Digging Deeper into Cluster System Logs for Failure Prediction and Root Cause Diagnosis

Xiaoyu Fu*,† Rui Ren*, Sally A. McKee‡, Jianfeng Zhan*, Ninghui Sun*
* State Key Laboratory Computer Architecture,Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences
‡ Computer Science and Engineering, Chalmers University of Technology

***Abstract*—As the sizes of supercomputers and data centers grow towards exascale, failures become normal. System logs play a critical role in the increasingly complex tasks of automatic failure prediction and diagnosis. Many methods for failure prediction are based on analyzing event logs for large scale systems, but there is still neither a widely used one to predict failures based on both non-fatal and fatal events, nor a precise one that uses fine-grained information (such as failure type, node location, related application, and time of occurrence). A deeper and more precise log analysis technique is needed.**

We propose a three-step approach to draw out event dependencies and to identify failure-event generating processes. First, we cluster frequent event sequences into event groups based on common events. Then we infer causal dependencies between events in each event group. Finally, we extract failure rules based on the observation that events of the same event types, on the same nodes or from the same applications have similar operational behaviors. We use this rich information to improve failure prediction. Our approach semi-automates diagnosing the root causes of failure events, making it a valuable tool for system administrators.

***Index Terms*—event causal dependency inference; failure prediction; root cause diagnosis; large-scale cluster systems;**

I. Introduction

Cluster systems are common platforms for high performance computing, cloud computing, and data centers. With the ever-growing scales and complexities of these platforms, failures become normal [19] and critical. Logs of large-scale clusters are the primary resources for implementing dependability: they track system behaviors by accurately recording detailed data about a system's changing states. Automatic failure prediction [11], [35], [16], [20] and root cause diagnosis [2], [6], [37] analyze these huge logs to extract information that helps administrators to predict when and where a failure is likely to happen or to infer why a failure has occurred.

The overwhelming volumes, wide variations in format, and complicated interleaving of log messages make such processing increasingly difficult, challenging the accuracy of current approaches. A deeper and more precise log analysis technique is needed to glean more useful information, both for automatic failure prediction and to assist administrators in root cause diagnosis.

The common problem with most approaches to both activities is that they cannot consider the entire log. The two main approaches to failure prediction — statistical counting of co-occurring events and data mining of *Frequent Event Sequences* (FESs, events that frequent occur together) — break the logs into discrete windows. Even data mining techniques that are more parallelizable [30] and that try to present event correlations between transactions throughout the whole log are still limited by the scope of the corresponding transactions they consider, and thus they may only generate a short segment of a full event causal dependency chain.

Root cause diagnosis tools — which analyze logs to identify event correlations associating failures with the events that trigger them [2], [6], [37], [7] — typically suffer from the same window-limited myopia due to the sheer number of events. This is problematic because the time lag between cause and effect may be unbounded [12], [2]. The ability to identify event correlations without window-size constraints would let us better establish complete event causal dependencies, which would, in turn, benefit diagnosis.

Prediction precision is the ratio of correctly identified failures to the number of all predicted failures, whereas prediction recall is the ratio of correctly predicted failures to the number of true failures. Previous failure prediction approaches by Fu et al. [11], Zheng et al. [35], Gujrati et al. [16], Gainaru et al. [12], and Liang et al. [20] deliver precisions of up to 90% but have recall rates of only about 50%. This low coverage may result from the fact that not all *Failure Generating Processes* (FGPs) can be mined as FESs. We find that such FGPs, or the reasons why a triggering event leads to a failure, represent repeated and persistent phenomena that may be nonobvious. Note that FGPs are usually found by experienced administrators through root cause searching. Bouguerra et al. [4] show that recall has more impact on failure prediction than precision, suggesting that researchers should focus on improving prediction recall.

Our previous work [11] identifies event correlations in FESs by data mining huge system logs. Here we follow a similar approach, attempting to glean deeper insights by inferring complete event causal dependencies based on FESs. We then extract *failure rules* to present FGPs in order to increase prediction recall. Our approach is based on two observations: (1) there are common events among different FESs in the total data-mined collection, indicating that these FESs are probably correlated; and (2) events are generated in a temporal order based on repeated and often nonobvious causal dependency relations between the application, system software, and hardware.

978-1-4799-5548-0/14/$31.00 ©2014 IEEE

Our contributions are five-fold:

1) We cluster correlated FESs to form event groups according to similarity, which we define to be the number of common events between each FES pair.
2) We infer an event *causal dependency graph* from each group through the temporal order of FES events.
3) We extract *failure rules* (i.e., FGPs) represented by the order of frequent event IDs to derive more potential FESs (in addition to those we mine) for better failure prediction.
4) We demonstrate the effectiveness of event *causal dependency graphs* in precise root cause diagnosis.
5) We validate our approaches on logs from a Bluegene/L system [1], a LANL systems [22], and a 260-node Hadoop cluster system at the Research Institution of China Mobile, achieving average recall rates as high as 85.3%, 69.4%, and 75.3%, respectively, with recall improvements of 8-15%.

To the best of our knowledge, this is the first failure analysis in large-scale cluster systems that infers causal dependencies between events and tries to determine the FGP. Note that this work assumes that the system logs fairly accurately depict system state and that system clocks are synchronized.

II. Background and Analysis Goals

Inferring event causal dependencies from system logs is strictly more difficult than just mining event correlations identified by statistical associations. Statistical associations cannot communicate causality since a statistical association between two events `a` and `b` could indicate that `a` causes `b`, that `b` causes `a`, or that some third event `c` causes both `a` and `b`. According to Jensen et al. [18], the causal inference that `a` causes `b` relies on two other classical conditions: (i) the direction of causality, and (ii) the lack of common causes of `a` and `b`. Although it is hard to define common causes of two log events, we can take the event temporal order shown in the FESs as the dependency direction based on the standpoint of causality theory [28]: "if `a` causes `b`, then `a` occurs earlier than `b`". This means the cause of an event is a preceding event.

