Adaptive Event Prediction Strategy with Dynamic Time Window for Large-Scale HPC Systems

Ana Gainaru
UIUC, NCSA, Urbana, IL, USA
UPB, Bucharest, Romania
againaru@ncsa.illinois.edu

Franck Cappello
INRIA, France
UIUC, Urbana, IL, USA
fci@lri.fr

Joshi Fullop
UIUC, NCSA, Urbana, IL, USA
jfullop@ncsa.uiuc.edu

Stefan Trausan-Matu
UPB, Bucharest, Romania
stefan.trausan@cs.pub.ro

William Kramer
UIUC, NCSA, Urbana, IL, USA
wkramer@ncsa.uiuc.edu

ABSTRACT

In this paper, we analyse messages generated by different HPC large-scale systems in order to extract sequences of correlated events which we lately use to predict the normal and faulty behaviour of the system. Our method uses a dynamic window strategy that is able to find frequent sequences of events regardless on the time delay between them. Most of the current related research narrows the correlation extraction to fixed and relatively small time windows that do not reflect the whole behaviour of the system. The generated events are in constant change during the lifetime of the machine. We consider that it is important to update the sequences at runtime by applying modifications after each prediction phase according to the forecast's accuracy and the difference between what was expected and what really happened. Our experiments show that our analysing system is able to predict around 60% of events with a precision of around 85% at a lower event granularity than before.

Keywords

HPC systems, logfile analysis, event prediction

1. INTRODUCTION

There is a growing demand for computational power for science and engineering applications. Large parallel HPC systems are designed and deployed in production facilities, with hundred thousands cores [14]. However, increasing component count in the system decreases the system-wide MTBF (mean time between failures), making reliability a major concern [15]. Most large-scale applications rely on checkpoint-restart techniques to recover from faults. Each fault in any of the components the application is using, could end in an interrupt and a restart of the application from the last checkpoint. Recent studies consider that this approach is no longer scalable to the size of future Exascale systems [16, 17, 24]. Recently, research has been focusing on characterizing the behaviour of the system in order to take proactive measures for detecting, tolerating and recovering from failures. For this, monitoring systems require a reliable prediction system to give information on what will be generated by the system and at what location. Still, being able to accurately predict what the system will experience at any given time is a holy grail that is not yet achieved.

In this paper, we propose a new methodology for analysing messages generated by the system. Our aim is to extract sequences of events that frequently occur together for both no-faulty events and failures. Our method is able to find correlations between events irrespectively on the time delay between them by using a different time window for analysis for each type of event and for each analysed log segment. Also we believe that updating the sequences during the runtime of the system in order to reflect the behaviour of events at any given moment is as important as the initial extraction of the sequences in the training phase, as we will show in the experiment part. Most of the related research does not make modifications on their rules once they have extracted them.

The contributions of this paper are the following:

- A mining model for frequent sequences using a dynamic time window.

  Most of the current research uses a pre-definite time window for which rules are investigated and extracted [7, 8, 9, 10, 13]. However a fix interval for all event types and for the whole life of the system is not realistic. We propose a novel model that adapt the analysed time intervals according to each event's behaviour. Our method investigates all the events that are generated between two adjacent occurrences for each event type and decides which are the ones that have a strong correlation. The adaptive behaviour of our model allows to find related events irrespective of the time delay between them.

- Online updates for the event chains extracted in the training phase.

  Events generated by the system are in constant change during the life-cycle of the machine, so every extracted rule or se-

SLAML '11, October 23, 2011, Cascais, Portugal.

quence must be updated to reflect the behaviour of failures at any given time. Our method updates the confidence of each frequent sequence and the list of message types that need to be spatial filtered, each time there is a modification in the event distribution and characterization.

The rest of the paper is organized as follows: Section 2 presents an overview of related research in the field of event prediction. Section 3 presents the methodology used by our system for extracting the frequent sequences of events and the updating modules used in the online phase. Section 4 presents experimental results for our system, and finally, in section 5 we provide conclusions and present possible directions for future work.

2. RELATED WORK

Over the last years, various prediction methods have been proposed, by using both statistical and data mining driven perspectives for analysing log files generated by large systems. There is also research that is not focusing on log files, but take into consideration different other characteristics of the system. For example, in [3], the authors inspects system performance metrics and then use probabilistic methods to establish correlations between events. However, in this section we will focus our attention on methods that extract chain of events that are related by inspecting logs generated by different components of the machine. The papers dealing with statistical modelling of failures [1, 2, 4] use data-driven models and all have a preprocessing phase where events are filtered using a fix time window and compressing multiple locations. In [1] the authors are using the a method to predict failures from different categories being able to predict 80% of the memory and network failures and 47% of application I/O failures. However they divided the events into 5 categories and are able to predict that one of them will happen without having further information. For next generation prediction system only 5 categories are not enough to describe all types of failures. For example it is important to know what type of memory error is predicted and the location where it will occur.

