2012 31st International Symposium on Reliable Distributed Systems

LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems

Xiaoyu Fu*,† Rui Ren*, Jianfeng Zhan*, Wei Zhou*, Zhen Jia*,† Gang Lu*,†
* State Key Laboratory Computer Architecture,Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences
Beijing, China
zhanjianfeng@ict.ac.cn

***Abstract*—This paper presents a set of innovative algorithms and a system, named *LogMaster*, for mining correlations of events that have multiple attributions, i.e., node ID, application ID, event type, and event severity, in logs of large-scale cloud and HPC systems. Different from traditional transactional data, e.g., supermarket purchases, system logs have their unique characteristics, and hence we propose several innovative approaches to mining their correlations. We parse logs into an n-ary sequence where each event is identified by an informative nine-tuple. We propose a set of enhanced apriori-like algorithms for improving sequence mining efficiency; we propose an innovative abstraction—event correlation graphs (ECGs) to represent event correlations, and present an ECGs-based algorithm for fast predicting events. The experimental results on three logs of production cloud and HPC systems, varying from 433490 entries to 4747963 entries, show that our method can predict failures with a high precision and an acceptable recall rates.**

***Keywords*—reliability; correlation mining; failure prediction; large-scale systems;**

I. Introduction

Cluster systems are common platforms for both high performance computing and cloud computing. As the scales of cluster systems increase, failures become normal [10]. It has been long recognized that (failure) events are correlated instead of independent. As early as 1992, Tang *et al.* [19] concluded that the impact of correlated failures on dependability is significant. Though we can not infer causalities among different events without invasive approaches like request tracing [17], we indeed can find out correlations among different events through a data mining approach. This paper focuses on mining *recurring event sequences with timing orders that have correlations*, which we call *event rules*. As an intuition, if there is an event rule that identifies the correlation of several warning events and a fatal event, the occurrence of warning events indicates a failure may happen in a near future, so mining event rules of all types is the basis for predicting failures.

Although considerable work on exploring event correlations from system logs has been done either by statistical model-based methods or some data mining techniques, the model-based methods are not practical for an increasing complexity of HPC systems [6], and most current data mining approaches just focus on several specific failure patterns which cannot be widely used. Effective failure prediction requires striking a balance between accuracy and more details of fine grain information, such as failure type, node location, related applications, when to happen, and etc. A failure event has many important attributions (i.e., node ID, application ID, event type, and event severity), which can not be omitted when predicting failures. For example, if one predict a failure without node information, the administrator can not take an appropriate action. Moreover, the time to predicted failures is also crucial for practical prediction use. Failure prediction often contains a training (knowledge mining) phase and a predicting phase. Previous related work has a high time complexity training phase and a complex machine learning based predicting phase, which is impractical for production use.

System logs are different from transactional data in three aspects. First, for event data, the timing order plays an important role. Moreover, a predicted event sequence without the timing constraints provides little information for a failure prediction, so we have to consider time between event occurrences rather than just their occurrence [14]; Second, logs are temporal. Only on the condition that events fall within a time window, we can consider those events may have correlations. Third, different events may happen interleavedly, and hence it is difficult to define a metrics for event correlation. For example, it is difficult to define the event correlation between two events A and B in an event sequence $BACBBA$, since A and B happen interleavedly. Consequently, mining correlations of multi-attribution events in system logs of large-scale cluster systems has its unique requirements.

To address the above problems, in this paper, we present a methodology to mine correlations of events and fast predict failures. Our contributions are threefold: first, taking into account the unique characteristic of logs, we propose an algorithm, named *Apriori-LIS*, to mine event rules, where a new operation is added to the classic Apriori algorithm and a simple metric of measuring event correlations is proposed. We also improve *Apriori-LIS* to significantly save mining time based on our findings that most of event rules are composed of events from same nodes or same applications, or they have same event types. Second, we design an innovative abstraction–*events correlation graphs* (ECGs), to represent event rules, and present an ECGs-based algorithm for event prediction. Compared with other prediction model, our method can save time by storing and updating event rules easier without re-training. Third, we validate our approaches on the logs of the

1060-9857/12 $26.00 © 2012 IEEE
DOI 10.1109/SRDS.2012.40

Bluegene/L system [12], a 260-node Hadoop cluster system at Research Institution of China Mobile and a production HPC cluster system—Machine 20 of 256 nodes at Los Alamos National Lab, the last two of which we call the Hadoop logs and the HPC cluster logs, respectively. The average precision rates are high as 83.66%, 81.19% and 79.82%, respectively. We also confirm that the correlations between non-failure and failure events are very important in predicting failures, which consists with the observation in [15] and contradict with other related works [8] [16] that only focused on failure events.

The rest of this paper is organized as follows. Section II explains basic concepts and metrics. Section III presents system design. Section IV gives out LogMaster implementation. Experimental results and evaluations on the Hadoop, the HPC cluster and the BlueGene/L logs are summarized in Section V. In Section VI, we describe the related work. We draw a conclusion and discuss the future work in Section VII.

II. Basic Concepts and Metrics

Definition 1: Event: an event is described by nine-tuples *(timestamp, log ID, node ID, event ID, severity degree, event type, application name, process ID, user ID)*.
Severity degrees include five levels: *INFO, WARNING, ERROR, FAILURE, FAULT,* and event types include *HARDWARE, SYSTEM, APPLICATION, FILESYSTEM, and NETWORK*. An event's attributions including *timestamp, node id, application name, process id and user name* are easily obtained. For an upcoming event, if a new 2-tuples ($severity\ degree$, $event\ type$) is reported, our system will generate and assign a new $event\ ID$ to this event. Similarly, if a new 4-tuples $(node\ ID, event\ ID, application\ name, process\ ID)$ is reported, we will create and assign a new $log\ ID$ to this event, and hence a $log\ ID$ will contain rich information including severity degree, event type, node ID, application name, and process ID.

