# Diagnosing the root-causes of failures from cluster log files

**Conference Paper** · December 2010

DOI: 10.1109/HIPC.2010.5713159 · Source: DBLP



**9 authors**, including:

Edward Chuah
Lancaster University
**13** PUBLICATIONS **138** CITATIONS
SEE PROFILE

Shyh-Hao Kuo
Institute Of High Performance Computing
**20** PUBLICATIONS **132** CITATIONS
SEE PROFILE

Paul Hiew
National SuperComputing Centre
**1** PUBLICATION **26** CITATIONS
SEE PROFILE

William Tjhi
National University of Singapore
**25** PUBLICATIONS **345** CITATIONS
SEE PROFILE

# Diagnosing the Root-Causes of Failures from Cluster Log Files

Edward Chuah*, Shyh-hao Kuo*, Paul Hiew†, William-Chandra Tjhi*, Gary Lee*, John Hammond‡,
Marek T. Michalewicz†, Terence Hung*, James C. Browne‡
*Institute of High Performance Computing, †A*Star Computational Resource Center
*† 1 Fusionopolis Way, #16-16 Connexis, Singapore 138632
*Email: {chuahty, kuosh, tjhiwc, leekk, terence}@ihpc.a-star.edu.sg, †{hiewnh, Michalewiczm}@acrc.a-star.edu.sg
‡University of Texas at Austin, Texas 78712, Email: jhammond@math.utexas.edu, browne@cs.utexas.edu

***Abstract*—System event logs are often the primary source of information for diagnosing (and predicting) the causes of failures for cluster systems. Due to interactions among the system hardware and software components, the system event logs for large cluster systems are comprised of streams of interleaved events, and only a small fraction of the events over a small time span are relevant to the diagnosis of a given failure. Furthermore, the process of troubleshooting the causes of failures is largely manual and ad-hoc. In this paper, we present a systematic methodology for reconstructing event order and establishing correlations among events which indicate the root-causes of a given failure from very large syslogs. We developed a diagnostics tool, FDiag, to extract the log entries as structured message templates and uses statistical correlation analysis to establish probable cause and effect relationships for the fault being analyzed. We applied FDiag to analyze failures due to breakdowns in interactions between the Lustre file system and its clients on the Ranger supercomputer at the Texas Advanced Computing Center (TACC). The results are positive. FDiag is able to identify the dates and the time periods that contain the significant events which eventually led to the occurrence of compute node soft lockups.**

***Keywords*-Resilient cluster systems; Reliability; Syslog files; Statistical correlation analysis;**

## I. Introduction

Large supercomputers, particularly those using open source software, are more and more being constructed from many different hardware and software components. Failures arise both in single component failures and from interactions among components, particularly software components. Diagnosis of failures is mostly done manually by systems administrators examining system event logs and applying their knowledge of the system and its components to identify the causes of a given failure. Syslog event logs are a stream of interleaved events in diverse forms from many components and only a small fraction of the events over the surrounding time span are relevant to the diagnosis of a given failure [1], [2]. The process of identifying the events that caused the system to fail is an effort intensive and time-consuming task.

Recognizing the impact that failures have on the productivity of large computing clusters [3] and the burden system administrators have to shoulder, increasing attention has been dedicated to: (a) the study of system logs [4]–[6], (b) algorithms for clustering log messages [7]–[14], (c) log processing and management tools [15]–[18], (d) fault detection techniques [19]–[21], (e) failure prediction techniques [22]–[26], and (f) visualization techniques [27] to help system administrators in their diagnosis of system failures. All these activities point towards growing interest of the HPC and Knowledge Discovery research communities in this field and the contributions these works have made.

Most of the work cited above focused on addressing the problems that are related to specific activities in the system fault detection and failure prediction space. There is, however, little published work on methodologies and tools that capture fault diagnostics processes and help the system administrators diagnose the root-causes of failures [28], [29] in large high-performance computing clusters. In this paper, we present a systematic methodology to aid the administrators of large HPC cluster systems diagnose the root-causes of failures from very large system logs. We developed a tool called FDiag which comprises of three main components: (1) a *Message Template Extractor* (MTE) that adds structure to the logs by extracting the message templates and producing a standard data format for further processing, (2) a *Statistical Event Correlator* (SEC) that identifies the strongly correlated message templates for any given symptom event, and (3) an *Episode Constructor* that constructs and identifies the period of time (referred to as *episodes* here after) that contain the strongly correlated message templates. These templates correspond to system events that have occurred. By identifying these system events and the episodes that contain these events, the system administrator can then flesh out the events that are very likely to have led to the occurrence of the system failure. Applying FDiag to a real case study, we identified the dates and episodes that are related to Lustre filesystem problems that resulted in the occurrence of compute node soft-lockups - a commonly occurring problem encountered by the administrators of the Ranger supercomputer at TACC. The diagnostics process and the dates on which these episodes occurred have been validated by the administrators from TACC and ACRC. Our key contributions are:

- We combine three activities in the systems fault detection and failure prediction space, i.e.: *preprocessing*, *system event correlation* and *episode discovery* and developed

a fault diagnostics tool, FDiag, that directly supports the system administrators fault diagnostics processes. Then, we describe how the fault diagnostics processes can be automated by providing detailed descriptions of workflows for each activity.

- We show that FDiag can identify from very large system logs, the episodes that contain the events and the dates on which the occurrence of these events led to the occurrence of compute node soft lockups on the Ranger supercomputer. Furthermore, we also show that FDiag can identify the *significant* events from these episodes, despite the fact that the log entries in the Ranger syslogs were not tagged with any priority information.
- We show the usefulness of text strings in the log messages for fault and failure detection, specifically in the diagnosis of the root-causes of system failures in a large computing cluster system.

The remainder of this paper is organized as follows: In the background section, we present an overview of the manual fault diagnostics processes and the system context in which we are working with. Then, we present the details of our fault diagnostics tool FDiag, and this is followed by a case study where we demonstrate how FDiag has helped the Ranger system administrators identify the root-causes of compute node soft lockups. We present a detailed survey of the related work in this area and conclude with a brief outlook on future work.

