A Practical Failure Prediction with Location and Lead Time for Blue Gene/P

Ziming Zheng [*], Zhiling Lan [*], Rinku Gupta [‡], Susan Coghlan [†], Peter Beckman [‡]

[*]*Department of Computer Science, Illinois Institute of Technology*
[‡]*Mathematics and Computer Science Division, Argonne National Laboratory*
[†]*Argonne Leadership Computing Facility, Argonne National Laboratory*
[*]{zzheng11,lan}@iit.edu, [‡]{rgupta,beckman}@mcs.anl.gov, [†]smc@alcf.anl.gov

Abstract

Analyzing, understanding and predicting failure is of paramount importance to achieve effective fault management. While various fault prediction methods have been studied in the past, many of them are not practical for use in real systems. In particular, they fail to address two crucial issues: one is to provide location information (i.e., the components where the failure is expected to occur on) and the other is to provide sufficient lead time (i.e., the time interval preceding the time of failure occurrence). In this paper, we first refine the widely-used metrics for evaluating prediction accuracy by including location as well as lead time. We, then, present a practical failure prediction mechanism for IBM Blue Gene systems. A Genetic Algorithm based method is exploited, which takes into consideration the location and the lead time for failure prediction. We demonstrate the effectiveness of this mechanism by means of real failure logs and job logs collected from the IBM Blue Gene/P system at Argonne National Laboratory. Our experiments show that the presented method can significantly improve fault management (e.g., to reduce service unit loss by up to 52.4%*) by incorporating location and lead time information in the prediction.*

1 Introduction

Over the past decades, various prediction methods have been studied. Representative research work includes association rule based methods [3], decision trees and Bayesian networks[18]. All these methods examine correlations between past events and fatal events to learn fault patterns for predicting potential failures in the future. As shown in Figure 1a, these methods analyze correlations among historical events to learn fault patterns and then examine events occurring in a fixed time window (denoted as *observation window*) to predict whether a fatal event will occur in a near future (denoted as *prediction period*). While these methods provide reasonable prediction accuracy, they ignore two critical issues in their design, thereby being impractical to use in real systems.

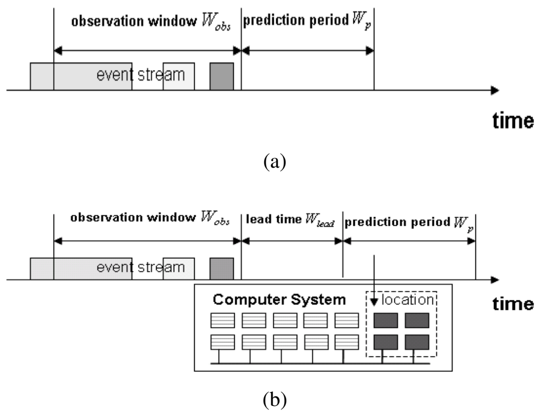


Figure 1: Failure prediction (a)without location and lead time information, (b) with location and lead time information

First, these methods do not provide *location information*, i.e., the components where the failure will occur. For a system composed of thousands or more components, location information is critical since it can narrow down the potential problematic parts and help system administrators take appropriate actions on these components. For example, without location information, it is difficult to determine which application processes should be migrated [22][15][21]). Even for reactive methods like checkpointing, predictions with location information will enable one to checkpoint data on those failure-prone components, thereby avoiding system-wide checkpointing which is significantly time consuming [12].

Secondly, these methods do not guarantee sufficient *lead time*, i.e., the time interval preceding the time of failure occurrence [19]. The time taken to perform fault tolerance (such as taking a checkpoint or performing a process migration) could be very high in large-scale systems consisting of hundred of thousands of nodes [12][13]. Hence, it is important from practical usability perspective, that the lead time be long enough to perform the desired action to achieve fault tolerance. On one hand, predictions with high accuracy but short lead time may be useless in practice because they cannot be used for effective failure management. On the other hand, a very long lead time is not desirable either since it tends to reduce prediction accuracy. How to choose an appropriate lead time thus becomes a question of crucial importance.