Element item	Description
timestamp	The occurrence time associated with the event
severity degree	Include five levels: INFO, WARNING, ERROR, FAILURE, FATAL
event type	Include HARDWARE, SYSTEM, APPLICATION, FILESYSTEM, and NETWORK
event ID	An event ID is a mapping function of a 2-tuple (severity degree, event type).
node ID	The location of the event
application name	The name of the application that associates with the event
process ID	The ID of the process that associated with the event
log ID	A log ID is a mapping function of a 4-tuple (node ID, event ID, application name, process ID).
user	The user that associated the event

TABLE I: The descriptions of each element in nine-tuples

Definition 2: $n-ary$ ***Log ID Sequence (LIS)***: an $n-ary$ LIS is a sequences of n events *that have different log IDs* in a chronological order. An $n-ary$ LIS is composed of an $(n-1)-ary$ LIS ($n \in N+, n>1$) and an event of log ID X ($X \notin (n-1)-ary \quad LIS$) with the presumed timing constraint that an event of log ID X follows the $(n-1)-ary$ LIS. We call the $(n-1)-ary$ LIS is the *preceding events* and the event of log ID X is the *posterior event*.

For example, for a $3-ary$ LIS (A,B,C), (A,B) are the *preceding events*, while C is the *posterior event*. In this paper, we simply use an event X instead of an event of log ID X.

Definition 3: subset of LIS : If the event elements of an $m-ary$ LIS are a subset of the event elements of an $n-ary$ LIS ($m<n$), meanwhile the timing constraints of the $m-ary$ LIS do not violate the timing constraints of the $n-ary$ LIS, we call the $m-ary$ LIS is the subset of the $n-ary$ LIS.

For example, for a $3-ary$ LIS (A,B,C), its $2-ary$ subsets include (A,B), (A,C) and (B,C). However, (B,A) is not its $2-ary$ subset, since it violated the timing constraints of (A,B,C).

Definition 4: $(n-1)-ary$ *adjacent subset of* $n-ary$ *LIS*: For an $(n-1)-ary$ LIS that is the subset of an $n-ary$ LIS, if all adjacent events of $(n-1)-ary$ LIS are also adjacent in the $n-ary$ LIS, we call this $(n-1)-ary$ LIS the adjacent subset of the $n-ary$ LIS.

For an $n-ary$ LIS $(a1,a2,\ldots,a(n-1),a(n))$, it has two $(n-1)-ary$ adjacent subsets: $(a1,a2,\ldots,a(n-1))$ and $(a2,\cdots,a(n-1),a(n))$.

Definition 5: the support count and *the posterior count*: The support count is the recurring times of the preceding events which are followed by the posterior event, while the posterior count is the recurring times of the posterior event which follows the preceding events.

For example, if an event sequence $BACBBA$ occurs in a time window, for a $2-ary$ LIS (A,B), the support count is one, and the posterior count is two; for a $3-ary$ LIS (A,C,B), the support count is one and the posterior count is two.

Definition 6: confidence: The confidence metric defined by Equation 1 is calculated to measure the event correlation.

$$Confidence = \frac{support\ count(LIS)}{posterior\ count(LIS)} \tag{1}$$

According to Equation 1, in $BACBBA$, the confidence of an $2-ary$ LIS (A,B) is 1/2. In other words, if an event A occurs, an event B will occur with the probability of 50%. Please note that our confidence metric is different from the classic one that produces association rules, since log entries represented by event sequences with timing orders are significantly different from the transaction data mentioned in Section I.

Definition 7: frequent LIS and *event rule*: for an LIS, if its adjacent subsets are frequent and its support count exceeds a predefined threshold, we call it a frequent LIS. For a frequent LIS, if its confidence exceeds a predefined threshold, we call it an event rule.

In our approches, a new method is proposed to mining frequent LIS and generate event rules, which is described in Section III-C1.

III. System Design

We introduce the LogMaster architecture in Section III-A. Log preprocessing and filtering method is described in Section III-B. In Section III-C, we will propose two event correlation mining approaches. We also design an innovative abstraction to represent event rules in Section III-D. And an ECGs-based algorithm for event prediction is presented in Section III-E.

A. LogMaster architecture

LogMaster includes three major components: *Log agent*, *Log server*, and *Log database*. On each node, *Log agent* collects, preprocesses logs, and filters repeated events and periodic events. And then *Log agent* sends events to *Log server* for mining event rules, which are stored in *Log database*. At the same time, *Log server* constructs a set of graphs - *event correlation graphs* (ECGs) to represent event correlations. Finally, *Log server* predict events or failures based on ECGs.

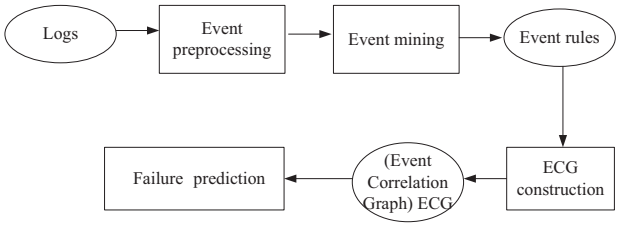

Fig. 1: The summary of event correlation mining approaches.

B. Log preprocessing and filtering

Since event logs are collected from different sources, they often have different formats: text files, databases, or special file formats generated by programs. In log preprocessing, LogMaster will parse logs with different log formats into a sequence of *Events* where each event is identified by an informative nine-tuples. After formatting event logs, we remove repeated events and periodic events. Note that spatial compression across physically close node locations is conducted to remove similar entries generated by parallel applications in other system log preprocessing systems [9] [18] [3]. This, however, might introduce high information loss and hide some important correlations of events from multiple notes.

We perform simple filtering according to two observations: (1) *Repeated events*. There are two types of repeated events: one is events recorded by different subsystems and the other is events repeatedly occurring in a short time window. For example, a switch failure may cause a series of network errors in a short time before we repair it. (2) *Periodic events*. Some events periodically occur with a fixed interval because of hardware or software bugs. Each type of periodic events may have two or more fixed cycles. For example, if a daemon in a node monitors CPU or memory systems periodically, it may produce large amount of periodic events. We only keep one event log for periodic events with the same fixed cycle. For periodic events, only events deviated from the fixed cycle are reserved.

We use a simple statistical algorithm and a simple clustering algorithm to remove repeated events and periodic events. The solution to removing repeated events is as follows: we calculate the interval between the current event and its preceding event that has the same log ID. If the current event has an interval less than a predefined value, we will treat the current event as a repeated event and remove it. The solution to removing periodic events is as follows: for each log ID, update the counts of events for different intervals. For periodic events with the same log ID, if the count and the percent (which is obtained against all periodic events with the same log ID) of events with the same interval is respectively higher than the predefined threshold values, we consider the interval as a fixed cycle. Lastly, we only keep one event log for periodic events with the same fixed cycle. In this step, we need to determine the two thresholds: the threshold of periodic count and the threshold of periodic ratio, which will affect the number of remaining events and log compression rate.