Other methods are using data mining algorithms in order to extract sequences of events that lead to a failure and are storing them with different methods that allow event prediction. In [5] and [6] the authors investigate parameter correspondence between different log messages in order to extract dependencies among different system components; [5] uses a modified Apriori method and [6] a PCA based algorithm. Their case study is on Hadoop where each message generated is characterized with a task and job information. In our experiments section we will show that this method is not practical for HPC system logs since not many of the messages share common parameters.

A common way of analysing logs is by examining correlation between past and fatal events in order to learn patterns for future failure prediction [7, 8]. Both [7, 8] extract rules for a fix time window and offer prediction only for fatal events. [7] is generating association rules between fatal events and between non-fatal and fatal events that leave a prediction window big enough so that proactive measures can be taken. Their analysis of the BlueGene/P system shows that their method is capable of capturing only around 15% of the total failures with a false alarm rate of around 60%.

In [8], the authors use the same based method as the previous paper, but they adapt the rule set dynamically considering the difference between what was predicted and what really happened. The experiments made on BlueGene/L show results of 70% recall and 80% accuracy. However the test case is chosen by a 10-fold cross-validation; they divide the log in 10 folds of equal size and choose 9 for extracting the rules, and one for testing them. This explains the high values in precision and recall since the algorithm uses more than one year of logs for extracting the rules.

In [10], the authors use a similar algorithm as the previous ones with the difference that the rule extraction uses a self defined way to measure correlations of events that may happen interleavedly. They use their method for predicting all events generated by the system or only fatal ones. The experiments made on Hadoop and on another unnamed HPC cluster show a low false positive rate but with the cost of capturing only around 15% of total events.

Another research direction is presented in [11], where the authors investigate both usage and failure logs. For each failure instance, past and future failure information is accumulated and based on these features, they applied different decision tree classifiers in order to predict failures with a fix time window. Their method is not very general since not all systems have a failure logs. However, the results are very good, showing a precision of around 70% and recall of about 80%.

All the methods that analyse the system log files have a preprocessing step to filter events in time and space and use a fix time window in their search for correlation. The only paper that shows good results for both precision and recall is the one that is analysing failure traces and not general system logs. However they only consider a small number of failure categories and require a up-to-date failure log where system admins must input information.

3. SYSTEM DESIGN

The methodology used for our system is divided into two major phases, the training methods where the initial sequences of frequent events are extracted and the online modules where prediction is being offered and according to what is generated by the systems, the sequences are updated. Figure 1 summarizes the whole event correlation mining process.

For a better understanding of the following sections we present some definitions. A sequence of events or a chain of events represents a ordered list of event types that frequently occur together. Event types, event categories or templates all represent regular expressions that describe different messages generated by the system.

Our system collects the data from log files and then parses them for preprocess purposes, extracts the categories and filter repeated events. The data is then analysed by the mining module where correlation chains between different events are generated. There needs to be a balance between how fast the mining systems can be started for extracting sequences and how much info is gathered in order to have a high precision for the correlation chains. We found that 3 months offer a fair trade-off since we observed these initial

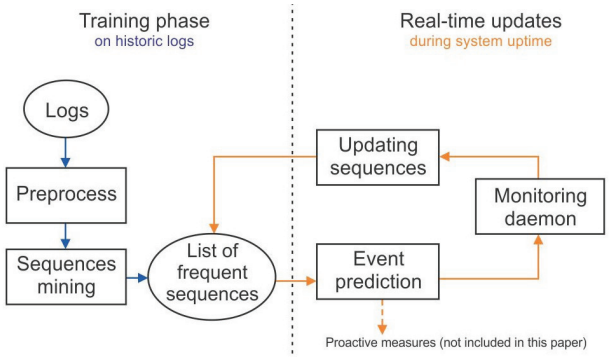

Figure 1: Event correlation mining process

System	Templates No	Example
BlueGene/L	207	node card vpd check: * node in processor card slot * do not match.
Mercury	321	xcatd: set boot request from *
LANL	52	failed subcommands *

Table 1: Log data examples

mined chains offer a good characterisation of the system behaviour. The chains are updated as the system progresses in time.

3.1 Training modules

3.1.1 Preprocessing

In the first step, the logs in raw format are parsed by our system in order to identify frequently occurring messages with similar syntactic patterns, resulting in a list of templates. We use the Hierarchical Event Log Organizer [12], which essentially extracts all event types generated by the system, for both fatal and informational events. Table 1. presents the number of templates for each analysed large-scale system and some examples. We will use the templates in the way previous work uses categories and investigate the correlation between them. The only difference between them is that HELO generates templates with a finer granularity by offering different types of network events instead of considering everything one.

