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

Rui Ren*, Xiaoyu Fu*, Jianfeng Zhan*, ‡, Wei Zhou*
*Institute of Computing Technology, Chinese Academy of Sciences
‡ Corresponding author: zhanjianfeng@ict.ac.cn

***Abstract*—This paper presents a methodology 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 cluster systems. Different from traditional transactional data, e.g., supermarket purchases, system logs have their unique characteristic, and hence we propose several innovative approaches to mine their correlations. We present a simple metrics to measure correlations of events that may happen interleavedly. On the basis of the measurement of correlations, we propose two approaches to mine event correlations; meanwhile, we propose an innovative abstraction—event correlation graphs (ECGs) to represent event correlations, and present an ECGs-based algorithm for predicting events. For two system logs of a production Hadoop-based cloud computing system at Research Institution of China Mobile and a production HPC cluster system at Los Alamos National Lab (LANL), we evaluate our approaches 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.**

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 [6]. It has been long recognized that (failure) events are correlated, not independent. In an early year as 1992, Tang *et al.* [30] 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 [15], 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 warning events and a fatal event, the occurrence of warning events indicates that a failure may happen in a near future, so mining event rules is the basis for predicting failures.

Considerable work has been done in mining frequent itemset in transactional data, e.g., supermarket purchases, [52] [53]. However, system logs are different from transactional data in four 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 among event occurrences rather than just their occurrence [10]. Second, logs are temporal. Only on condition that events fall within a time window, we can consider those events may have correlations. Third, a failure event has many important attributions, i.e., node ID, application ID, event type, and event severity. For a failure prediction, those attributions are ingredient. For example, if you predict a failure without node information, an administrator can not take an appropriate action. Last, 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.

On the basis of Apriori-like algorithms [52] [53], several previous efforts propose new approaches for frequent itemset, event bursts, periodical events, mutual-dependent events [10], frequent episodes [57], sequential patterns[62] or closed sequential patterns [58] [59], however mining multi-attribution event rules in system logs of large-scale cluster systems has its unique requirements as mentioned above. For example, in [59], instead of mining the complete set of frequent subsequences, Yan *et al.* mine frequent closed subsequences only, i.e., those containing no super-sequence with the same occurrence frequency, which is difficult to directly apply in mining event rules for the purpose of predicting failures. Some work proposed Apriori-like algorithms in predicting failures [5] without providing details (multi-attribution) or rare events [2] [63], which are limited to the specified target events.

Our effort in this paper focuses on mining event rules in system logs. Taking into account the unique characteristic of logs, we propose a simple metrics to measure event correlations in a sliding time windows. Different from Apriori that generates item set candidates of a length k from all item sets of a length $k-1$ [52] [53], 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 hence we significantly decrease the time complexity. We validate our approaches on the logs of 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, which we call the Hadoop logs and the HPC logs, respectively. For predicting events in the Hadoop logs and the HPC logs, the precision rates are high as 78.20%, 81.19%, respectively. We also evaluate our approaches 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.

Our contributions are four-fold. First, we propose a simple metrics to measure event correlations. Second, on the basis of the measurement metrics, we propose two approaches (*Apriori-LES* and *Apriori-semiLES*) to mining event correlations. Third, we design an innovative abstraction–*events correlation graphs (ECGs)*, to represent event rules, and present an ECGs-based algorithm for event prediction. Fourth, for the first time, we compare the breakdown of events of different types and events rules in two typical cluster systems for cloud and HPC, respectively.

The rest of this paper is organized as follows. Section II explains basic concepts. Section III presents LogMaster design. Section IV gives out LogMaster implementation. Experiment results and evaluations on the Hadoop logs and the HPC 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. Background, and Basic Concepts

A. Background of two 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 by using /dev/error, /var/log, IBM Tivoli, HP openview, and NetLogger, then store them on the management nodes. So these logs include system service and kernel logs, such as crond, mountd, rpc.statd, sshd, syslogd, xinetd, and so on. The HPC logs are available from (http://institutes.lanl.gov/data/fdata/). TABLE I give the summary of system logs from the Hadoop and HPC cluster systems.

Log name	*Days*	*Start Date*	*End Date*	*Log Size*	*No. of Records*
Hadoop	*67*	*2008-10-26*	*2008-12-31*	*130 MB*	*977858*
HPC	*1005*	*2003-07-31*	*2006-04-40*	*31.5 MB*	*433490*

TABLE I: The summary of logs

We use nine-tuples *(timestamp, log ID, node ID, event ID, severity degree, event type, application name, process ID, user ID)* to describe each event. For an upcoming event, if a new 2-tuple ($severity\ degree$, $event\ type$) is reported, our system will generate and assign a new $event\,ID$ associated with this event. Similarly, if a new 4-tuple $(node\ ID, event\ ID, application\ name, process\ ID)$ is reported, we will create and assign a new $log\,ID$ associated with this event, and hence a $log\ ID$ will contain rich information, including severity degree, event type, node ID, application name, and process ID.

B. Basic concepts

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

For example, for a $3-ary$ LES (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 2: A subset of LES : If the event elements of an $m-ary$ LES are a subset of the event elements of an $n-ary$ LES ($m<n$), meanwhile the timing constraints of the $m-ary$ LES do not violate the timing constraints of the $n-ary$ ES, we call the $m-ary$ LES is the subset of the $n-ary$ LES.

For example, for a $3-ary$ LES (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 3: $(n-1)-ary$ *adjacent subset of* $n-ary$ *LES*: For an $(n-1)-ary$ LES that is the subset of an $n-ary$ LES, if all adjacent events of $(n-1)-ary$ LES are also adjacent in the $n-ary$ LES, we call the $(n-1)-ary$ LES the adjacent subset of the $n-ary$ LES.

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

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 II: The descriptions about the elements of nine-tuples

III. SYSTEM DESIGN

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.

In Section III-C, we will propose two event correlation mining approaches. At the same time, Log server constructs

a set of graphs - event correlation graphs (ECG) to represent event correlations. Section III-D will introduce the details of ECG. LogMaster will mine event rules and their presentations - ECG for other systems, for example a failure prediction or fault diagnose system. Fig. 1 summarizes our event correlation mining approaches.

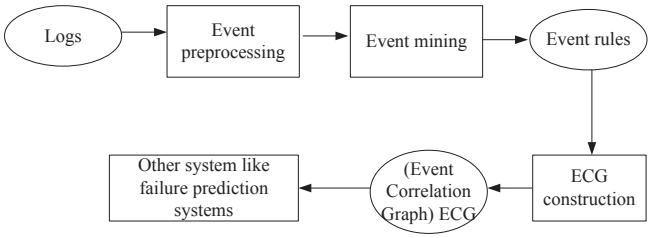

Fig. 1: The summary of event correlation mining approaches.

B. An metrics for measuring event correlations

In this subsection, we define a simple metrics to measure event correlations.

We assume that time synchronization services are deployed on large-scale systems. The largest clock skew is easy to be estimated using a simple clock synchronization algorithm [33] [34] or the ntptrace [35] tool. With a time synchronization service, like NTP, we can ignore the effect of clock skew in our algorithm, since a NTP service can guarantee time synchronization to a large extent. For example, NTPv4 can achieve an accuracy of 200 microseconds or better in local area networks under ideal conditions.

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.

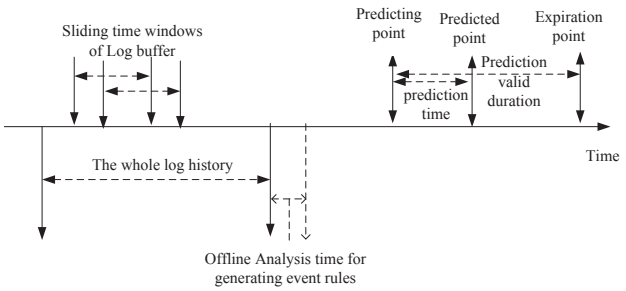

Fig. 2: The time relations in our event correlation mining and event prediction systems.

