System Behavior Characterization via Information Content Clustering of System Logs

Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada. B3H 1W5.
+1-902-494-2093
{makanju, zincir, eem}@cs.dal.ca

***Abstract*—Self-awareness or the ability to be informed about system internal state is an important attribute for any system to have before it is capable of self-management. This is irrespective of which of the self-* properties of self-management in autonomic systems we choose to achieve. A system needs to have a continuous stream of real-time data to analyze to allow it be aware of its internal state. To this end, previous approaches have utilized system performance metrics and system log data as a means of characterizing system behaviour and internal state.**

In this work, we propose a scheme which utilizes the entropy-based information content to group spatio-temporal partitions of system log data into *conceptual clusters*. We evaluate our method using cluster cohesion, cluster separation and cluster conceptual purity as metrics on High Performance Cluster (HPC) system log data. The results show that our proposed method not only produces well-formed clusters but also clusters that can mapped to different kinds of alert behavior with a high degree of confidence. These results provide evidence that clusters produced by the proposed method characterize the different behaviors of the system and hence capture information about internal state. Hence they have value for the enhancement of self-awareness.

The ability to differentiate among types of behavior (both normal and abnormal) is also valuable for self-monitoring and fault detection as deviations from types of normal behavior could be indicative of a fault.

***Index Terms*—Algorithms; Autonomic Computing; Networked Systems; System Management; Modeling and Assessment**

I. Introduction

An autonomic computer system is one, which is capable of self-management to a certain degree. The need for autonomic computer systems has become more apparent recently, due to the ever growing size and complexity of modern computer systems [1]. As most of today's systems such as cloud computing infrastructure require high availability, this need becomes even more apparent when fault resolution is required. System administrators need to be able to resolve system problems quickly despite the complexity of the systems they manage. Having systems that can provide pointers or aid the understanding of the problem or possible solutions are therefore desirable [2], [3].

For a computer system to be capable of self-management, it needs to possess four attributes [4]. These are self-awareness, self-situation, self-monitoring and self-adjustment. Self-awareness refers to the system's ability to be informed about its internal state, while self-situation implies the ability to be informed about its external operating conditions. Self-monitoring is the ability to detect changes while the ability to adapt to these changes is self-adjustment. To achieve self-awareness, the system needs to be able to differentiate between its different internal states or system behavior.

Previous work has leveraged on either the collection of system performance metrics [2] or system logs [5], [3], [6] as sources of data, which could be analyzed by a system to make it aware of its internal state. Event logs collect information about the various pieces of software running on a system, whereas system logs collect this information at the operating system level. System logs are usually good indicators of system state as they consist of information written by the several interrelated components of complex systems [3]. We therefore explore the use of system logs for the purpose of characterizing system behavior in our work.

In this work, we explore the use of entropy-based information content clustering of spatio-temporal partitions (node-hours) of system logs as a means of characterizing system behavior. Leveraging on previous work in the entropy-based approach to *alert* detection in system logs [5], [7], [8], we extend the approach to characterization of the behavior of a system. We utilize the information content scores derived from the entropy values of the message types that occur within spatio-temporal partitions of system logs as attribute values to propose a clustering technique, which groups the partitions into *conceptual clusters*. *Conceptual clusters* are clusters where objects in a cluster can be described by a concept, not just based on their distance from each other, as is done in conventional clustering [9]. Moreover, message types are textual templates, which abstract the natural language messages in system logs. The clusters formed by this method are then assumed to describe different internal states of the system.

We evaluate our method using three measures, namely, cluster cohesion, cluster separation and cluster conceptual purity, on datasets derived from the system logs of four High Performance Cluster (HPC) systems. The results show that not only are the clusters formed by our proposed information content based clustering method well formed but they can also be described *conceptually* with a high level of confidence with regards to the different alert types in the log data. While it is possible to generalize our method to other types of event logs, our work using HPCs is significant since it is well known that the rate of failure on HPC systems is significantly higher than on uniprocessor machines [10]. Enhancing self-awareness through system behaviour characterization could help reduce the rate of failure and failure resolution times on these systems i.e. improving their availability and dependability.

