Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study

Jiexing Gu[1], Ziming Zheng[1], Zhiling Lan[1]
John White[2], Eva Hocks[3], Byung-Hoon Park[4]

Illinois Institute of Technology[1]
Revision3 Company[2]
San Diego Supercomputer Center[3]
Oak Ridge National Laboratory[4]

{jgu5, zzheng11, lan}@iit.edu[1]
jwhite@moltar.org[2],hocks@sdsc.edu[3]
parkbh@ornl.gov[4]

Abstract

Despite great efforts on the design of ultra-reliable components, the increase of system size and complexity has outpaced the improvement of component reliability. As a result, fault management becomes crucial in high performance computing. The advance of fault management relies on effective failure prediction. Despite years of research on failure prediction, it remains an open problem, especially in large-scale systems. In this paper, we address the problem by presenting a dynamic meta-learning prediction engine. It extends our previous work by exploring dynamic training, testing and prediction. Here, the "dynamic" part is from two perspectives: one is to continuously increase the training set during the system operation; and the other is to dynamically modify the rules of failure patterns by tracing prediction accuracy at runtime. Our case study indicates that the proposed predictor is promising by being capable of capturing more than 70% of failures, with the false alarm rate less than 10%.

1. Introduction

In the next few years production systems are expected to contain tens to hundreds of thousands of computing nodes and thousands of I/O nodes [35]. Such a scale, combined with the ever-growing system complexity, is introducing a key challenge on fault management in high performance computing (HPC). Despite great efforts on the design of ultra-reliable components, the increase of system size and complexity has outpaced the improvement of component reliability. Recent studies have pointed out that the mean-time-between-failure (MTBF) of teraflop and soon-to-be-deployed petaflop machines are only on the order of 10 - 100 hours [19,22].

To address the above reliability problem, considerable research has been done on improving fault resilience of systems and their applications through various technologies. Representative works include failure-aware resource management and scheduling [20], checkpointing [2,4,7,24,25], and run-time resilience support [3,16,31]. Nevertheless, the advance of these fault tolerant technologies is hindered by the lack of *fault prediction* support in HPC. For instance, proactive fault tolerant methods require failure forecasting to enable cost-effective failure prevention. For reactive fault tolerant methods such as checkpointing, an efficient failure prediction could substantially reduce their operational cost by telling when and where to perform checkpoints, rather than blindly invoking actions periodically with an unwisely chosen frequency [18].

Previous work on failure prediction can be classified into two categories: *model-based* and *data-driven*. A model-based method derives an analytical or probabilistic model of the system and then triggers a warning when a deviation from the model is detected [9,12,13,14,24,27,28]. Considering the size and complexity of HPC systems, the model-based methods are too complicated to be practical for failure prediction in these systems. *Data-driven methods*, such as those using data mining techniques, attempt to learn and classify occurring failure patterns from historical data without building an a priori model ahead of time [10,17,23].

Existing prediction studies mainly focus on *static analysis* by applying one specific method. Here, "static" means that the method generates rules in a static manner, such as using a fixed training set. Although they are effective in forecasting some failures, they have three inherent drawbacks. First, the sources of failures are numerous and complex in a large-scale system, thus it is improper to expect a single method to detect and capture all of them alone. Second, in order to obtain sufficient failure patterns, most of existing methods require a long training phase (e.g. a year), thereby making failure prediction unavailable for a long period of time. Considering that most HPC systems at supercomputing centers only have a couple of years in production, this requirement must be removed. Lastly, existing studies mainly focus on static analysis in which the training set remains unchanged. Since the upgrade of hardware and software is common in a typical HPC system, the failure patterns obtained from static analysis may become outdated very soon, thereby resulting in low prediction accuracy.

To address the first problem, in our previous work we have proposed the use of meta-learning to boost prediction accuracy [10]. By integrating multiple data mining techniques, meta-learning aims at discovering various failure patterns and thus improving prediction accuracy. While this work is promising, it is also based on static analysis.

In this paper, we address the other two issues by extending our previous work to dynamic meta-learning. In particular, we present *a dynamic meta-learning prediction engine for large-scale systems*. It does not require a long training phase by *dynamically increasing the training set* during system operation. As we will show in Section 3-4, it can start to provide an acceptable failure prediction service after only two weeks of training. As the time goes by, it provides better failure prediction by *dynamically adjusting its rules of failure patterns* according to accuracy tracing and dynamic re-training.