Considering the events may happen interleavedly, we propose a *confidence* metrics to measure the correlation of an LES as follows:

we count on two important attributes : *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 $BACBBA$ occur in a time window, for a $2-ary$ LES (A, B), the support count is one, and the posterior count is two; for a $3-ary$ LES (A, C, B), the support count is one and the posterior count is two. The confidence metrics is calculated to measure the event correlation according to Equation 1.

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

According to Equation 1, in $BACBBA$, the confidence of an $2-ary$ LES (A, B) is 1/2. In other words, if an event A occurs, an event B will occur with the probability of 50%.

Based on the above definitions, we formally define *frequent LES* and *event rule* as follows: for an LES, if its adjacent subsets are frequent and its support count exceeds a predefined threshold, we call it a frequent LES. For a frequent LES, if its confidence exceeds a predefined threshold, we call it an event rule.

C. Event correlation mining algorithms

In this section, we propose two event correlation mining algorithms: an Apriori-LES algorithm and its improved version—Apriori-simiLES.

The notations of the Apriori-LES and Apriori-simiLES algorithm are listed in TABLE. III.

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

TABLE III: Notations of the Apriori-LES and Apriori-simiLES

1) Apriori-LES algorithm: In data mining approaches, Apriori is a classic algorithm for learning association rules in transactional data [52] [53]. Apriori uses a breadth-first search and a tree structure to count item set candidates. According to the downward closure lemma [52] [53], a $k-length$ item set candidate contains all frequent $(k-1)-length$ item sets. The Apriori algorithm generates frequent $k-length$ item set candidates from frequent $(k-1)-length$ item sets. After that, it scans transaction data to determine frequent item sets among the candidates.

As mentioned in Section I, logs are significantly different from transaction data. We propose an algorithm to mine event rules, which we call the Apriori-LES algorithm. The Apriori-LES algorithm is 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 $k-ary$ LES candidates. We generate the frequent $k-ary$ LES candidate by the *LINK operation* of two frequent $(k-1)-ary$ adjacent subsets, and add it into $C(k)$.

The LINK operation is defined as below: for two frequent (k-1)-ary LES, if the last (k-2) log IDs of the one (k-1)-ary LES are same like the first (k-2) log IDs of the other (k-1)-ary LES, 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 $3-ary$ LES in $F(3)$, then a $4-ary$ LES (A, B, C, D) is a frequent $4-ary$ LES candidate, which we will add into $C(4)$.

Step 5: Scan the logs to validate each frequent $k-ary$ LES candidates in $C(k)$. The support count and posterior count of each $k-ary$ LES candidate in $C(k)$ is counted. For a frequent $k-ary$ LES 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 increment the support count and the posterior count of the $k-ary$ LES candidate by m and n, respectively.

Step 6: Generate frequent $k-ary$ LES and $k-ary$ event rules. For a $k-ary$ frequent LES candidate, if its support count is above the threshold Sth, add it into $F(k)$. For a $k-ary$ LES 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 LES 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-simiLES algorithm: As shown in Section V, the Apriori-LES algorithm still suffers from inefficiency, and generates a large amount of frequent LES candidate, which may lead to a long analysis time. In order to improve performance and save costs while ensuring the algorithm's efficiency, we observe the breakdown of event rules through mining about ten days's logs of the Hadoop system (in Nov 2008) and the HPC cluster system (in Jan 2004), respectively.

We set the following configuration in the Apriori-LES algorithm: the sliding time window (Tw), the support count threshold (Sth), and the confidence threshold (Cth) are 60 minutes, 5, and 0.25, respectively. We only mine $2-ary$ event rules so as to simplify the experiments.

In all, we get 517 $2-ary$ event rules in the Hadoop logs and 156 $2-ary$ event rules in the HPC logs. When we analyze the breakdown of 2-ary event rules generated by the Apriori-LES algorithm, we find that most of $2-ary$ event rules are composed of events that occur on the same nodes or the same applications, or have the same event types. This phenomenon is probably due to: (a) error may spread in a single node. For example: 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 an 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. The analysis results are shown in TABLE. IV and TABLE. V.

Description	*All*	*Same nodes*	*Same event types*	*Same applications*
count	*517*	*168*	*172*	*159*
Percent(%)	*100%*	*32.5%*	*33.3%*	*30.8%*

TABLE IV: The breakdown of $2-ary$ event rules in the Hadoop logs.

Description	*All*	*Same nodes*	*Same event types*	*Same applications*
count	*156*	*10*	*48*	*52*
Percent(%)	*100%*	*6.4%*	*30.8%*	*33.3%*

TABLE V: The breakdown of $2-ary$ event rules in the HPC logs.

On the basis of these observations, we propose an improved version of the Apriori-LES algorithm: Apriori-simiLES. The distinguished difference of Apriori-simiLES from Apriori-LES 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-simiLES algorithm includes two rounds of analysis: a single-node analysis and a multiple-node analysis. In the single-node analysis, we use the Apriori-LES algorithm to mine event rules that have same $node\ IDs$. And in the multiple-node analysis, we use the Apriori-LES 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.

A ECG is a directed acyclic graph (DAG). A vertex in a ECG represents a event. For a vertex, its children 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.

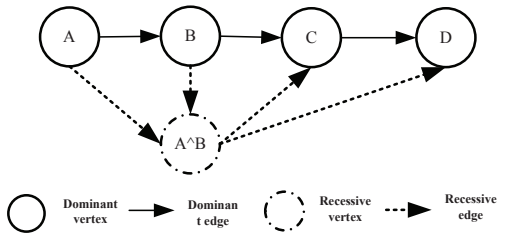

Fig. 3: An example of ECG.

Choosing the ECG 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.

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 event rules generated in the analysis, 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.

E. Event Prediction

For each prediction, there are three important timing points: *predicting point*, *predicted point*, and *expiration point*. 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.

In addition, 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.

The event prediction algorithm based on ECGs is 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.

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

$$Probability(C) = probability(B) * confidence(B \rightarrow C)$$
$$= confidence(A \rightarrow B) * confidence(B \rightarrow C) \quad (3)$$

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 $A \wedge B$ are marked too. The probability of events C and D are also calculated according to the new edges.

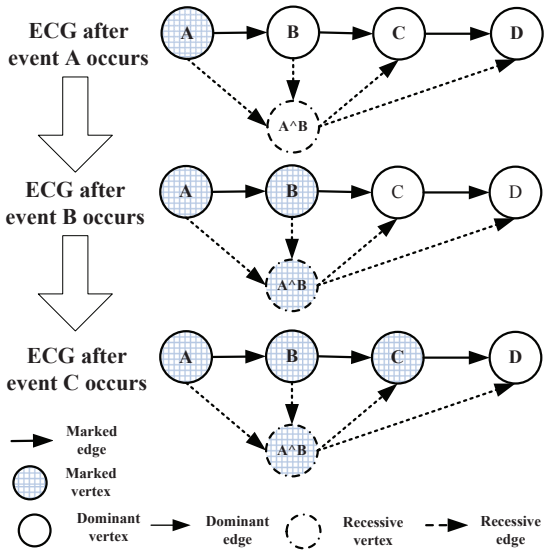

Fig. 4: An example of event prediction.

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

We have implemented an event correlation mining system and an event prediction system. The main modules of LogMaster are 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.

Log database is a MySQL database, including the tables of the formatted logs, the filtered logs, and the event rules. We define event attributions in the section "definitions", and the definitions include $timestamp$, $node\,name$, $application$, $process$, $process\ id$, $user$, $description$, and so on. The formats of event logs are defined in the section "format". All keywords that are used to decide the severity degree and event type of logs are defined in the section "keywords".