In the following, we discuss concepts important to understanding our work and previous work in Section 2, while Section 3 discusses entropy-based analysis of system logs in-depth. Section 4 discusses the methodology of proposed method and the experiments we carried out to evaluate the method, whereas the results of those experiments and discussions on the generalization of the proposed method and results are discussed in Section 5 and Section 6, respectively. Finally, conclusions are drawn and the future work is discussed in Section 7.

II. Background and Previous Work

A. Definitions

A system log can be defined as a collection of lines of text reporting occurrences on a computer system setup in temporal order. Each event can be further decomposed into several fields, the exact nature and order of which would differ from system log to system log, but will generally consist of a timestamp, reporting computer or node, severity information and a free form message. A nodehour is thus one hour of log information from a single node.

The free form message, which we refer to here as *message*, is a natural language description of the event and is therefore unstructured. Each message in an event can be sub-divided into tokens, using a delimiter, which is usually whitespace. This unstructured nature of messages makes it difficult for the message field of events to be used in building models. Message type extraction or message type clustering is a way of building structured context into the unstructured message fields of events. Its goal is to find a set of textual templates, defined by constant tokens and variable tokens (wildcards) that abstract all the messages in a system log. Each message can be produced by one and only one template. These templates are referred to as *message types*.

B. Self-Monitoring for Self-Management in Computer Systems

The goals of any automatic system monitoring application in regard to the goal of self-healing in autonomic systems can usually be summarized into one of three inter-related goals.

1) **System Characterization :** The goal here is to model or describe the characteristic of the system under normal operation. These models, once formulated can then be used to monitor the state(s) and state transitions of the system. Essentially this improves self-awareness in a deployed system.
2) **Error/Failure Identification :** We differentiate here between errors as symptoms of a fault and failures that are actual faults. Faults usually leave signatures on systems before and after they occur, these signatures manifest themselves in the form of errors (alerts) in the system. The goal here is the automatic identification of these error conditions that ultimately link to the actual fault conditions or failures, thereby reducing troubleshooting time.
3) **Failure Prediction :** Failure prevention can sometimes be more desirable than recovery from a failure. The error signatures associated with faults sometimes occur before the actual fault. In cases where such errors are non-fatal and always ultimately lead to a fault condition, they can be used as failure predictors, thus allowing pre-emptive action to be carried out before failure occurs.

To be able to create applications that will be capable of automatically monitoring systems with the above goals in mind, we must have access to continuous stream of real-time data from the systems. Such data must explicitly or implicitly contain information about the state of the system and its components.

A literature review shows significant effort has been carried out by various researchers in automatic system monitoring and most have followed the approach of analyzing data from the system. Two important sources of data include system metrics [2], [17] and system logs [10], [12], [13], [14], [3], [5], [15], [6]. While these sources seem to be more prevalent in the literature, other sources have been explored too. For example, Chen et al. model system behavior by collecting and modeling statistics of the paths that system requests follow as they move through a system [16]. They demonstrate how this knowledge can be used in failure and change management.

Metrics of system activity can be used in the automatic analysis of such systems. These metrics could be either low level system metrics or application level metrics. System metrics include measurements such as resource utilization (memory and CPU), length of system queues and latency of disk I/O operations, while application level metrics include transaction response time and request throughput [2].

In [2], Cohen et al., successfully demonstrate a method of using system metrics to define system states. Such observed states are clustered and indexed for similarity based retrieval. Their assumption was that if an indexed system state can be used to identify a prior system problem, then future system states, which are sufficiently *similar* can also be associated with the problem. This in turn can lead to quick diagnosis and repair. In [17], Jiang et al. propose the utilization of entropy based analysis of management metrics for the problem of fault detection. Their work suggests that systems can be monitored by observing the changes in the *in-cluster* entropy of clusters identified using the normalized mutual information of metric pairs. In [11], the authors apply a dependency-aware framework based on the Tanimoto coefficient to the problem of fault diagnosis in enterprise software systems, using metric-correlation models built form collected application level metrics such as resource utilization and response time.

In summary, previous work in the analysis of system logs typically follow, encompass or require one or more of the following steps.