Some component failures may cause logging of large numbers of messages. From our previous research we observed that for example memory and processor cache failures usually result in a single faulty component generating hundreds or thousands of messages in less than a day [19].

The filtering method used in previous related research contains temporal compression at a single location, and spatial compression across multiple locations. Both methods use a fix time window for which two occurrences of the same events type is considered one. For our system, on each location, we use a temporal compression using a fixed window, storing both the beginning and the end of the burst of messages.

In order to optimize the spatial filtering process, we investigate which are the event messages that propagate on different locations. We parse the logs one more time and extract the propagation confidence for each template. The propagation confidence is defined as the ratio between the number of occurrences where the message propagated on different locations and the total number of occurrences for the specific template. We use the same time window as for the temporal filtering process to establish if two occurrences share the same root problem.

The templates that have a propagation confidence over a threshold are considered more likely to appear on multiple locations and are spatial filtered in the analysis process. However, the rest of the templates are not spatial compressed at all, the cases of two occurrences in a small time window on two locations is treated as two separate errors. The propagation confidence for each template is update after the training phase.

The method has little effect in the sequence extraction phase since, we are looking at frequent patterns and the chance of having two of the same events that happen on two different locations and that have different root causes frequent in a historic log in close to zero [19, 20, 23]. However the improvement is given in the prediction process when those two events could lead to two different effects and need to be both monitored.

The preprocessing method is presented in Figure 2.

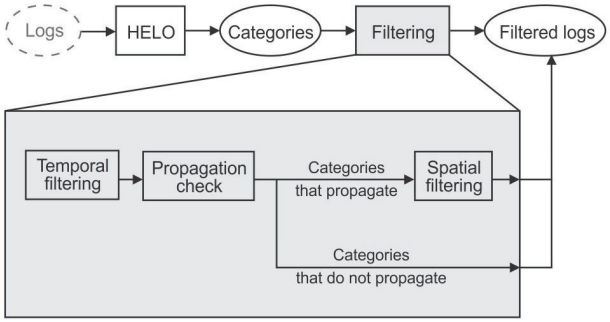

Figure 2: Preprocessing strategy

3.1.2 Mining frequent sequences

The mining module is in charge of extracting sequences of events that frequently occur together. Usually a system is experiencing different correlations of events: between information events, for example installation messages generated by different components, between non-faulty event and faults, for example chain of warnings messages leading to failure events, or between failure events. From our experience, the time delay between two correlated events may range from seconds to hours and days, and can be constant or vary from one occurrence to another. For example "job start" and "job end" could happen days apart in some cases or minutes apart in others. Previous work fail to discover events that happen at wider time rage than the time window chosen in the mining process or events that are separated by variable time intervals. Also, sequences made only of info messages are usually ignored by related research. However we consider that they are also useful to monitor since incomplete messages sequences could indicate a failure.

The mining process is illustrated in Figure 3. We analyse each event type separately and combine the results in the final step. The middle module is the one responsible for finding frequent sequences correlated with each message type of interest. In the figure the 'a' character represents each time occurrence for the specific event. The time window used for extracting frequent events is defined by the time space between two adjacent time occurrences. After all chains are extracted, the last module intersects them and returns only the common ones. The log entries are once again parsed and the system computes the final confidence for all selected rules.

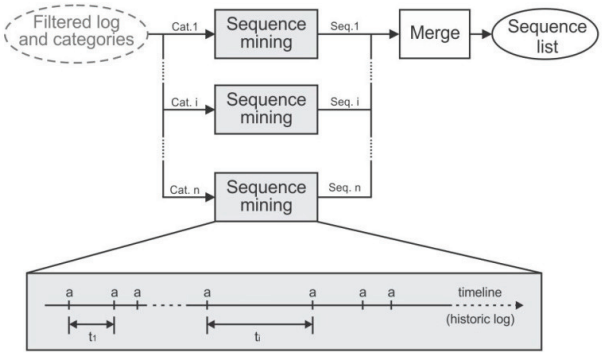

Figure 3: Mining frequent sequences

Algorithm 1 Mining Process

Input: Support Threshold Sth
Output: Collection G[] of reorganized groups

```
1: Add all templates with the support count above Sth to L(1);
2: F={}
3: for L(k) not null do
4:     k++; C(k)={}; L(k) = {};
5:     Create C(k) by concatenating two (k-1)-ary adjacent subsets
       from L(k-1) - for example concatenate(a,b):
6:     if a[1-len(a)]==b[0:len(b)-1] then
7:         Return the set(a[0],common part, b[len(b)])
8:     end if
9:     Scan the logs to validate each frequent kâĹŠary candidates from
       C(k) and store the frequent ones in L(k)
10:    Keep the validated k-ary in F
11:    Delete from F all i-ary sets (with i<k) that are part of one of the
       k-ary introduced into F previously
12: end for
13: return F
```