The importance of fault location information and lead time for fault management is further demonstrated via our analysis of the RAS (Reliability, Availability, and Serviceability) log data collected from the IBM Blue Gene/P (named 'Intrepid') system at ANL over a period of few months (Refer to Section 2.2 for additional details). Out of a total of 520 reported fatal events, about 71% of them were reported at a single midplane or rack of the Blue Gene/P system. In other words, most of fatal events were found to occur within a single midplane or rack, rather than being dispersed throughout the entire system. Such information, indicating location of the event, can be very useful for fault management. For instance, instead of invoking a system-wide checkpoint, one may need to checkpoint only on a midplane or rack. Studies show that a system-wide checkpointing on such a system may take up to 1500 seconds [6][13], whereas a midplane- or rack-level checkpointing may only take about 120 seconds, thereby saving valuable processing cycles and reducing I/O traffic. Further analysis of the RAS logs revealed that typically a long stream of warnings are often present before the occurrence of fatal event [25]. This indicates that for many fatal events, the system does provide some warnings ahead of time and the time interval between the first warnings and the fatal event could be the lead time for failure prediction.

In this study, we first refine the traditional metrics (i.e., *precision* and *recall*) for evaluating prediction accuracy by including location as well as lead time information in the definitions. A prediction is considered correct (or true positive) only if it correctly pinpoints the location with a sufficient lead time. Next, we present a Genetic Algorithms (GA) based prediction method for practical use in Blue Gene systems. The refined metrics are used during both the training phase and the testing phase. As shown in Figure 1b, our method analyzes the events occurring during the *observation window*, and aims to predict whether a failure will occur after the *lead time* at a specific *location*. Our method differs significantly from other existing methods (illustrated in Figure 1a) because we incorporate the location and lead time information for failure prediction. The granularity of the location may be fine-grained (e.g., node-level) or coarse-grained (e.g., rack-level). For Blue Gene systems, typically the midplane is the minimum unit for job scheduling [21], so in this study we focus on prediction at the midplane and rack-level.

Compared to other predictive techniques like association rule based methods [3][18], GA has two main advantages. First, while it may be difficult for some predictive methods to consider location or lead time during their training phase, GA contains various interacting parts which can be carefully designed to directly address prediction accuracy, location and lead time together [11]. Second, without generating a large number of candidate rules, GA can converge rapidly, with a high probability, to the rules with optimal or suboptimal accuracy [3][11]. Note that the main objective of this study is to show the importance of incorporating location and lead time information for practical failure prediction. We believe other predictive methods may be augmented similarly by incorporating location and lead time for failure forecasting.

Further, to demonstrate the effectiveness and practical use of our method, we evaluate it with the actual RAS logs and job logs collected from the 'Intrepid' system at ANL. Our experiments show that our GA-based method can significantly improve fault tolerance (e.g., to reduce service unit loss by up to 52.4%), as compared to the GA-based method that does not consider the location and lead time information. Moreover, in terms of the extended prediction metrics, our GA-based method can substantially boost prediction accuracy, especially with regards to *precision*.

2 Background

2.1 Blue Gene/P Architecture

IBM Blue Gene/P is a scalable, low-power high performance computing system. It is scalable to the upwards of 80 racks with a peak performance above 1 petaflop. In each rack, there are 2 midplanes, which consists of 512 quad core PowerPC450 compute nodes, 1 service card and 4 link cards. Compute nodes are connected into a 3-D torus for collective communication. In Blue Gene/P, 64 compute nodes are served by an I/O node. A tree network is used to connect compute nodes to dedicated I/O nodes. The I/O nodes are connected through a 10-Gigabit Ethernet network to 136 file servers, that in turn, connect to back-end InfiniBand-based storage. More details of the system architecture is available in [1].