We demonstrate that the proposed failure predictor can effectively forecast failures by evaluating it with a 130-week RAS log (about two and half years) from the Blue Gene/L system at SDSC. Our results show that the proposed predictor is capable of capturing more than 70% of failures, with the false alarm rate less than 10%. Moreover, the proposed prediction engine can adapt to the changing system environment by dynamically adjusting its rules of failure patterns, even after a major system reconfiguration.

The rest of the paper is organized as follows. Section 2 discusses the related work on failure prediction. Section 3 presents our dynamic meta-learning prediction engine. The case study on a RAS log is presented in Section 4. Finally, Section 5 summarizes the paper.

2. Related Work

Recognizing the importance of fault management, the community has paid much attention to failure prediction. Exiting predictive approaches can be broadly classified as *model-based methods or data-driven methods*. Model-based approach derives a probabilistic or analytical model of the system and triggers the warning when a deviation from the model is detected [28]. For example, Gross et al. have presented an adaptive statistical data fitting method called MSET to forecast the system dependability [29]. In [12], a naive Bayesian based algorithm is used to predict disk drive failures. In [26], a specific analytical model is developed for quickly detecting anomalies in I/O systems. While model-based methods are effective for forecasting some failures, it is hard, if not impossible, to construct a precise model for large-scale HPC systems composed of tens of thousands of components.

A data-driven method, such as using data mining techniques, attempts to learn failure patterns from historical data for failure prediction, without constructing an accurate model ahead of time. For example, the group at the RAD laboratory has applied statistical learning techniques for failure diagnosis in Internet services [33]. Sahoo et al. apply association rules to predict failure events in a 350-node IBM cluster [23]. In [17], Liang et al. examine several statistical based prediction techniques for failure forecasting in a Blue Gene/L system. In our own previous works [10, 15], we have investigated a meta-learning based method by adaptively combining the merits of various data mining techniques. The above studies mainly utilize statically learned rules for failure analysis and prediction.

While this paper is built upon many previous studies, it distinguishes from the above studies in that it emphasizes dynamic training, testing and prediction. By dynamically discovering failure patterns and tracing prediction accuracy during the system operation, the proposed predictor aims at finding an optimal set of fault patterns, even when the patterns are changing over time due to hardware or software reconfigurations. Another benefit of using dynamic training and prediction is that it does not require a long training phase to start with. By gradually increasing training data, the proposed predictor is able to build up its knowledge base as time goes by. To the best of our knowledge, we are not aware of any such service for failure prediction in HPC that can dynamically learn failure patterns and further adapt to the changing system state.

3. Dynamic Meta-Learning Prediction

Figure 1 presents a high level diagram of the proposed dynamic meta-learning prediction engine. It consists of two processing phases: one for *data preprocessing* and the other for *failure prediction*. Given that raw logs generally contain many repeated or useless information, the data preprocessor takes the raw logs from the underlying system as input and produces clean data for online prediction [10]. Here, the categorizer provides a standard categorization of RAS events, and the filter removes redundant data by conducting both a temporal compression at a single location and a spatial compression across multiple locations. Upon completion, the data preprocessor intends to provide a list of unique events for failure prediction.