- **Unstructured Message Analysis :** Events in system logs are typically not homogenous entities. They contain structured and unstructured information. The unstructured information, namely free-form messages, typically pose a stumbling block to the automatically analysis of event data. Therefore, analysis of unstructured messages is required for further understanding of system logs. The methods proposed in [14], [18], [6], [15] show different approaches to carrying out unstructured message analysis.
- **Indexing/Feature Creation :** This involves the creation of indexable features from the unstructured data in the form of message type *IDs* and message variables. This is referred to as message type transformation (MTT) in [7], [19].
- **Event Correlation :** In most cases, it is unlikely that a single event in a system log could characterize system behavior. As such, it is important to find message types that are correlated in the system logs. Correlated messages are usually better indicators of system state. In [6], the authors track the variables reported in message types as a means of identifying correlated messages. They argue that messages which report the same variable(s) are likely to be correlated.

In the following, we summarize some previous work done with log data in system monitoring.

Loghound is a log data mining tool, which is an implementation of a frequent itemset mining algorithm for mining both lines (message types) and event types (correlated messages) from an event log [20]. In [14], a case study of Loghound is presented to analyze Cisco Netflow logs for patterns that describe the behavior of network traffic. In [10], Liang et al. propose a 3-step filtering algorithm for filtering failure logs from a high performance cluster (a BlueGene/L prototype), which compresses and categorizes the events in the log to better understand failure behavior.

In [12], the authors propose a modified Naive Bayes algorithm for the categorization of messages in system logs. Unlike other approaches where such message categories are based on the textual representation of the messages, the categorization done here is based on previously defined categories associated with the IBM CBE (Common Base Event) format [21]. The authors then discover temporal relationships between these message categories. These relationships are then visualized to monitor system behavior.

In [13], the authors assume the existence of known *event types* (message types) and use the process they refer to as *event summarization* to mine and rank temporal dependencies between event types. Temporal dependencies are mined using time series analysis and are ranked using a *forward entropy* technique [22]. These dependencies are then visualized using an Event Relationship Network (ERN) [23], [24], which is used to interpret system behavior and derive rules for system management. In [3], Lim et al. utilize message de-parameterization to create message types from enterprise telephony system logs. Messages in the logs are then replaced by these message types, which they refer to as *message codes*. The logs are then further analyzed using frequent itemset mining to discover correlated messages, which were useful in determining failure states in the system.

In [15], Aharon et al. propose the PARIS (Principal Atom Recognition in Sets) algorithm. This algorithm is able to detect *atoms*, i.e. sets of correlated message types, which are produced as part of a normal process or failure activity. In this case, the message types have been previously mined. The authors then propose the monitoring of these atoms through visualization as a means of detecting failure in system logs. Xu et. al proposed a Principal Component Analysis (PCA) based framework for the detection of system problems through the analysis of console logs [6]. In their case message types were extracted from source code.

On the other hand, Nodeinfo is an alert detection method based on the entropy-based information content analysis of system logs [25], [5]. Based on the assumption that *"Similar computers correctly executing similar code should produce similar logs"*, Nodeinfo introduces the more complex log.entropy term weighting scheme to the work of Liao [] and Reuning [26]. Nodeinfo has been shown to achieve an operationally acceptable false positive rate of 0.05% at a Recall rate of 50% in the detection of system logs from High Performance Clusters (HPC). Nodeinfo does not fully capture message context as it does not use message types. Instead, it utilizes the concept of encoding token and token position pairs for dealing with unstructured messages. In [7], [8], [19], Makanju et al. incorporate the Iterative Partition Log Mining (IPLoM[1]) [18] message type extraction algorithm into Nodeinfo, to allow it to fully capture message context. The authors show that with their modifications, Nodeinfo was much faster and was able to achieve a false positive rate of 0% at a Recall (detection) rate of 100% under the best case scenario.

Our work builds on this entropy-based information content approach to system monitoring. Previous applications are focused on the use of an entropy based approach to alert detection or error identification. In this work, we demonstrate how an entropy based approach can be extended to the characterization of system behavior. To achieve this, we build *conceptual clusters* using the portions of the system log that have similar information content. In the following, we present this approach in more detail.