As for causality mining, classical Bayesian network learning uses powerful algorithms, and many such learners produce good results on benchmark datasets. Unfortunately, using current Bayesian network learning is infeasible for extracting a reasonable structure from the huge number of events in a log: the number of possible network structures grows super-exponentially with the number of events. Nevertheless, event correlations *can* be deduced from parallelized data mining techniques [30]. Here we explore a new way to obtain causal dependencies by means of FESs.

A. Terminology

To clarify concepts, we first review some notation. We leverage the LogMaster [11] event correlation mining framework, adopting its preprocessing phases, FES mining, prediction model construction, and failure prediction approach. In LogMaster, system log messages with multiple fields (i.e., timestamp, node ID, severity degree, event type, or application type) are parsed into event sequences. If a new 3-tuple (severity degree, event type, keywords) is reported, a new *event ID* will be assigned to the event. The operational states of a running system are represented by a sequence of *log ID*s. Note that events are represented by log IDs in the form of natural numbers, and events with different log IDs can have the same event IDs. Logmaster mines frequent log ID sequences representing events that co-occur often. The frequency of co-occurrence of two events in a given sequence is the *support count*. Here we also use LogMaster's event IDs and log IDs, but we rename log ID sequences to be Frequent Event Sequences (FESs).

B. Causal dependency graphs

Each system state represented in an event log induces the following state, and so on, up to failure. Implicit and dynamic event correlations are formally described by the FESs mined from the logs. Event correlations indicate that events occurring early in an FES cause events occurring later and that these patterns occur often in the log. The FESs reflect the temporal order of events in a segment of the complete log. Combining FESs having common events forms an event *Causal Dependency Graph* (CDG). The example in Fig. 1 shows FESs mined from log transactions. Here, FES {a→d→c} appears three times and FES {d→e→m} appears twice in close proximity. The "→" represents "happened before" relations between events. Because these two FESs have a common event, `d`, they are clustered together. The resulting CDG for these FESs is shown at the bottom of the figure.

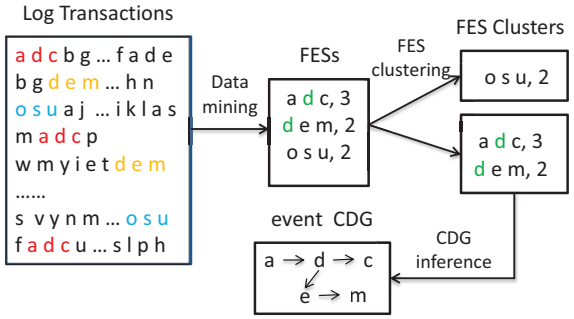

Fig. 1: Overall approach

C. Failure rules

The FESs mined directly from the logs are insufficient by themselves to achieve high failure prediction recall rates, since not all of the FGPs can be mined as FESs. This difficulty in identifying FESs may be because: (a) breaking the logs into discrete windows to form transactions cuts off some FESs; and (b) low support counts in some log segments cause weakly correlated FESs to be ignored, even if they are strongly correlated elsewhere in the log.

Close examination of the detailed information of each FES cluster reveals identical semantics between different nodes. The example shown in Table I displays information from part of the events from an FES cluster mined from the Hadoop system log. Grouping events by node ID reveals identical semantics among groups. For example, event group {682, 683} in node 69 has the same event IDs, application types, and keywords as event group {3162, 3170} in node 237 and event group {3749, 3752} in node 278. The identical semantics indicates that the three nodes share the same FGP ("EXT3- fs error" causes "failed to read"). Checking the log trace reveals that this FGP is also shared by other nodes in this cluster.

The system log messages comprise streams of interleaved events that may have resulted from many FGPs. FGPs can be extracted from corresponding FESs in each cluster. We find that different log ID orders in different FESs may have similar event ID orders. Take Table I in Section III-C as an example: although event log ID order {2542→3162} and event log ID order {2542→682} appear in different FESs, they share an event ID order {138→6}. This event ID order is just one of the FGPs in this cluster. We extract such cause and effect relationships among event IDs as failure rules. By following these failure rules, we can reconstruct the event log ID order from the root cause to the failure event to create new FESs that improve failure prediction recall rates. Here we would generate the FES {683→682}, {3170→3162}, {3752→3749}.

III. System Design

We introduce the framework of our technique in Section III-A and describe event CDG mining in Section III-B, including FES clustering and event CDG construction. We present our failure rule extraction-based failure prediction in Section III-C. We also demonstrate precise root cause diagnose from event CDGs in Section III-D.

A. Framework

The hybrid integrated system architecture is shown in Fig. 2; uncolored components represent our new approach, which consists of the following steps. First, we identify *FES clusters* and *event groups*. Second, we reconstruct event log ID orders in each group to form event CDGs from the temporal order of events in the FESs. Third, we extract specific event ID orders from the CDG as a failure rules. We derive additional failure FESs. We then use this rich information for failure prediction and root cause diagnosis.

B. Event CDG mining

It is unreasonable to assume that all events in a system log would be statistically correlated such that a single (big) event CDG could be deduced. In fact, there will be a set of event CDGs. Here we explore a new way to establish these event CDGs from FESs. First, we cluster FESs to form statistically correlated event groups, and then we deduce the dependency of each event pair from the temporal order of the FESs.

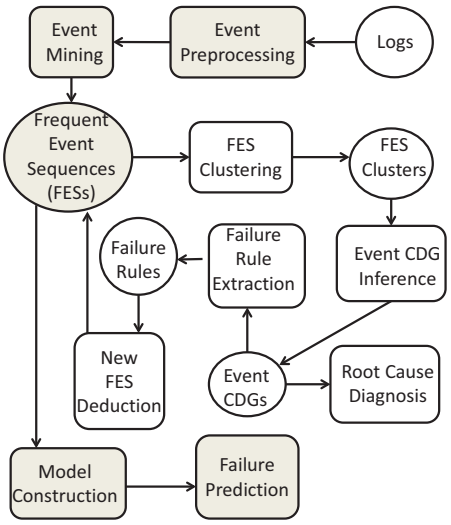

Fig. 2: Framework

1) FES clustering: The presence of common events among different FESs in the data-mined collection means that some FESs are correlated, and these correlated events are statistically dependent. Clustering partitions the FESs into event groups having like features (where an event group constitutes all events in an FES cluster). Here we cluster correlated FESs according to similarity, or the number of common events between FESs. Note that our frequent event sequence clustering differs from previous event filtering work in which events are assumed to be temporally and spatially redundant and thus can be represented by just a few events [17], [19], [21]. Our event clustering also differs from semantic based clustering techniques that extract message templates [3], [6].