'Intrepid' is a 40-rack Blue Gene/P system at ANL. It consists 40,960 compute nodes with a total of 163,840 cores, which offers a peak performance of 556 TFlops. It ranks the #8 on the latest TOP500 supercomputer list (as of Nov. 2009)[2].

Table 1: Summary of the RAS log and job log from the Blue Gene/P system.

Log Name	Days	Start Date	End Date	Log Size	No. of Records
RAS	81	2008-03-11	2008-05-31	3.5 GB	2715668
Job	31	2008-05-01	2008-05-31	4.5 MB	14108

Table 2: An example of RAS events from the Blue Gene/P

RECID	13718190
MSG_ID	CARD_0411
COMPONENT	CARD
SUBCOMPONENT	PALOMINO_S
ERRCODE	DetectedClockCardErrors
SEVERITY	FATAL
EVENT_TIME	2008-04-14-15.08.12.285324
LOCATION	R-04-M0-S
MESSAGE	An error(s) was detected ...

Table 3: An example of job records from the Blue Gene/P

Submission Time	05/01/2008 00:00:43
Job ID	8935
Job Name	XXX
Queuing Time	1209614949.07
Starting Time	1209618043.1
End Time	1209621636.96
Location	R10-R11

2.2 Understanding RAS Logs

The IBM Blue Gene/P Core Monitoring and Control System (CMCS) is responsible for monitoring the hardware components, including the compute nodes, I/O nodes and various networks. Monitored information is reported by CMCS as RAS (Reliability, Availability and Serviceability) event messages. RAS messages are stored in back-end DB2 databases and the event stream from the back-end databases is utilized to provide efficient failure prediction facility.

All analysis and studies in this paper have been conducted using the RAS logs and job submission logs collected from the 'Intrepid' system. Evaluation of the proposed GA-based method has also been performed using these logs. Table 1 summarizes these logs.

An example of event record from 'Intrepid' is shown in Table 2. The raw RAS log was found to contain a large amount of redundant records. In order to analyze the RAS event information effectively, we first applied pre-processing techniques [25] on the logs to remove the redundant records. The pre-processing stage was used to combine the sequence of redundant or correlated records into one event, and to keep track of various attributes such as the start time, end time, count, and location list for each event. All this information is essential for incorporating the location and lead time information for failure prediction.

The job log on 'Intrepid' was collected by the COBALT [21] job scheduler. An example of a job record from *Intrepid* is shown in Table 3. By examining the location and time information in both the RAS log and the job log, one can determine which jobs were interrupted by fatal events.

3 Refining Prediction Metrics

Precision and *recall* are two widely-used metrics to measure prediction accuracy. *Precision* is defined as the proportion of correct predictions to all the predictions made, and *recall* is the proportion of correct predictions to the number of failures.

Depending on whether we consider the location and lead time information, the meaning of correct prediction may be different. Without considering location and lead time, the calculation of *precision* and *recall* is simple. However, when location and lead time are involved, more complicated cases should be considered to define these metrics. For instance, if the rule correctly predicts the occurrence of a failure but at a wrong location, it can lead to both a false positive at the wrong location and a false negative at the actual location. If the rule states that a failure will occur throughout the entire system, but actually only at several midplanes, then other midplanes will get false alarms. Furthermore, the lead time of a rule may be shorter than the time required for a fault management action. In this case, the rule is useless and the prediction should be considered as a false negative.

To address the above problems, we refine the term of true positive (T_P), false negative (F_N) and false positive (F_P) as follows:

- T_P: denotes prediction result that accurately gives a positive reading of a fatal event that will occur during the W_{fatal} prediction period with the correct *location*. Furthermore, the prediction provides a lead time W_{lead} which is larger than a predefined threshold.

- F_N: denotes an incorrect prediction which erroneously fails to detect something when, in fact, it is present. It may occur at a failure-prone location when (1) the prediction fails to give a warning, or (2) the prediction gives a warning, but does not provide a sufficient lead time, or (3) the prediction gives a warning, but lists a wrong location.