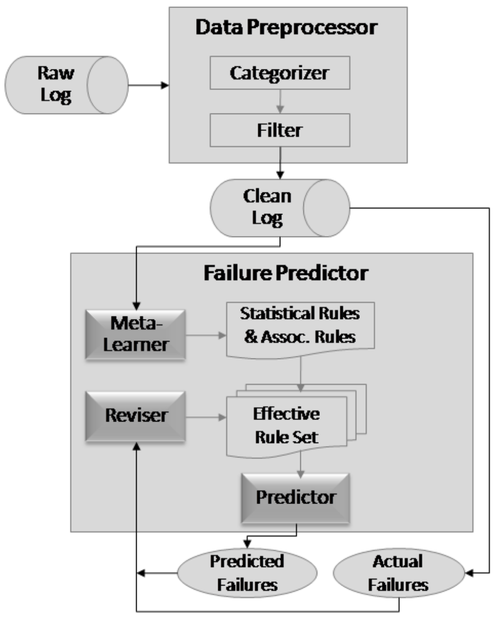


Figure 1. Dynamic Meta-learning Prediction Engine

The online predictor consists of three major components: *the meta-learner, the predictor, and the reviser*. It first triggers *the meta-learner* on the clean data to discover various fault patterns by applying multiple predictive methods. The generated rules, including both statistical rules and association rules, form the base for online prediction. These rules are subjected to modifications done by the reviser at runtime. The *reviser* monitors prediction accuracy by comparing the predicted results and the actual failures, and then constructs an effective rule set for failure prediction. Note that the effective rule set is dynamically adjusted to reflect the current state of the system and the prediction accuracy. The *predictor* continuously examines the runtime data collected by system monitor tools. In case that it discovers a matching pattern in the effective rule set, it will trigger a warning.

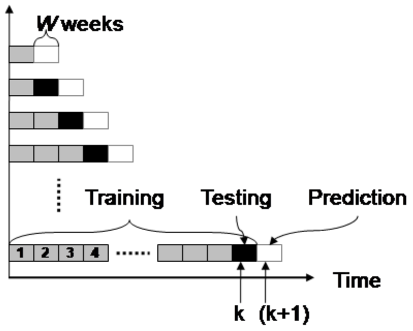


Figure 2. Dynamic training, testing and prediction. The grey and black boxes together represent the training set, the black box indicates the testing set, and the white box denotes the prediction set. According to the results on the testing set, the reviser dynamically modifies the rules generated by the meta-learner.

In the proposed dynamic meta-learning prediction engine, the *"dynamic"* feature comes from two perspectives: one is to continuously increase the training set during the system operation; and the other is to dynamically modify the rules of failure patterns by tracing prediction accuracy at runtime. Figure 2 gives an illustrative example on dynamic training, testing and prediction. Specifically, assume that dynamic retraining is triggered every W weeks (denoted as *dynamic window size*) and it is the start of the $[(k+1)\cdot W]^{th}$ week:

1. The meta-learner first generates rules using the training set composed of date from the previous $k\cdot W$ weeks (i.e. *the training set*);
2. The reviser constructs an effective rule set by verifying the rules on the $[k\cdot W]^{th}$ week (i.e. *the testing set*). This includes adding new rules and removing those rules resulting in high false alarm.
3. Once the effective rule set is obtained, the predictor uses it for failure prediction on the $[(k+1)\cdot W]^{th}$ week (i.e. *the prediction set*).

Meta-Learner. We have presented a meta-learning method to improve failure prediction in large-scale systems, as shown in Figure 3 [10]. *Meta-learning*, also known as *ensemble-learning*, can be loosely defined as learning from learned knowledge [21]. It is a technique that deals with the problem of computing a "global" predictor from the separately learned "base predictors" to boost overall predictive effectiveness. In particular, it learns to identify preferable combinations of based classifiers as well as their quantitative performance effects from previous results.

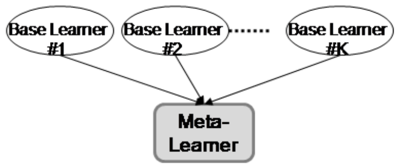


Figure 3. Meta-learner

In our case study, our meta-learner integrates two predictive techniques, i.e. *the statistical based method and the association rules,* with the objective to improve the coverage and accuracy of failure prediction. *The statistical based method* generates the statistical characteristics of fatal events, e.g. how often and with what probability will the occurrence of one failure influence subsequent failures. It is in the form of $f_k \rightarrow \{f_i, ..., f_j\}, conf$, where f is a fatal event and *conf* is the corresponding confidence value. *Association rules* examine the causal correlations between non-fatal and fatal events. For example, an association rule is in the form of $\{e_1, e_2, ..., e_k\} \rightarrow f, conf$, where e_i is a nonfatal event, f is a fatal event and *conf* is the corresponding confidence value.

Reviser. The reviser is responsible for modifying the rules generated by the meta-learner on the training set to construct an effective rule set. The principle is based on a key observation of failure characteristics in large-scale systems. In [5], Song et al. show that failures have temporal locality, meaning that a failure may re-appear multiple times before its root problem is solved. Hence, the data set from the latest period is crucial for representing the failure patterns in the next period. The reviser adjusts the rule set based on the testing set (the data collected immediately before the prediction set), thereby making it possible to better capture failure patterns in the prediction set.

