: (1) *SEVERITY* of the event, which can be one of the following levels - INFO, WARNING, SEVERE, ERROR, FATAL, or FAILURE - in the increasing order of severity (our primary focus in this study is consequently on predicting the occurrence of FATAL and FAILURE events); (2) *EVENT_TIME*, which is the time stamp associated with that event; (3) *JOB_ID*, which denotes the job that detects this event; (4) *LOCATION* of the event; and (5) *ENTRY_DATA*, which gives a short description of the event. More detailed information can be found in [4].

In this study, we collected RAS event logs from BlueGene/L in the period from August 2, 2005 to December 21, 2005. Before we develop failure prediction models, we first need to preprocess the RAS event logs, which involves eliminating the redundancy of the event logs using the adaptive semantic filter proposed in [5].

3 Problem Definition and Methodology Overview

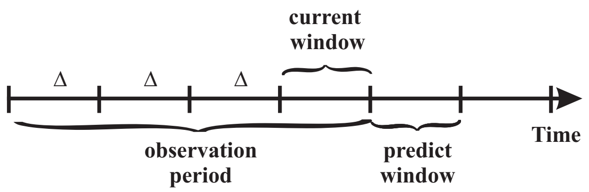


Figure 1. The illustration of our prediction methodology. We use Δ to denote the duration of a time window.

Figure 1 illustrates the basic idea of our prediction methodology. As shown in the figure, we seek to predict whether there will be fatal events in the next Δ interval, which we call a *prediction* window, based on the events in the *observation period* (with duration $T = 4\Delta$ in this example), which usually consists of several windows. Though we consider the event characteristics during the entire observation period, we consider events in the current window (the one immediately preceding the prediction window) more important. In this paper, we attempt to forecast whether fatal events will occur in the prediction window; we classify the prediction window as FATAL if fatal events are predicted to occur during the period, and NONFATAL if no fatal event is predicted.

In order to forecast whether the next time window will encounter fatal events, we identify the following features from those events that occurred during the observation period:

1. The first group of features represent the number of events (at all 6 severity levels) that occurred during the current time window. Specifically, we have *InfoNum*, *WarningNum*, *SvrNum*, *ErrNum*, *FlrNum*, and *FatNum*. For example, *FatNum* is used to signify how many fatal events occurred during the current time window. In total, we have 6 features in this group.
2. The second group of features represent the number of events (at all 6 severity levels) accumulated over the entire observation period. Specifically, we have *AcmInfoNum*, *AcmWarningNum*, *AcmSvrNum*, *AcmErrNum*, *AcmFlrNum*, and *AcmFatNum*. In total, we have 6 features in this group.
3. The third group of features describe how the events (at all 6 severity levels) are distributed over the observation period. To this end, we break down a time window (with duration Δ) into a number of smaller sampling intervals (with duration δ), and count the number of events in each sampling interval over the entire observation period. For instance, if we have $\Delta = 1$ hour, $\delta = 10$ minutes, and the observation period of 5Δ, then we will have 30 sampling intervals in total, and therefore, 30 samples for each severity level. Each of these features is named to include the following three parts: (1) its severity level, (2) the keyword "Sample_", and (3) the index of the corresponding sampling interval. For example, *FatalSample_23* signifies the number of FATAL events in the $23rd$ sampling interval. In addition to the individual sample values, we also report

the mean and variance of the samples for each severity level, and name these features as *InfoSampleMean*, *InfoSampleVariance*, etc. As a result, the total number of features we have in this group is $6(\frac{k\Delta}{\delta}+2)$, where $k\Delta$ is the observation period duration.

4. The fourth group of features are used to characterize the inter-failure times. There is only one feature that belongs to this group, i.e. the elapsed intervals since last FATAL interval, which we call *ItvNum2Fatal*.
5. The last group of features describe how many times each entry data phrase occurred during the current period. Entry data phrases are entry data keywords after we take out numbers, punctuation, file names, directory names, etc. from the entry data [5]. In our study, there are totally 529 phrases from 1,792,598 RAS events. Therefore, we have 529 features in this group, each corresponding to one phrase, and these features are named in the format of *Phrase_245*, where 245 is the phrase ID.