## II. Background

The diagnosis of system faults begin when an anomaly is detected as depicted in Fig 1. The system administrator then attempts to find the cause (locating the fault) by tracing back the events leading up to this anomaly. Due to the size of the syslog files, the administrator does not read the log in a sequential manner. Instead, a common approach is to jump to a log entry that is consistent with the anomaly observed. For instance, in a system crash, the obvious starting point would be the last entry in the log before the crash. Alternatively, one could begin at specific time instances if these can be deduced from the anomaly. Once this log entry is found, there are two things to look for. Firstly, the sequence of log entries leading up to it, and secondly, other related log entries that are known to be associated with it. A general assessment of the sequence of events (episode) leading up to the fault is then hypothesized and tested until the administrator is confident in his diagnosis. A positive diagnosis generally requires assessment of multiple possibilities to arrive at the most likely scenario.

Therefore, the problem is to determine and present to the systems administrator, the events and correlations among the events which are required to enable the diagnosis of system failures. For the rest of this paper, we will refer to the sequence of log messages associated with the events caused by a failure as an episode. Thus, the job of diagnosing a failure is to extract this episode from the system logs and associate it with the cause of the failure, a process known as root-cause analysis. The aim of our methodology is to automatically extract

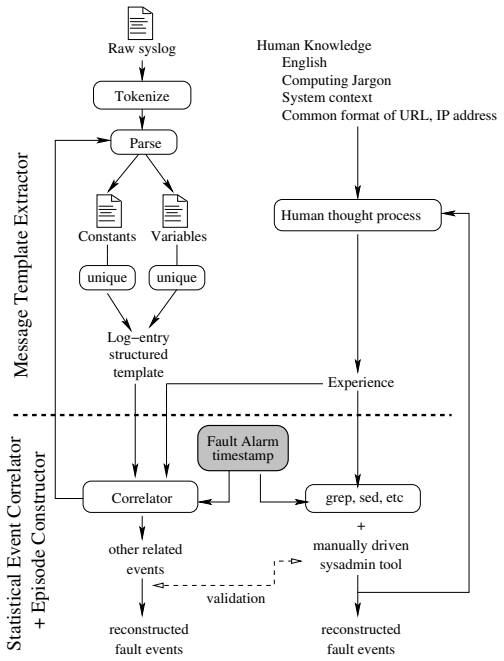



Fig. 1. General fault diagnostics workflow. On the right, the traditional manual process a system administrator goes through is shown. On the left, the processes of our methodology implemented by FDiag is shown.

and present the episodes to aid the system administrators in diagnosing a failure. In addition, the episodes may enable implementation of an automated online detection monitor. A typical process of extracting the episodes is shown in Fig 1. On the right hand side, we show the traditional manual process of extracting this information. This process is largely manual and extensive knowledge of the system is required. The target of our methodology is to capture the flow and semantics of the manual process in a fault diagnostics framework shown on the left hand side of Fig 1.

### A. System Context

Our work is carried out on the Ranger supercomputer at the Texas Advanced Computing Center (TACC)[1]. It is a Linux cluster comprising of 3,936 nodes (16 processors per node) each with its own Linux kernel. As such, all nodes are capable of generating log messages independent of each other. These messages are combined and interleaved in time as a centralized "syslog" file for collection purposes. However, due to the sheer number of nodes, two consecutive messages from a single node may be separated by thousands of other messages (from other nodes) in this interleaved format. This makes manual fault finding difficult as the system administrator can no longer easily *scan for correlated events* in the logs as these events are separated by thousands of other events.

As the system becomes more complex, a typical sequence of messages that leads up to a failure generally involve multiple nodes. Furthermore, such multi-node failures are the most difficult to diagnose manually. This motivates our approach

[1]Details of the Ranger supercomputer is available from the following URL: http://www.tacc.utexas.edu/resources/hpc/#constellation

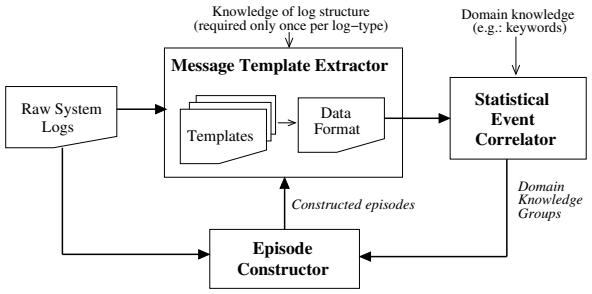



Fig. 2. FDiag Architecture

in extracting the temporal (across time) and spatial (across nodes) correlation between the log entries.

## III. FDiag: A Tool for Failure Diagnosis

Discovering the events that are associated with a system failure is a largely manual and time-consuming task. Hence the role of FDiag is to aid the systems administrators in diagnosing the root-causes of failures in large cluster systems. In this section, we describe in detail, the components in FDiag as shown in Fig 2 and how FDiag can be used. For the convenience of our reader, a summary of the terms and definitions[2] used in our paper is provided in Table I. FDiag will be available at the Institute of High-Performance Computing open-source webpage.

TABLE I
Definition of Terms used in this paper.

| Term | Definition |
|---|---|
| $AC_n$ | ASCII code number $n$ |
| Alphabet | $A = \{A-Z\} \cup \{a-z\}$ |
| Symbol | $S = \{AC_{33-47}\} \cup \{AC_{58-64}\} \cup \{AC_{91-96}\} \cup \{AC_{123-126}\}$ |
| Number | $N = \{0-9\}$ |
| Token | $T$, a sequence of $x \in A \cup S \cup N$ *and* $x \notin \{AC_{32}\}$ |
| Constant | $C$, a sequence of $x \in A \cup S$ *and* $x \notin \{AC_{32}\}$ |
| Variable | $V$, a sequence of $x \in A \cup S \cup N$ *and* $x \notin \{AC_{32}\}$ |
| Constants | $CS$, a sequence of $C_i$ such that $\forall C_i \in CS$ $1 \leq i \leq I$, each $C_i$ is delimited by $\{AC_{32}\}$ |
| Variables | $VS$, a sequence of $V_j$ such that $\forall V_j \in VS$ $1 \leq j \leq I$, each $V_j$ is delimited by $\{AC_{32}\}$ |
| Message | $M$, a sequence of $T$ such that $\forall T_i \in M$, each $T_i$ is delimited by $\{AC_{32}\}$ |
| Domain Knowledge Group | $DKG$, a set of pairs $\{(CS_{dkt}, CS_i)\}$ such that $CS_{dkt} \neq CS_i$ |

### A. Message Template Extractor

The POSIX standard [30] for logging system events allows complete freedom in formatting the log entries, hence one can not guarantee that the same piece of information is presented consistently throughout. For example, given the message body of two syslog entries from the Ranger supercomputer:

```
kernel: LustreError: 15949:0:(quota_ ↵
master.c:514:mds_quota_adjust()) mds ↵
adjust qunit failed! (opc:4 rc:-122)
```

and

```
kernel: LustreError: 15817:0:(quota_ ↵
master.c:514:mds_quota_adjust()) mds ↵
adjust qunit failed! (opc:4 rc:-122)
```

an administrator can easily pick out the key phrase, i.e.: `mds adjust qunit failed!` to search for similar messages. The administrator would not search for `kernel:` as it will return too many false positives. Furthermore, he/she may attempt to search for `opc:` just to see if there are other message bodies that may contain this cryptic *token* to shed some light as to what `opc` stands for. It is these types of relationships that we are trying to extract automatically. Our definition of a *token* is given in Table I.