The proposed reviser applies the ROC (Receiver Operating Characteristic) analysis, and the detailed algorithm is illustrated in Figure 4. ROC analysis aims at selecting possibly optimal models and discarding suboptimal ones independently from the class distribution [11].

On the testing set:

```
For each rule r generated by the meta-learner {
    1) count its true positives TP, false positives FP, and
       false negatives FN on the testing set;
    2) calculate precision(r) and recall(r) as described in
       Section 4.2;
    3) calculate ROC distance:
       $d_{ROC}(r) = \sqrt{precision(r)^2 + recall(r)^2}$
    4) put the rule r into the effective rule set if its ROC
       value is larger than a predefined threshold
       MinROC. For example, we only put those rules
       outside of the grey area into the effective rule set.
}
```

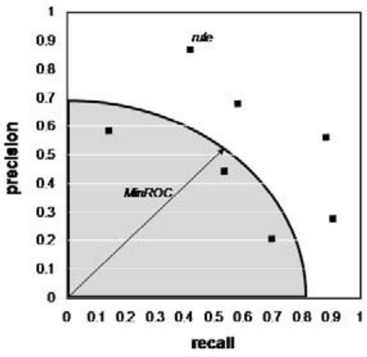


Figure 4. The Pseudo-code for the Reviser

Based on the effective rule set, first create **two lists**:

$F - List = \{ f_i \rightarrow \{e_{i1}, e_{i2}, ..., e_{ik}\} : 1 \le i \le N_f \}$

$E - List = \{ e_m \rightarrow \{ f_{m1}, f_{m2}, ..., f_{mn}\} : 1 \le m \le N_e \}$

Where f_i is a fatal event and e_j is an event (nonfatal or fatal)

During prediction, when an event *e* occurs:

```
(1) Append e into the prediction event set
    $E = \{e_1, e_2, ...., e_n, e\}$ where the events are sorted
    in an increasing order of their occurrence times,
    and remove $e_i$ when $|T_e - T_{e_i}| > T_{predict\_window}$
(2) Obtain potential failures that may be triggered by
    e according to the E-List: $e \rightarrow \{f^1, f^2, \ldots, f^k\}$
(3) For each failure in the set of $\{f^1, f^2, \ldots, f^k\}$, go
    through its event list according to the F-List:
    $f^i \rightarrow \{e^i_{i1}, e^i_{i2}, \ldots, e^i_{ik}\}$
(4) If $\{e^i_{i1}, e^i_{i2}, ..., e^i_{ik}\} \subseteq E$ , then produce a
    warning that the failure $f^i$ may occur within
    $T_{predict\_window}$
```

Figure 5. The Pseudo-code for the Predictor

Predictor. The predictor actively monitors runtime events and triggers a warning when a rule is observed within a fixed time window $T_{predict\_window}$ (i.e. *prediction window*). The specific prediction method is presented in Figure 5.

The online prediction engine is implemented in Java. It is connected to Weka [32] for generating association rules, and is also connected to an Oracle database for storing and querying the knowledge base.

4. Case Study

In this section, we evaluate the proposed online predictor by testing it with a RAS log collected from the production Blue Gene/L system at SDSC. We first describe the RAS log, followed by presenting the results. We conduct three sets of experiments. The first set is to examine prediction accuracy by using different dynamic window sizes; the second set is to study the benefit brought by using the reviser; and the last set is to analyze the number of rules changed by using dynamic training and testing.

4.1. The Blue Gene/L RAS Log

The Blue Gene system at SDSC (*San Diego Supercomputing Center)* consists of three racks with 3,072 compute nodes (4,144 processors) and 384 I/O nodes. The configuration is chosen to support data-intensive computing. Each node consists of two PowerPC processors that run at 700 MHz and share 512 MB of memory, giving an aggregate peak speed of 17.2 teraflops and a total memory of 1.5 TB [34]. We have acquired a 130-week RAS log (i.e. about two and half years) from this production system.