- F_P: denotes an incorrect prediction which erroneously detects something when, in fact, it is not present. It may occur at a failure-free location when (1) the prediction gives a warning, or (2) the prediction gives a warning at the location, whereas the failure actually occurs at a different location.

Note that T_P, F_N and F_P are measured and counted per location. For example, if the fault predictor generates a warning about a failure across the entire system, then all the midplanes get a F_P except for the failed midplane. Based on these terms, *precision* and *recall* can be defined by considering location and lead time, i.e. $precision = \frac{T_P}{T_P+F_P}$ and $recall = \frac{T_P}{T_P+F_N}$.

4 GA-based Prediction Method

4.1 Prediction Rule

The objective is to use non-fatal events preceding a fatal event to predict whether the fatal event will occur after a specific *lead time* at the specific *location*. Our methodology involves training phase followed by a testing phase (to verify the rules learnt during the training phase).

During the training phase, for each fatal event, we identify the set of non-fatal events preceding it within the observation window T_{obs}. Our method intends to generate a set of prediction rules. Prediction rules are in the format of $< e_i, e_j, ..., e_k > \rightarrow f$ where e_i is a non-fatal event and f is a fatal event. Suppose during the training phase, there are M instances satisfying the rule of $< e_1, e_2, ..., e_k > \rightarrow f$. We calculate the lead time for each instance as the time interval between the fatal event and the latest non-fatal event. That is, for each instance m, its lead time W^m_{lead} is estimated as $min(T^f - T^{e_i})$, where T^f is the start time of the fatal event f and T^{e_i} is the start time of the non-fatal event e_i. Consequently, we estimate the lead time for $< e_i, e_j, ..., e_k > \rightarrow f$ as the average value of the lead times of M instances, i.e., $W_{lead} = \frac{\sum_{m=1}^{M} W^m_{lead}}{M}$.

For each predication rule, we also need to give a possible failure location. For the Blue Gene series of machines, the midplane is the minimum unit for job scheduling [21]. Hence, a fault action can only be conducted at the level of midplane, rack or across the entire system. As a result, our prediction is also provided at these levels. Again, suppose during our training phase, there are M instances satisfying the rule of $< e_1, e_2, ..., e_k > \rightarrow f$. For each non-fatal event e_i, we count the number of instances where e_i has the same location as f. The location of the non-fatal event $\hat{e}$ which has the maximum number will be selected as the location for the predicted failure. In case of a tie, we will randomly choose one.

In short, a prediction rule with a lead time and a location (i.e., $< e_1, e_2, ..., e_k > \rightarrow f, location, W_{lead}$) means that during the testing phase, when we observe the occurrence of $e_1, e_2, ..., e_k$ within the observation window T_{obs}, we may anticipate that the fatal event f will possibly occur at the specified location after the lead time of W_{lead}.

4.2 Rule Generation

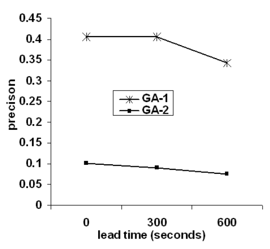
Genetic algorithm (GA) is a widely used technique for optimization problems [17]. It starts from an initial population of randomly generated individuals. During each successive generation, multiple individuals are stochastically selected from the current population based on a fitness function, and then are modified via genetic operations like crossover and mutation to construct a new population. The new population is then used in the next generation. The algorithm terminates when a satisfactory fitness level has been reached.

In our GA-based method, an initial population containing some candidate rules is selected, and then the Michigan encoding method is adopted to transform these rules into genetic individuals, where each rule is represented as an array of bits [17]. We chose the Michigan encoding due to its low computation cost of fitness function [23]. Initialization generates the first population of individuals for evolution. The population can be randomly initialized. However, it is possible that many non-fatal events are uncorrelated with fatal events, thereby resulting in a poorly fitted population. A poor population may cause a slow convergence or a trapping at a local optima.