The mining module uses a modified version of Apriori, similar with the one presented in [10]. The Apriori algorithm is used in the data mining community for learning association rules in transactional data. In our case an item set is define by all events that appear for one time window. The algorithm generates frequent k-length item set, that represent frequent sequence of k different events in our case. Frequent sequences are defined by a support count, which represents the percentage of the time intervals that contain the sequence. For each event type, the time intervals are computed as the time window between every two adjacent occurrences of the corresponding template. The pseudocode for the method we use is illustrated in Algorithm 1.

Apriori searches for the longest possible item set without returning any of the other lower length sets. In our case, we are interested in any frequent sequence no matter of its length. For this we save all k-any candidates that are validated for all values of k over 2 and eliminate all sequences that are included into larger ones. For example if (A B C) is a frequent sequence we do not need to store the sub-sequence (A B).

The output of this module is represented by a set of frequent sequences, one for each template given by the first module. In the last phase the sets are merged and only the ones that are frequent for all templates that compose them are kept. For example if (A B C) is found as a frequent sequence for event A, and (B C) is not frequent for event B, we discard the set.

Another advantage for our method is visible for HPC systems that do not use a central clock for synchronizing the nodes. This could result in time inconstancies that affect the way the correlations are extracted with previous methods. However, by using a dynamic time window, we believe that with our method, the effect of the induced time lag on the analysis is minimal.

3.2 Runtime modules

3.2.1 The prediction method

Our prediction method uses the sequences extracted in the previous phase by storing them in a simple list structure; each element keeps informations about the probability of the next event in the chain appearing. Also each element stores information about the average time interval to the next event and the standard deviation for this value. An example is presented in Figure 4.

The list differs from Markov chains since it keeps memory information. For each element we store the probability for the future

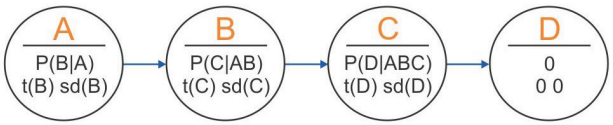

Figure 4: Sequence storing structure example

event to be predicted based on information about all past events in the list:

$$P(A_n|A_1A_2..A_{n-1}) = \frac{Count(A_1A_2..A_n)}{Count(A_1..A_{n-1})}$$

In our example, for C to be predicted we look at element B and inspect the probability of generating event C after receiving both AB, $P(C|AB)$. Based on all probability values from all active lists, we predict the future event. In case of elements for which we have a low time standard deviation, we are also able to predict the time of occurrence.

In the future we plan to incorporate in the module also the probability P(Clnot(AB)) of event C happening in case A or B did not happen.

3.2.2 Updating the sequence list

Configuration changes or updates are a constant part of a system life cycle so changes must be taken into consideration [18]. None of the previous work studies the lifespan of prediction rules and how these can be updated to reflect the behaviour of the system. We propose a monitoring daemon who oversees the modifications in events distribution and re-calculates the confidence values for each sequence accordingly. The formula used is the same as the one presented in the previous section having the count values modified by the new data. If there is no difference between what was predicted and what the system generated, the confidence values are still modified, since the rule is now more trustworthy. Also, the daemon checks the propagation behaviour of each template and switches templates from one group to another depending on the case, for the preprocessing phase of new events.

Figure 5 shows the number of novel templates generated by the clustering tool we used, HELO [12], for every week for the three analysed systems. During a systems lifetime there are many modifications on the initial templates, due to configuration changes or to new components installation. We monitor the changes each week and for new templates we compute new sequences that contain them by applying the same algorithm on the time intervals obtained between the first and last occurrences of the event.

4. EXPERIMENTS

The measures used are the evaluation units presented in all research related to prediction: precision and recall. Precision in our case is defined as the number of corrected predicted events over total number of predicted events, basically showing how trustworthy are the generated sequences. We consider a correctly predicted event if the event is generated by the system and the estimated time, considering the standard deviation, is correct. Recall determines how many of the total existing rules are we able to extract. For this we use two definitions for recall. The first one represents the number of corrected predicted events over the total number of existing events in the logs and is called total recall and the second one, called type recall, determines the recall value for each event type by computing the ratio between number the corrected predicted oc-