Firstly, we observed from several messages of the same type that tokens in the English dictionary tend to exhibit common patterns while the alpha-numerical tokens exhibit less common patterns. Furthermore, we observed that the English-type tokens contain strings such as `eviction`, `recovery`, `communication` etc. while the alpha-numerical tokens contain strings such as `scratch-ost001d`. Consulting the system administrators on our observations, we learnt that (1) the English-type tokens could serve as indicators of the event that has occurred in the system, and (2) the alpha-numerical tokens could serve as indicators of which component/software function is in need of help.

Leveraging these observations, we defined two categories of tokens called *Constants* and *Variables*. *Constants* are sequences of tokens that comprise of "letters in the English alphabet and punctuations" while *Variables* are sequences of tokens that comprise of "letters in the English alphabet, punctuations and at least one number". The formal definitions of *Constants*, *Variables*, *alphabets*, *punctuations* (represented by the term *symbol*), *numbers* and *messages* are given in Table I. The job of the *Message Template Extractor* is to extract *Constants* and *Variables* from the messages and to create a standard data format for analysis.

Next, we show in Fig 3, an MTE workflow and the steps to extract the *Constants* templates and create the data format. A similar workflow for extracting *Variables* templates can be performed by using FDiag function 1.10 in Step 3 (S3).

- S1: Split the raw logs into logs for individual days.
- S2: Extract the message bodies from the log entries.
- S3: Extract each *Constant* from the message bodies.
- S4: Merge the *Constants*.
- S5: Identify the unique *Constants* and obtain the *Constants* templates.
- S6: Compute the daily frequency occurrences of the *Constants* templates.

We extracted the *Constants* and *Variables* templates from the Ranger and Turing syslogs, and the BlueGene/L RAS logs [5]. A summary of the logs and the distribution of *Constants* and *Variables* templates is shown in Table II. Samples of the

[2]For the list of ASCII codes and their characters, we refer the reader to the ASCII table website at http://www.asciitable.com/.

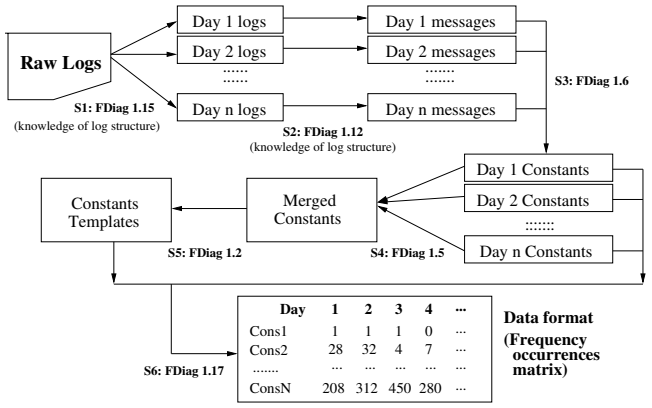



Fig. 3. An MTE workflow for extracting all *Constants* templates and producing the daily frequency occurrence counts of the *Constants* templates.

logs from the Ranger, Turing[3] and BlueGene/L systems are shown in Table III. The purpose of applying MTE to the logs from Turing and BlueGene/L is to show that it can extract templates from the logs produced by different supercomputers. Note that FDiag, being a *flexible framework*, can been extended to include finer-grained temporal resolutions and new problem mappings.

### B. Statistical Event Correlator

The *Statistical Event Correlator* or SEC - the second component of FDiag - attempts to discover trends in the occurrence of system events by analyzing the strength of pair-wise associations between system events. A goal of SEC is to provide an answer to the following question:

*Given knowledge of any system event (e.g.: evict), find all the system events that are associated with the given event, and how strongly associated these system events are to the given event*.

More specifically, we are interested to know if an increase or a decrease in the occurrence of the given event will likely be followed by a respective increase or a decrease in another event. Pearson Correlation Coefficient, arguably the most widely used method for measuring the association between two measured quantities, can be used to discover such trends [31]. Note that the objective of our methodology is not to compare the performance of different algorithms, but to enable validation of the analysis results against system administrators domain knowledge and to enable them acquire domain knowledge.

In the context of SEC, *system events are represented as Constants templates*. The workflow for SEC, shown in Fig 4, provides three steps for discovering trends of pair-wise associations between templates. These steps are: (1) create the Pearson Correlation matrix, (2) generate a heat-map to visualize the strength of pair-wise associations of template pairs, and (3) create Domain Knowledge Groups. A formal definition of a Domain Knowledge Group or DKG is provided in Table I. A detailed description of each step in the SEC workflow follows:

[3]Turing is a HPC cluster managed by A*Star Computational Resource Center, Singapore

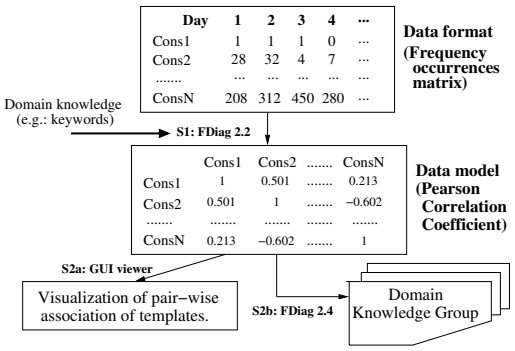



Fig. 4. A SEC workflow for discovering trends in the occurrences of system events.