After we establish the above raw feature sets, we next need to preprocess these features to make them suitable for the later prediction study. Feature preprocessing consists of two steps: first normalizing the numeric feature values and then calculating each feature's significance major. For a numeric feature value v that belongs to feature V, we calculate the normalized value as follows:

$$v' = \frac{v - median(V)}{std(V)}. \tag{1}$$

After normalizing each feature value, we next calculate feature V's significance major $SIM(V)$ as follows:

$$SIM(V) = |\frac{mean(V_F) - mean(V_{NF})}{std(V)}|. \tag{2}$$

Here, we partition all the values of feature V into two parts: V_F containing the feature values of the FATAL class (i.e. these features collected from an observation period followed by a FATAL window), and V_{NF} containing the feature values of the NONFATAL class (i.e. these features collected from an observation period followed by a NONFATAL window). A feature will have a high significance major if its values differ significantly between FATAL records and NONFATAL records. In the prediction study, we sometimes only use features whose significance index values are above a certain threshold (T_{SIM}).

4 Prediction Techniques

In this study, we exploited three well-known prediction techniques: RIPPER (a rule-based classifier), Support Vector Machines (SVMs) [3], and Nearest Neighbor based classification for predicting failure events.

4.1 RIPPER

As a rule-based classifier, RIPPER has been shown able to effectively predict rare classes [2]. When using RIPPER, we first feed the feature values as described in Section 3 to RIPPER as input, and RIPPER will output a set of classification rules. Some example rules are provided below:

1. FATAL :- Phrase_216>=97, InfoSample_55<=22 (46/9)
2. FATAL :- Phrase_213>=3, InfoNum<=1669, InfoSample_98<=1 (37/20)
3. FATAL :- InfoSample_105>=34 (12/6)

We note that the numbers included in the parenthesis at the end of each rule denote the number of hits and misses caused by that rule. Finally, rules induced from the training data will be applied on test data to make prediction.

4.2 Support Vector Machines

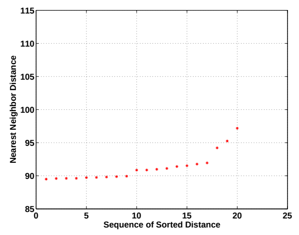
The second classifier we employed in this study is Support Vector Machines (SVMs), which are a set of generalized linear classifiers. The primary advantage of SVMs is that, after the training phase is completed, the actual prediction can be fast. Also, in many real-word cases [1], SVMs have best classification performance. A drawback of SVMs is that the training cost can be expensive if there are a large number of training samples. For example, in our study, it usually took 10 hours to run the training data that contain 2890 records each with 986 features. In this study, we chose LIBSVM [1] with the Radial Basis Function (RBF) kernel and the five-fold CV.

4.3 A Bi-Modal Nearest Neighbor Predictor

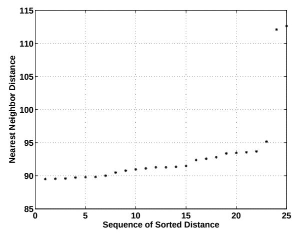
In addition to using off-the-shelf prediction tools, we also designed a nearest neighbor predictor.

For our nearest-neighbor prediction, we partition the data sets into three parts: *anchor* data, *training* data and *test* data. From the anchor data, we identify all the FATAL intervals in which at least one FATAL event occurred, and compute the feature values in the corresponding (preceding) observation periods. These feature values serve as a base for distance computation, and are organized into a two-dimensional matrix where rows correspond to each FATAL interval and columns correspond to each feature. We call this matrix the *anchor matrix*, and each row of this matrix a *feature vector*.

After establishing the anchor matrix from the anchor data, we next process all the FATAL features from the training data, and calculate the nearest distance between each



(a) FATAL distance map



(b) NONFATAL distance map

Figure 2. The distance maps with $\Delta = 12$ hours, $T = 36$ hours and no sampling.