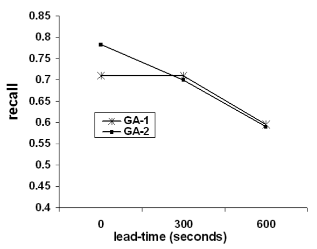
To overcome this problem, event correlation is adopted to select some elite individuals, mixed with randomly generated individuals, as the initial population. In particular, we calculate Pearson's correlation between a non-fatal event and a fatal event [8], and then select the pair with a high Pearson's correlation value as our elite individual. These elite individuals can help fast convergence and avoid local optima.

Fitness function is the most important component of GA. It is used to evaluate each individual and determine the search direction. In this study, we choose the following fitness function: $fitness = (w_1 \cdot recall + w_2 \cdot precision) \cdot W_{lead}$ Here, w_1 and w_2 are user defined weights (between 0 and 1). If $w_1 > w_2$, then the method tends to a searching direction which maximizes recall, and vice versa. Note that the refined metrics *recall* and *precision* presented in Section 3 are used here. It is clear that our method aims to select the individuals with high accuracy. In addition, a large lead time is preferred here, which will provide sufficient time for possible fault management.

Three genetic operations, namely selection, crossover, and mutation, are used to evolve the population. Selection involves choosing some individuals from the population based on their fitness values. Typically individuals with higher values are more likely to be selected. The one-point



(a)



(b)

Figure 2: (a) Precision (b) Recall

crossover operation breeds new individuals from two selected parents by copying some bits from each parent. The mutation operation makes small random changes to a single bit in a genetic sequence from its original state. In general, crossover produces elite children, whereas mutation maintains genetic diversity [11].

5 Experiments

In our experiments, a real RAS log and a job log collected from the 'Intrepid' system are studied (see Table 1). We first pre-process the RAS log using the method presented in our previous work [25]. For the purpose of training and testing, we split the RAS log into two parts: the log events in first 50 days with 430 fatal events are used for training, and log events in the last 31 days with the rest 190 fatal events is used for testing. The job log is collected in the same period of 31 days as the testing part of the RAS log, which is used to examine the impact of failure prediction results on fault management.

As mentioned earlier, the main objective of this study is to show the importance of incorporating location and lead time information for practical failure prediction. Toward this end, we compare our GA-based method (denoted as *GA-1*) as against a standard GA method which uses a fitness function of $(w_1 \cdot recall + w_2 \cdot precision)$ (denoted as *GA-2*). Note that the standard GA method uses the traditional prediction metrics without considering location and lead time.

Our evaluation is divided into two parts: (1) Examining prediction accuracy between GA-1 and GA-2; and (2) Examining their impact on service unit loss (SUL).

5.1 Prediction Accuracy

In our experiments, we set the lower bound of lead time at 120 seconds to train our GA-based method (GA-1). This generates 37 rules - 10 rules providing midplane-level locations, 7 rules giving rack-level locations, and 20 rules without any specific location information. GA-2 generates 41 rules without location and lead time information.

To study the impact of lead time on prediction accuracy, we vary the lead time from 0 to 600 seconds for testing. Figure 2 presents prediction accuracy by using GA-1 and GA-2. Here, we use the refined metrics defined in Section 3 for evaluation. As we can see, overall trend shows that both $precision$ and $recall$ decrease with a growing lead time. This is due to the fact that an increasing lead time means that more precursor events cannot be used for prediction. For both GA-1 and GA-2, this scenario will introduce more false negatives.

When lead time is set as 0 for testing, GA-2 provides better $recall$ over GA-1. This is because GA-2 only provides prediction on system-level, and some results of GA-1 on midplane- or rack-level will inevitably introduce more false negatives. As lead time increases, GA-1 outperforms GA-2 in $recall$. The main reason is that GA-2 is prone to rely on non-fatal events immediately preceding fatal events for prediction. When lead time increases, many of these precursor non-fatal events may be discarded because these non-fatal events are too close to the fatal events. Thereby we will see more false negatives. On the other hand, GA-1 explicitly incorporates lead time in its fitness function during the training phase, hence it is less sensitive to the growth of lead time.