In Blue Gene/L, the Cluster Monitoring and Control System (CMCS) service is implemented on the service nodes for the purpose of system monitoring and error checking. The service node, which is available in each midplane, acquires specific device information, such as RAS (Reliability, Availability and Serviceability) events, directly through the control network. Runtime information is collected from computer and I/O nodes by a polling agent running on each BLC, reported to the CMCS service, and finally stored in a centralized DB2 repository. This system event logging mechanism works in a granularity of less than 1 millisecond. More details of the system architecture can be found in published literature [6].

The entries in the RAS log include hard errors, soft errors, machine checks, and software problems. Information about scheduled maintenance, reboot, and repair is not included. Each record of the logs has a number of attributes, including event type, event time, job ID, location, facility, entry data, and severity. Here, the *SEVERITY* attribute can be one of the following levels - INFO, WARNING, SEVERE, ERROR, FATAL, or FAILURE - which also denotes the increasing order of severity. Our primary focus in this study is *to predict FATAL and FAILURE events* (denoted as *fatal events*, while other events are denoted as *non-fatal events*).

Table 1 summarizes the RAS log from the Blue Gene/L system at SDSC. The raw log has more than one million entries. By applying data preprocessing, i.e. temporal compression at a single location and spatial compression across multiple locations where the threshold for compression is set to 300 seconds, we have obtained a cleaned log, whose information is listed in Table 1.

	SDSC
Start date	12/6/2004
End date	06/11/2007
No. of records after data preprocessing	559,211
Size of cleaned log	704 MB

Table 1. The RAS log from the production Blue Gene/L System at SDSC

4.2. Evaluation Metrics

Two evaluation metrics are used to measure prediction accuracy:

- *Precision*: defined as the proportion of correct predictions to all the predictions made

$$precision = \frac{T_p}{T_p + F_p}$$

- *Recall:* defined as the proportion of correct predictions to the number of failures

$$recall = \frac{T_p}{T_p + F_n}$$

Here, T_p is number of correct predictions (i.e. *true positives*), and F_p is number of false alarms (i.e. *false positives*), and F_n is number of missed failures (i.e. *false negatives*). Obviously, a good prediction engine should achieve a high value (closer to 1.0) for both metrics.

4.3. Results

In our experiments, the *support* and *confidence* values used in the meta-learner are set to 0.01 and 0.1 respectively. The *MinROC* value used in the reviser is set to 0.7. The prediction window is set to 300 seconds.

Figure 6 presents our prediction results by using different *dynamic window* sizes (2 weeks, 4 weeks, 6 weeks, 8 weeks, and 10 weeks). The x-axis shows the sequence number of the week, which is up to 130. Each plot has two curves, one for *precision* and the other for *recall*. We can see that both *precision* and *recall* are not stable during the first 10 weeks. In most cases, *precision* monotonically increases to 0.9, whereas the recall monotonically increases to 0.7. The reason is that during the initial phase, the effective rule set may not well capture failure patterns due to the limited size of the training set. We shall point out that even when the training set is two weeks, the predictor is still capable of capturing more than 47% of failures.

After the initial phase, we notice that both *precision* and *recall* become stable. In general, after the initial phase (i.e. the first ten weeks), no matter how frequent the online predictor re-train its rule set, the recall value is maintained around 0.7-0.8, whereas the precision value is between 0.9-1.0. It indicates that the online predictor is capable of capturing more than 70% of actual failures, with the false alarm rate less than 10%.

Further, we notice that both *precision* and *recall* decrease more than 10% during the 64th week. According to our record, the system went through a major system reconfiguration around this time. As a consequence, failure patterns are changed, thereby resulting in lower prediction accuracy during this period of time. Nevertheless, our predictor is capable of capturing newly discovered failure patterns. As we can see, both *precision* and *recall* are changed back to 0.9 and 0.7 on the next prediction set.

Comparing these plots generated by using different adaptation windows, we have made a key observation. In the initial phase, a small window size is needed for training and re-training. This can help the reviser to rapidly build up the effective rule set for online prediction. Once entering a stable phase, we can start to use a large window size to reduce the re-training cost.

In the second set of experiments, we investigate the benefit introduced by using the reviser. Specifically, we compare prediction accuracy produced by using and not using the reviser. Note that the main functionality of the reviser is to dynamically adjust the rules of failure patterns according to accuracy tracing. Figure 7 presents our results where the dynamic window size is set to four weeks.