We represent each FES, s, by a tuple $(f, \overrightarrow{fes})$, where f is the frequency of this pattern and $\overrightarrow{fes}$ is an event vector represented as $\overrightarrow{fes} = \{e_1, e_2, \ldots, e_n\}$. Here e_i is the event log ID in the ith position, and n is the number of events in this pattern. The set of FESs is $S = \{s_1, s_2, \ldots, s_m\}$, where m is the number of FESs in the total collection. The problem is to discover a set of FES clusters $C = \{c_1, c_2, \ldots, c_k\}$ (where k is the number of clusters ($k << |S|$)) and to map each FES $s(f, \overrightarrow{fsp})$ to one of the clusters.

Formally, similarity is defined by Eq. 1:

$$similarity(\overrightarrow{fes1}, \overrightarrow{fes2}) = \frac{n_{both}}{\sqrt{n_1 * n_2}} \quad (1)$$

Here n_{both} is the number of common events in $\overrightarrow{fes1}$ and $\overrightarrow{fes2}$, and n_1 and n_2 are the numbers of events in each pattern. We define a set of *event groups* $E = E_1, E_2, \ldots, E_k$ (k is again the number of clusters) and map each e_i to a group.

Our FES clustering approach is shown in Algorithm 1. We begin with an empty set of clusters. We compare each new FES to representative FESs in the current clusters in the order in which the clusters were created, and we assign the new

FES to the first cluster that exceeds the similarity threshold. If none of the clusters surpasses this threshold, we create a new cluster and use the FES as its representative. The events in each cluster constitute an event group.

Algorithm 1 FES clustering

Input: A set of m FES sequences $s(f, \overrightarrow{fes})$: S, SimilarityThreshold
Output: A set of k sequence clusters: C, a set of k event groups: E

```
1. Initialize C = {}, E = {};
2. for all sequence s(f, fes) in S do
3.   for all C{i} in C do
4.     if similarity(fes, C{i}) > SimilarityThreshold then
5.       Add s(f, fes) to C{i};
6.       Add {e_1, e_2, ..., e_n} from s(f, fes) to E{i};
7.       break;
8.     end if
9.   end for
10.  if no cluster for s(f, fes) then
11.    Add fes as a new cluster in C;
12.  end if
13. end for
14. return C, E
```

2) Event CDG inference: After clustering FESs, we look for statistically correlated events in each event group. Each FES is just a segment of the desired complete event CDG, but we can combine all FESs in each cluster to form a complete CDG. In other words, we can take the event temporal orders of the FESs as the dependency directions.

For each FES in each cluster, we generate a directed edge from a precursor event `a` to a successor event `b` in the form of an event transition {a→b} (i.e., $e(a, b)$) indicating that `a` (potentially) causes `b`. Also, the support count is assigned as a weight on the edge. We then form a graph from all the event pairs, as in Fig. 1. Formally, the CDG for a given FES cluster can be described as a weighted directed acyclic graph $G(V, D, w(e(i, j)))$, where the vertices V represent the set of all event log IDs in the given cluster, the directed edges D represent the causal dependencies between events, and the weights $w(e(i, j))$ represent the numbers of co-occurrences of the vertex events e_i and e_j.

Our inferral method may introduce cycles in the graph. For example, both event transition {a→b} and {b→a} may appear in an FES cluster, which means that the event order is not fixed and the presence of both transitions violates cause-and-effect theory. To prune such edges, we examine their support counts. A low support count indicates a weak statistical dependency between the events, and the co-occurrence may merely be coincidence. We thus remove such edges from the graph. Algorithm 2 shows our CDG inference algorithm. The advantages of using event CDGs over FESs lie in the completeness and correctness of the deduced event dependencies.

Algorithm 2 Event causal dependency inference

Input: A set of k FES clusters: C, a set of k event groups: E
Output: Directed causal graph $G = (V, D, w(e(i, j)))$ where V is the set of events with $|V| = k$ and an edge $e(i, j) \in D$ indicates e_i is a cause of e_j with weights $w(e(i, j))$ as support counts.

```
1. Initialize D = {}; V_k = E_k;
2. for all sequence s(f, fes) in C do
3.   for all events e_i and e_j with w(e(i,j)) in fsp do
4.     Add e(i,j) with w(e(i,j)) to D{k};
5.   end for
6.   for all cyclic edge in D{k} do
7.     Delete e(i,j) with min w(e(i,j)) from D{k};
8.   end for
9. end for
10. return G
```

C. Failure-rule based prediction

Different workloads may generate different FESs, which, in turn, lead to different FES clusters that reveal different FGPs. El-Sayed et al. [8] note that a failure significantly increases the probability of a follow-up failure of the same event type, and thus failure prediction should consider these event types. In fact, any repeated event (not just a failure event) increases the probability of a follow-up event of the same type, from the same node ID, or from the same application. The order of event types is thus critical for failure process identification.

Examining each FES cluster reveals that many FESs have the same event type order (here we represent the event type order by the event ID order). Consider Table I: log ID sequence {2542→3162} appears multiple times, and the event ID order of this sequence {138→6} represents an FGP. We therefore call this event ID order a *failure rule*. Similar event IDs occur on other nodes. We can therefore apply this event ID order to those nodes to reconstruct the event log ID order, from which we can deduce potential new FES patterns.