III. Entropy Based Analysis of System Logs

Entropy based analysis of system logs proceeds from the work of Oliner et al. [25], [5] and has so far been limited to the task of unsupervised alert detection. We refer to *alerts* as events (or group of events) in a system log that are symptomatic of failure or require the attention of an

[1]An open source implementation of the IPLoM algorithm is available for download from http://web.cs.dal.ca/∼makanju/iplom/

administrator.*Unsupervised alert detection* is the task of automatically identifying such events without human intervention. An entropy based approach to alert detection has so far been shown to scale to large datasets [7] and achieve an F-Measure detection accuracy of up to 100% leading to an effective FPR of 0% in the best case [19], [8].

Since entropy based approaches are based on a *"Similar Computers, Similar Code, Similar Logs"* assumption, a logical pre-processing step is to partition the contents of the system log based on the similarity of their source. In [5], [7], [8], similarity was based on the functionality of the nodes in the HPC files used in the evaluations.

After partitioning, the rest of the alert detection is carried out following a three step process:

- **Step-1:** Calculate an entropy based information content score for each term that appears in the system log.
- **Step-2:** Calculate the information content score for spatio-temporal partitions of the system log based on the information content of the terms that occur in it.
- **Step-3:** Create a ranking of the nodehours based on their information content score to facilitate alert detection.

In the first step, entropy based information content scores are calculated for each individual *term* that appears in the free form message fields of the events in the log. A *term* could either refer to a concatenation of the individual tokens in the free form message with a number corresponding to its ordinal position in the free form message (thereby capturing the token's context in the message) [5] or the tokens produced through the use of Message Type Transformation (MTT) [19] to convert the free-form messages in the system log based on their message types. Three message transformations are proposed in [19], these are: (i) Phrasal MTT, (ii) MTT with variables, and (iii) Full MTT. It is shown that Full MTT can provide up to a 99% reduction in the number of unique terms found in a system log without reducing the detection accuracy of the framework [7]. Thus, in this work, we take *terms* to mean tokens formed by Full MTT, too.

In our work, we do not assume that these message types are known, but we instead extract them automatically using the Iterative Partitioning Log Mining (IPLoM) message type extraction algorithm [18]. We utilize IPLoM for message type extraction because we do not assume access to source code as in the work of Xu et. al. [6]. IPLoM had been shown to produce message types, which matched manually produced messages types closely. In addition, it is also capable of finding not only frequent patterns in the data but also infrequent ones. The use of IPLoM to extract message types ensures that we have a framework that can be made fully automatic.

Once the set of unique terms W in an system log is identified, we can calculate the entropy based information content of each term using Eqs. 1 and 2. If we let C be the set of nodes on the network, then matrix $\mathbf{X}$ represents a $|W| \times |C|$ matrix where $x_{w,c}$ is the count of the number of times term w appears in messages having node c as source. The output of this stage is vector $\mathbf{G}$ with cardinality $|W|$, where each element g_w of $\mathbf{G}$ represents that entropy based information content of term w. Its values are in the range $[0, 1]$, with *0* signifying low information content and *1* signifying the highest information content possible.

$$g_w = 1 + \frac{1}{\log_2(C)} \sum_{c=1}^{C} p_{w,c} \log_2(p_{w,c}) \quad (1)$$

$$p_{w,c} = \frac{x_{w,c}}{\sum_{c=1}^{C} x_{w,c}} \quad (2)$$

Entropy based alert detection in system logs does not attempt to identify alerts on an event by event basis. It instead attempts to identify spatio-temporal partitions of the log, which are more likely to contain alerts than others. The spatio-temporal partitions used in previous work and in this work are referred to as nodehours [5]. A nodehour is basically one hour of log information produced by a single node on the network. Given any system log E, a Nodehour can be defined as any grouping of lines produced by a single node (c) within a one hour interval in tune with wall clock time [5].