C. Event correlation mining algorithms

In this section, we propose two event correlation mining algorithms: an Apriori-LIS algorithm and its improved version—Apriori-simiLIS. As shown in Fig.2, we analyze the whole log history to generate event rules. In our approach, we use a sliding time window to analyze logs. For each current event, we save events within a sliding time window (according to timestamps) to the log buffer, and analyze events in the log buffer to mine event rules. After an event log has been analyzed, we will advance the sliding time window according to the timestamp of the current event. Fig. 1 summarizes our event correlation mining approaches.

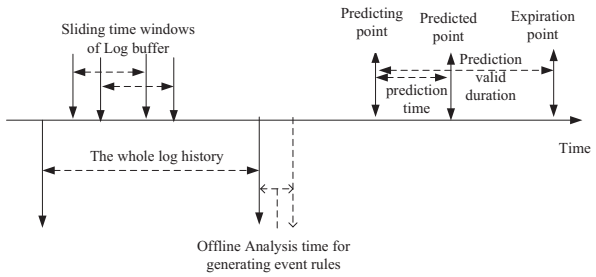

Fig. 2: The time relations in our event correlation mining and event prediction systems.

The notations later used in our algorithms are listed in TABLE. II.

1) Apriori-LIS algorithm: In data mining approaches, Apriori is a classic algorithm for mining frequent itemsets from transactional data [20] [21], which use Apriori property (any subset of frequent itemset must be frequent). Log entries represented by event sequences with timing orders are significantly different from transaction data as mentioned in Section I, and

Notation	Description
Tw	the size of sliding time window
Sth	the threshold of the support counts of LIS
Cth	the threshold of confidence of event rules
C(k)	a set of frequent k-ary LIS candidates
F(k)	a set of frequent k-ary LIS
R(k)	a set of k-ary event rules

TABLE II: Notations of the Apriori-LIS and Apriori-simiLIS

the Apriori property does not hold here. Different from Apriori that generates item set candidates of a length k from all item sets of a length $k-1$, we proposed a simplified algorithm that generates a $n-ary$ event rule candidate if and only if its two $(n-1)-ary$ adjacent subsets are frequent, and then we define a new operation— a LINK operation as follows:

For two frequent (k-1)-ary LIS, if the last (k-2) log IDs of one (k-1)-ary LIS is same like the first (k-2) log IDs of the other (k-1)-ary LIS, the result of the LINK operation is: $LINK((a1, a2, \cdots, a(k-1)), (a2, \cdots, a(k-1), a(k)) = (a1, a2, \cdots, a(k-1), a(k))$. For example, if (A, B, C) and (B, C, D) are frequent $3-ary$ LIS in $F(3)$, then a $4-ary$ LIS (A, B, C, D) is a frequent $4-ary$ LIS candidate, which we will add into $C(4)$.

By computing the confidence metric defined in Section II, the Apriori-LIS algorithm refined by *the LINK operation* directly generate event rules on each frequent sequence pattern, which is described as follows:

Step 1: Predefine two threshold values Sth and Cth for the support count and the confidence, respectively.

Step 2: Add all events that have different log IDs with the support count above the threshold value Sth to $F(k=1)$;

Step 3: $K = k+1$; $C(k) = \{\}$; $F(k) = \{\}$; $R(k) = \{\}$;

Step 4: Get all frequent $k-ary$ LIS candidates.We generate the frequent $k-ary$ LIS candidate by the *LINK operation* of two frequent $(k-1)-ary$ adjacent subsets, and add it into $C(k)$.

Step 5: Scan the logs to validate each frequent $k-ary$ LIS candidates in $C(k)$. The support count and posterior count of each $k-ary$ LIS candidate in $C(k)$ is counted. For a frequent $k-ary$ LIS candidate $(a1, a2, \cdots, a(k-1), a(k))$, if event $a(k-1)$ occurs after any event in $(a1, a2, \cdots, a(k-2))$ and before the posterior event m times, and the posterior event occurs after any event in $(a1, a2, \cdots, a(k-2), a(k-1))$ n times, we increase the support count and the posterior count of the $k-ary$ LIS candidate by m and n, respectively.

Step 6: Generate frequent $k-ary$ LIS and $k-ary$ event rules. For a $k-ary$ frequent LIS candidate, if its support count is all above the threshold Sth, add it into $F(k)$. For a $k-ary$ LIS candidate, if its support count and confidence are above the threshold values: Sth and Cth, respectively, add it into $R(k)$.

Step 7: Loop until all frequent LIS and event rules are found, and save them in Log database. If $R(k)$ is not null, save $k-ary$ event rules in $R(k)$. If $F(k)$ is not null, go to *step 3*; else end the algorithm.

2) Apriori-simiLIS algorithm: From large-scale experiments, we found the *Apriori-LIS* algorithm suffers from inefficiency, and generates a large amount of frequent LIS candidates, which may lead to a long analysis time. In order to improve efficiency, save costs and ensure the algorithm's performance, we observe the breakdown of event rules respectively mined by *Apriori-LIS* from ten days's logs of each system in TABLE III.

Description		All	Same nodes	Same events	Same applications
Hadoop	*count*	517	168	172	159
	Percent	100%	32.5%	33.3%	30.8%
HPC cluster	*count*	156	10	48	52
	Percent	100%	6.4%	30.8%	33.3%
BlueGene/L	*count*	1221	318	466	361
	Percent	100%	26.0%	38.2%	29.6%

TABLE III: The breakdown of event rules on three logs.

We notice that most of event rules (about 96.3%) from the hadoop logs are composed of events that occur on the same nodes or the same applications, or have the same event types. Similar observations are found in the HPC cluster and BlueGene/L logs. This phenomenon is probably due to: (a) error may spread in a single node: one application or process error can lead to another application or process error; (b) replicated applications in multiple nodes may have same errors or software bugs, and same failure events may appear in multiple nodes; (c) nodes in a large-scale system need to transfer data and communicate with each other, so a failure on one node may cause failures of same event types on other nodes; (d) a failure on one node may change the cluster system environment, which may cause failures of same event types on other nodes.