1) Failure rule extraction: Next we extract failure rules from the CDG of each event group. From the log ID of a failure event in a given event group, we trace back in the corresponding CDG to locate the precursor events of the same node ID or application type. We define a potential failure rule for a given failure event to be the order of event IDs from the first precursor event to each successive event up to the failure. We repeat this procedure to generate potential failure rules for all failure events in the given event group.

In Table I, the FES {3170→3749→3162} appears multiple times. Tracing backwards from the corresponding subgraph in the CDG reveals that event 3170 in node 237 causes event 3162 in the same node. The event ID order of the sequence {138→6} is thus a potential failure rule. From the FES {458→21→3162}, event 458 from application "smartd" causes event 3162 from the same application. The event ID order of the sequence {1→6} is also a potential failure rule.

log ID	node ID	event ID	application	severity	eventtype	keywords
21	0	137	kernel	failure	link	Link is Down
458	50	1	smartd	error	filesystem	unreadable
631	64	8	smartd	warning	filesystem	FAILED SMART
680	69	1	smartd	error	filesystem	unreadable
682	69	6	smartd	failure	filesystem	failed to read
683	69	138	kernel	error	filesystem	EXT3-fs error
685	69	21	sshd	warning	memory	Address already in use
689	69	8	smartd	warning	filesystem	FAILED SMART
693	69	145	kernel	failure	memory	Memory for crash
697	69	225	kernel	error	hardware	ec object not found
1808	150	8	smartd	warning	filesystem	FAILED SMART
2498	195	13	ntpd	error	filesystem	can't open
2542	197	138	kernel	error	filesystem	EXT3-fs error
3161	237	2	smartd	error	filesystem	uncorrectable
3162	237	6	smartd	failure	filesystem	failed to read
3170	237	138	kernel	error	filesystem	EXT3-fs error
3749	278	6	smartd	failure	filesystem	failed to read
3752	278	138	kernel	error	filesystem	EXT3-fs error

TABLE I: A motivating example from the Hadoop log

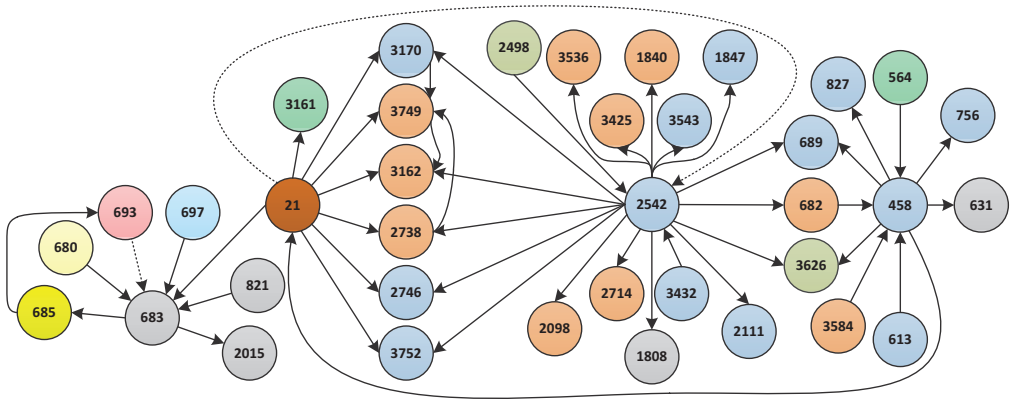

Fig. 3: Part of an event CDG from the Hadoop log

Following every dependency path of a given failure event may yield several failure rule candidates. The candidates that are useful for failure prediction are deemed true failure rules. Those that are useless and should be discarded are false failure rules. We apply two heuristics to remove false candidates.

First, the time interval between the precursor and the failure event of each candidate should not exceed the *prediction valid duration* predefined by system administrators [11]. Longer intervals make it harder to identify causality among events and are not proper for failure prediction. Second, our method examines whether a given failure rule candidate can represent the FGP of other failure events of the same event ID. This requires that we examine the whole log sequence to determine whether these failure events have precursor events with the candidate's event type. Specifically, we define the term *true positive* (*TP*) and *false negative* (*FN*) for a given failure rule candidate as follows: TP is the number of events in the entire log that have the candidate's event type and that lead to failure events with the candidate's failure event type. FN is the number of events in the entire log that have the candidate's event type but do not lead to failure events with the candidate's failure event type. Eq. 2 defines the *confidence* of a given candidate in terms of TP and FN.

$$confidence = \frac{TP}{TP + FN} \tag{2}$$

Failure rule candidates whose confidences are lower than a predefined threshold are removed as false candidates. For example, the confidence of failure rule $\{1 \rightarrow 6\}$ is lower than that of failure rule $\{138 \rightarrow 6\}$ and therefore $\{1 \rightarrow 6\}$ is more likely to be removed as a false candidate. This confidence metric is also useful for reducing the true negatives in failure prediction. We explain the choice of this threshold in the following section.

2) New FES deduction: Having extracted failure rules to represent the FGPs for each failure event type, we deduce new potential failure FESs from each failure rule. These new FESs can be added to the FES database [11] for further model construction and failure prediction. Apparently mining more FESs covering more failures improves prediction recall.

In order to deduce potential failure FESs from an FES cluster, we first search all events with the same event ID of

a given failure rule and then order events for the same nodes or same applications in a sequence according to the event ID direction from the given failure rule. This creates new FESs.

Again taking Table I as an example, after we extract the failure rule {138→6}, searching for events with failure event type 6 finds the event group {3162,3749,682}. If we search for events of type 138, we find the event group {683,3170,3752}. We then reconstruct the order of event log IDs from the event type order {138→6} to form the new potential patterns {683→682}, {3170→3162}, and {3752→3749}.

For simplicity, we set the support count of each new FES to be the average of the support counts of the corresponding FESs from which the failure rule is extracted.

D. Root cause diagnosis

Unlike approaches that rely on window generation [35], [16], [20], our technique associates events by globally clustering FESs to infer event CDGs *without any window-size constraint*. Thus we have effectively extended the window to be the length of time between the first and last events in each CDG. We find this approach to be simpler and more robust.