GA-1 significantly outperforms GA-2 with respect to *precision*. The figure shows that GA-2 can only achieve about 0.1 on $precision$, while GA-1 can provide up to four times improvement on $precision$. The major reason is that GA-2 leads to 12 false alarms at the system level. Based on our refined metrics, GA-2 totally generates 960 false positives on the 80 midplanes. On the other hand, GA-1 only generates 5 false alarms at the system level, 7 at the midplane-level, and 3 at the rack-level. As a result, number of false positives introduced by GA-1 is much less than GA-2. However, precision is not high even with GA-1. The reason is that not every rule generated by GA-1 includes specific location information. As a result, there are non-trivial amount of false positives.

5.2 Impact on Fault Management

Our next goal is to examine the impact on fault management by using different predictive methods. *Service unit loss (SUL)* is a widely used metric to measure the amount of system resource loss caused by failure [21]. It is defined as product of wasted wall clock hours and number of CPUs. This metric directly indicates the computing cycles lost due to failure and checkpointing overhead. Checkpoint and restart is commonly used in the field of high performance computing for fault management. On 'Intrepid', IBM Checkpoint library is available for user-level checkpointing. Hence, we chose it as our fault management strategy.

By cross examining the RAS log and the job log, we identified the jobs interrupted by fatal events, i.e., the jobs with the end time close to a fatal event occurring in the same location. Without prediction, these jobs will have to restart from scratch and their execution hours would be wasted. By using GA-1 or GA-2, a job will perform a checkpoint when a warning is reported from the predictor. In this study, SUL consists of three parts: (1) false negative which leads to a job termination (F_N), (2) false positive which stops the job to issue a useless checkpointing (F_P), (3) the overhead of successful checkpointing (CKP). We examine these SUL parts in our experiments (see Figure 3).

In our experiments, we estimate checkpoint overhead O_{ckp} as $O_{ckp} = \frac{n \times size}{B(n)}$ [12][13]. Here, n is the number of nodes used by the application, $size$ is checkpoint image size per node, and $B(n)$ is the available bandwidth for checkpoint on n nodes. In terms of $size$, since the job log does not provide information regarding checkpoint size for each job, we estimate $size$ based on memory utilization. According to [13], typically HPC applications use 10-50% memory per compute node. In 'Intrepid', each node is equipped with 2 GB memory. Thus, we estimate that their checkpoint images range from 200MB to 1 GB per node for the jobs in the job log. We consider three possible cases for the checkpoint images size per node (for all the nodes): (1) Images are 200-400MB per node for all the jobs (Case 1); (2) Images are about 400-800MB per node for all the jobs (Case 2); (3) Images are 0.8-1G per node for all the jobs (Case 3).

To calculate the $B(n)$, we first analyze the job log and get the number of nodes in each job. If a midplane- or rack-level prediction is provided, then checkpoint is issued per application and we calculate B(n) according to application size n. If $n < 16,384$ (i.e., 16 racks), then O_{ckp} varies from 120 to 600 seconds as $size$ increases from 200 MB to 1 GB; if $n > 16,384$, O_{ckp} is ranging between 240 to 1200 seconds. The reason is that $B(n)$ scales roughly linear with n at first but becomes to level off at about 25 GB/sec when $n = 16,384$ [6]. If the prediction does not provide any location information, a system-wide checkpoint is invoked. In this case, O_{ckp} per application varies from 300 to 1500 seconds with the growth of $size$. They are higher than the numbers listed in the previous case due to the bandwidth contention between different applications.