On the basis of these observations, we propose an improved version of the *Apriori-LIS* algorithm: *Apriori-simiLIS*. The distinguished difference between *Apriori-simiLIS* and *Apriori-LIS* is that the former uses an event filtering policy before the event correlation mining. The event filtering policy is described as below: to reduce the number of the analyzed events and decrease the analysis time, we only analyze correlations of events that occur in (a) *the same nodes* or (b) *the same applications*, or have (c) *the same event types*.

The *Apriori-simiLIS* algorithm includes two rounds of analysis: a single-node analysis and a multiple-node analysis. In the single-node analysis, we use the *Apriori-LIS* algorithm to mine event rules that have same $node\ IDs$. And in the multiple-node analysis, we use the *Apriori-LIS* algorithm to mine event rules that are of the same application names or the same event types but with different $node\ IDs$.

D. ECGs construction

After two rounds of analysis, we get a series of event rules. Based on the event rules, we propose a new abstraction—event correlation graphs (ECGs) to represent event rules.

An ECG is a directed acyclic graph (DAG). A vertex in an ECG represents an event. For a vertex, its child vertexes

are its posterior events in event rules. There are two types of vertexes: dominant and recessive. For a $2-ary$ event rule, such as (A, B), the vertexes representing events A, B are dominant vertexes. An additional vertex $A \wedge B$ represents the case that A and B occurred and B occurred after A. The vertex $A \wedge B$ is a recessive vertex.

Vertexes are linked by edges. An edge represents the correlation of two events linked by the edge. Each edge in ECG has five attributes: *head vertex*, *tail vertex*, *support count*, *posterior count* and *edge type*. Similar to vertexes, edges have two types: *dominant* and *recessive*. If two vertexes linked by an edge are dominant, the edge type is dominant; otherwise, the edge type is recessive. For a $2-ary$ event rule (A, B), the head vertex is B, and the tail vertex is A; the support count and posterior count of the edge is the support count and posterior count of the rule event (A, B), respectively. For a $3-ary$ event rule (A, B, C), the head vertex is C, and the tail vertex is the recessive vertex $A \wedge B$; the support count and the posterior count of the edge is the support count and the posterior count of the event rule (A, B, C), respectively.

An example is shown in Fig. 3. There are three $2-ary$ event rules: (A, B), (B, C) and (C, D), and two $3-ary$ (A, B, C) and (A, B, D). We generate four dominant vertexes, which represent events A, B, C, and D. We also generate three dominant edges, which represent the event rules (A, B) and (B, C) and (C, D). In additional, we also generate a recessive vertex $(A \wedge B)$. A recessive edge $(A \wedge B \rightarrow C)$ represents the event rule (A, B, C), and a recessive edge $(A \wedge B \rightarrow D)$ represents the event rule (A, B, D). The additional vertex $A \wedge B$ is the child of both A and B.

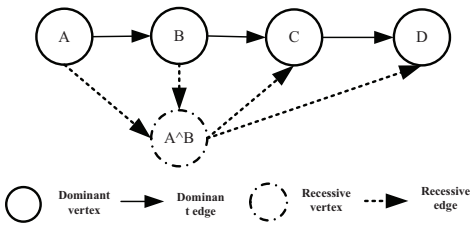

Fig. 3: An example of ECG.

The construction of ECGs includes three main steps:

Step 1: Construct a group of ECGs based on event rules found in the single-node analysis. Each ECG represent correlations of events in one node.

Based on generated event rules, the vertexes and edges of ECGs are created. For each event rule, dominant vertexes and recessive vertexes are generated, and edges between vertexes are created too.

Step 2: ECGs that represent correlations of events on multiple nodes are constructed based on event rules found in multiple-nodes analysis.

Step 3: The index of ECGs is created, and the positions of events in ECGs are also saved. We can locate events by using these indexes.

The ECG ID and the ECG entrance vertex are the index of the ECGs. The ECG position is the index of each event in an ECG. So it is convenient to locate events in the ECGs.

After these three steps, a series of ECGs that describe the correlation of events are constructed. Choosing the ECGs abstraction has three reasons: first, the visualized graphs are easy to understand by system managers and operators; second, the abstraction facilitates modeling sophisticated correlations of events, such as $k-ary$ $(k > 2)$ event rules; third, ECGs can be easily updated in time since the attributes of edges and vertexes are easily updated when a new event comes.

E. Event Prediction

For each prediction, there are three important timing points: *predicting point*, *predicted point*, and *expiration point*, which is similar to Salfner's way in [25]. The relations of those timing points are shown in Fig. 2. The prediction system begins predicting events at the timing of the predicting point. The predicted point is the occurrence timing of the predicted event. The expiration point refers to the expiration time of a prediction, which means this prediction is not valid if the actual occurrence timing of the event passed the expiration point.

Consequently, there are two important derived properties for each prediction: *prediction time*, and *prediction valid duration*. The *prediction time* is the time difference between the predicting point and the predicted point, which is the time span left for system administrators to respond with the possible upcoming failures. The *prediction valid duration* is the time difference between the predicting point and the expiration point, which can be predefined by system administrators to meet their expectation. We also have to define the *prediction probability threshold* to give warnings if the probability of predicted events over the threshold.

The event prediction algorithm based on ECGs is described as follows:

Step 1: Define the prediction probability threshold Pth, and the prediction valid duration Tp.

Step 2: When an event comes, the indexes of events are searched to find matching ECGs and the corresponding vertexes in ECGs. The searched vertexes are marked. For a recessive edge, if its tail vertex is marked, the recessive edge is marked, too. For a recessive head vertex, if all recessive edges are marked, the recessive vertex is also marked. We mark vertexes so as to predict events; and the head vertexes that are dominant vertexes are searched according to the edges (both dominant and recessive edge types) linked with marked vertexes.

Step 3: The probabilities of the head vertexes are calculated according to the attributions of vertexes that are marked and their adjacent edges in the ECGs. For a head vertex, we calculate its probability as the probability of tail vertex times the confidence of the edge. If a head vertex is linked with two marked vertexes with different edges, we will calculate two probabilities, and we choose the largest one as the probability of the head vertex.