The second step proceeds to assign an information content score to each nodehour. This information content score then acts as an indicator of the *interestingness* of the nodehour. Let's define H_j^c as the j^{th} Nodehour for node c, we can assign an information content (Nodeinfo) score using either of Eqs. 3, 4, 5. A comparison of these approaches can be found in [8]. In this work, we utilize *NodeinfoPlus_Uniq*, a modified nodeinfo score defined by Eq. 4. In this approach, we define a matrix $\mathbf{Z}$, where $z_{w,j}^c$ effectively only records unique occurrences of terms in the event data. This is different from the matrix $\mathbf{Y}$ used in Eq. 3, where $y_{w,j}^c$ is the count of the number of times term w appears in Nodehour H_j^c. Therefore *NodeinfoPlus_Uniq* assigns an information content score to a nodehour based on the magnitude of the vector of information content values of the terms contained in Nodehour H_j^c.

Results highlighted in [19], [8] have shown that *NodeinfoPlus_Uniq* results in improved alert detection accuracy and reduced computation of the alert detection process.

$$NodeInfo(H_j^c) = \sqrt{\sum_{w=1}^{|W|} (g_w \log_2(y_{w,j}^c))^2} \quad (3)$$

$$NodeInfoPlus\_Uniq(H_j^c) = \sqrt{\sum_{w=1}^{|W|} (g_w * z_{w,j}^c)^2} \quad (4)$$

$$NodeInfoPlus\_Max(H_j^c) = max_w(g_w * z_{w,c,j}) \quad (5)$$

In the third step, a ranking of nodehours based on their information content scores is established. Nodehours with high information content scores are then considered more likely to contain alerts than those that come up lower in the ranking.

A. Entropy-Based Information Content System Characterization

In this section, we describe the intuition behind our method of extending an entropy-based approach to system behavior characterization and the datasets used in our work. In our

TABLE I
SYSTEM LOG DATA STATISTICS

System	# Days	Size(GB)	# Events
Blue-Gene/L (BGL)	215	1.21	4,747,963
Liberty	315	22.82	265,569,231
Spirit	558	30.29	272,298,969
Thunderbird(Tbird)	244	27.37	211,212,192

evaluation of entropy-based alert detection, we tested the approach(es) on four HPC logs. These system logs are *Blue-Gene/L (BGL), Liberty, Spirit and Thunderbird(Tbird)*, which are part of a set of HPC system logs, publicly available in the USENIX Computer Failure Data Repository [27]. These datasets were collected on HPC systems with varied configurations and usage patterns; hence results generated using this data should be generalizable. Some statistics for these system logs can be found in Table I. The events in these system log datasets have been previously labelled as alerts and non-alerts by domain experts, giving us ground truth when evaluating our results. More details of the hardware architecture, configuration, characteristics, log collection methods and alert identification policies of these datasets and the systems that produced them can be found in [10], [28].

Entropy based alert detection relies on the assumption that *"Similar computers correctly executing similar work should produce similar logs"* [25]. For this reason, log events from similar nodes need to be analyzed together for the framework to work effectively. To this end, we separated the messages in the datasets based on the functional roles of the nodes that produced them, leading to fourteen categories. These categories are listed in the first column of Table II. The **-Other* categories are not functional groupings of messages but consist of all messages that could either not be placed in any of the other categories or have ambiguous source information. The data statistics of the resultant datasets based on functional groupings is detailed in Table II.

In our evaluations a strong clustering of nodehours around single information content score values was observed Fig. 1. This was particularly pronounced when *NodeinfoPlus_Uniq* i.e. Eq. 4 was used. Such clustering of values could be considered odd, since information content scores are real numbers that theoretically can take any value in the range $[0, \infty)$. The graphs in Fig. 1 shows select scatter plots for nodehours from four of the datasets shown in Table II. In each graph, the y-axis represents the information content score for a nodehour using Eq. 4 while the x-axis represents each individual nodehour sorted according to their information content score. We can see the clustering manifest itself in all the graphs, while being most pronounced with the BGL-Link category.

In the following, we try to explain what could be responsible for this observation. Consider a set of distinct objects X, which you wish to sample (with replacement) and distribute into a number of bins (each bin acting as a *bag*, which can contain several instances of the same object from X), with the following constraints:

1) If the number of bins is n, then $|X|$ should be $<< n$.
2) If Y_i is the set of unique objects in bin i, then $|Y_i|$ should be $<< |X|$ for most i.