(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-LES and Apriori-semiLES 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. Evaluations

In this section, we use LogMaster to analyze logs of a production Hadoop cluster system and a HPC cluster system logs, the detail of two logs are described in Section II-A. The server we used has two Intel Xeon Quad-Core E5405 2.0GHZ processors, 137GB disk, and 8G memory.

A. Filtering Events

In this step, we remove repeated events and periodic events.

After collecting logs, Log sever will perform a simple filtering according to two observations: first, there are two types of repeated events: one kind of repeated events are recorded by different subsystems, and the other kind of repeated events repeatedly occur in a short time windows. Second, 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 use a simple statistical algorithm and a simple clustering algorithm to remove repeated events and periodic events, respectively. The solution to removing repeated events is as follows: for the first step, we treat events with the same $logID$ and the same $timestamp$ as repeated events. For the second step, we treat events with the same $logID$ occurring in a small time windows as repeated events. In this experiment, we set this interval threshold as 10 seconds, and the reason is that we consider the repeated events should occur in a short time windows.

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 event percent of the same interval, which is obtained against all periodic events with the same log ID, is higher than the predefined threshold values, respectively, we consider the interval as a fixed cycle. In our experiment, we

set two predefined threshold values of the Hadoop logs and the HPC logs as (20, 0.2) and (20, 0.1), respectively. The effects of different threshold values on the number of filtered events can be found at Appendix A. Lastly, we only keep one event for periodic events with the same fixed cycle. For periodic events, only events deviated from the fixed cycle are reserved.

TABLE VI shows the experiments results. In preprocessing, our python scripts parse about 977,858 original Hadoop event entries in 4 minutes 24 seconds, and interpret those events into nine-tuples, which are stored into the MySQL database. We parse 176,043 original HPC cluster event entries in 2 minutes 28 seconds. Please note that we only select the node logs from the HPC logs without including other events, e.g., that of "switch module" , since the Hadoop logs only include node logs.

logs	*raw logs*	*Pre processing*	*Removing repeated events*	*Removing periodic events*	*Compression rate*
Hadoop	*977,858*	*977,858*	*375,369*	*26,538*	*97.29%*
HPC cluster	*433,490*	*176,043*	*152,112*	*132,650*	*69.4%*

TABLE VI: The results of preprocessing and filtering logs.

In this experiment, the compression rate of the Hadoop logs can achieves 97.29%, but the compression rate of the HPC logs only achieves 69.4%. The reason is probably that the Hadoop logs have a large amount of repeated events, while the HPC logs have relatively small number of events for a long period, and hence have less repeated events. The periodic events of HPC logs are small, and the reason is that the HPC logs have a long time span.

B. Comparison of two event correlation mining algorithms

In this step, we compare two proposed algorithms: Apriori-LES and Apriori-simiES. We use (a) *the average analysis time per events* and (b) *the number of event rules* to evaluate the computational complexity and the efficiency of Apriori-LES and Apriori-simiES algorithms, respectively.

For the Hadoop logs, we analyze 43 days' logs from 2008-10-26 04:04:20.0 to 2008-12-09 23:21:28.0. For the HPC logs, we analyze 48 days' logs from 2003-12-26 22:12:30.0 to 2004-02-13 03:02:39.0.

Before reporting experiment results of two algorithms, we pick the following parameters as the baseline configuration of LogMaster for comparisons. Through comparisons with an large amount of experiments, we set the baseline parameters in LogMaster: Hadoop logs—[Tw60/Sth5/Cth0.25] and HPC logs—[Tw60/Sth5/Cth0.25]. [Twx/Sth$y$/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. The effect of varying parameters (Tw, Sth and Cth) on the average analysis time per event and the number of event rules can be found in Appendix B.

The comparison experiments show that: for the Hadoop logs, the average analysis time of Apriori-simiLES is about 10%-20% of that of Apriori-LES, while Apriori-simiLES obtains about 60%-70% event rules of that of Apriori-LES; For the HPC logs, the average analysis time of Apriori-simiLES is about 10%-20% of that of Apriori-LES algorithm, while Apriori-simiLES obtains about 80%-90% event rules of that of Apriori-LES.

C. The summaries of events and event rules in two typical cluster systems

In two typical cluster systems for Cloud and HPC, respectively, we give the summaries of the events and events rules, which are generated by the Apriori-LES algorithm with the baseline parameters mentioned above.

(a) In the Hadoop logs, the number of events of different types ranks according to the order: FILESYSTEM, HARDWARE, SOFTWARE, SYSTEM, MEMORY, NETWORK and OTHER. In the HPC logs, the number of events of different types ranks according to the order: HARDWARE, SYSTEM, NETWORK, FILESYSTEM, CLUSTERSYSTEM, and KERNEL. Please note that the event types of two logs are slightly different. For the HPC logs, the event types are recorded in the original logs, while for the Hadoop logs the event types are parsed by ourself. The breakdown of logs is shown in Fig. 5.

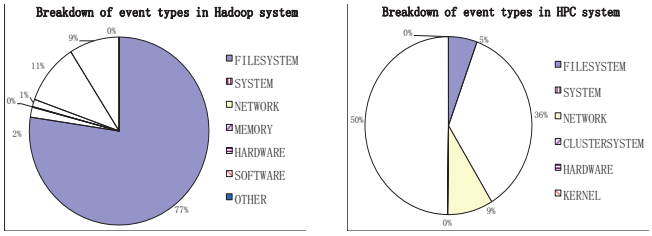

Fig. 5: The breakdown of event types of two system logs.

(b) 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; However, most of event rules in the HPC 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 HPC cluster systems.

With the same baseline configuration like that in V-B, we obtain 2 or 3-ary event rules with Apriori-LES and Apriori-simiLES as shown in TABLE VII.

Event Rules	Hadoop logs		HPC cluster logs	
	Apriori-LES	*Apriori-simiLES*	*Apriori-LES*	*Apriori-simiLES*
$2-ary$	*2413*	*1520*	*4726*	*3990*
$3-ary$	*1603*	*1603*	*1695*	*1695*

TABLE VII: The event rules obtained from two algorithms.

We also analyze the breakdown of event rules that lead to failure events. As shown in TABLE VIII, with the exception of the event rules in the HPC logs obtained with the Apriori-simiLES algorithm, event rules that identify correlations between non-failure events (including "INFO", "WARNING",

"ERROR") and failure events ("FATAL" and "FAILURE")) dominate over the event rules that identify the correlations between different failure events.

Failure Rules(No.)	Hadoop logs		HPC cluster logs	
	Apriori-LES	*Apriori-simiLES*	*Apriori-LES*	*Apriori-simiLES*
Configuration	[Tw60/Sth5/Cth0.25]		[Tw60/Sth5/Cth0.25]	
Non-failures→Failures	*377*	*180*	*97*	*5*
Failures→Failures	*86*	*78*	*26*	*26*

TABLE VIII: The event rules obtained from two algorithms

logid1	*logid2*	*nodeid1*	*type1*	*nodeid2*	*type2*	*confidence*
3314	*3311*	*249*	*memory*	*249*	*system*	*0.997487*
370	*359*	*42*	*hardware*	*42*	*filesystem*	*0.993789*
91	*89*	*4*	*software*	*4*	*system*	*0.961538*
2034	*2035*	*164*	*filesystem*	*164*	*filesystem*	*0.952381*
1412	*1413*	*120*	*software*	*120*	*software*	*0.947368*
66	*64*	*2*	*system*	*2*	*software*	*0.947368*
147	*148*	*12*	*filesystem*	*12*	*hardware*	*0.9375*
3632	*3628*	*260*	*system*	*260*	*software*	*0.933333*
3627	*3628*	*270*	*filesystem*	*270*	*software*	*0.933333*
172	*169*	*22*	*hardware*	*22*	*filesystem*	*0.928571*