**Description of SEC Workflow:** As an example, let $FM$ be a *Constants* templates frequency occurrence matrix with $i$ rows and $n$ columns. Let $X$ = row vector 1 $FM_{1,j}$, $Y$ = row vector 2 $FM_{2,j}, 1 \leq j \leq n$. The Pearson Correlation Coefficient [31] for $X$ and $Y$ is:

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left( \frac{X_i - \overline{X}}{s_x} \right) \left( \frac{Y_i - \overline{Y}}{s_y} \right) \quad (1)$$

where $\overline{X}$, $\overline{Y}$ = mean of $X$ and $Y$, $s_x$, $s_y$ = standard deviation of $X$ and $Y$, $X_i \in X$, $Y_i \in Y$, and $n$ = the sample size of $X$ and $Y$. The value of the correlation coefficient should range between 1 to -1, i.e.: if $X$ and $Y$ are identical, then $r = 1$, but if $X$ and $Y$ are exactly opposite, then $r = -1$.

Next, to construct the correlation matrix, the system administrator can provide as input to SEC, domain knowledge which is represented as keyword(s) (e.g.: "evict", "lockup", etc.). The inclusion of domain knowledge performs three important roles: (a) it validates the analysis results against the system administrators knowledge of the system, (b) it demonstrates FDiag's usefulness in assisting system administrators acquire domain knowledge, (c) it enables customization of FDiag to support diagnosis of multiple types of failures and execution environments. SEC will search and extract all the row vectors in $FM$ that correspond to the *Constants* templates that contain tokens matching these keyword(s). From these row vectors, the Pearson Correlation coefficient between each row vector and all the row vectors in $FM$ is calculated. For example, if $FM$ contains 494 row vectors, each row vector contain 101 columns and 12 of these row vectors correspond to *Constants* templates that contain tokens matching the keyword "evict", then the Pearson Correlation coefficient is calculated for each of the 12 row vectors with all the 494 row vectors, resulting in a 12 rows by 494 columns correlation matrix.

Next, a heat-map to visualize the strength of the pair-wise associations between the domain knowledge and all the *Constants* templates is created. An example is shown in Fig 5. To find out which of the *Constants* templates are significantly associated with a domain knowledge, the system administrator

TABLE II
SUMMARY OF LOGS AND DISTRIBUTION OF *Constants* AND *Variables* TEMPLATES FROM RANGER, TURING AND BLUEGENE/L (BG/L) SYSTEMS.

| **System** | **Log-size** | **Start-date** | **Days** | **Messages** | **Constants** templates | **Variables** templates |
|---|---|---|---|---|---|---|
| Ranger | 1.2 GB | Jan 10 09 | 101 | 10,513,072 | 5469 | 203,693 |
| Turing | 780 MB | Feb 27 09 | 167 | 7,048,103 | 1146 | 137,008 |
| BG/L | 730 MB | June 03 05 | 215 | 4,747,963 | 764 | 351,804 |

TABLE III
SAMPLES OF RANGER, TURING AND BLUEGENE/L LOG ENTRIES AND THEIR *Constants* AND *Variables*.

| **A syslog entry from Ranger** | | | | | |
|---|---|---|---|---|---|
| **Month** | **Date** | **Timestamp** | **Node** | **Application** | **Message** |
| Apr | 3 | 14:39:54 | build | kernel: | Out of Memory: Kill process 23397 (pickup) score 6327 and children. |
| *Constants*: Out of Memory: Kill process (pickup) score and children. *Variables*: 23397 6327 | | | | | |
| **A syslog entry from Turing** | | | | | |
| **Month** | **Date** | **Timestamp** | **Node** | **Application** | **Message** |
| Apr | 1 | 00:00:32 | 4A:turing | kernel: | sharc.x(17186): floating-point assist fault at ip 2000000000073be2, isr 0000020000003001 |
| *Constants*: floating-point assist fault at ip isr *Variables*: sharc.x(17186): 2000000000073be2, 0000020000003001 | | | | | |
| **A RAS log entry from BlueGene/L** | | | | | |
| **Category** | **Date** | **Timestamp** | **Source** | | **Message** |
| - | 2005.06.03 | 1117845188 | R20-M0-NB-C:J13-U11 | | RAS KERNEL INFO generating core.3609 |
| *Constants:* RAS KERNEL INFO generating *Variables*: core.3609 | | | | | |

just have to (1) look for the brightly colored regions on the heat-map; these regions indicate strong correlations between a domain knowledge and the *Constants* templates, and (2) use the color-bar on the heat-map to find the desired correlation threshold (higher values indicate stronger correlations) and use it to create what we call *Domain Knowledge Groups* or DKG(s). Each DKG contain one domain knowledge template and the *Constants* templates that are associated with it. The number of associated *Constants* templates is determined by the correlation threshold.

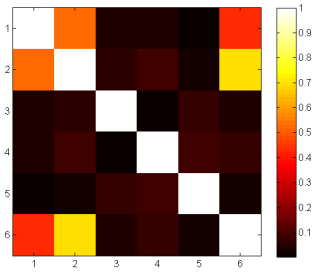


Fig. 5. An example heat-map for *Constants* templates. Domain knowledge are represented by their indexes on the Y-axis while all the *Constants* templates are represented by their indexes on the X-axis.

## C. Episode Constructor

The *Episode Constructor* or ECR - the third component of FDiag - completes the fault diagnostics workflow by (1) extracting from the processed system logs all the log entries that correspond to the templates in each Domain Knowledge Group that has been created by the Statistical Event Correlator, and (2) grouping the log entries into *episodes* such that each *episode* contain the events that have occurred over a specified time window. The workflow for ECR is shown in Fig 6 and entails the following steps:

- S1a: Extract from the corresponding merged constants logs, the indices where the templates in each Domain Knowledge Group occur in the merged constants logs.
- S1b: Merge the logs that previously, have been separated into individual days.
- S2: Extract from the merged logs, the log entries where their indices match the indices produced in Step 1a.
- S3: Extract from the significant log entries produced in Step 2, all the *episodes*.