FATAL feature vector and all the rows in the anchor matrix. The resulting distance values form a FATAL nearest distance map. Similarly, we can also prepare the NONFATAL feature vectors and thus the NONFATAL nearest distance map. Examining these two maps carefully, we can characterize the properties of the nearest neighbor distributions for both FATAL and NONFATAL classes, and based on these properties (which hopefully are distinct), we can make prediction on the test data.

d_1/d_2	f_{FF}	f_{FN}	f_{NF}	f_{NN}	p (%)	r (%)	F (%)
89.8/91.4	13	15	7	10	46.43	65.0	54.17
89.9/91.4	15	17	5	8	46.88	75.0	57.69
90.0/91.4	16	17	4	8	48.48	80.0	60.38
90.1/91.4	16	18	4	7	47.06	80.0	59.26
90.0/91.5	15	16	5	9	48.39	75	58.82

Table 1. Prediction results using different (d_1, d_2) values on the training data in Figure 2.

In order to understand the mechanism of the nearest neighbor predictor, let us look at an example first. Figures 2 (a) and (b) show the FATAL and NONFATAL nearest neighbor maps when we have $\Delta = 12$ hours, $T = 36$ hours, and no sampling. In both figures, we sort all the distances and plot them in the increasing order. From Figure 2, our first observation is that the distance distributions from both classes look rather alike. For example, in both classes, most of the distances are between 89.5 and 94, i.e. 16 out of 17 FATAL distances in this range, and 23 out of 27 NONFATAL distances in this range. This similarity makes it impossible to adopt a single threshold to differentiate these two classes. Fortunately, after examining the figures more carefully, we find a sparse region in FATAL distance distribution (Figure 2(a)), i.e. [90, 91], and meanwhile, many NONFATAL distances are within this range. This observation suggests that we can employ two distance thresholds (d_1 and d_2 with $d_1 < d_2$), and adopt the following prediction rule: $r_{nnp} : ((d_1 > distance) \vee (distance > d_2)) \longrightarrow FATAL$.

We also observed the above trend in other parameter settings, and therefore, our nearest neighbor predictor adopts this simple prediction rule across all the settings. Since this nearest neighbor predictor utilizes two distance thresholds, we refer to it as *bi-modal nearest neighbor* predictor (BMNN, in short). Traditional nearest neighbor prediction, on the other hand, only uses the lower bound threshold, and it classifies an event as FATAL as long as the nearest distance is smaller than the threshold.

The performance of BMNN is largely dependent on the values of d_1 and d_2. Here, we use the training data to learn the appropriate values for these two thresholds; for a given setting, we vary their values, and choose the pair which leads to the best prediction results for the training data. Later on, we apply the same threshold values learnt from the training data to the test data. For the example shown in Figure 2, Table 1 summarizes the prediction results with a range of (d_1, d_2) values. Based on the results shown in Table 1, we choose $d_1 = 90.0$ and $d_2 = 91.4$, as they lead to the highest F_measure (60%), and more importantly, the highest recall value (80%) (refer to 5.1 for the definitions of F_measure and recall).

5 Experimental Results

In this study, we choose to horizontally (along the time axis) partition the event logs into large consecutive blocks. For RIPPER and SVMs, we partition the event logs into two consecutive parts which are training and test data respectively, while for BMNN, we partition the event logs into three parts.

In order to conduct a comprehensive evaluation of the predictors, we need to run them against multiple test cases. In this study, we constructed 15 test cases, $T(1), \ldots, T(15)$. Our event log consists of failure events collected over a 20-week period (In fact, the log duration is 20 weeks and 2 days, and we merged the last two days into the last week), and our test data always contain events from a three-week period. In the 15 test cases, the test data start from week 18, 17, …, and week 4 respectively. As far as RIPPER and SVMs are concerned, the data from the remaining 17 weeks are regarded as training data. In the case of BMNN, we need to further split these 17-week data into two parts, events from the immediately preceding 3 weeks are the training data, and the remaining 14-week data are the anchor data. Please note that test data are the same for all the classifiers across the 15 test cases.