An example is shown in Fig. 4. When an event A occurs, the vertex A and the edge $A \rightarrow A \wedge B$ are marked. We can calculate the probability of B according to the confidence of the edge $A \rightarrow B$. The probability of event C also can be calculated by the dominant edges $A \rightarrow B$ and $B \rightarrow C$. If the probability of event B or event C is above the prediction probability threshold Pth, it is predicted. So does the event D.

$$Probability(B) = confidence(A \rightarrow B) \tag{2}$$

$$\begin{aligned} &Probability(C) \\ &= probability(B) * confidence(B \rightarrow C) \\ &= confidence(A \rightarrow B) * confidence(B \rightarrow C) \end{aligned} \tag{3}$$

When an event B occurs later, the vertex B and the edge $B \rightarrow A \wedge B$ are marked. Because the edge $A \rightarrow A \wedge B$ and $B \rightarrow A \wedge B$ are both marked, so the vertex $A \wedge B$ is marked too. The probability of events C and D are also calculated according to the new edges.

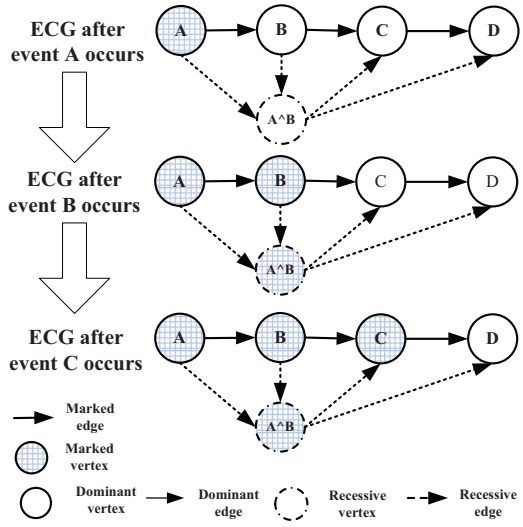

Fig. 4: An example of event prediction.

Step 4: If the probability of a head vertex is above the prediction probability threshold Pth, then the head vertex is the predicted event.

Step 5: Loop *Step 2*.

IV. LogMaster Implementation

As mentioned in Section III-A, We have implemented an event correlation mining system and an event prediction system. The main modules of LogMaster are described as follows:

(1) The configuration file and Log database. A XML-format configuration file named config.xml includes the regular expressions of important definitions, formats, and keywords that are used to parse logs and the configuration of Log database.

(2) The Python scripts are used to parse system log files according to the regular expressions defined in the configuration file. The parsed logs are saved to the table of the formatted logs in Log database.

An example of the original event logs and the corresponding formatted event logs is shown as bellows:

[**Original event log**] Oct 26 04:04:20 compute-3-9.local smartd [3019]: Device: /dev/sdd, FAILED SMART self-check. BACK UP DATA NOW!

[**Formatted event log**] timestamp =”20081024040420”, node name =”compute-3-9.local”, format =”format1”, keyword = ”FAILED SMART”, application=”smartd”, process=”NULL”, process id=”3019”, user=”NULL”, description=”Device: /dev/sdd, FAILED SMART self-check. BACK UP DATA NOW!”.

(3) Log server is written in Java. The simple filtering operations are performed to remove repeated events and periodic events. The filtered logs are saved in the table of filtered logs in Log database. We implemented both Apriori-LIS and Apriori-semiLIS algorithms. The attributions of each event rules, including event attributions, support count, posterior count, and confidence are saved to the table of event rules in Log database.

(4) We implement the event rules based prediction system in C++.

V. Experiments

In this section, we use LogMaster to analyze three real logs generated by a production Hadoop system, a HPC cluster system and a BlueGene/L system. The details of the first two logs are described in Section V-A and the BlueGene/L system [12] is a baseline since much work is performed on it. The server we used has two Intel Xeon Quad-Core E5405 2.0GHZ processors, 137GB disk, and 8G memory.

A. Background of three real system logs

The 260-node Hadoop cluster system is used to run MapReduce-like cloud applications, including 10 management nodes, which are used to analyze logs or manage system, and 250 data nodes, which are used to run Hadoop applications. We collect the Hadoop logs from /dev/error, /var/log, IBM Tivoli, HP openview, and NetLogger, then store them on the management nodes. So these logs include system services and kernel logs, such as crond, mountd, rpc.statd, sshd, syslogd, xinetd, and so on. The HPC cluster logs are available from (http://institutes.lanl.gov/data/fdata/). TABLE IV give the summary of system logs from the Hadoop, HPC cluster and BlueGene/L.

B. Evaluation metrics

(1) *Average analysis time*

As shown in Fig.5, the analysis time is the time difference between the beginning and ending timing points of event correlation mining. In this experiment, we use the *average analysis time* per event to evaluate the computational complexity of

Properties	Logs		
	Hadoop	*HPC cluster*	*BlueGebe/L*
Size (MB)	130	31.5	118
Records count	977858	433490	4399503
Days	67	1005	215
Start Date	2008-10-26	2003-07-31	2005-06-03
End Date	2008-12-31	2006-04-04	2007-01-02

TABLE IV: The summary of logs.

the correlation mining algorithms. The *average analysis time* is defined by Equation 4.

$$average\ analysis\ time\ = \frac{\text{total analysis time}}{\text{the count of filtered events}} \tag{4}$$

(2) *Average prediction time*

As described before, the prediction time is the time difference between the predicting point and the predicted point. In this experiment, we use the a*verage prediction time* per event to evaluate the computational complexity of the failure predicting algorithms. The *average prediction time* per events is defined by Equation 5.

$$average\ prediction\ time\ = \frac{\sum(\text{the predicted point} - \text{the predicting point})}{\text{count of all predicted events}} \tag{5}$$

(3) *Precision* and *Recall*

In addition to predicting failure occurrence, our approach also predict fine grain information, like failure type, node and application location etc.. Hence the meaning of correct prediction may be different: only the prediction of failure with right fine grain information is considered to be correct. Similar to the one presented in [5], we refine the term of *the true positive (TP)*, *the false positive (FP)* and *the false negative (FN)* as follows: *The true positive (TP)* is the count of events which are correctly predicted. *The false positive (FP)* is the count of events which are predicted but not appeared in the prediction valid duration. *The false negative (FN)* is the count of events which are not predicted but appeared in the prediction valid duration.The precision rate is defined by Equation 6. and the recall rate is defined by Equation 7.

$$precision = \frac{TP}{TP + FP} \tag{6}$$

$$recall = \frac{TP}{TP + FN} \tag{7}$$

C. Events preprocessing and filtering

For preprocessing, our python scripts parse three raw event logs into structured LIS and save them into the Mysql database. Each event in event logs is transacted to a nine-tuples (*timestamp*, *log id*, *node id*, *event id*, *severity degree*, *event type*, *application name*, *process id*, *user*). Please note that we only select the node logs from the HPC cluster logs except for logs of "switch module", and 348460 warning message in the BlueGene/L logs is also omitted. After event preprocessing, 2878 log ids for Hadoop, 3665 for HPC cluster and 99183 for BlueGene/L are kept.