Figure 3 compares the amount of SUL brought by using GA-1, GA-2, and without failure prediction. As compared to the situation without any failure prediction, GA-1 reduced SUL by 52.4% for Case 1, whereas GA-2 only reduced it by 25.1%. The major difference lies on the amount of SUL caused by F_P. On analyzing the logs, we found that only 21.6% of the fatal events will actually interrupt the jobs since other fatal events occur in the components which are not running any productive jobs. As a result, with the location information provided by GA-1, we can significantly

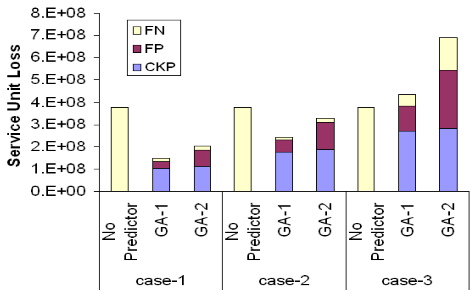


Figure 3: The service unit loss (CPU*Seconds).

avoid meaningless checkpointing. Meanwhile, GA-1 generates 7 less false alarms on the whole system than the GA-2 predictor, which significantly reduce overall checkpointing overhead.

In Case 2, again as compared to the situation without failure prediction, GA-1 reduces SUL by 26.6% but GA-2 increases it by 18.6%. Obviously the benefit brought by failure prediction decreases. With GA-1, the amount of SUL caused by CKP and F_P increases as the growth of checkpoint image. For GA-2, since it does not provide location information, any possible false alarm will invoke a system-wide checkpoint, thereby causing a significant overhead. The overhead eventually eliminates the benefit brought by failure prediction. Meanwhile, 12% of predictions have a lead time less than 300 seconds (i.e., the minimum overhead for conducting an application-level checkpoint). As a result, these predictions cannot be effectively used by checkpointing.

Both GA-1 and GA-2 cannot help much in Case 3. The main reason is that not all the rules provide the location information, which will lead to extreme high checkpointing overhead. With GA-2, the amount of SUL caused by F_N also significantly increases as the growth of checkpoint image. This essentially means that when system-wide checkpointing has a high overhead, failure prediction is not a good idea unless it is accompanied by location information.

6 Related Work

Considerable research has been performed on failure prediction methods based on system logs. Take several examples, Sahoo et al. evaluate the time-series, the rule-based classification and Bayesian network methods to predict failure events in a 350-node IBM cluster [18]. In [7], several statistical based techniques are studied for failure forecasting in a Blue Gene/L system. In our own previous work [3], we present a dynamic meta-learning prediction engine in large-scale systems. In [9], Liang et al. mention the usability of prediction methods for Blue Gene/L systems. In [19], Salfner et al. consider lead time in their Hidden

Semi-Markov Models for failure prediction in a commercial telecommunication system, without addressing the location issue. Research presented in this paper distinguishes itself from the above research by emphasizing on generating usable rules for failure prediction by addressing prediction accuracy, location, and lead time problems together. A widely used approach to address fault location is to develop fault propagation models (FPMs), such as causality graphs or dependency graphs [20]. However, this requires a priori knowledge about the system structure and dependencies among different components. This information is hard to obtain and maintain in large-scale systems.

Research in this paper is inspired by the work of Weiss [23], which explores a genetic-based method for failure prediction in a telecommunication equipments. Our work fundamentally distinguishes from [23] in two key aspects. First, the above method is applied to predict failure in telecommunication equipments, in which the location information is not a big concern. In contrast, our research focuses on failure prediction in large-scale systems, where the location information is critical for effective fault management. Second, the lead time is explicitly considered in our fitness function to guarantee enough time for fault tolerance actions like checkpointing.

7 Conclusions

In this paper, we have presented a practical failure prediction method for high performance computing systems. It provides a holistic approach to address prediction accuracy, location, and lead time together. We have evaluated it with the logs collected from the IBM Blue Gene/P system at ANL. Experimental results have shown that our method can substantially boost prediction accuracy, especially with regards to refined *precision* as against the method without considering location and lead time. Furthermore it can reduce service unit loss caused by failure by up to 52.4%.