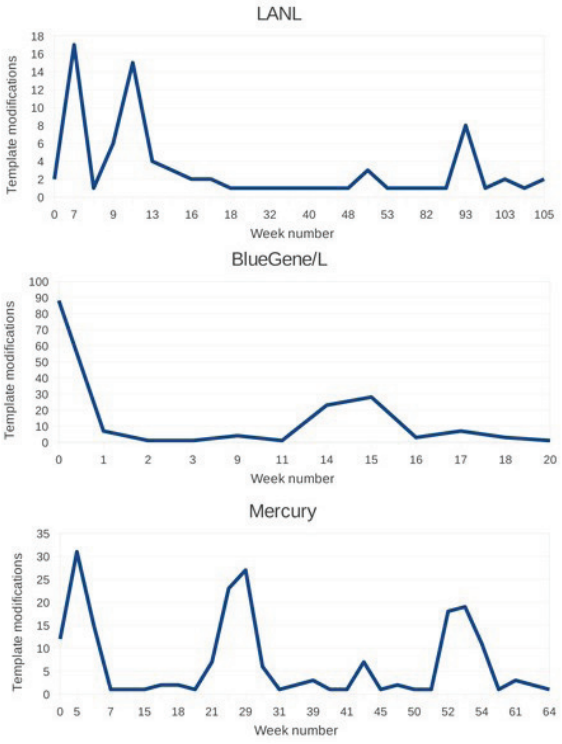

Figure 5: Template modifications per week

currences over the total number of occurrences of the same type of event.

We made two type of experiments. In the first one we analyse log files from three real systems, use the first three months for the training phase and the next months to simulate the runtime behaviour of the system for the online phase. The description of systems is given in table 2.

Recall is a measure of completeness. However it is hard to define it for prediction, since the total list of events that are part of some sequences is unknown. For this reason, the second type of experiments are made on synthetic logs that we randomly generate using characteristics showed by HPC systems. We build a log generator that takes as input the number of events, the number of sequences of events, the minimal and maximal number of events in each sequence and the minimal and maximal number of repetitions for each sequence. The sequences are randomly plotted with different timestamps within a whole timespan of six months. Also among sequences the generator plots a given number of events that are not part of any sequence. For the future we plan to generate logs that follow the same failure and event distribution that we observed in current HPC systems.

We create different test sets by choosing the parameters so that the total number of events and total number of generated lines is similar to the logs generated by the three systems. We investigate the ratio between the corrected predicted events over all events that could have been predicted.

For the rest of the paper, we define noise as messages generated by the system that are not part of any sequence, or occurrences of templates that usually are part of sequences but not for current entry. We call them noise because they are unpredictable but interfere in the prediction phase.

System	Mercury	BlueGene/L	LANL
Number of nodes	635	5252	256
Number of msg	100G	10G	433K
Initial templates	215	171	30
Templates after update	321	207	52
Analyse interval	Jan 2008 to July 2008	June 2005 to Dec 2006	2003 to 2006
References	[22]	[21]	[20]

Table 2: System information

System	Mercury	BlueGene/L	LANL
Precision	0.81	0.87	0.79
Total Recall	0.01	0.03	0.1
No of correlation chains	3	5	6

Table 3: Variable correlation results

4.1 Results

4.1.1 Variable correlation

We implemented the method described in [5] that investigates variable correspondence between different log messages. We made experiments in order to see how applicable is the method for HPC large-scale system logs. The results are presented in table 3.

The method is able to find sequences of strong correlated events shown by the high value of precision. However the total number of sequences found is very low, resulting in low values for recall for all HPC systems. The results show that not many generate messages share variables, the complexity of these systems making the method unusable.

4.1.2 Parameters

Our method uses a number of parameters. In this section we will investigate how tuning these parameters influences the results. In Figure 6b we present changes in precision and total recall as we change the threshold used by the Apriori algorithm to extract frequent event chains. With higher similarity, the total number of extracted sequences decreases, so it is normal to see a decrease in recall. However the sequences extracted have a high accuracy, resulting in higher values for precision.

Figure 6a presents how the update interval influences the results. We investigated intervals for one day, one week and one month. As the time interval increases the update module has more information when extracting new sequences but the prediction module uses old sequences for a longer time. Each system has a different template modification distribution so the best value for the update window differs depending on the system.

Next we compared the results obtained using the spatial filtering described in the previous section and the one that does not consider how events propagate. The precision of the extracted rules seem not to be effected by this parameter. On the other hand, recall values are higher for two of the systems, when using our filtering method, with differences of around 5%. This can be explained by the fact that the other methods filter some events that are part of active chains, making it impossible to predict the future event from the respective chain.