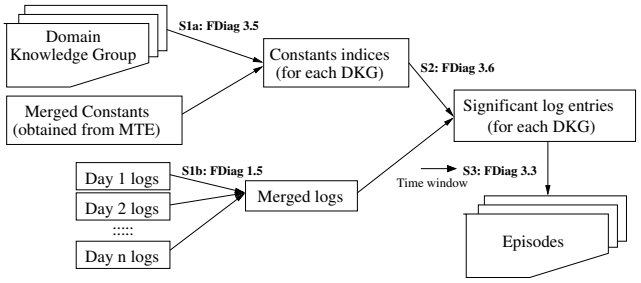



Fig. 6. A ECR workflow for constructing the *episodes*

**Time-Group and Tupling heuristic schemes for episode construction:** Before the episodes can be constructed, two issues must be addressed: (1) a choice of using either a time grouping heuristic [32] or a tupling heuristic [1] scheme to group the sequence of events into episodes, and (2) a choice of the appropriate time window for grouping these events into episodes based on the chosen heuristic. The tupling heuristic considers all events that occur within a fixed time interval of each other while the time grouping heuristic considers all events that occur within a fixed time interval of the first event. The advantage of the time grouping heuristic scheme is that the grouping of events will not occur indefinitely. However the events will be forced within a fixed time interval. The tupling

heuristic scheme can allow for closely spaced events while at the same time, events that have long range dependencies can be captured by the tuple - if the time grouping heuristic was used, then the events that have long range dependencies will be artificially captured in multiple groups [1]. This problem has been addressed by building higher order groupings using statistical techniques [32]. That said, the focus of our methodology is not to compare and recommend either time or tupling heuristic schemes but to present detailed analysis reports to aid the system administrators in their diagnosis of system failures. As such, FDiag's ECR has been equipped to construct episodes using the time grouping and tupling heuristic schemes.

To conclude, the system administrators have the option of providing different time windows for each of the heuristic schemes - of course, each of the heuristic schemes have a recommended time window (e.g.: 3 minute intervals for the tupling heuristic [1] and 5 minutes for the time group heuristic [32]). The resulting episode definitions are presented to the systems administrator.

### D. Implementation

Currently, FDiag provides 18 MTE functions, 4 SEC functions, 8 ECR functions and 10 miscellaneous functions. Each function is implemented in C++ and is designed such that it performs only the assigned task. This approach is flexible and allow us to create meta-functions for the purpose of automating the MTE, SEC and ECR workflows. In addition, the correlation results are stored in comma-delimited files. As such, any statistical package or off-the-shelf / custom-built GUI can be used to create the heat-maps.

The size of main memory in most high-end workstations range from 4GB to 8GB. Hence, in order for FDiag to process system logs when their file sizes exceed the capacity of main memory, we have decided that FDiag should work directly on flat files. This is, however, at the expense of incurring a longer processing time. That said, the time required to process the logs can be shortened by using map-reduce platforms such as Hadoop [33].

## IV. The Lustre Filesystem

Lustre, an object-based high-performance networked filesystem designed for high-throughput I/O tasks, is used by several of the world's top 10 supercomputers. These include Ranger at TACC, Kraken at NICS, Jaguar at ORNL, Pleiades at NASA and Red Sky at SNL/ERNL. The components in the Lustre system work together to provide file management and directory operations to its clients. Fig 7 shows the interactions of Lustre system components [34] and a brief description follows.

- **Metadata Server (MDS):** Metadata (which holds information such as filenames, directories, permissions, etc.) which are stored in one or more Metadata Targets (MDTs), is made available to Lustre clients via the MDS. The management of names, directories and the handling of network requests are performed by each MDS.

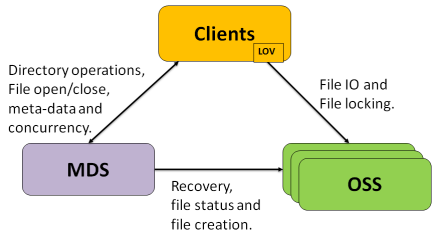



Fig. 7. The Lustre System Component Interactions.

- **Object-Storage Server (OSS):** File I/O services and network request handling are provided by the OSS for one or more Object-Storage Targets (OSTs) - the OST stores user files on one or more OSSs.
- **Clients:** Lustre clients are typically computational, visualization and desktop nodes that run Lustre software to allow them to mount the filesystem.

### A. Soft Lockups in Lustre Client/Server Interactions

The case study we used to illustrate the application of FDiag is the diagnosis of soft lockups of compute nodes which result from timing mismatches in Lustre server/client interactions on the Ranger supercomputer at TACC. When a Lustre server fails to respond to a client request within the client's timeout period, then the client initiates a recovery action, a reconnection to the server. When a Lustre server has not heard from a client within the server's timeout period, the server will begin to evict the client from its list of active clients. If the server eviction takes place while a client recovery is in progress, then on some but not all occasions, the server will refuse the reconnection. If this occurs, eventually the client node will go into a soft lockup state. When one node of a job goes into soft lockup, eventually the job will become hung and must be removed by a systems administrator. Note that the cause of the eventual soft lockup of the client node may occur in the system log long before the actual soft lockup of the client node and its appearance in the system logs.

Such soft lockups occurred on several dates and required administrator action. Since the Lustre file system is typically used by compute nodes for high-throughput I/O tasks, we focused our diagnosis on a subset of the Lustre syslogs that contain the log entries reported by the compute, MDS and OSS nodes. We applied FDiag to the logs to determine if the instances of soft lockups resulting from the Lustre client recovery/evict sequences could be automatically identified through our approach. The case study was executed by processing 3 months worth of the Lustre syslogs from January 10 2009 to April 20 2009, and having a senior systems administrator examine the results of the diagnostic process and use them for diagnosis of the source of the eventual soft lockups which occurred on those dates. The feedback we received from the administrators of the Ranger and Turing systems are positive; they found the FDiag analysis effective in supporting their diagnosis of system failures.

### B. Identify Eviction/Recovery Episodes

The first step in our diagnosis of compute node soft lockups is to identify the episodes in which server eviction and client recovery events took place. Applying FDiag's MTE and SEC workflows and using the domain knowledge of "evict", we obtained the correlation matrix which is represented as a heat-map shown in Fig 8. On the heat-map, there are 12 domain knowledge templates (represented by their indexes on the Y-axis) and 494 constants templates (represented by their indexes on the X-axis). We observed that there are altogether 8 domain knowledge templates with strongly correlated constants templates. The next step is to decide on a correlation threshold for extracting the significant *Constants* templates. We experimented with different correlation threshold values and found that a value of 0.75 produces a substantial set of strongly correlated *Constants* templates for each of the 8 domain knowledge templates. Next, we applied FDiag's SEC to extract the 8 domain knowledge groups. A summary of the domain knowledge groups is shown in Table IV.