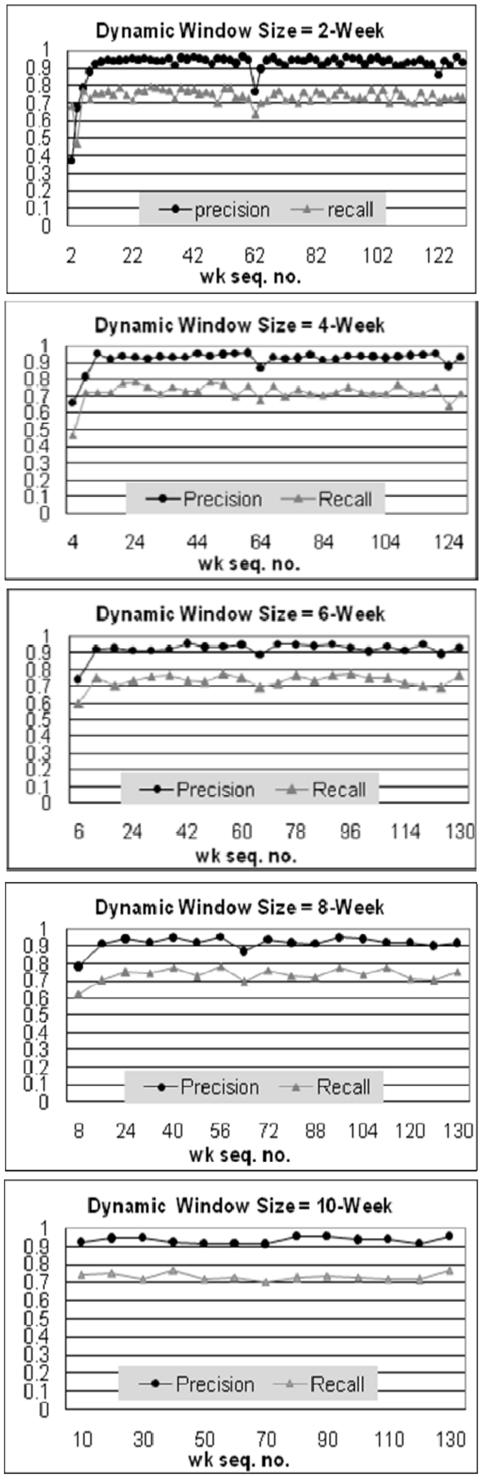
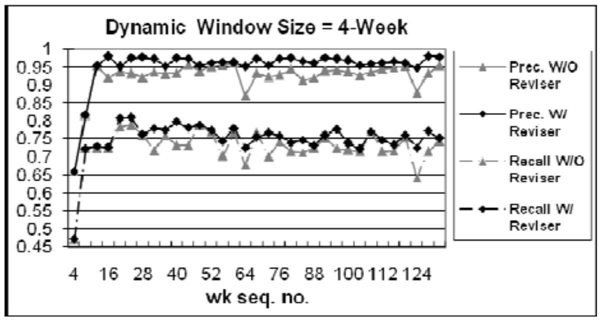
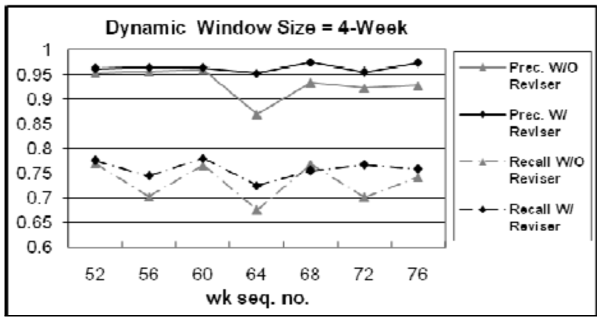


Figure 6. Prediction accuracy using different dynamic window sizes (2 weeks, 4 weeks, 6 weeks, 8 weeks, and 10 weeks)



(a)



(b)

Figure 7. Prediction results W/ and W/O the reviser. Figure 7(a) plots the prediction results for the entire log, whereas Figure 7(b) highlights the period between the 52nd week – the 76th week. We have observed similar results for other window sizes, so we omit them.