For filtering, we remove repeated events and periodic events. For all of the three logs, the time interval threshold of removing repeated events are set as 10 seconds and the periodic count threshold and the periodic ratio threshold of removing periodic events are set as (20, 0.2), (20, 0.1) and (20, 0.2), respectively. Note that threshold values are chosen after careful comparison of the effects of different threshold values on the number of filtered events. After event filtering, the compression rate of the Hadoop and BlueGene/L logs are above 90%, but the compression rate of the HPC cluster logs only achieves 69.4% as shown in TABLE V.

Results	Logs		
	Hadoop	*HPC cluster*	*BlueGebe/L*
No. of logid	2,878	3,665	99,183
No. of failure logid	543	275	15,523
Raw logs	977,858	433,490	4,747,963
Preprocessing	977,858	176,043	4,399,503
Removing repeated events	375,369	152,112	442,527
Removing periodic events	26,538	132,650	422,585
Compression. rate	97.29%	69.40%	91%

TABLE V: The results of preprocessing and filtering three logs.

D. Evaluation of two event rules mining algorithm

In this step, we choose the first two thirds of the three logs for mining event rules and the last one third for predicting failures respectively. Through comparisons with a large amount of experiments, we find that *the average analysis time* and *number of event rules* increase with the *sliding time window (Tw)*, but decrease with the threshold of *support count (Sth)* or the threshold of *confidence (Cth)*. We properly set the baseline parameters in LogMaster as [Tw60/Sth5/Cth0.25]. [Tw*x*/Sthy/Cth*z*] indicates that the sliding time window Tw is x minutes, the threshold of support count Sth is y, and the threshold of confidence Cth is z.

Fig 5 and Fig 6 show the experiment results of the average analysis time and the number of event rules by conducting *Apriori-LIS* and *Apriori-simiLIS* on the three logs, respectively. For the Hadoop logs, the *average analysis time* of *Apriori-simiLIS* is about 22% of that of *Apriori-LIS*, while *Apriori-simiLIS* obtains about 89% event rules of that of *Apriori-LIS*; For the HPC cluster logs, the *average analysis time* of *Apriori-simiLIS* is about 18% of that of *Apriori-LIS* algorithm, while *Apriori-simiLIS* obtains about 87% event rules of that of *Apriori-LIS*. For the BlueGene/L logs, the *average analysis time* of *Apriori-simiLIS* is about 15% of that of *Apriori-LIS* algorithm, while *Apriori-simiLIS* obtains about 95% event rules of that of *Apriori-LIS*. Apparently, the results are in accord with our expectations that *Apriori-simiLIS* can significantly

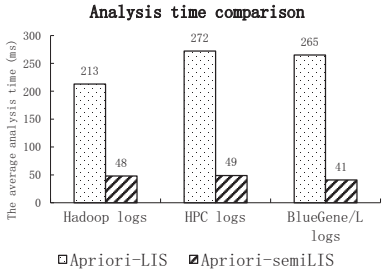

Fig. 5: The average analysis time of two algorithms on three logs.

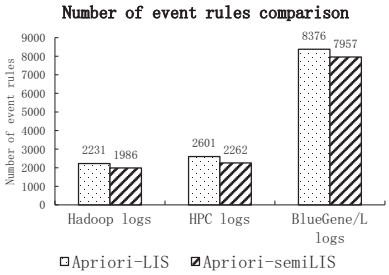

Fig. 6: The number of event rules from three logs by two algorithms.

improve the time efficiency of *Apriori-LIS* with small rule loss.

After obtaining all event rules and storing them in Mysql database, we analyse the breakdown of event rules and failure rules and obtain two findings as follows:

(a) Most of event rules are composed of events with the FILESYSTEM type in the Hadoop logs, and the reason may be that applications running on the Hadoop cluster system are data-intensive, and need to access the file system frequently; Most of event rules in the HPC cluster logs are composed of events of *"HARDWARE"* and *"SYSTEM"*types, and it is probable that hardware and system level errors more easily lead to failures in the HPC cluster system. Most of event rules are composed of events with the *"application"* type in the BlueGene/L logs, and the reason may be that much more jobs and applications are running on this super cluster.

(b) As shown in TABLE VI, failure rules that identify correlations between non-failure events (including *"INFO"*,*"WARNING"*, *"ERRO"*) and failure events (*"FATAL"* and *"FAILURE"*) dominate over the event rules that identify the correlations between different failure events, which indicate that non-failure events are very important to predict failures.

E. Evaluation of predication

After mining the event rules, we need to consider whether these event rules are suitable for predicting events. We evaluate our algorithms in three scenarios: (a) predicting all events on the basis of both failure and non-failure events; (b) predicting only failure events on the basis of both failure and non-failure events; (c) predicting failure events after removing non-failure events.

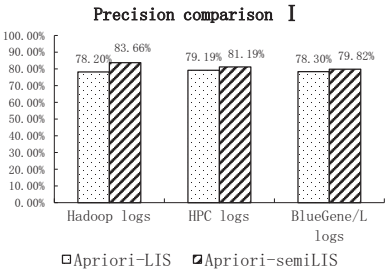

Fig. 7: The precision rate of predicting events on the basis of both failure and non-failure events.