TABLE IX: Top 10 $2-ary$ event rules in the order of the descending confidence in the Hadoop logs

logid1	*logid2*	*nodeid1*	*type1*	*nodeid2*	*type2*	*confidence*
4671	*4682*	*260*	*hardware*	*260*	*hardware*	*0.975*
2598	*2580*	*153*	*hardware*	*153*	*hardware*	*0.975*
2601	*2619*	*154*	*hardware*	*154*	*hardware*	*0.928571*
2193	*2180*	*131*	*hardware*	*131*	*hardware*	*0.923077*
2774	*2883*	*162*	*hardware*	*167*	*hardware*	*0.923077*
2796	*2883*	*163*	*hardware*	*167*	*hardware*	*0.923077*
2819	*2985*	*164*	*hardware*	*172*	*hardware*	*0.923077*
2556	*2539*	*151*	*hardware*	*151*	*hardware*	*0.916667*
3195	*3212*	*182*	*hardware*	*182*	*hardware*	*0.916667*
2661	*2643*	*156*	*hardware*	*156*	*hardware*	*0.909091*

TABLE X: Top 10 $2-ary$ event rules in order of confidence in the HPC logs.

The top 10 $2-ary$ event rules in the order of the descending confidence in the Hadoop logs and the HPC logs are shown in TABLE IX and TABLE X, respectively. For the top one $2-ary$ event rule in the Hadoop logs—$(3314, 3311)$, the original logs are shown in TABLE XI.

[Log id=3314] 2008-12-06 05:04:27 compute-12-9.local looks like a 64bit wrap, but prev!=new [Log id=3311] 2008-12-06 05:04:57 compute-12-9.local c64 32 bit check failed

TABLE XI: The original logs of $(3314, 3311)$.

D. 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.

On the basis of event rules obtained in Section V-C, we predict 21 days' logs from 2008-12-10 00:00:38.0 to 2008-12-31 15:32:03.0 in the Hadoop logs, and 14 days' logs from 2004-02-13 03:02:41.0 to 2004-02-27 19:02:00.0, respectively.

We use *the precision rate*, *the recall rate*, and *the average prediction time of event prediction* to evaluate the prediction. *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 precision rate is the ratio of the correctly predicted events (TP) to all predicted events, including TP and FP. The recall rate is the ratio of correctly predicted events (TP) to all filtered events. We calculate the average prediction time according to Equation 4.

$$\text{The average prediction time} = \frac{\sum(\text{the predicted point} - \text{the predicting point})}{\text{count of all predicted events}} \quad (4)$$

There are two parameters that affect the prediction accuracy, including the prediction probability threshold (Pth) and the prediction valid duration (Tp). Before reporting experiment results, we pick the following parameters as the baseline configuration for comparisons. Through comparisons with large amount of experiments, we set the baseline parameters of the Hadoop logs-[Tw60/Sth5/Cth0.25/Pth0.5/Tp60] and the baseline parameters of the HPC logs-[Tw60/Sth5/Cth0.25/Pth0.5/Tp60]. Please note that Tw, Sth, Cth just keep the same baseline parameters in Section V-B.

[Pth*u*/Tp*v*] indicates that the prediction probability threshold Pth is u, and the prediction valid duration Tp is v minutes. The effects of varying Pth and Tp on failure predictions can be found at Appendix C.

First, on the basis of events 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 and the HPC logs are shown in Fig. 6. From Fig. 6, we can observe that with Apriori-LES, the precision rates of the Hadoop logs and the HPC logs are high as 78.20%, 81.19%, respectively, while the recall rates of the Hadoop logs and the HPC logs are 33.63% and 20.73%, respectively. The reason for the low recall rates is that we still keep rich log information after filtering events, including 26,538 entries (2.71% of the original Hadoop logs) and 132,650 entries (30.6% of the original HPC logs), respectively. We also notice that adopting a more efficient algorithm—Apriori-semiLES, which mines fewer event rules, results in higher precision rates. This is because with Apriori-LES we can obtain more event rules, which predicts more events which not happen.

Second, on the basis of events of all types, we only predict failure events (Failure and FATAL types), of which

the precision rates and the recall rates of two logs are shown in Fig. 7.

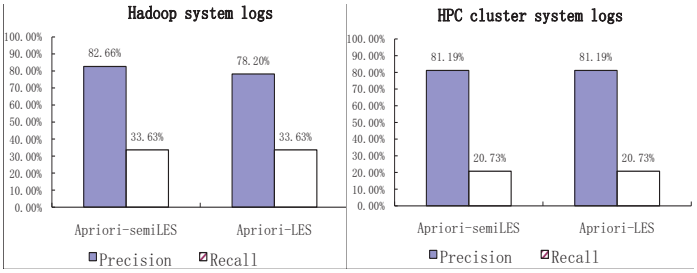

Fig. 6: The precision rate and recall rate of predicting events (including failure and non-failure events) on the basis of failure and non-failure events.

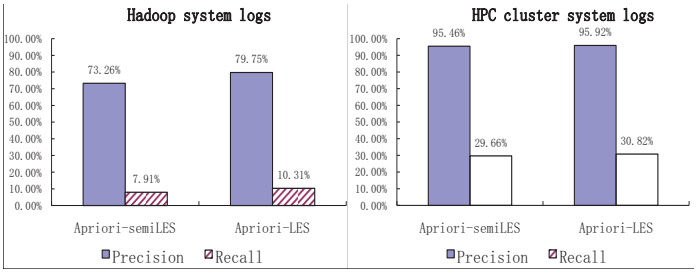

Fig. 7: The precision rate and recall rate of predicting *failure* events on the basis of both failure and non-failure events.

For predicting failure events, Liang *et al.* [6] suggest removing events with the types of *INFO, WARNING, and ERROR* in preprocessing events. Lastly, following their idea, we remove non-failure events, and then predict failure events in the subsequent experiments.

TABLE XII reports the filtered events at different stages. With respect to TABLE VI, TABLE XII show after removing non-failure events, only a small fraction of events are reserved: 1,993 v.s. 132,650 entries (not removing non-failure events) for the HPC logs; 3,112 v.s. 26,538 entries (not removing non-failure events) for the Hadoop logs. Then we predict failure events after removing non-failure events, and the precision rates and recall rates of predicting failure events are shown in Fig. 8.

From Figs.7 and 8, we can observe two points. First, after removing non-failure events, the precision rates in predicting failures are lower than that without removing non-failure events. The reason is that in both logs 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 VIII in Section V-C. Second, in predicting failures (FAILURE and FATAL), the recall rates are low (especially for the Hadoop logs). This observation has two reasons. (a) Some events are independent, 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 long time-correlated event rules may be discarded.

logs	*raw logs*	*Removing repeated events*	*Removing non-failure events*	*Removing periodic events*	*Compression rate*
Hadoop	*977,858*	*375,369*	*53,259*	*3,112*	*99.68%*
HPC	*433,490*	*152,112*	*5,427*	*1,993*	*99.54%*

TABLE XII: The results of filtered events after removing non-failure events.

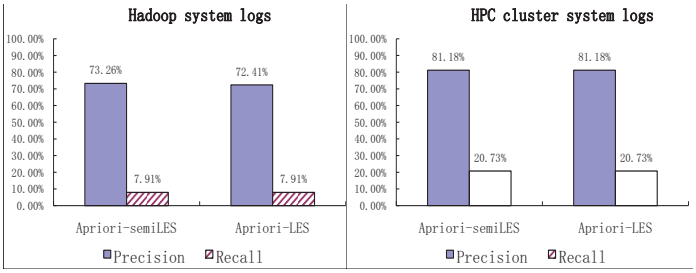

Fig. 8: The precision rate and recall rate of predicting failure events after removing non-failure events.