Next we want to help administrators diagnose the root cause of each failure event from the CDGs. Locating the cause here is quite different from identifying triggering events in failure prediction because the latency between cause and failure may well exceed the prediction valid duration [11]. The chain of dependencies between the triggering event and the failure event may be long and have a low support count. Hence, we need to explore dependency paths that are as long as possible, considering many event co-occurrences with no cause-and-effect relationships. When tracing back in the CDGs, we may find multiple follow-up events. It is reasonable to choose events with the same node ID or from the same application as the precursor. When multiple events in the CDG share an event type, we need only trace one. Careful manual inspection is required to locate the actual trigger event.

IV. Experiments

In this section, we use our approach to analyze three real logs generated by a production Hadoop system, a Los Alamos National Lab (LANL) system, and a BlueGene/L system. The first set of logs is not publicly available due to privacy issues. The second and third sets are openly available; we take previous analysis on them as our baseline for comparison. Table II gives a summary of these system logs.

Our method uses LogMaster [11] to improve failure-prediction recall. Fig. 2 illustrates our approach; uncolored components represent our new approach. In this work, we process data-mined FESs to extract richer information on which to predict failures. We also present case studies demonstrating our root-cause diagnosis approach.

A. FES dataset description

We use LogMaster to parse and filter the raw logs and then conduct our experiments on its intermediate results. We use the first two thirds of the three logs for mining FESs and the last one third for predicting failures. Table III shows details of each set with respect to the numbers of FESs and the numbers of included log IDs, event IDs, node IDs, and application IDs. Our event preprocessing finds 2,878 log IDs for the Hadoop system, 3,665 for the LANL system, and 99,183 for the BlueGene/L system. LogMaster mines 5,247, 16,433, and 220,956 patterns for each system, respectively.

Properties	Logs		
	Hadoop	LANL	BlueGene/L
Size (MB)	130	31.5	118
Record count	977,858	433,490	4,399,503
Days	67	1,005	215
Start Date	2008-10-26	2005-07-31	2005-06-03
End Date	2008-12-31	2006-04-04	2006-01-02

TABLE II: Log summary

Results	Logs		
	Hadoop	LANL	BlueGene/L
# log ids	2,878	3,665	99,183
# failures logid	543	275	15,523
# nodes	273	274	69,244
# event id types	402	36	74
# application types	42	16	12
# frequent sequences	5,247	16,433	220,956

TABLE III: Summary of frequent sequences in the three logs

B. FES clustering results

We cluster the FESs mined by LogMaster according to Algorithm 1. We characterize the similitude of patterns within a group according to sequence similarity, or the percentage of common events in any FES pair. The similarity threshold should be chosen carefully. Increasing this parameter creates more clusters of smaller size, which may lead to the discovery of fewer failure rules. Decreasing it creates fewer large clusters; in the limit, a zero threshold causes our algorithm to yield just one cluster that includes all patterns. In our experiments, we set this parameter to 0.1 in order to generate bigger event groups, which helps to improve the recall rate in failure prediction. We end up with 216, 89, and 125 event groups for the Hadoop, LANL, and BlueGene/L systems, respectively.

Recall that Table I shows part of an event group from the Hadoop system. Table IV and Table V show examples from the other two systems. A closer look at these event groups in the three logs reveals that there are few event IDs in each group, but the number of log IDs varies from dozens to hundreds. This indicates that many events share the same event types within each group, which results in a small number of event IDs. Events on different nodes may have identical semantics, which indicates that these nodes experience similar run-time behaviors (and perhaps suffer the same FGP). This is why we conduct FES clustering before CDG deduction.

C. CDG inference results

Next we infer an event CDG from each FES cluster. We generate the graph from all event transitions in an FES cluster acording to Algorithm 2, removing edges with minimum

log ID	node ID	event ID	severity	keywords
3613	266	23	fatal	no clusterfilesystem server
3615	266	29	warning	ServerFileSystem full
3616	266	30	fatal	ServerFileSystem not served
3617	266	31	warning	node inconsistent
3619	267	18	info	responding
3622	267	30	fatal	ServerFileSystem not served
3623	267	31	warning	node inconsistent
3639	270	29	warning	ServerFileSystem full
3640	270	30	fatal	ServerFileSystem not served
3641	270	31	warning	node inconsistent
3657	273	23	fatal	no clusterfilesystem server
3659	273	29	warning	ServerFileSystem full
3660	273	30	fatal	ServerFileSystem not served
3661	273	31	warning	node inconsistent

TABLE IV: Part of an event group from the LANL log

log ID	node ID	event ID	severity	keywords
24	16	73	fatal	KERNFPLB
800	1445	73	fatal	KERNFPLB
919	1634	58	failure	MONTEMP
920	1635	53	warning	DISWARN
923	1635	56	warning	HARDWARN
1160	1938	54	error	DISERROR
1494	2546	53	warning	DISWARN
1496	2546	55	Severe	DISSEVE
1969	3259	56	warning	HARDWARN
1976	3261	55	severe	DISSEVE
2114	3529	40	failure	MONILL
2115	3529	53	warning	DISWARN

TABLE V: Part of an event group from the BlueGene/L logs

support counts to eliminate cycles. Fig. 3, Fig. 4, and Fig. 5 show the event CDGs for our three examples. Pruned edges are represented by dotted lines, and events with the same event ID are the colored alike. These CDGs show that among the nodes there are a limited number of event ID orders that represent the run-time behaviors and even FGPs well. We find that the depths of the CDGs are relatively short considering the number of vertices. The CDGs thus exhibit a "small-world phenomenon" [29] where several vertices are connected to many others. Further, examining the node IDs of these central events reveals that these server nodes fail more frequently than others. This may indicate that such nodes are at the heart of failure propagation and thus merit attention in failure analysis.