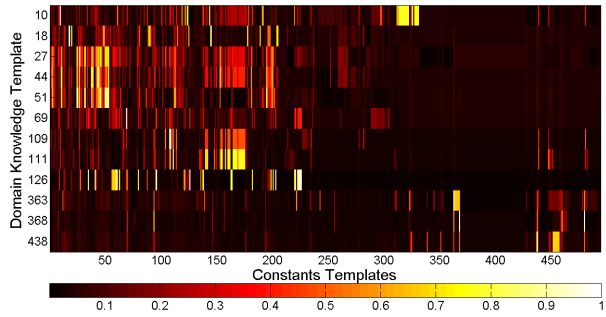


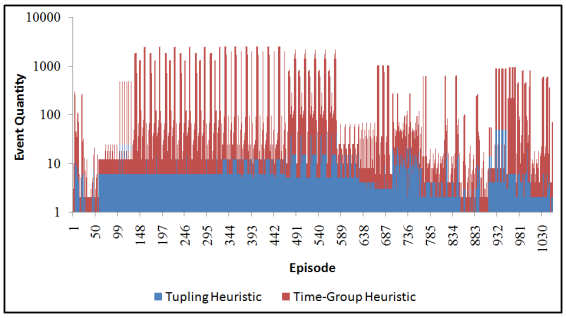

Fig. 8. Heat-map for visualizing the correlation strength between the domain knowledge templates and all the *Constants* templates for the compute, MDS and OSS nodes on the Ranger Supercomputer at TACC.

TABLE IV
SUMMARY OF DOMAIN KNOWLEDGE GROUPS.

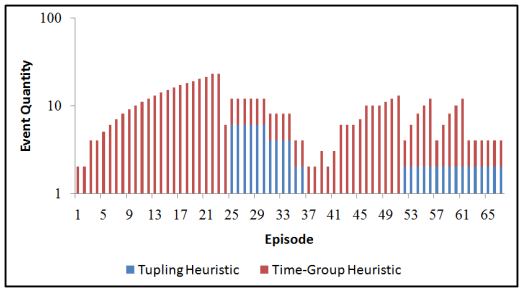
| DKG | *Constants* templates | DKG | *Constants* templates | DKG | *Constants* templates |
|---|---|---|---|---|---|
| 1 | 16 | 2 | 5 | 3 | 2 |
| 4 | 3 | 5 | 7 | 6 | 7 |
| 7 | 17 | 8 | 2 | | |

Note that the number of *Constants* templates in each DKG includes the domain knowledge template. However, not all the 8 domain knowledge groups contain *Constants* templates with the keyword "recover". As we are interested in identifying Lustre's server eviction and client recovery episodes, we added a function to FDiag's MTE to search through the 8 DKGs for the keyword "recover". We found three DKGs in which each DKG contain at least one constants template with the keyword "recover". As such, our episode construction will be based on DKGs 5, 6 and 7.

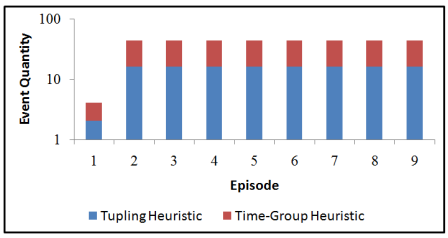
Next, we applied FDiag's ECR to construct all the episodes using the Time-Group and Tupling heuristic schemes. The time windows used for the Time-Group and Tupling heuristic schemes are 5 and 3 minutes respectively. For each episode, the end-point is an "evict" event and all the other events that preceed it are captured in the episode. A total of 1053, 67 and 9 episodes were constructed for DKGs 5, 6 and 7 respectively. The distribution of events per episode are shown in Fig 9(a), Fig 9(b) and Fig 9(c). By observing the event distribution in the episodes, there are still a lot of episodes the system administrator have to sift through to find out which episodes actually contain the client recovery events. Note that these numbers include single "evict" event episodes. Hence, there is a need to find the eviction/recovery episodes. To do so, we added a function to FDiag's ECR to identify the episodes given a set of keywords.




(a) Domain Knowledge Group 5.




(b) Domain Knowledge Group 6.




(c) Domain Knowledge Group 7.

Fig. 9. Distribution of events over episodes from DKG 5, 6 and 7.

A total of 412 time-grouped and 37 tuple episodes from DKG 5 were found to contain server eviction/client recovery events. No such episodes were found from DKGs 6 and 7, however further experiments using different time windows should reveal the presence of these episodes. We chose the episodes that were constructed using the Tupling heuristic scheme as the first set to find out on which dates server evictions and client recoveries have occurred. The number of episodes and dates are shown in Table V. We communicated the episodes and the dates to the Ranger system administrators

and received confirmation that these episodes eventually led to compute node soft lockups on these dates. The workflows that support the diagnostics processes in our methodology will enable the implementation of an automated root-cause failure diagnostics system.

TABLE V
SUMMARY OF EVICTION/RECOVERY EPISODES.

| Date | Episodes | Date | Episodes |
|---|---|---|---|
| Jan 12 2009 | 6 | Jan 15 2009 | 4 |
| Feb 6 2009 | 5 | Feb 15 2009 | 9 |
| Feb 16 2009 | 2 | Feb 23 2009 | 8 |
| Feb 25 2009 | 1 | Mar 10 2009 | 2 |

### C. Identify Significant Events from Domain Knowledge Group 5 Episodes

The second step in our diagnosis of compute node soft lockups is to identify the significant events from the Lustre server eviction/client recovery episodes. Using the episodes that were identified in Section IV-B, we applied FDiag's MTE and SEC to identify the *Constants* templates and the correlation between these templates. Note that for the frequency occurrence matrix generated in this diagnosis, the *Constants* templates are represented as rows while the *episodes* are represented as columns. We obtained the heat-map and *Constants* templates as shown in Fig 10 and Table VI respectively.

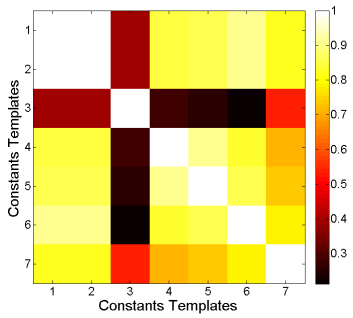



Fig. 10. Heat-map for visualizing the correlation strength between the *Constants* templates for the eviction/recovery episodes from DKG 5.

From Fig 10, we observed that *Constants* templates 4, 5, 6 and 7 are strongly correlated with *Constants* templates 1 and 2, these two being the "evict" and "recover" templates respectively. Moreover, *Constants* templates 4, 5, 6 and 7 are also strongly correlated with each other. We also observed that *Constants* template 6 has the strongest correlation with *Constants* template 1 and 2. This is followed in decreasing correlation strength by *Constants* templates 5, 4, 7 and 3.