4.1.3 Real traces

For our experiments we use real traces generated by three HPC systems: BlueGene/L and LANL were chosen for comparison purposes since many papers make their experiments on these systems;

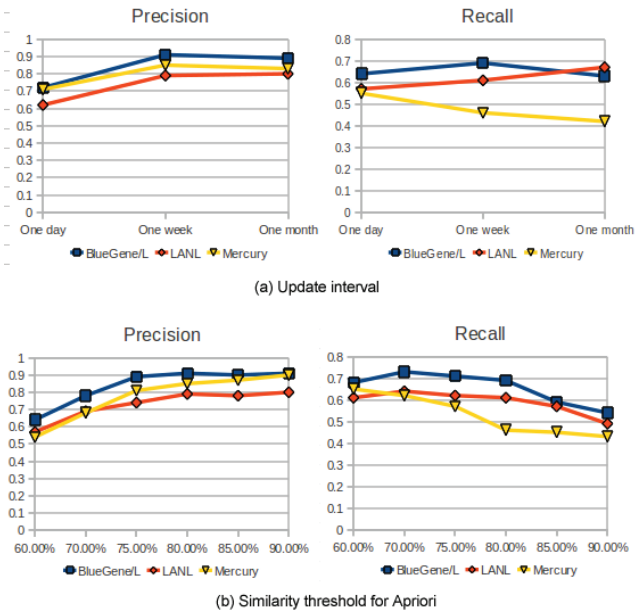

Figure 6: Parameter analysis

Mercury is a system at NCSA that generated a high number of messages per day and that had a lot of changes during the systems lifetime. We used this system to test how our method deals with many updates in the update process.

Figure 7 presents the distribution of the sequences containing different number of event types. As the figure show most of the chains are composed of two or three event types. After manual inspection we observe that a large number of small sequences could have been merged into one. For example sequence AB and BC are frequent, but not ABC even though from our manual inspection it should be. We investigate the reason for this, in the next section. We plan to analyse more in the future the event sequences themselves and offer a more in-depth characterization since they affect the results and time complexity for the prediction process.

Figure 8 presents the precision and total recall obtained using out method for all three systems. Compared to other methods that analyse the same system, the results are equivalent to the ones presented in [8]. However we use a much lower granularity for our categories, we do not just predict hardware failures, but give details on what hardware problem will occur. For example the template "pbs_mom: sister could not communicate(*) in *, job_start_error from node * in job_start_error" indicates a failure of a PBS (Portable Batch System) daemon to communicate.

The recall values are low because not all event types are part of sequences, for example human errors or events like "job start" will not be able to be predicted without other context information. We investigate the type recall in Figure 9, by looking at each individual template.

The results show that for all systems almost a third of the templates have a very low recall the rest having recall values over 40%. We believe the first category represents the templates that are unpredictable as we mention earlier. From the ones predictable, 2 thirds have high values, of over 90%. The exact reason for all different recall values obtained for these category of templates is unknown. After inspecting the sequences generated by our system, we observed that messages that are very frequent in the logs, usu-

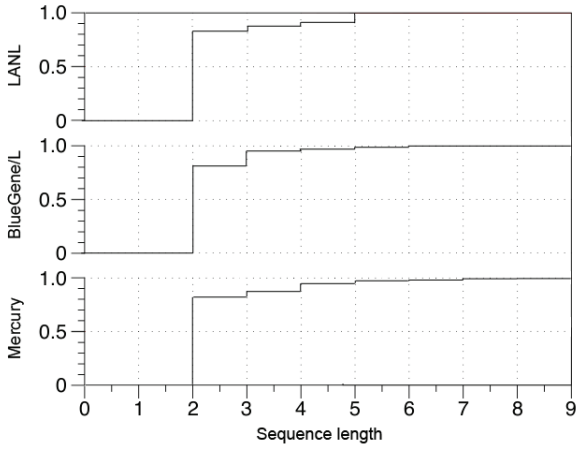

Figure 7: Sequence size distribution

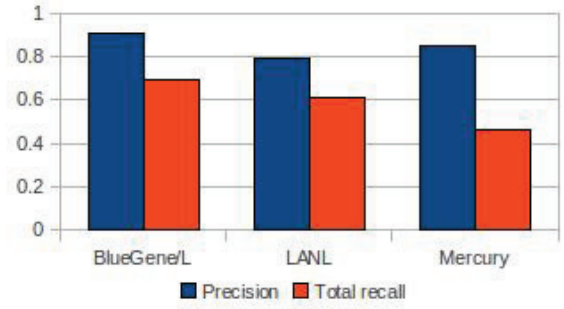

Figure 8: Precision and total recall

ally informational messages, tend to be part of many sequences and tend to have a low type recall. An explanation for this phenomena is that event types that are part of multiple sequences usually have a low probability value in our storing structure and are rarely predicted.

4.1.4 Synthetic logs

We developed a log generator that simulates the behaviour of a HPC system, but for which the sequences are known beforehand. We generate three months message logs for the training phase and three more months for the prediction phase, where we extract the precision and the recall value. We made experiments by tuning parameters, with the purpose of understanding what are the factors in the behaviour of a systems log file that influences the precision and recall of our method.