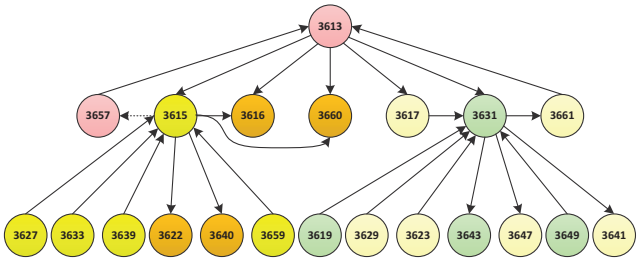

Fig. 4: Part of an event CDG from the LANL log

D. Failure rule extraction and new FES deduction

In the experiments to extract failure rules from each event CDG, our approach first gathers all failure rule candidates

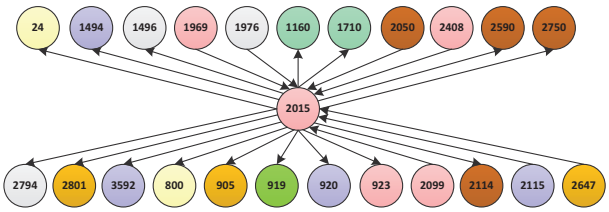

Fig. 5: Part of an event CDG from the BlueGene/L log

represented by event ID orders for a given failure event. To remove the false candidates, we apply our two heuristics in separate elimination rounds. The first round eliminates false candidates based on the prediction valid duration, which we set to 20 minutes based on empirical observation. The second round eliminates candidates according to their confidence values (Eq. 2). Lower values generate larger numbers of failure rules and new FESs, which tends to improve the failure prediction recall rate but may reduce the precision by introducing false positives. We therefore prefer lower values, and we set the minimum value to be 0.2.

We scan the entire log sequence to calculate 1) the event counts of the same event ID as the precursor of the candidate and 2) the counts of event IDs leading to failures with the same failure event ID as the candidate within a 20-minute interval. Table VI summarizes the numbers of failure rules after candidate elimination for the three datasets. We then derive 1) new potential failure FESs based on all failure rule candidates and 2) new true failure FESs based on just the true failure rules. Table VI summarizes the results.

Results	Logs		
	Hadoop	LANL	BlueGene/L
# of candidates	35	8	65
# after 1st round	21	4	34
# after 2nd round	16	4	12
# of potential FESs	1,200	587	12,384
# of true FESs	320	213	3,456

TABLE VI: Failure rule extraction and FES deduction results

E. Failure Prediction

After deducing potential new failure FESs, we consider their suitability for predicting events. A standard way to measure the effectiveness of failure prediction is by calculating the precision and the recall rate. Precision is defined as the ratio of correct predictions to all predictions made, and recall is the ratio of correct predictions made to all possible predictions. A good prediction has a high value (closer to 1.0) for both precision and recall. However, there is often an inverse proportionality between high recall and high precision. Improving recall in most cases lowers precision and vice versa. Bouguerra et al. [4] suggest that when failure predictors provide flexible precision/recall tradeoffs, one should favor first high recall over to high precision.

We evaluate our failure prediction two ways: by predicting failures on the basis of potential FESs (Scenario 1), and by

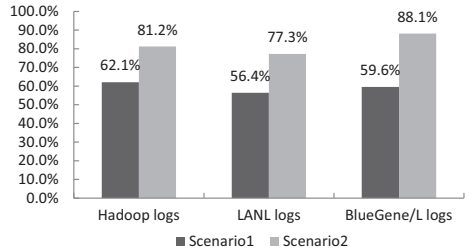

Fig. 6: Precision of the three logs

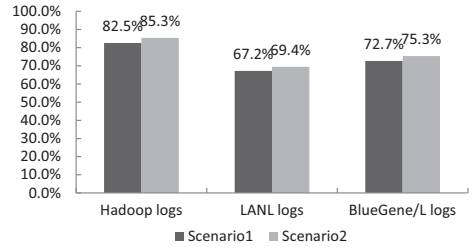

Fig. 7: Recall rates of the three logs

predicting failures on the basis of only true FESs (Scenario 2). Fig. 6 and Fig. 7 show precision and recall results, respectively. Precision results for Scenario 1 are about 20% below those for Scenario 2, and the recall rate increases by an unacceptably low 2-3%. This demonstrate the effectiveness of our culling of failure rules that we deem of little use.

Fig. 8 and Fig. 9 compare our results with those from previous work [12]. Here "dynamic" represents the work of Gu et al. [15], and "adaptive" represents the work of Gainaru et al. [12]. Not all papers experiment with all logs, so we only show the results we can obtain. By comparison, our approach delivers much better recall rates (as high as 85.3%, 69.4%, and 75.3%), improving 8-15% over prior approaches. Furthermore, it does so with little loss in precision.

F. Root cause diagnosis

We examine the effectiveness of our root cause diagnosis based on event CDGs with three case studies from the three logs. In the Hadoop system CDG subgraph in Fig. 3, there are multiple paths we can follow to trace back from the failure event 697 to event 685 and then to event 683. We choose event 21 for in-depth tracing and finally draw two cause-and-effect

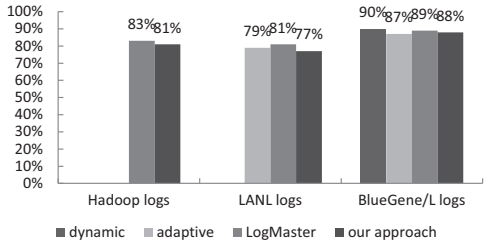

Fig. 8: Comparative precisions

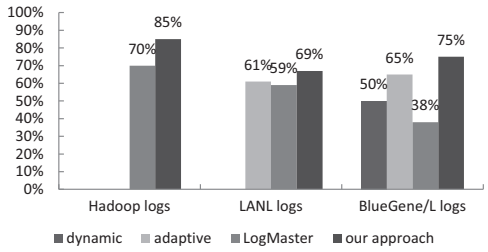

Fig. 9: Comparative recall rates

paths: {2498→2542→682→458→21→683→685→693} and {697→683→685→693}. Analyzing these events in detail reveals that error event 697 with the message "ec object not found" in the second path is more causally related to failure event 693 with the message "Memory for crash", and so we choose it for further diagnosis. In the LANL system CDG subgraph from Fig. 4, we can draw a cause-and-effect path {3661→3613→3615→3660} from warning event 3661 with keywords "node inconsistent" to failure event 3660 with keywords "ServerFileSystem not served". In the BlueGene/L system CDG subgraph from Fig. 5, we can draw a cause-and-effect path {2115→2015→2114} from warning event 2115 with keywords "MONILL" to failure event 2114 with keywords "DISWARN". These dependency paths can help in understanding system operational behaviors.