In the third set of experiments, we analyze the number of rules changed by using dynamic training and testing, and the result is presented in Figure 8 where the dynamic window size is set to four weeks. As expected, the effective rule set always changes and the change becomes less significant with time. At the beginning, there are 310 rules and 392 rules added into the effective rule set in the 4th week and the 8th week respectively. Starting from the 16th week, the number of rules in the effective set is stabilized. For instance, only 17 rules are added and 19 rules are removed from the effective rule set in the 128th week. We notice a substantial change occurs during the 64th week, where 76 rules are added and 85 rules are removed. During this period of time, a system reconfiguration occurs, thereby resulting in significant rule changes.

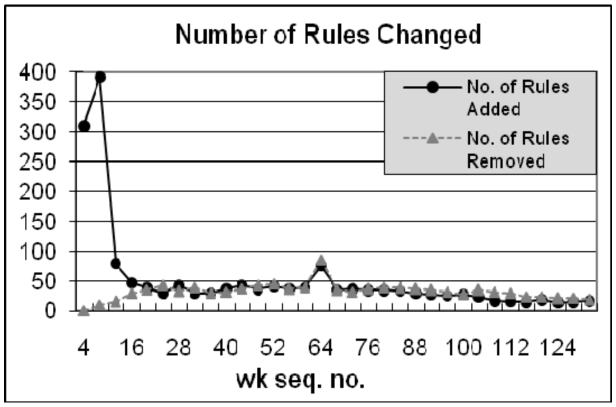


Figure 8. Number of rules changed by using our dynamic meta-learning prediction engine where the dynamic window size is set to four weeks. Similar patterns are observed with other window sizes.

5. Summary

In this paper, we have presented *a dynamic meta-learning prediction engine for large-scale systems*. It does not require a long training phase by dynamically increasing the training set during system operation. For instance, it can start to provide an acceptable failure prediction service after only two weeks of training phase. Our case study on a 130-week RAS log from the production Blue Gene/L system at SDSC has shown that it can effectively forecast failures with a precision of 0.9-1.0 and a recall of 0.7-0.8 by dynamically modifying its rule set according to accuracy tracing and dynamic re-training.

Our study has some limitations that remain as our future work. First, in the current design, the *dynamic window* size is fixed. Our on-going work includes adaptively changing this window size such that the system can automatically tune its size to reduce the training cost, without sacrificing the prediction accuracy. Second, we plan to investigate different revising algorithms to further improve prediction accuracy. Lastly, more case studies with a variety of HPC systems are needed. We are in the process of acquiring RAS logs from supercomputing centers, such as the Cray XT3 at ORNL, for the purpose of evaluating the proposed prediction engine.

Acknowledgement

This work is supported in part by US National Science Foundation grants CNS-0720549, CCF-0702737, and a TeraGrid Compute Allocation. The work of B.H. Park is supported by the U.S. Dept. of Energy, Office of Science, Advanced Computing Science Research Division.

References

[1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules", *VLDB*. Sep 12-15 1994, Chile, 487-99

[2] A. Bouteiller, T. Herault, et al. , "MPICH-V: A Multiprotocol Automatic Fault Tolerant MPI", *International Journal of High Performance Computing and Applications,* 2005.

[3] S. Chakravorty, C. L. Mendes and L. V. Kale, "Proactive Fault Tolerance in Large Systems", *Proc. of HPCRI Workshop in conjunction with HPCA*, 2005.

[4] E. Elnozahy and J. S. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery", *IEEE Transactions on Dependable and Secure Computing*, Volume 1, Number 2, 2004, pp. 97-108.

[5] S. Fu, C. Z. Xu, "Exploring Event Correlation for Failure Prediction in Coalitions of Clusters", Proc of SC 2007, 2007.

[6] A. Gara, M. A. Blumrich et al., "Overview of the Blue Gene/L System Architecture", *IBM J. Res. & Dev*. 49, No. 2/3, 195–212, 2005.

[7] E. Gabriel, G. Fagg, and et al., "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *Proc. of The 11th European PVM/MPI Users' Group Meeting*, 2004.