5.1 Evaluation Methodology

Since detecting FATAL intervals is more important than detecting NONFATAL intervals, we use *recall* and *precision* to measure the effectiveness of our predictors. Using the confusion matrix as shown in Table 2, we define these two metrics as below.

$$Precision, p = \frac{f_{FF}}{f_{FF} + f_{NF}} \tag{3}$$

$$Recall, r = \frac{f_{FF}}{f_{FF} + f_{FN}} \tag{4}$$

A predictor with a high precision commits fewer false positive errors, while a predictor with a high recall commits fewer false negative errors. Finally, we also use a combined metric, $F_$measure.

$$F\_measure = \frac{2rp}{r+p} = \frac{2f_{FF}}{2f_{FF} + f_{NF} + f_{FN}} \tag{5}$$

A high $F_$measure ensures that both precision and recall are reasonably high.

5.2 Comparison of the Predictors

In the first set of experiments, we compare the performance of the following predictors: RIPPER, SVM, BMNN, and the traditional nearest neighbor predictor. We have run the four predictors over all 15 test cases and we report the average $F_$measure (across all 15 test cases), the average precision, and the average recall in Figure 3.

Regardless of which predictor we use, they all take feature data derived from raw logs as input. Nonetheless, we can render different versions of feature data by tuning several parameters, i.e. the significance major threshold T_{sim}, the sampling period δ, and the observation period T. We found that both RIPPER and SVMs prefer raw feature data without applying any significance major filtering, i.e. $T_{sim} = 0$. On the other hand, we had $T_{sim} = 2.5$ for nearest neighbor predictors. Incidentally, the same value was also recommended in [7]. In the experiments, we tuned the parameters for each predictor to report the best results.

From the results in Figure 3, our observation is that the prediction window size Δ has a substantial impact on the prediction accuracy. With $\Delta = 12$ hours, RIPPER, SVM, and BMNN can perform reasonably well: $F_$measure higher than 60%, precision higher than 50%, and recall higher than 70%. As prediction window becomes smaller, the prediction difficulty rapidly increases, leading to much degraded performance, especially for RIPPER and SVMs.

		Predicted Class	
		FATAL	NONFATAL
Actual Class	FATAL	f_{FF} (TP)	f_{FN} (FN)
	NONFATAL	f_{NF} (FP)	f_{NN} (TN)

Table 2. The confusion matrix

The $F_$measure for these two predictors is less than 5% when Δ is 4 hours or shorter (0 in some cases), which is hardly of any practical use for our application. The traditional nearest neighbor also fares poorly. Fortunately, our nearest neighbor predictor, BMNN, managed to sustain a much slower degradation. Even with $\Delta = 1$ hour, its $F_$measure is above 20%, and its recall is as high as 30%. Figure 3 shows that our nearest neighbor classifier greatly outperforms the other classifiers, especially for smaller prediction window sizes. The main lesson learnt from this study is that a prediction window from 12 to 6 hours balances the prediction accuracy (above 50%) and prediction utility the best.

6 Related Work

In this section, we broadly classify the related work in two categories: (1) failure prediction in computer systems, and (2) rare class classification.

Predicting failures in computer systems has received some attention in the past. However, most of the prediction methods focused only on prediction accuracy, but not the usability of the prediction. For example, Vilalta et al. [11] presented both long-term failure prediction based on seasonal trends and short-term failure prediction based on events observed in the system. While these two methods can provide insights, they cannot help runtime fault tolerance because the long term prediction does not tell when the failure will occur while the short term prediction only predicts failures a few minutes before their occurrence. Similarly, prediction studies in [8] and [9] suffer from the same insufficiency.

At the same time, the data mining community has witnessed a number of studies on rare class analysis. For instance, Joshi et al. [3] discussed the limitations of boosting algorithms for rare class modeling and proposed PNrule, a two-phase rule induction algorithm, to carefully handle the rare class cases [2]. Other algorithms developed for mining rare classes include SMOTE, RIPPER, etc. However, we note that, these research efforts merely focused on the algorithm-level improvement of the existing classifiers for rare class analysis on generic data, but not on event log data in a specific application domain. Indeed, as previously described, IBM BlueGene/L event log data have uniquely challenging characteristics which thus requires domain expert knowledge to transform data into a form that is appropriate for running off-the-shelf classifiers. More importantly, we would like to point out that due to these characteristics, off-the-shelf tools fail to classify BlueGene/L event data.