Next, we analyzed the distribution of *Constants* templates over each episode shown in Fig 11. A summary of the distribution together with the correlation coefficients of these *Constants* templates with respect to the "evict" and "recover" *Constants* templates is shown in Table VII. From Table VII, *Constants* template 6 is present in 81% of the episodes while *Constants* templates 4, 5 and 7 are present in 45%, 57% and 54% of the episodes respectively. This result shows that FDiag can identify the significant system events from the set of episodes.

TABLE VI
*Constants* TEMPLATES FROM DKG 5 EVICTION/RECOVERY EPISODES.

| Template | Description |
|---|---|
| 1 | LustreError: lock callback timer expired after evicting ⤸ client at ns: lock: lrc: mode: PW/PW res: rrc: type: ⤸ EXT (req flags: remote: expref: pid: |
| 2 | LustreError: lock timed out (enqueued at ago); ⤸ not entering recovery in server code, just going ⤸ back to sleep ns: lock: lrc: mode: –/PW res: ⤸ rrc: type: EXT (req flags: remote: expref: pid: |
| 3 | LustreError: dumping log to |
| 4 | LustreError: Dropping timed-out request from ⤸ deadline ago |
| 5 | LustreError: DROPPING req from old connection ⤸ lens e to dl ref fl rc |
| 6 | LustreError: obd_truncate fails ino |
| 7 | LustreError: enqueue wait took from ns: lock: lrc: ⤸ mode: PW/PW res: rrc: type: EXT (req flags: ⤸ remote: expref: |

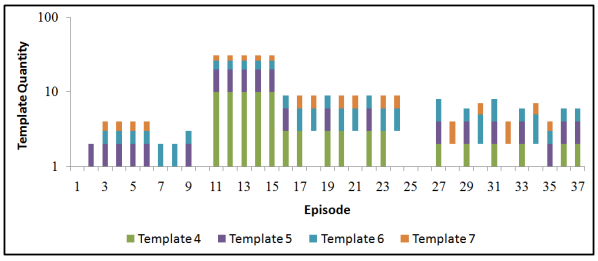



Fig. 11. Distribution of *Constants* templates over episodes from DKG 5.

TABLE VII
SUMMARY OF DISTRIBUTION OF *Constants* TEMPLATES OVER THE 37 EPISODES FROM DKG 5.

| Template | Episodes Quantity | Percentage | r |
|---|---|---|---|
| 6 | 30 | 81% | 0.9 |
| 5 | 21 | 57% | 0.86 |
| 7 | 20 | 54% | 0.81 |
| 4 | 17 | 46% | 0.85 |

## V. RELATED WORK

In [9], a method for ranking log messages that are important to the users was presented. The assumption is that if a message appears more times than it normally should, then this message is important to the user. An unsupervised clustering technique based on the idea that each computer system would produce more messages that is representative of its behavior was developed and used by IBM xSeries support personal. In [10], two algorithms, the first which uses text clustering to discover templates of log messages and the second that discovers patterns of events which correspond to the occurrences of processes in the system were developed as tools and used for application performance debugging and detection of erroneous behavior in IT management software. These tools are designed for uncovering anomalous event patterns and has been shown

to be effective in assisting system administrators identify parts of the computer systems that are faulty or failing.

In [19], an algorithm for detecting faults in syslogs was presented. The algorithm leverages two insights: the first insight is importance of the location of each word text in a message, the second insight is computers that correctly execute similar workloads tend to generate similar logs. Experiments with several metrics showed that information entropy together with the node aggregated matrices results in very accurate detection of faults and produces a false positive rate of less than 0.05%. In [21], an automated mechanism for identifying faulty nodes in large cluster systems was presented. A high-dimensional feature matrix is used as a standard data format for representing a number of features collected from a number of nodes. Principal Component and Independent Component Analysis were compared and used to reduce noise, false alarms, etc. in the data. Faulty nodes were identified using a cell-based outlier algorithm. On one hand, these works focused on methods to detect faulty or failing nodes from information sources such as cluster log files [9], [10], [19], [20] and from tools such as *netstat*, *iostat*, *mpstat* and *vmstat* [21]. On the other hand, FDiag, which is tailored to support the administrators diagnostics processes, extends these frameworks and tools by providing system administrators with (1) the episodes that contain significant system events, (2) the dates and the number of episodes and the events that led to the occurrence of system failures, and (3) significant details on how the fault diagnostics processes have been put together into a tool which will enable the implementation of automated root-cause diagnosis of failures from cluster log files.

Severity levels that mark the messages in the system logs has been shown to be especially useful for the design of failure analysis and prediction models [4], [22], [23], [26]. However the messages in the system logs produced by the Ranger supercomputer are not marked by any severity levels. As such, it is more difficult to identify which are the significant events that likely caused the system failures. In this paper, FDiag provides a solution to this problem by combining *message template extraction*, *statistical event correlation* and *episode construction* to identify the significant events that eventually led to the occurrence of compute node soft lockups.

A number of works have showed that the location of text strings in the log messages are very useful for developing methods to detect faulty nodes [10], [19], and to classify log messages into semantic categories that accurately replicated the manual annotation of messages by system administrators [11]. In this paper, we verified this finding and provided additional details on how templates based on the sequence of text strings can be used to diagnose the root-causes of system failures.

Several tools have been proposed and developed to extract templates from the log messages [15]–[17], [24]. In [24], a log message is broken down into message types and message variables. By examining the source codes, an abstract syntax tree was built and a list of all the possible message types, variable values and variable names were generated and validated in the logs. However the method relies on the availability of source codes and this unfortunately, limits its application to a wider-variety of systems. On the other hand, FDiag uses information that is directly available from the system logs. We have shown in Section III-A that FDiag can be used to extract message templates from system logs produced by different supercomputers.

An automated monitoring and notification toolkit called SWATCH was developed to sift through system logs and provide facilities for simple and user-customized notification actions (e.g.: echo, bell, mail and pipe) [15]. Users of SWATCH create configuration files to specify search criterias using regular expressions and follow-up actions using either the standard list of actions (provided by the SWATCH library) or customized their own. Similarly, LoGS [16] provides facilities for the user to specify rulesets for discovering interesting messages and enable runtime monitoring of the cluster status. Both SWATCH and LoGS are powerful, flexible and customizable. Unfortunately, the need to create and maintain rulesets will still require system administrators' time and effort. On the other hand, the *Message Template Extractor* - the first of three components we developed for FDiag - uses a built-in set of regular expressions, i.e.: the *Constants* and *Variables*. As such, system administrators using FDiag do not have to create and maintain constants and variables.