[8] R. Gioiosa, J. Sancho, S. Jiang, F. Petrini, K. Davis, "Transparent Incremental Checkpointing at Kernel Level: A Foundation for Fault Tolerance for Parallel Computers", *Proc. of SC2005*, 2005.

[9] A. Goyal, S. Lavenberg, and K. Trivedi, "Probabilistic Modeling of Computer System Availability", *Annals of Operations Research*, 1987.

[10] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-learning Failure Predictor for Bluegene/L Systems," *Proc. of ICPP'07*, 2007.

[11] J. Han and M. Kamber "Data Mining: Concepts and Techniques", 2nd Edition, Morgan Kaufmann, 2006.

[12] G. Hamerly and C. Elkan, "Bayesian Approaches to Failure Prediction for Disk Drives", *Proc. of ICML*, 2001.

[13] J. Hellerstein, F. Zhang, P. Shahabuddin, "A Statistical Approach to Predictive Detection", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 2001.

[14] R. K. Iyer, L. T. Young, V. Sridhar, "Recognition of error symptoms in large systems", *Proceedings of 1986 ACM Fall joint computer conference*, 1986.

[15] Z. Lan, Y. Li, P. Gujrati, Z. Zheng, R. Thakur, and J. White, "A Fault Diagnosis and Prognosis Service for TeraGrid Clusters", *Proc. of TeraGrid'07* , 2007.

[16] Y. Li, Z. Lan, "Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing", *Proc. of IEEE CCGrid'06*, 2006.

[17] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramanium, R. Sahoo, "BlueGene/L Failure Analysis and Prediction Models", *Proc. of DSN*, 2006.

[18] A. Oliner, L. Rudolph, and R. Sahoo, "Cooperative checkpointing theory," in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[19] A. Oliner and J. Stearly, **"**What Supercomputers Say: A Study of Five System Logs", *Proc. of DSN* 2007.

[20] A. Oliner, Ramendra K. Sahoo, José E. Moreira, Manish Gupta, Anand Sivasubramaniam, "Fault-Aware Job Scheduling for BlueGene/L Systems", *IPDPS,* 2004.

[21] R. Polikar, "Ensemble Based Systems in Decision Making", *IEEE Circuits and Systems Magazine*, vol.6, no. 3, pp. 21-45, 2006.

[22] D. Reed, C. Lu, and C. Mendes, "Big systems and big reliability challenges," in *Proc. of Parallel Computing,* Germany, 2003.

[23] R.K. Sahoo, A.J. Oliner et al., "Critical event prediction for proactive management in large-scale computer clusters", *Proc. of KDD*, 2003, pp. 426-435.

[24] Hoffmann, Salfner et al, "Advanced Failure Prediction in Complex Software Systems", *Proc. of SRDS,* 2004.

[25] M. Schulz, G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, P. Stodghill, "Implementation and Evaluation of a Scalable Application-level Checkpoint-Recovery Scheme for MPI Programs", *Supercomputing 2004*. November 6-12, 2004.

[26] Kai Shen, Ming Zhong, and Chuanpeng Li, "I/O System Performance Debugging Using Model-driven Anomaly Characterization", *4th USENIX Conference on File and Storage Technologies*, 2005.

[27] D. Turnbull, N. Alldrin, "Failure Prediction in Hardware Systems", *UCSD CSE221 Project*, 2003.

[28] K. Trivedi and K. Vaidyanathan, "A Measurement-based Model for Estimation of Resource Exhaustion in Operational Software Systems", *Proc. of the 10th Int'l Symposium on Software Reliability Engineering*, 1999.

[29] K. Vaidyanathan and K. Gross, "MSET Performance Optimization for Detection of Softtware Aging", *Proc. of ISSRE*, 2003.

[30] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains", *Proc. of IEEE Intl. Conf. On Data Mining*, 2002.

[31] C. Wang and F. Mueller and C. Engelmann and S. Scott, "A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance", *Proc. of IPDPS* ,2007.

[32] Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", *2nd Edition, Morgan Kaufmann*, San Francisco, 2005.

[33] RAD Lab: Reliable Adaptive Distributed Systems Laboratory. http://radlab.cs berkeley.edu/

[34] SDSC Blue Gene/L Homepage. www.sdsc.edu/us/resources/bluegene

[35] The TOP500 Supercomputer Sites. www.top500.org