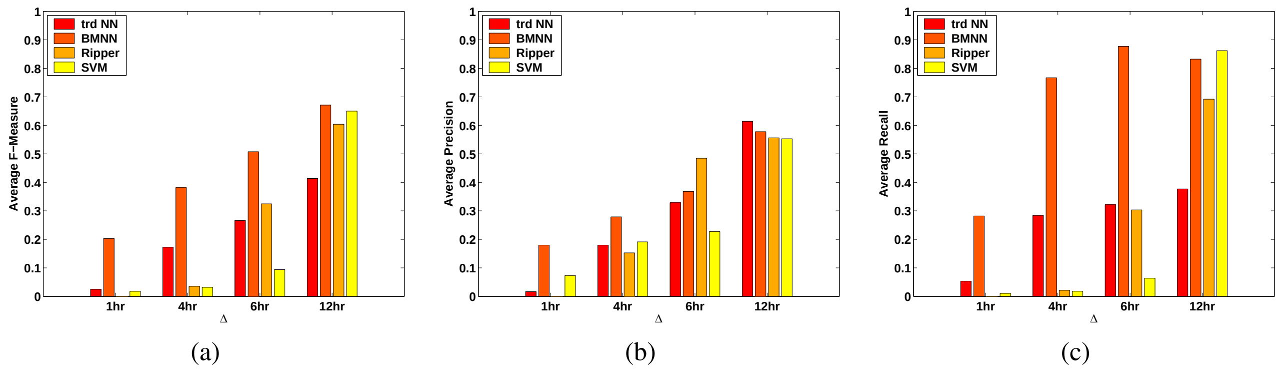


Figure 3. (a) The average $F_$measure, (b) the average precision, and (c) the average recall with varying prediction window size.

7 Concluding Remarks

As failures become more prevalent in large-scale high-end computing systems, the ability to predict failures is becoming critical to ensure graceful operation in the presence of failures. A good failure prediction model should not only focus on its accuracy, but also focus on how easily the predicted results can be translated to better fault tolerance. To address this need, we collected event logs over an extensive period from IBM BlueGene/L, and developed a prediction model based on the real failure data. Our prediction methodology involves first partitioning the time into fixed intervals, and then trying to forecast whether there will be failure events in each interval based on the event characteristics of the preceding intervals.

Our prediction effort addressed two main challenges: feature selection and classification. We carefully derived a set of features from the event logs. We then designed a customized nearest neighbor classifier, and compared its performance with standard classification tools such as RIPPER and SVMs, as well as with the traditional nearest neighbor based approach. Our comprehensive evaluation demonstrated that our nearest neighbor predictor greatly outperforms the other two, leading to an $F_$measure of 70% and 50% for a 12-hour and 6-hour prediction window size. These results indicate that it is promising to use the nearest neighbor predictor to improve system fault-tolerance.

References

[1] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at *http://www.csie.ntu.edu.tw/ cjlin/libsvm*.

[2] M. Joshi, R. Agarwal, and V. Kumar. Mining needle in a haystack: Classifying rare classes via two-phase rule induction. In *SIGMOD*, pages 91–102, 2001.

[3] M. Joshi, R. Agarwal, and V. Kumar. Predicting rare classes: Can boosting make any weak learner strong? In *KDD*, 2002.

[4] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2006.

[5] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. An Adaptive Semantic Filter for Blue Gene/L Failure Log Analysis. In *Proceedings of the Third International Workshop on System Management Techniques, Processes, and Services (SMTPS)*, 2007.

[6] A. J. Oliner, R. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware Job Scheduling for BlueGene/L Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[7] R. J. Roiger and M. W. Geatz. *Data Mining: A Tutorial Based Primer*. Addison-Wesley, 2003.

[8] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2003.

[9] F. Salfner, M. Schieschke, and M. Malek. Predicting Failures of Computer Systems: A Case Study for a Telecommunication System. http://samy.informatik.hu-berlin.de/ schiesch/papers/salfner06predicting.pdf.

[10] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance-computing systems. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks*, June 2006.

[11] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3):461–474, 2002.

[12] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo. Performance Implications of Failures in Large-Scale Cluster scheduling. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2004.