These cause-and-effect paths are often deduced from multiple FESs with a large time gap and cannot be directly derived from simple FESs. This is consistent with our previous work [11] and our initial idea that a better approach should take global information into account to identify triggering events. We find that choosing events with the same node ID and from the same application as the precursor event when analyzing root causes of failures works well for a given event type.

V. Related Work

We briefly discuss related work on event clustering, failure prediction, event causality inference, and root cause diagnosis.

1) Event clustering: Several previous efforts cluster temporally and spatially correlated events to reduce log size and to perform statistical failure correlation. Sahoo et al. [24] observe that failure events are strongly correlated spatially. Schroeder et al. [25] analyze logs from two HPC installations to study statistics like root cause of failures, mean time between failures, and mean time to repair. Liang, et al. [19] filter duplicate failure events for a given location and coalesce failure reports across different locations. Their approach removes 99.96% of the entries from the logs of an 8192 processor BlueGene/L prototype. Taerat et al. [26] also filter repeated events and then analyze the compressed logs to predict application time to interrupt. Hacker et al. [17], present a model for predicting the probability of node failure and assess the effects of differing rates of failure on job failures. Yigitbasi et al. [32] investigate the time correlation of failures

to derive a model that characterizes duration of peak failure periods, peak inter-arrival time, inter-arrival time of failures during peaks, and duration of failures during peaks.

Gainaru et al. [14] propose a novel, signal analysis approach to characterize normal and faulty system behaviours. Tati et al. [27] develop algorithms to diagnose large-scale, clustered failures from incomplete symptoms and to more accurately diagnose both independent and clustered failures. Lim et al. [21] mine logs of large enterprise telephony systems. They de-parameterize messages and cluster them according to Levenshtein distance to reduce the number of unique messages to less than 1% of the original set. Chen et al. [5] also categorize failures in enterprise telephony logs, hierarchically clusteringing messages with the same semantic form. Several prior efforts employ context-based clustering after extracting message templates [3], [6], [13] or statistics-based clustering for proactive fault management [35], [16], [10], [9].

2) Failure prediction: Many studies adopt statistical techniques to predict failures. Liang et al. [20] compare using a rule-based classifier, Support Vector Machines, a traditional nearest-neighbor method, and a customized nearest-neighbor method, finding that the latter performs best with respect to both recall and precision. Gujrati et al. [16] and Gu et al. [15] combine statistical methods with meta-learning techniques to boost failure prediction accuracy. Several previous studies mine frequent sequences for failure prediction [11], [12], [34], [31], as we do here. Zheng et al. [36] improve on widely used metrics for failure prediction by including location and lead-time information. Yu et al. [33] compare period-based and event-driven prediction approaches via a Bayesian prediction model, finding that the latter has higher accuracy. Unfortunately, even when highly accurate, all these previous prediction methods suffer from low recall.

3) Event causal dependency inference: Mahimkar et al. [23] organize correlated events into a causality graph using the timing information among events. Zheng et al. [35] propose apriori association rule mining and identify parameters to measure whether events are causality-related. Zheng et al. [37] perform causation pruning via co-analysis of multiple logs.

4) Root cause diagnosis: Present root causes diagnosis techniques [2], [6], [37] mostly associate events correlated with the failure by statistical analysis and some typical heuristics. Chuah et al. [6] use statistical correlation analysis to establish probable cause-and-effect relationships. Agarwal et al. [2] develop an approach to identifying failure causes resulting from system change events resulting from system administrator attempts to fix existing problems. Both Agarwal et al. [2] and Zheng et al. [37] acknowledge the insuitability of statically chosen window sizes, given that the time between the cause event and the ensuing failure may be unbounded. Zheng et al. [37] identify the failure layer, the time, and the location of the problem-causing event by employing a dynamic window-generation scheme that tunes the tracing window based on event correlation analysis.

VI. Conclusion

We propose a methodology for failure prediction and root cause diagnosis, and we demonstrate its usefulness on logs from three production large-scale systems. We mine Frequent Event Sequences, which we cluster into event groups. Then we infer causal dependency between events in each group. We extract failure rules based on the observation that events on the same nodes or from the same applications have similar operational behaviors. We then derive new potential Frequent Event Sequences from identified failure rules.

We compare the effectiveness of our event CDG-based approach against two classical algorithms, finding that on the logs from a Hadoop system, a LANL system, and a BlueGene/L system, the average precision rates are as high as 85.3%, 69.4%, and and 75.3%, respectively. In addition, we improve recall by 8-15%. Furthermore, our technique can automatically draw long event dependency paths for precise root cause diagnosis without any window-size constraint.

We are currently working to streamline our system to better meet the demands of the process of large-scale log data in online failure prediction and root cause diagnosis. And we are continuing to study improved causal dependency inference for more precise root cause diagnose and system behavior understanding.

References

[1] N. Adiga *et al.*, "An overview of the BlueGene/L supercomputer". In *Supercomputing*, Nov. 2002 p. 60.

[2] M. Agarwal and V. Madduri, "Correlating failures with asynchronous changes for root cause analysis in enterprise environments". In *International Conference on Dependable Systems and Networks*, June 2010, pp. 517-526.

[3] M. Aharon, G. Barash, I. Cohen, and E. Mordechai, "One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs". In *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases*, 2009. Volume 5781, 2009, pp 227-243

[4] M. Bouguerra, A. Gainaru, L. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing". In *International Parallel and Distributed Processing Symposium*, May 2013, pp. 501-512.