There are two parameters that affect the prediction accuracy, including the *prediction probability threshold* (Pth) and the *prediction valid duration* (Tp). Through comparisons with large amount of experiments, we set the baseline parameters as [Tw60/Sth5/Cth0.25/Pth0.5/Tp60]. Please note that Tw, Sth, Cth just keep the same baseline parameters in Section V-D. [Pth*u*/Tp*v*] indicates that the *prediction probability threshold* Pth is u, and the *prediction valid duration* Tp is v minutes.

Firstly, on the basis of both failure and non-failure events, we predict all events, including *"INFO"*, *"WARNING"*, *"ERROR"*, *"FAILURE"*, and *"FATAL"* events. The precision rates and recall rates of predicting events in the Hadoop logs, the HPC cluster logs and the BlueGene/L logs are shown in Fig. 7 and Fig. 8, respectively. We notice that the precision rates of *Apriori-semiLIS* on three logs are 1%-4% higher than that of *Apriori-LIS* and the recall rates of *Apriori-semiLIS* on three logs are 1%-2% lowerer than that of *Apriori-LIS*. This is because we obtain more event rules by *Apriori-LIS*, which predicts more events that not happen. We also notice a relatively low recall rate compared with other works [2] [6]. The reason for this is that we keep a richer log information without spatial filtering.

In the second set of experiments, on the basis of all type events, we only predict failure events (*"Failure"* and *"FATAL"* types). The precision rates and the recall rates on three logs are shown in Fig. 9 and Fig. 10, respectively. The results are close to the first one since failure is a special type of event.

Liang *et al.* [10] suggest removing events with the types of *"INFO", "WARNING", and "ERROR"* in preprocessing

Logs	Non-failures→Failures		Failures→Failures	
	Apriori-LIS	*Apriori-simiLIS*	*Apriori-LIS*	*Apriori-simiLIS*
Hadoop	377	312	86	78
HPC cluster	97	291	26	26
BlueGene/L	532	524	123	109

TABLE VI: The failure event rules obtained from two algorithms on three logs

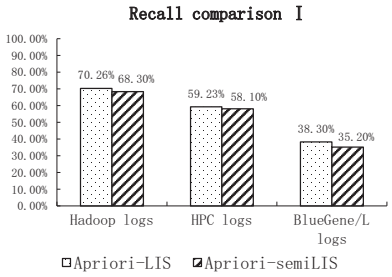

Fig. 8: The recall rate of predicting events on the basis of both failure and non-failure events.

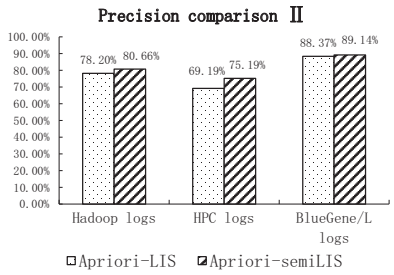

Fig. 9: The precision rate of predicting failures on the basis of both failure and non-failure events.

events. Following their idea, we remove non-failure events, and then predict failure events in the third set of experiments.

From Fig. 11 and Fig. 12, we can observe that, after removing non-failure events, the recall rates in predicting failures on three logs are 20%-26% lower than that without removing non-failure events. The reason is that the numbers of the event rules that identify correlations between non-failure events and failure events are higher than that of the event rules that identify the correlations between different failure events as shown in TABLE VI in Section V-D, which confirms that non-failure events are very important to predict failures in addition to failure events.

From the three set of experiments, we notice that the average recall rate of our approaches is lower than expected and the reasons may be as follows: (a) Some events are independent,

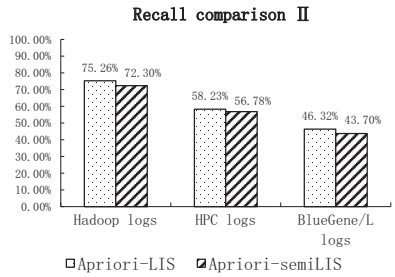

Fig. 10: The recall rate of predicting failures on the basis of both failure and non-failure events.

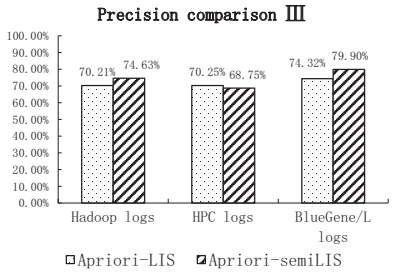

Fig. 11: The precision rate of predicting failures after removing non-failure events.

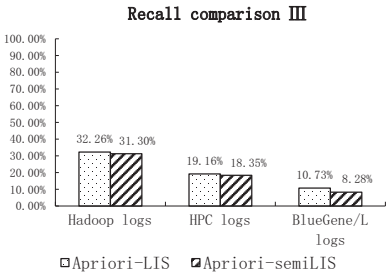

Fig. 12: The recall rate of predicting failures after removing non-failure events.

and they have not correlated events. (b) Because of the setting of the sliding time window, the support count and the posterior count threshold values, some weak-correlated or a long time-correlated event rules may be discarded. (c) The logs are not long enough to contain all failure patterns, which change over time.

With the same baseline configurations, the *average prediction time* of three logs is shown in TABLE VII, which indicates that system administrators or autonomic systems should have enough time to monitor and handle predicted events.

prediction time (minutes)	**Logs**					
	Hadoop		*HPC cluster*		*BlueGene/L*	
	A-L	*A-s*	*A-L*	*A-s*	*A-L*	*A-s*
Event prediction	15.75	15.01	23.57	23.34	30.01	29.58
Failure prediction	9.79	8.98	17.57	16.28	17.01	16.54
Failure prediction after removing non-failure	3.03	2.45	2.57	1.57	3.24	2.96

TABLE VII: The *average predicting time* by the two algorithms (A-L represent *Apriori-LIS* and A-s represent *Apriori-simiLIS*) on the three logs.

VI. Related Work

In the past decades, considerable work has been performed on proactive fault management, especially failure prediction. Based on the type of data used, Salfner *et al.* [25] divide failure prediction methods into two main categories: symptom monitoring way on analyzing periodically system variables and

detected error reporting way on dealing with various event logs. Li *et al.* [22] confirm that the latter is more suitable in large-scale systems than the former. There's also a significant amount of work [23] [24] on preprocessing and analyzing event logs in order to help fault tolerant.