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

Average prediction time(minutes)	*Hadoop logs*	*HPC cluster logs*
event prediction	*42.78*	*4.01*
failure prediction	*52.01*	*25.57*
failure prediction after removing non-failure	*52.01*	*25.57*

TABLE XIII: The average prediction time.

VI. Related Work

We summarize the related work from five perspectives: characterizing failure characteristics, log preprocessing, event correlation mining, anomaly (failure or performance bottleneck) prediction, and failure diagnosis.

A. characterizing failure characteristics

It has been long recognized that failure events are correlated, not independent. For example, in the year of 1992, Tang *et al.* [30] concluded that the impact of correlated failures on dependability is significant. The work in [1] has observed that there are strong spatial correlations between failure events and most of the failure events occur on a small fraction of the nodes. The work in [6] presents that some failure events such as network failure events and application I/O failure events show more pronounced skewness in the spatial distribution. The work of [3] and [6] has found that failure events can propagate in the systems.

Some work uses statistical analysis approach to find simple temporal and spatial laws or models of system events [13] [3] [28] in large-scale cluster systems like Bluegene/L. When the obtained knowledge is used in failure prediction, it may bring good precision rate and recall rate, but the prediction results are coarse and high level without the detail.

Daniel *et al.* [64] characterize the availability properties of cloud storage systems based on an extensive one year study of Googles main storage infrastructure and present statistical models that enable further insight into the impact of multiple design choices, such as data placement and replication strategies.

Edmund *et al.* [66] present the first large-scale analysis of hardware failure rates on a million consumer PCs. They found that many failures are neither transient nor independent. Instead, a large portion of hardware induced failures are recurrent: a machine that crashes from a fault in hardware is up to two orders of magnitude more likely to crash a second time.

B. Log Pre-processing

Zheng *et al.* [25] proposes a log pre-processing method, and adopt a causality-related filtering approach to combining correlated events for filtering through apriori association rule mining.

In these research efforts, the concept of event cluster [6] is proposed to deal with multiple redundant records of fatal events at one location for event filtering; an apriori association rule mining [10] is presented to identify the sets of fatal events co-occurring frequently and filter them together; an automated soft competitive learning neural-gas method [28] is used for cluster analysis to reduce dependent events.

C. Event Correlation Mining

In the data mining field, [59] [58] concern about mining closed sequential patterns, [55] [57] discusses the frequent pattern mining, [54] [53] [52] focus on generalizing association rules to correlations.

Hellerstein *et al.* [10] present efficient algorithms to mine three types of important patterns from historical event data: event bursts, periodic patterns, and mutually dependent patterns, discuss a framework for efficiently mining events that have multiple attributes, and finally build a tool—Event Correlation Constructor that validates and extends correlation knowledge.

Lou [31] propose an approach to mine inter-component dependencies from unstructured logs: parse each log message into keys and parameters; find dependent log key pairs belong to different components by leveraging co-occurrence analysis and parameter correspondence; use Bayesian decision theory to estimate the dependency direction of each dependent log key pair.

Mannila *et al.* [57] give efficient algorithms for the discovery of all frequent episodes from a given class of episodes, and present detailed experimental results.

Though lots of previous efforts have proposed failures mining approaches for different purposes, for example event filtering [6][25], event coalescing [28], or failure prediction [2], little work proposes the event correlation mining system for large-scale cluster systems.

D. Anomaly (Failure or Performance Bottleneck) Prediction

1) Performance bottleneck prediction: Zhang *et al.* [15] proposes a precise request tracing algorithm for multi-tier services of black boxes, which only uses application-independent knowledge and constructs a component activity graph abstraction to represent causal paths of requests and facilitate end-to-end performance debugging.

Gu *et al.* [37] focus on predicting the bottleneck anomaly, the most common anomaly in data stream processing clusters. Their approach integrates naive Bayesian classification method, which captures the distinct symptoms of different bottlenecks caused by various reasons, and Markov models, which capture the changing patterns of different measurement metrics that are used as features by the Bayesian classifiers, to achieve the anomaly prediction goal.

Tan *et al.* [38] presents the context-aware anomaly prediction model training algorithm to predict various system anomalies such as performance bottlenecks, resource hotspots, and service level objective (SLO) violations. They first employ a clustering algorithm to discover different execution contexts in dynamic systems, and then train a set of prediction models, each of which is responsible for predicting anomalies under a specific context.

Shen *et al.* [47] propose a model-driven anomaly characterization approach and use it to discover operating system performance bugs when supporting disk I/O-intensive online servers.

2) Failure prediction: Gujrati *et al.* [5] presents a meta-learning method based on statistical analysis and standard association rule algorithm. They not only obtain the statistical characteristics of failures, but also generate association rules between nonfatal and fatal events for failure predictions.

Fu *et al.* [3][4] develops a spherical covariance model with an adjustable timescale parameter to quantify the temporal correlation and a stochastic model to describe spatial correlation. They cluster failure events based on their correlations and predict their future occurrences.

Fulp *et al.* [19] describes a spectrum-kernel Support Vector Machine (SVM) approach to predict failure events based on system log files. The approach described use a sliding window (sub-sequence) of messages to predict the likelihood of failure.

E. Failure Diagnosis

Chen *et al.* [40] presents an instance based approach to diagnosing failures in computing systems. Their method takes advantage of past experiences by storing historical failures in a database and developing a novel algorithm to efficiently retrieve failure signatures from the database.

Oliner *et al.* [42] propose a method for identifying the sources of problems in complex production systems where,

due to the prohibitive costs of instrumentation, the data available for analysis may be noisy or incomplete.

John *et al.* [41] present a fault localization system called Spotlight that essentially uses two basic ideas. First, it compresses a multi-tier dependency graph into a bipartite graph with direct probabilistic edges between root causes and symptoms. Second, it runs a novel weighted greedy minimum set cover algorithm to provide fast inference.

Console logs rarely help operators detect problems in large-scale datacenter services, for they often consist of the voluminous intermixing of messages from many software components written by independent developers. Xu *et al.* [49] [36] propose a general methodology to mine this rich source of information to automatically detect system runtime problems.

Tucek *et al.* [48] propose a system Triage, that automatically performs onsite software failure diagnosis at the very moment of failure. It provides a detailed diagnosis report, including the failure nature, triggering conditions, related code and variables, the fault propagation chain, and potential fixes.

Tan *et al.* [60] propose SALSA—their approach to automated system-log analysis, which involves examining the logs to trace control-flow and data-flow execution in a distributed system, and derive state-machine-like views of the systems execution on each node. Based on the derived state machine views and statistics, they illustrate SALSA's value by developing visualization and failure-diagnosis techniques for three Hadoop workloads.

VII. CONCLUSION AND FUTURE WORK

In this paper, we designed and implemented an event correlation mining system and an event prediction system. We presented a simple metrics to measure correlations of events that may happen interleavedly. On the basis of the measurement of correlations, we proposed two approaches to mining event correlations; meanwhile, we proposed an innovative abstraction—event correlation graphs (ECGs) to represent event correlations, and presented an ECGs-based algorithm for event prediction. As two typical case studies, we used LogMaster to analyze and predict logs of a production Hadoop-based cloud computing system at Research Institution of China Mobile, and a production HPC cluster system at Los Alamos National Lab (LANL), respectively. For the first time, we compared the breakdown of events of different types and events rules in two typical cluster systems for Cloud and HPC, respectively.

In the new future, we will investigate two issues. a) How to use ECGs for fault diagnose in large-scale production cluster systems? b) How to combine causal path-based solutions [15] with the log mining approach to diagnosis failure events and performance problems?

REFERENCES

[1] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, *et al.* "Failure data analysis of a large-scale heterogeneous server environment". In *Proc. of DSN*, 2004.