Recently, researchers have developed a fault-diagnostics tool called NICE that discovers the root-causes of failures in a large telecommunications network [28]. Similar to FDiag's *Statistical Event Correlator*, NICE receives a symptom event as its input and uses Pearson Correlation Coefficient to identify all the events that are correlated to the symptom event. Then, to address the issue of high false-positive rates that result from auto-correlation between the time-series events, a circular permutation based significance test was developed. GIZA [29] goes beyond the pair-wise analysis of NICE and performs multi-dimensional analysis, addresses auto-correlation, discovers edge-directionality automatically and handles collinearity when performing regression. The spirit behind NICE and GIZA lie in a novel infrastructure that enable flexible analysis of the root-causes of failures at different levels of granularity in large-scale network environments. In contrast, our paper describes a systematic methodology and a framework that enables flexible analysis of root-causes of failures in large-scale supercomputing environments.

## VI. Conclusion

We presented a systematic methodology for reconstructing event order and establishing correlations among the events which indicate the root-causes of system failures. We developed a Fault Diagnostics tool FDiag, tailored to provide direct support for our methodology, and validated FDiag using real system log data and a real case study involving the diagnosis of the root-causes of compute node soft lockups on the Ranger supercomputer at TACC. The feedback we received from the Ranger and Turing systems administrators are positive; they have found FDiag's analysis effective in supporting their

diagnosis of system failures. The results from FDiag indicate that it has the potential to become a powerful diagnostics tool for root-cause diagnosis of system failures in large computing clusters.

We plan to extend FDiag in several ways. The first is to create meta-functions for automating the end-to-end *Message Template Extractor*, *Statistical Event Correlator* and *Episode Constructor* workflows so that FDiag is readily used in a production environment. The second is to generate additional functions applying FDiag to other commonly occurring system failures, and to further collaborate with systems administrators in application and enhancement of FDiag. The longer term research is to determine if the message/event patterns that has been discovered to be predictive of future faults/failures can be captured and analyzed by online real-time monitoring to detect potential faults/failures before they occur, and perhaps even to prevent them from occurring.

## ACKNOWLEDGEMENTS

The authors would like to thank Tommy Minyard from the Texas Advanced Computing Center (TACC) for providing the Ranger system logs, Chris Jordan from TACC for his suggestion of looking at evictions as a case study, Stephen Wong from ACRC for access to his system administrators and for the Turing system logs, and Ivor Tsang from Nanyang Technological University (Singapore) for his input. This research was supported by the National Science Foundation under a grant from the Office of CyberInfrastructure.

## REFERENCES

[1] J. P. Hansen and D. P. Siewiorek, "Models for time coalescence in event logs," in *Proceedings of 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, 1992.

[2] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, 2004.

[3] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of 36th IEEE International Conference on Dependable Systems and Networks*, 2006, pp. 249–258.

[4] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a bluegene/l prototype," in *Proceedings of 35th IEEE International Conference on Dependable Systems and Networks*, 2005.

[5] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proceedings of 37th IEEE International Conference on Dependable Systems and Networks*, June 2007.

[6] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems," *Journal of Parallel and Distributed Computing*, vol. 69, no. 7, 2009.

[7] T. Li and W. Peng, "A clustering model based on matrix approximation with applications to cluster system log files," in *Proceedings of 16th European Conference on Machine Learning (ECML 2005)*, 2005.

[8] J. Stearley, "Towards informatic analysis of syslogs," in *Proceedings of IEEE International Conference on Cluster Computing*, 2004, pp. 309–318.

[9] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, "Analyzing system logs: A new view of what's important," in *2nd USENIX workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007.

[10] 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 *Proceedings of 20th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2009.

[11] S. Jain, I. Singh, A. Chandra, Z.-L. Zhang, and G. Bronevetsky, "Extracting the textual and temporal structure of supercomputing logs," in *Proceedings of 16th International Conference on High Performance Computing, Kochi India*, December 2009.

[12] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[13] sisyphus Log Data Mining Toolkit, http://www.cs.sandia.gov/sisyphus.

[14] R. Vaarandi, "Mining event logs with slct and loghound," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, 2008.

[15] S. E. Hansen and E. T. Atkins, "Automated system monitoring and notification with swatch," in *USENIX Large Installation Systems Administration Conference*, 1993.

[16] J. E. Prewett, "Listening to your cluster with logs," in *5th LCI International Conference on Linux Clusters, The HPC Revolution*, 2004.

[17] J. Rouillard, "Real-time log file analysis using the simple event correlator (sec)," in *Proceedings of 18th USENIX Conference on System Administration*, 2004.

[18] M. J. Baum, D. Carasso, R. K. Das, B. Hall, B. Murphy, S. Sorkin, A. Stechert, and E. M. Swan, "A method for building a machine data web from machine data," *US Patent*, no. 11459632, 2006.

[19] J. Stearly and A. J. Oliner, "Bad words: Finding faults in spirit's syslogs," in *Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2008.

[20] A. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Proceedings of 8th IEEE International Conference on Data Mining*, December 2008.

[21] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, 2010.

[22] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *IEEE International Conference on Dependable Systems and Networks*, 2006.

[23] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B.-H. Park, "Dynamic meta-learning for failure prediction in large-scale systems: A case study," in *37th International Conference on Parallel Processing*, 2008.

[24] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Mining console logs for large-scale system problem detection," in *Proceedings of 3rd Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, December 2008.

[25] F. Salfner and S. Tschirpke, "Error log processing for accurate failure prediction," in *1st UNIX Workshop on the Analysis of System Logs*, December 2008.

[26] Z. Zheng, Z. Lan, B. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Proceedings of 39th IEEE International Conference on Dependable Systems and Networks*, 2009.

[27] S. Tricaud, "Picviz: finding a needle in a haystack," in *1st UNIX Workshop on the Analysis of System Logs*, December 2008.

[28] "Troubleshooting chronic conditions in large ip networks," in *ACM International Conference On Emerging Networking Experiments And Technologies*, no. 2, 2008.

[29] A. 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 *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009.

[30] IEEE, *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Base Definitions, Issue 6*. IEEE Standards, 2001.

[31] A. Agresti and C. Franklin, *Statistics: The Art and Science of Learning From Data*. Prentice Hall Pearson International Edition, 2009.

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

[33] Hadoop, http://hadoop.apache.org/.

[34] Lustre, http://http://wiki.lustre.org/manual/LustreManual18_HTML/index.html.