[5] C. Chen, N. Singh, and S. Yajnik, "Log analytics for dependable enterprise telephony". In *European Dependable Computing Conference*, May 2012, pp. 94-101.

[6] E. Chuah, S. Kuo, P. Hiew, W. Tjhi, G. Lee, and J. Hammond, "Diagnosing the root-causes of failures from cluster log files". In *High Performance Computing* , Dec. 2010, pp. 1-10.

[7] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, J. Browne, and B. Barth, "Linking resource usage anomalies with system failures from cluster log data". In *International Symposium on Reliable Distributed Systems*, Oct. 2013, pp. 111-120.

[8] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how HPC systems fail". In *International Conference on Dependable Systems and Networks*, June 2013, pp. 1-12.

[9] S. Fu and C. Xu, "Quantifying temporal and spatial correlation of failure events for proactive management". In *International Symposium on Reliable Distributed Systems*, Oct. 2007, pp. 175-184.

[10] S. Fu and C. Xu, "Exploring event correlation for failure prediction in coalitions of clusters". In *International Conference on Supercomputing*, Nov. 2007, pp.1-12.

[11] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "LogMaster: mining event correlations in logs of large-scale cluster systems". In *International Symposium on Reliable Distributed Systems*, Oct. 2012, pp. 71-80.

[12] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer, "Adaptive event prediction strategy with dynamic time window for large-scale HPC systems". In *SOSP Workshop on Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, Oct. 2011, p. 4.

[13] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, and Bill Kramer, "Event log mining tool for large scale HPC systems". In *Euro-Par*, Aug. 2011, pp. 52-64.

[14] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: modeling the normal and faulty behaviour of large-scale HPC Systems". In *International Parallel and Distributed Processing Symposium*, May 2012, pp. 1168-1179.

[15] J. Gu, Z. Zheng, and Z. Lan, "Dynamic meta-learning for failure prediction in large-scale systems: a case study". In *International Conference on Parallel Processing*, Sept. 2008, pp. 157-164.

[16] P. Gujrati, Y. Li, and Z. Lan, "A meta-learning failure predictor for Blue Gene/L systems". In *International Conference on Parallel Processing*, Sept. 2007, p. 40.

[17] T. Hacker, F. Romeroa, and C. Carothers, "An analysis of clustered failures on large supercomputing systems". In *Elsever Trans. Journal of Parallel and Distributed Computing*, 69(7):652 July 2009.

[18] D. Jensen, A. Fast, B. Taylor, and M. Maier, "Automatic identification of quasi-experimental designs for discovering causal knowledge". In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Aug. 2008, pp. 372-380.

[19] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a BlueGene/L prototype". In *International Conference on Dependable Systems and Networks*, June 2005, pp. 476-485.

[20] Y. Liang, Y. Zhan, H. Xiong, and R. Sahoo, "Failure prediction in IBM BlueGene/L event logs". In *International Conference on Data Mining*, Oct. 2007, pp.583-588.

[21] C. Lim, N. Singh , and S. Yajnik, "A log mining approach to failure analysis of enterprise telephony systems". In *International Conference on Dependable Systems and Networks*, June 2008, pp. 398-403.

[22] The Computer Failure Data Repository: The LANL data. http://www.usenix.org/node/167208.

[23] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large IPTV network". In *of ACM SIGCOMM Conference on Data Communication*, Oct. 2009, pp. 231-242.

[24] R. Saho, A. Sivasubramaniam, M. Squillant, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment". In *International Conference on Dependable Systems and Networks*, July 2004, pp. 772-781.

[25] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems". In *International Conference on Dependable Systems and Networks*, June 2006, pp. 249-258.

[26] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostroucho, S. Scott, and C. Engelmann, "Blue Gene/L log analysis and time to interrupt estimation". In *International Conference on Availability, Reliability, and Security*, March 2009, pp. 173-180.

[27] S. Tati, B. Ko, G. Cao, A. Swami, and T. La Porta, "Adaptive algorithms for diagnosing large-scale failures in computer networks". In *International Conference on Dependable Systems and Networks*, June 2012, pp. 1-12.

[28] M. Tooley, "Causation: a realist approach". *Oxford: Clarendon Press*, 1987.

[29] D. Watts and S. Strogatz, "Collective dynamics of small-world networks". *Nature*, vol. 393, June 1998, pp. 440-442.

[30] G. Wu, H. Zhang, M. Qiu, Z. Ming, J. Li, and X. Qin, "A decentralized approach for mining event correlations in distributed system monitoring". In *Elsevier Journal of Parallel and Distributed Computing*, 73(3):330-340, Mar. 2013.

[31] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs". In *IEEE International Conference on Data Mining*, Dec. 2009, pp. 588-597.

[32] N. Yigitbas, M. Gallet, D. Kondo, A. Iosup, and D. Epema, "Analysis and modeling of time-correlated failures in large-scale distributed systems". In *International Conference on Grid Computing*, Oct. 2010, pp. 65-72.

[33] L. Yu, Z. Zheng, Z. La, and S. Coghla, "Practical online failure prediction for Blue Gene/P: period-based vs event-driven". In *International Conference on Dependable Systems and Networks Workshops*, June 2011, pp. 259-264.

[34] D. Yuan, Y. Xie, R. Panigrahy, J. Yang, C. Verbowski, and A. Kumar, "Context-based online configuration-error detection". In *USENIX Annual Technical Conference*, June 2011, p. 28.

[35] Z. Zheng, Z. Lan, B. Park, and A. Geist, "System log pre-processing to improve failure prediction". In *International Conference on Dependable Systems and Networks*, June 2009, pp. 572-577.

[36] Z. Zheng , Z. Lan , R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for Blue Gene/P". In *International Conference on Dependable Systems and Networks Workshops*, June 2010, pp. 15-22.

[37] Z. Zheng, Li Yu, Z. Lan, and T. Jones, "3-Dimensional root cause diagnosis via co-analysis". In *International Conference on Autonomic Computing*, Sept. 2012, pp. 181-190.