Acknowledgment

Zhiling Lan is supported in part by US National Science Foundation grants CNS-0834514, CNS-0720549, and CCF-0702737. The CIFTS project is supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] Blue Gene Team, “Overview of the IBM Blue Gene/P project,” *IBM Journal of Research and Development*, 2008.

[2] Top500 supercomputing sites http://top500.org/.

[3] J. Gu, Z. Zheng, Z. Lan, J. White, and B. Park. Dynamic meta-learning for failure prediction in large-scale systems: A case study. *Proc. of ICPP*, 2008.

[4] R. Gupta, P. Beckman, B.-H. Park, E. Lusk, and P. Hargrove. CiFTS: A coordinated infrastructure for fault-tolerant systems. *Proc. of ICPP*, 2009.

[5] Z. Lan and Y. Li. Adaptive fault management of parallel applications for high performance computing. *IEEE Trans. on Computers*, 57(12):1647–1660, 2008.

[6] S. Lang, P. Carns, K. Harms, and W. Allcock. I/O performance challenges at leadership scale. *Proc. of Supercomputing*, 2009.

[7] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramanium, and R. Sahoo. Blue Gene/L failure analysis and prediction models. *Proc. of DSN*, 2006.

[8] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. An adaptive semantic filter for Blue Gene/L failure log analysis systems. *Workshop on SMTPS*, 2007.

[9] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure prediction in IBM Blue Gene/L event logs. *Proc. of ICDM*, 2007.

[10] C. Lim, N. Singh, and S. Yajnik. A log mining approach to failure analysis of enterprise telephony systems. *Proc. of DSN*, 2008.

[11] T. Mitchell. Machine learning. *McGraw-Hill Companies, Inc*, 1997.

[12] H. Naik, R. Gupta, and P. Beckman. Analyzing checkpointing trends for applications on the IBM Blue Gene/P system. *Workshop on P2S2 in conjunction with ICPP*, 2009.

[13] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, and M. Varela. Modeling the impact of checkpoints on next-generation systems. *Proc. of MSST*, 2007.

[14] A. Oliner, L. Rudolph, and R. Sahoo. Cooperative checkpointing: A robust approach to large-scale systems reliability. *Proc. of ICS*, 2006.

[15] A. Oliner, R. Sahoo, J. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for Blue Gene/L systems. *Proc. of IPDPS*, 2004.

[16] A. Oliner and J. Stearly. What supercomputers say: A study of five system logs. *Proc. of DSN*, 2007.

[17] M. Sagger, A. Agrawal, and A. Lad. Optimization of association rule mining using improved genetic algorithms. *Proc. of SMC*, 2004.

[18] R. Sahoo and A. Oliner et al. Critical event prediction for proactive management in large-scale computer clusters. *Proc. of SIGKDD*, 2003.

[19] F. Salfner and M. Malek. Using hidden semi-markov models for effective online failure prediction. *Proc. of SRDS*, 2007.

[20] M. Steinder and A. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2), 2004.

[21] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware utility-based job scheduling on Blue Gene/P systems. *Proc. of Cluster*, 2009.

[22] C. Wang, F. Mueller, C. Engelmann, and S. Scott. Proactive process-level live migration in HPC environments. *Proc. of Supercomputing*, 2008.

[23] G. Weiss. Timeweaver: A genetic algorithm for identifying predictive patterns in sequences of events. *Genetic and Evolutionary Computation Conference*, 1999.

[24] Z. Zheng, R. Gupta, Z. Lan, and S. Coghlan. FTB-enabled failure prediction for Blue Gene/P systems. *Proc. of Super-Computing (research poster)*, 2009.

[25] Z. Zheng, Z. Lan, B. Park, and A. Geist. System log pre-processing to improve failure prediction. *Proc. of DSN*, 2009.