The difference in precision and total recall between different systems is given by many factors. Systems have different event types and total number of generated messages. We investigate the influence of these factors in Figure 10a and 10b. The ones with more total messages are more likely to repeat the normal behaviour sequences often, decreasing the probability of introducing false event types in sequences. This is shown in figure 10a by the increase in the recall value, without losing precision. However, the more event types the system generates, sequences get more complex so the chains we extract are less precise. Surprisingly, as the sequences contain more elements, the recall value actually increases until it reaches a top. This can be explained by the fact that more inaccu-

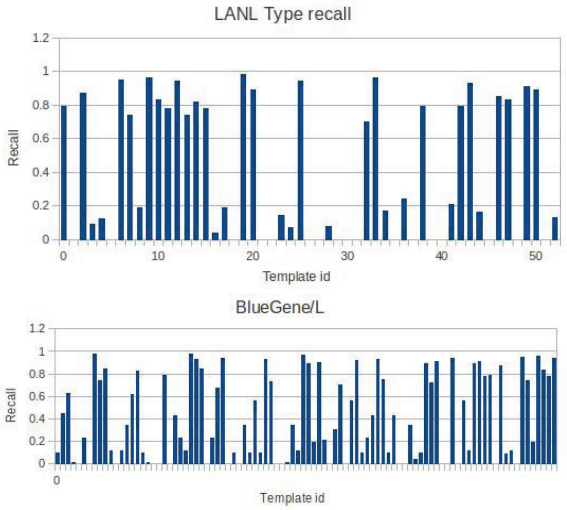

Figure 9: Recall for each event type

rate sequences are found, thus covering more events. This explains why even though Mercury has a higher number of event types and log messages, it presents a lower recall than for BlueGene/L.

Next we investigate the influence of sequence complexity on precision and recall. The results are presented in Figure 10c and 10d. The first figure investigates the average number of events in each sequence and the second one looks at the effect of noise on sequences. The number of events in a sequence has an effect on recall and precision from an early point. When all sequences are made out of 2 or 3 event types, the results are the best, afterwards values get balanced having the recall at low values.

The main reason is that we generate the sequences randomly, so for higher numbers of elements in each chain, there is a higher probability for having the same template in multiple sequences. After forcing all sequences to have different event types, the recall balanced at much higher values. This is another reason for the low recall value on Mercury, since it returned sequences with the most number of duplicates.

Due to noise, the sequences found are not the ones used for generating the logs, but they are sub-sequences of them. The recall of our sequences seems to not be affected by the noise; on the other hand precision is decreasing as the noise becomes larger. The discovered sequences cover most of the events in the system, but the overall accuracy becomes lower when noise becomes larger. This could explain the difference in precision obtained for our systems.

In the last test case, we simulate the logs generated by the three systems by tuning parameters so that the files have the same characteristics as the logs described in the previous section and that the results present the same precision for each system. We computed the recall for each of them, and the values are much higher than the ones found in the previous section: 0.85 for BlueGene/L; 0.78 for LANL; and 0.71 for Mercury.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented a new approach for mining correlations among different events generated by large-scale HPC systems, correlations that are used for predicting the output of the system. Our method is able to extract correlation between events regardless of the time delay between them and is adapted to each event type behaviour. We show that updating the sequences is important when dealing with HPC systems, and propose a method for modifying the correlations over time so that the correlations extracted reflect the system nature at any given moment. Also, we analyse the events at a finer granularity than any other related research without loosing prediction accuracy.