[2] R. K. Sahoo, A. J. Oliner, *et al.* "Critical Event Prediction for Proactive Management in Large scale Computer Clusters". In *Proc. of SIGKDD*, 2003.

[3] S. Fu, C. Xu, *et al.* "Exploring Event Correlation for Failure Prediction in Coalitions of Clusters". In *Proc. of ICS*, 2007.

[4] S. Fu and C. Xu, *et al.* "Quantifying Temporal and Spatial Correlation of Failure Events for Proactive Management". In *Proc. of SRDS*, 2007.

[5] P. Gujrati, Y. Li, Z. Lan, *et al.* "A Meta-Learning Failure Predictor for Blue Gene/L Systems". In *Proc. of ICPP*, 2007.

[6] Y. Liang, *et al.* "Filtering Failure Logs for a BlueGene/L Prototype". In *Proc. of DSN*, 2005.

[7] Z. Xue, et al. "A Survey on Failure Prediction of Large-Scale Server Clusters". In ACIS Conf. on SNPD, 2007.

[8] C. Yuan, *et al.* "Automated Known Problem Diagnosis with Event Traces". In *Proc of Eurosys*, 2006.

[9] H. J. Wang, J. C. Platt, Y. Chen, *et al.* "Automatic Misconfiguration Troubleshooting with PeerPressure". In *Proc. of OSDI*, 2004.

[10] J. L. Hellerstein, S. Ma, and C. Perng, *et al.* "Discovering actionable patterns in event data". In *IBM Systems Journal*, 41(3), 2002.

[11] R Agrawal, R Srikant, *et al.* "Fast algorithms for mining association rules". In *Proc. of VLDB*, 1994.

[12] K. Yamanishi, Y. Maruyama, *et al.* "Dynamic Syslog Mining for Network Failure Monitoring". In *Proc. of SIGKDD*, 2004.

[13] Y. Liang, Y. Zhang, *et al.* "BlueGene/L Failure Analysis and Prediction Models". In *Proc. of DSN*, 2006.

[14] T. T. Y. Lin, D. P. Siewiorek, *et al.* "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis". *IEEE Trans. on Reliability*, vol. 39, pp. 419-432, October 1990.

[15] Z. Zhang, J. Zhan, Y. Li, *et al.* "Precise Request Tracing and Performance Debugging for Multi-tier Services of Black Boxes". In *Proc. of DSN*, 2009.

[16] W. Zhou, J. Zhan, D. Meng. "Multidimensional Analysis of Logs in Large-scale Cluster Systems". In *FastAbstract, Proc. of DSN*, 2008.

[17] A. J. Oliner, A. Aiken, J. Stearley. "Alert Detection in Logs". In Proc. of ICDM, 2008.

[18] S. Zhang, I. Cohen , M. Goldszmidt, *et al.* "Ensembles of models for automated diagnosis of system performance problems". In *Proc. of DSN*, 2005.

[19] E. W. Fulp, G. A. Fink, J. N. Haack. "Predicting Computer System Failures Using Support Vector Machines". In *First USENIX Workshop on the Analysis of Logs (WASL)*, 2008.

[20] J. C. Knight. "An Introduction To Computing System Dependability". In *Proc. of ICSE*, 2004.

[21] L. Lamport. "Time, Clocks and the Ordering of Events in a Distributed System". *Communications of the ACM*, 21(7), 1978, pp.558-565.

[22] http://institutes.lanl.gov/data/fdata/

[23] http://www.python.org/

[24] J. Han, M. Kamber. "Data mining: Concepts and Techniques". *Morgan Kaufmann Publishers*, 2000.

[25] Z. Zheng, Z. Lan, B. H. Park, A. Geist. "System Log Pre-processing to Improve Failure Prediction". In *Proc. of DSN*, 2009.

[26] R. Zhang, E. Cope, L. Heusler, F. Cheng. "A Bayesian Network Approach to Modeling IT Service Availability using System Logs". In *USENIX workshop on WASL*, 2009

[27] L. Wu, D. Meng, W. Gao, J. Zhan. "A proactive fault-detection mechanism in large-scale cluster systems". In *Proc of IPDPS*, 2006.

[28] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems". *Journal of Parallel and Distributed Computing*. 69, 7 (Jul. 2009), 652-665.

[29] L. Chinghway, N. Singh, and S. Yajnik. "A log mining approach to failure analysis of enterprise telephony systems". In *Proc. of DSN*, 2008.

[30] D. Tang and R. K. Iyer, "Analysis and Modeling of Correlated Failures in Multicomputer Systems". *IEEE Transactions on Computers*. 41, 5 (May. 1992), 567-577.

[31] J. Lou, Q. FU, Y. Wang, J. Li. "Mining Dependency in Distributed Systems through Unstructured Logs Analysis", In *USENIX workshop on WASL*, 2009.

[32] W. Zhou, J. Zhan, D. Meng, and Z. Zhang. "Online Event Correlations Analysis in System Logs of Large-Scale Cluster Systems", In *Proc. NPC*, 2010.

[33] R. Gusella, and S. Zatti, "The accuracy of clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD", *IEEE Transactions Software Engineering*, 1989. 15: p. 847-853.

[34] F. Cristian, "Probabilistic clock synchronization. Distributed Computing", 1989. 3: p. 146-158.

[35] ntptrace - trace a chain of NTP servers back to the primary source. http://www.cis.udel.edu/ mills/ntp/html/ntptrace.html.

[36] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Experience Mining Google's Production Console Logs", In *USENIX Worksip on SLAML*, 2010.
[37] X. Gu, H. Wang, "Online Anomaly Prediction for Robust Cluster Systems", In *Proc. ICDE*, 2009.
[38] Y. Tan, X. Gu, and H. Wang, "Adaptive Runtime Anomaly Prediction for Dynamic Hosting Infrastructures", In *Proc. PODC*, 2010.
[39] P. Zhou, B. Gill, W. Belluomini, and A. Wildani, "GAUL: Gestalt Analysis of Unstructured Logs for Diagnosing Recurring Problem in Large Enterprise Storage Systems", In *Proc. SRDS*, 2010.
[40] H. Chen, G. Jiang, K. Yoshihira, and A. Saxena, "Invariants Based Failure Diagnosis in Distributed Computing Systems", In *Proc. SRDS*, 2010.
[41] D. John, P. Prakash, R. R. Kompella, R. Chandra, "Shedding Light on Enterprise Network Failures using Spotlight", In *Proc. SRDS*, 2010
[42] A. J. Oliner, A. V. Kulkarni, A. Aiken, "Using Correlated Surprise to Infer Shared Influence", In *Proc. DSN*, 2010.
[43] M. K. Agarwal, V. R. Madduri, "Correlating Failures with Asynchronous Changes for Root Cause Analysis in Enterprise Environments", In *Proc. DSN*, 2010.
[44] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs", In *Proc. of ICDM*, 2007.
[45] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Comparison of machine learning methods for predicting failures in hard drives", *Journal of Machine Learning Research*, 2005.
[46] B. Schroeder and G. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean too you", In *Proc. of FAST*, 2007.
[47] K. Shen, M. Zhong, and C. Li, "I/O system performance debugging using model-driven anomaly characterization", In *Proc. of FAST*, 2005.
[48] J. Tucek, S. Lu, C. Huang, S. Xanthos, and Y. Zhou, "Triage: Diagnosing Production Run Failures at the Users Site", In *Proc. of SOSP*, 2007.
[49] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Large-scale system problems detection by mining console logs", In *Proc. of SOSP*, 2009.
[50] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems". In *Proc. of DSN*, 2006.
[51] W. Zhou, J. Zhan, D. Meng, D. Xu, and Z. Zhang. "LogMaster: Mining Event Correlations in Logs of Large scale Cluster Systems". *Technical Report*. http://arxiv.org/abs/1003.0951v1.
[52] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules", In *Proc. VLDB*, 1994.
[53] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," In *Proc. of SIGMODE*, 1993.
[54] S. Brin, R. Motiwani, and C. Silverstein, "Beyond Market Baskets: Generalizing Association Rules to Correlations", *Data Mining and Knowledge Discovery 2*, No. 1, 39C68 (1998).
[55] 15. J. Pei and J. Han, "Can We Push More Constraints into Frequent Pattern Mining", In *Proc. KDD*. 2000.
[56] Adetokunbo Makanju, A.Nur Zincir-Heywood, Evangelos E.Milios. "Extracting Message Types from BlueGene/L's Logs". *WASL 2009, Big Sky Resort, Big Sky, Montana*, October 14, 2009.
[57] H. Mannila, Hannu Toivonen, and A. Inkeri Verkamo. "Discovery of Frequent Episodes in Event Sequences". *Data Min. Knowl. Discov*. 1, 3 (January 1997), 259-289.
[58] P. Tzvetkov, X. Yan, and J. Han. "Tsp: Mining top-k closed sequential patterns". In *Proc. ICDM*, 2003.
[59] X. Yan, J. Han, and R. Afshar. "Clospan: Mining closed sequential patterns in large datasets". In *Proc. SDM*, 2003.
[60] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. "SALSA: analyzing logs as state machines". In *Proc. WASL*, 2008.
[61] S. Ma and J. L. Hellerstein, "Mining Mutually Dependent Patterns", In *Proc. ICDM*, 2001.
[62] R. Agrawal and R. Srikant, "Mining Sequential Patterns", In *Proc. ICDE*, 1995.
[63] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains", In *Proc. of ICDM*, 2002.
[64] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems", In *Proc. OSDI*, 2010.
[65] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability", In *Proc. of SoCC*, 2010.
[66] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer PCs". In *Proc. of EuroSys*, 2011.