In this paper, we only pay close attention to exploring event correlations and predicting failures on system logs generated by large cluster systems. Tang *et al.* [19] conclude that the impact of correlated failures on dependability is significant and the work in [1] has observed that there are strong spatial correlations between failure events. Schroeder *et al.* [13] find some simple failure characteristics by analyzing huge amount of failure data. Further, some work uses statistical analysis approach to model failure behaviors. For example, Liang *et al.* [15] investigate the statistical characteristics of failure events, as well as the statistics about the correlation between the occurrence of non-fatal and fatal events; Fu *et al.* [7] [8] develop a spherical covariance model with an adjustable timescale parameter to quantify the temporal correlation and a stochastic model to describe spatial correlation in order to cluster failure events, and later train Bayesian network for prediction; Yigitbasi *et al.* [4] analyze and model failures taking into account the time-varying behaviors.

However, the incompleteness of model-based methods are not practical for an increasing complexity of HPC systems [6] and hence some work uses data mining techniques to get frequent sequences as failure patterns. Gujrati *et al.* [9] present a meta-learning method based on statistical analysis and standard association rule algorithm, but only focus on some specific failure patterns and predict one of them will happen without fine-grain information. The work in [5] is generating event rules between fatal events and between fatal and non-fatal events to predict failures just including location and lead time. Inspired by our former technical report [11], Gainaru *et al.* [2] use an Apriori-modified algorithm by a dynamic time window to mine frequent LIS for each event type they defined, which can increase recall rate relatively because of the small amount of filtered events. The time window, however, can range up to hours, even days, which is not practical.

VII. Conclusion

In this paper, we designed and implemented an event correlation mining system and an event prediction system, LogMaster. With LogMaster, we parsed logs into event sequences where each event is represented as an informative nine-tuple. We proposed an algorithm, named *Apriori- LIS*, to mine event rules. We also improve *Apriori-LIS* to significantly save mining time based on our findings that most of event rules are composed of events from same nodes or same applications, or they have same event types. We designed an innovative abstraction—events correlation graphs (ECGs), to represent event rules, and presented an ECGs-based algorithm for event prediction. We validated our approaches on three logs of production cloud and HPC systems. The average precision rates are high as 83.66%, 81.19% and 79.82%, respectively. We also confirmed that the correlations between non-failure and failure events are very important in predicting failures, contradicting with the previous related work, which filters warning information in the early stage of log preprocessing and ignores their effects in predicting failures.

In near future, we will gain insight and extract useful information from the event rules mined from the logs by LogMaster, so as to increase the precision and recall rates in failure prediction.

Acknowledgment

We are very grateful to anonymous reviewers. This work is supported by the NSFC project (Grant No.60933003) and the Chinese 973 project (Grant No.2011CB302500).

References

[1] R. K. Sahoo *et al.* "Failure data analysis of a large-scale heterogeneous server environment". In *Proc. of DSN*, 2004.
[2] A. Gainaru *et al.* "Adaptive event prediction strategy with dynamic time window for large-scale HPC systems". In *Proc. of SLAML*, 2011.
[3] A. Gainaru *et al.* "Event log mining tool for large scale HPC systems". In *Proc. of Euro-Par*, 2011.
[4] N. Yigitbasi *et al.* "Analysis and modeling of time-correlated failures in large-scale distributed systems". In *Proc. of GRID*, 2010.
[5] Z. Zheng *et al.* "A practical failure prediction with location and lead time for Blue Gene/P". In *Proc. of DSN-W*, 2010.
[6] J. Gu *et al.* "Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study". In *Proc. of ICPP*, 2008.
[7] S. Fu *et al.* "Exploring Event Correlation for Failure Prediction in Coalitions of Clusters". In *Proc. of ICS*, 2007.
[8] S. Fu *et al.* "Quantifying Temporal and Spatial Correlation of Failure Events for Proactive Management". In *Proc. of SRDS*, 2007.
[9] P. Gujrati *et al.* "A Meta-Learning Failure Predictor for Blue Gene/L Systems". In *Proc. of ICPP*, 2007.
[10] Y. Liang *et al.* "Filtering Failure Logs for a BlueGene/L Prototype". In *Proc. of DSN*, 2005.
[11] R. Ren *et al.* "LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems". CoRR abs/1003.0951, 2011.
[12] The BlueGene/L Team, "An Overview of the BlueGene/L Supercomputer". In *Proc. of Supercomputing*, 2002.
[13] B. Schroeder *et al.* "A Large-Scale Study of Failures in High-Performance Computing Systems". In *Proc. of DSN*, 2006.
[14] J. L. Hellerstein *et al.* "Discovering actionable patterns in event data". In *IBM Systems Journal*, 41(3), 2002.
[15] Y. Liang *et al.* "BlueGene/L Failure Analysis and Prediction Models". In *Proc. of DSN*, 2006.
[16] Y. Liang *et al.* "Failure Prediction in IBM BlueGene/L Event Logs". In *Proc. of ICDM*, 2007.
[17] Z. Zhang *et al.* "Precise Request Tracing and Performance Debugging for Multi-tier Services of Black Boxes". In *Proc. of DSN*, 2009.
[18] Z. Zheng *et al.* "System Log Pre-processing to Improve Failure Prediction". In *Proc. of DSN*, 2009.
[19] D. Tang *et al.* "Analysis and Modeling of Correlated Failures in Multicomputer Systems". *IEEE Transactions on Computers*. 41, 5 (May. 1992), 567-577.
[20] R. Agrawal *et al.* "Fast Algorithms for Mining Association Rules", In *Proc. VLDB*, 1994.
[21] R. Agrawal *et al.* "Mining Association Rules Between Sets of Items in Large Databases," In *Proc. of SIGMODE*, 1993.
[22] Yu Li *et al.* "Practical online failure prediction for Blue Gene/P: Period-based vs Event-driven," In *Proc. of DSN-W*, 2011.
[23] N. Taerat *et al.* "Blue Gene/L Log Analysis and Time to Interrupt Estimation," In *Proc. of ARES*, 2009.
[24] C. Chen *et al.* "Log analytics for dependable enterprise telephony," In *Proc. of EDCC*, 2011.
[25] F. Salfner *et al.* "A survey of online failure prediction methods," In *ACM Comput. Surv.*,42(3): 1-42,March 2010.