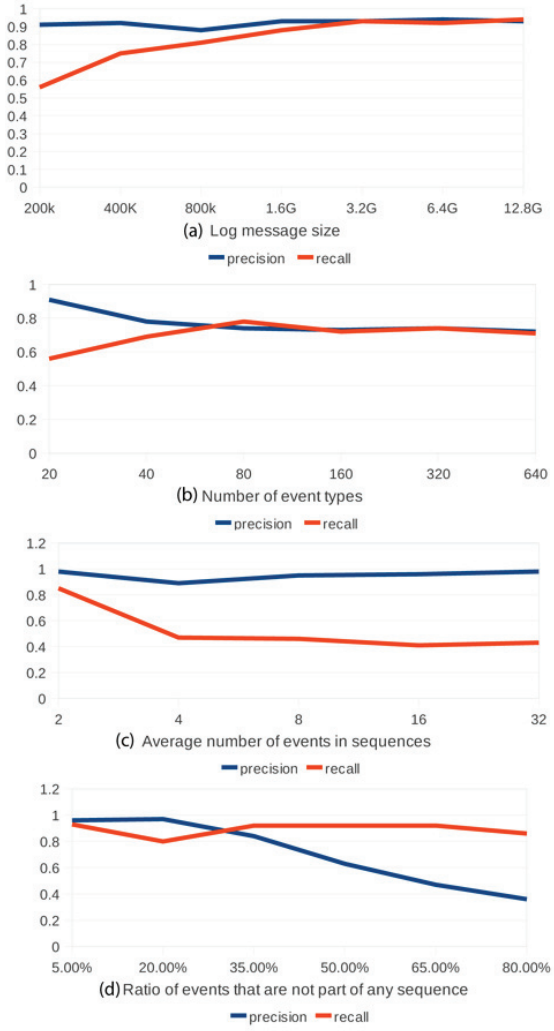

Figure 10: Recall for each event type

We made experiments on logs collected from three production HPC systems. We also implemented a log generator that allows us to better understand what characteristics of the system influence the precision and recall of our extracted sequences and to what extent. The precision and recall obtained are higher to the ones presented by similar work in this field, offering a higher level of event detail. For future work we plan to better understand the behaviour of correlations in real systems and to investigate different structure where the sequences and confidence values can be stored in a more efficient way.

6. ACKNOWLEDGEMENTS

This work was supported in part by the DoE 9J-30281-0008A grant, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

7. REFERENCES

[1] Y. Liang: BlueGene/L Failure Analysis and Prediction Models. Proceedings of the International Conference on Dependable Systems and Networks, 2006

[2] N. Taerat et al: BlueGene/L Log Analysis and Time to Interrupt Estimation. International Conference on Availability, Reliability and Security, pp.173-180, 2009

[3] R. K. Sahoo et al: Critical Event Prediction for Proactive Management In Large-scale Computer Clusters. International conference on Knowledge discovery and data mining, pp 426-435, 2003

[4] N. Yigitbasi et al: Analysis and Modeling of Time-Correlated Failures in Large-Scale Distributed Systems. IEEE/ACM International Conference on Grid Computing, pp 65-72, 2010

[5] J. G. Lou et al: Mining Dependency in Distributed Systems through Unstructured Logs Analysis ACM SIGOPS Volume 44 Issue 1, January 2010

[6] W. Xu et al: Online System Problem Detection by Mining Patterns of Console Logs IEEE International Conference on Data Mining, pp 588-597, 2009

[7] Z. Zheng et al: A Practical Failure Prediction with Location and Lead Time for Blue Gene/P International Conference on Dependable Systems and Networks Workshops, pp 15-222010

[8] J. Gu et al: Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A case Study International Conference on Parallel Processing, pp 157-164, 2008

[9] E. Chuah et al: Diagnosing the Root-Cause of Failures from Cluster Log Files International Conference on High Performance Computing, pp 1-10, 2010

[10] R. Ren et al: LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems CoRR abs/1003.0951, 2010

[11] N. Nakka et al: Predicting Node Failure in High Performance Computing Systems from Failure and Usage Logs IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems, 2011

[12] Ana Gainaru, Franck Cappello et al: Event log mining tool for large scale HPC systems. Euro-Par conference, Sep 2011.

[13] N. Taerat and all: Blue gene/l log analysis and time to interrupt estimation. International Conference on Availability, Reliability and Security, pp 173-180 (2009).

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

[15] D. Kerbyson et al: Use of Predictive Performance Modeling During Large-scale Systems Installation International Workshop on Hardware/Software Support for Parallel and Distributed Scientific and Engineering Computing, 2002.

[16] R.A. Oldfield et al: Modeling the impact of checkpoints on next-generation systems. IEEE Conference on Mass Storage Systems and Technologies, pp 30-46, Sept, 2007.

[17] E. Elnozahy and J Plank: Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. Dependable and Secure Computing, pp 97-108, Apr, 2004.

[18] Adam Oliner et al: What Supercomputers Say: A Study of Five System Logs. International Conference on Dependable Systems and Networks, June, 2007

[19] Eric Heien, Derrick Kondo, Ana Gainaru, Franck Cappello: Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems - Supercomputing 2011, November 2011

[20] B. Schroeder et al: A large-scale study of failures in high-performance computing systems. IEEE DSN, pages 249-258, June 2006

[21] A. Gara et al: Overview of the blue gene/l system architecture. IBM Journal of Research and Development, 49(2):195-212, March 2005.

[22] National Center for Supercomputing Applications at the University of Illinois. www.ncsa.illinois.edu. Accessed on 2010.

[23] Y. Li and Z. Lan: Using Adaptive Fault Tolerance to Improve Application Robustness on the TeraGrid TeraGrid, 2007.

[24] Y. Tan, X. Gu et al: Adaptive Runtime Anomaly Prediction for Dynamic Hosting Infrastructures Symposium on Principles of Distributed Computing, 2010.