APPENDIX A
CHOOSING THE THRESHOLD VALUES OF THE PERIODIC COUNT AND THE PERIODIC RATIO IN FILTERING PERIODIC EVENTS

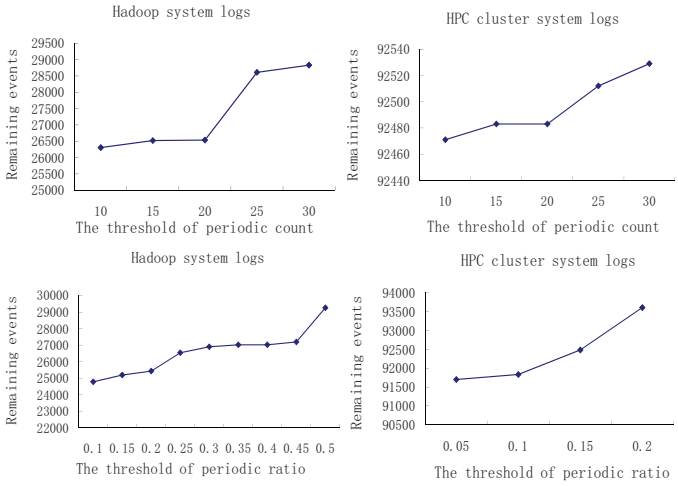

Fig. 9: Relationships between remaining events and two thresholdsłperiodic count and periodic ratio.

As shown in Fig. 9, when the threshold of the periodic count is less than 20, the remaining numbers of Hadoop system logs and HPC cluster system logs change little. When the threshold of the periodic count is above 20, the remaining events change significantly, so we set the threshold of the periodic count as 20.

When the threshold of the periodic ratio of Hadoop system logs is less than 0.2 and the threshold of periodic ratio of HPC cluster system logs is less than 0.1, the remaining events change little. So we set the threshold of the periodic ratio of Hadoop system logs to 0.2 and the threshold of periodic ratio of HPC cluster system logs to 0.1, respectively.

logs	*the threshold of periodic count*	*the threshold of periodic ratio*
Hadoop	*20*	*0.2*
HPC cluster	*20*	*0.1*

TABLE XIV: Threshold selection of two logs

APPENDIX B
PARAMETERS EFFECTS IN EVENT CORRELATION MINING

The effects of parameters(Tw, Sth and Cth) on the average analysis time per event and the number of event rules are shown in Fig. 10 and Fig. 11, respectively.

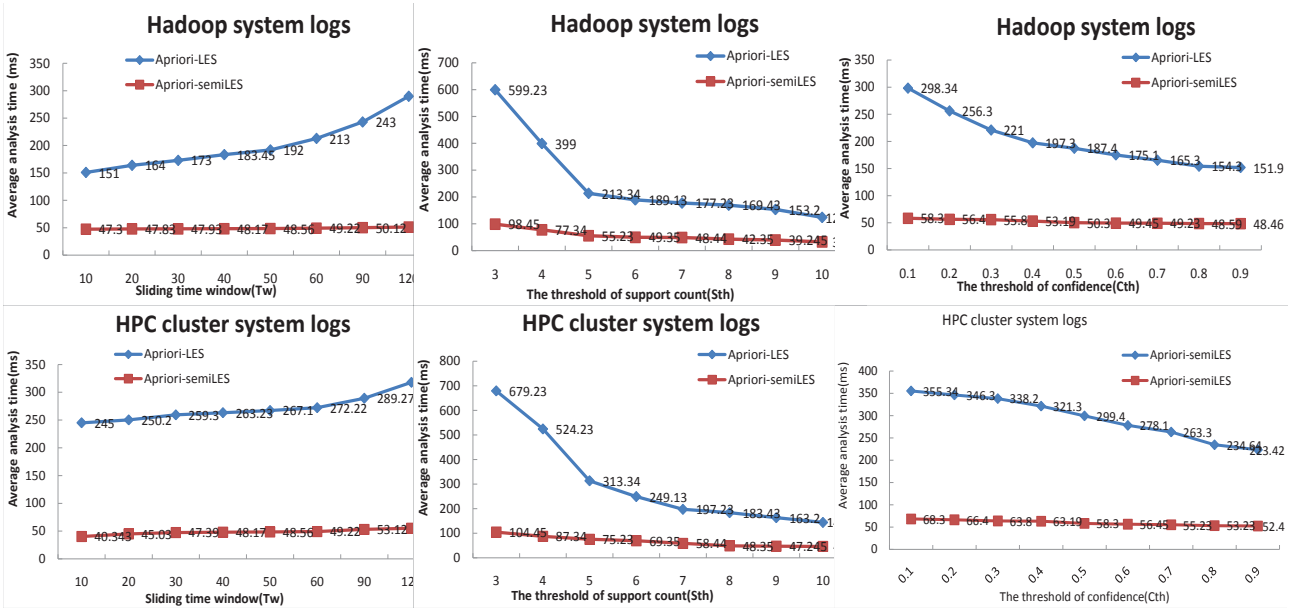

Fig. 10: The effect of parameters(*Tw*, *Sth* and *Cth*) on the average analysis time per event.

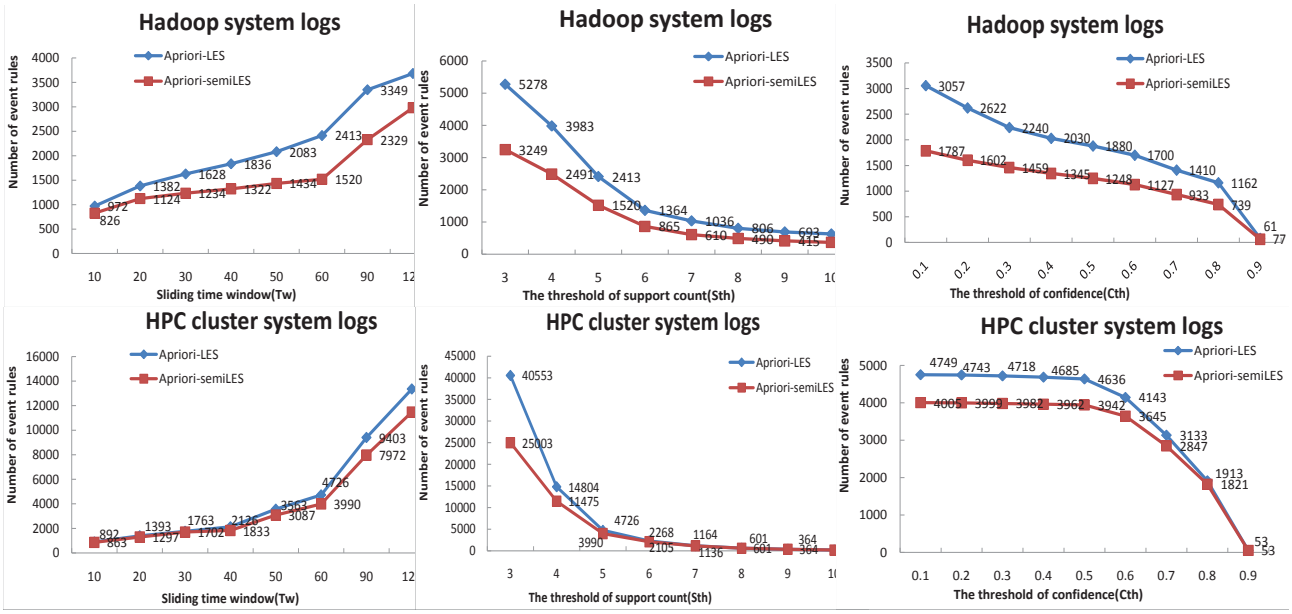

Fig. 11: The effect of parameters(*Tw*, *Sth* and *Cth*) on the number of event rules.

APPENDIX C
PARAMETERS EFFECTS IN EVENT PREDICTIONS

The effects of parameters(Pth and Tp) on the precision rates and recall rates are shown in Fig. 12, Fig. 13, respectively.

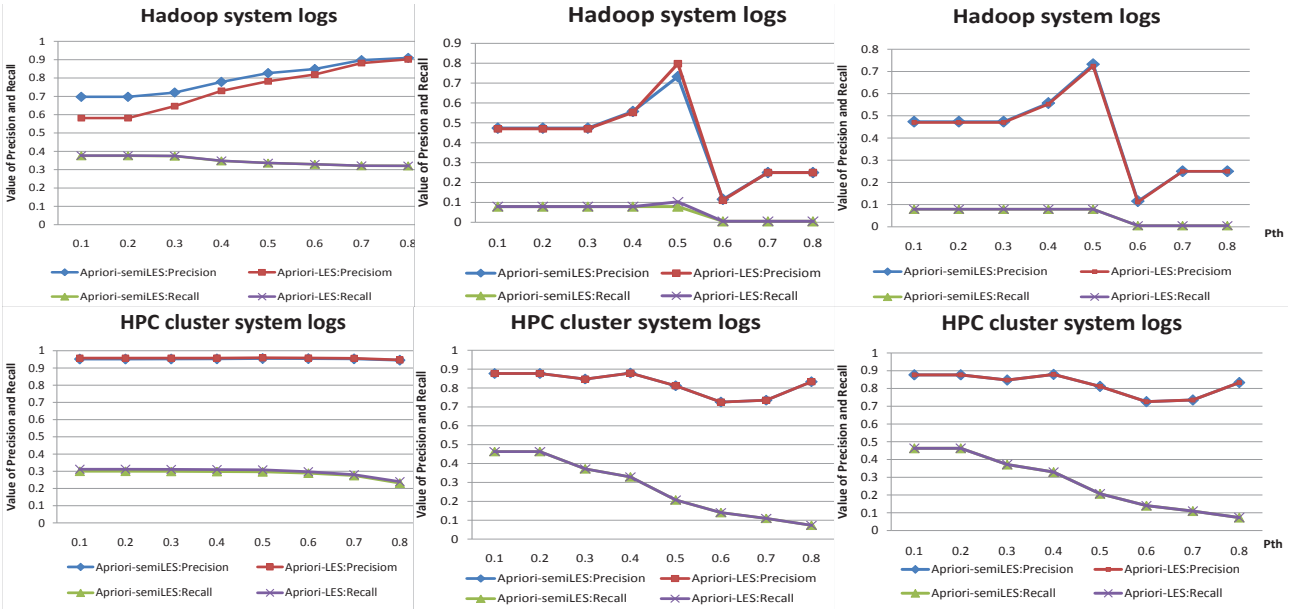

Fig. 12: The effects of the parameter *Pth* on the precision rate and the recall rate of the Hadoop system logs and the HPC cluster system logs in predicting all events on the basis of both failure and non-failure events, predicting only failure events on the basis of both failure and non-failure events, and predicting failure events after removing non-failure events (from left to right).

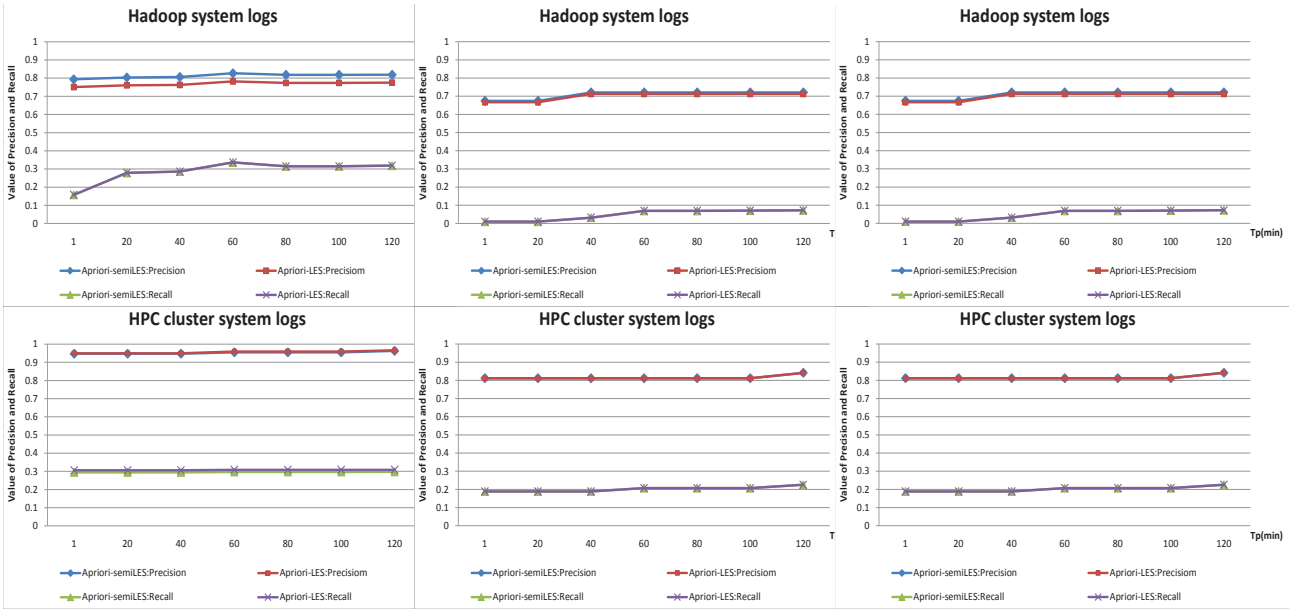

Fig. 13: The effects of the parameter *Tp* on the precision rate and recall rate of the Hadoop system logs and the HPC cluster system logs in predicting all events on the basis of both failure and non-failure events, predicting only failure events on the basis of both failure and non-failure events, and predicting failure events after removing non-failure events (from left to right).