If the sampling from X described above is carried out even with a random sampling method, it is easy to see how we could end up with several bins containing the same set of distinct objects. Constraint 2 effectively reduces the number of possible distinct object combinations that can exist in any bin, while constraint 1 ensures that the chance for a combination to repeat itself is high. If the number of possible of combinations of objects from X given constraint 2 is less than n, a combination repetition is guaranteed.

If we take X to be set of message types that exist in a system log and the bins as the nodehours in the system log, then we can reduce the process described to the system log analysis domain with one major difference; the sampling from X will follow a Pareto distribution rather than a random distribution. Previous work [29], [20] has shown that the distribution of messages in system logs by nature typically follow a Pareto distribution. This means that the sampling of objects from X will biased in such a way that a small subset of the objects would be sampled more frequently than others. This biased sampling should accentuate the result of having several bins containing the same set of distinct objects.

We conjecture that the process described above is responsible for the strong clustering of nodehours around a single information content score as in Fig. 1. The information content score value derived from the use of Eq. 4, is in a way a *hash* value for the set of unique message types in a nodehour, so we can link a distinct information content score to a (some) set(s) of unique message type combinations. This would also explain why this observation is more pronounced when Eq. 4 is used to calculate information content scores for nodehours. Eq. 3 weights its results with frequency of occurrence of each message type, thus two nodehours can only have the same information content score, if they have the same message type combination appearing at the same frequency. On the other hand, Eq. 5 does not take all the message types in a nodehour into consideration. This highlights another advantage of using Eq. 4 over the other methods evaluated in [8]. Previous work suggests that temporal filtering of system log messages could be beneficial for system log analysis [10] and Eq. 4, represents a form of implicit temporal filtering of system log messages.

From Table II, we can see that the constraints we described earlier hold true for our datasets. The "# Msg-Types" column represents $|X|$, the "# Nodehours" column represents the number of bins while the "Msg-Types/Nodehour (Max)" and "Msg-Types/Nodehour (Avg.)" columns represent the maximum and average number of message types that can be found in each nodehour respectively. Based on these observations, we arrived at the following hypotheses:

- Nodehours with the same information content score (based on Eq. 4), contain the same unique set of message types.
- Information content scores, which occur frequently, represent nodehours, which contain strongly correlated mes-

TABLE II
SYSTEM LOG DATA FUNCTIONAL GROUPING STATISTICS

	# Events	# Nodes	# Nodehours	# Msg-Types	Msg-Types/Nodehour (Max)	Msg-Types/Nodehour (Avg)
BGL-Compute	4,153,009	65,554	1,581,845	399	117	1.37
BGL-IO	400,923	1,024	219,722	49	5	1.11
BGL-Link	2,935	517	1,395	13	4	1.23
BGL-Other	191,096	2,167	13,666	97	16	2.71
Liberty-Compute	200,940,735	236	1,748,865	481	214	1.87
Liberty-Admin	52,211,676	2	27,162	601	195	5.9
Liberty-Other	12,416,820	6	44,447	510	166	11.89
Spirit-Compute	218,697,851	512	6,648,719	854	349	2.29
Spirit-Admin	41,847,257	2	26,216	443	151	6.79
Spirit-Other	11,753,861	7	57,532	707	228	10.93
Tbird-Compute	155,403,254	4,514	14.520,204	1,262	325	3.62
Tbird-Admin	15,306,749	20	100,740	627	179	4.69
Tbird-SM	19,109,810	1,319	8,859	597	254	12.54
Tbird-Other	21,392,379	2	626,030	1,387	356	2.13

sage types and represent some system behavior or characteristic.

Our work in this paper was carried out to validate these hypotheses.

IV. METHODOLOGY

In this section, we describe the *conceptual clustering* technique we developed based on the hypotheses above. We also discuss the methods to evaluate the quality of the clusters formed using our proposed technique.

A. Information Content Based Clustering

The pseudo-code in Algorithm 1 describes our method for the clustering of the nodehours based on their information content. Essentially the algorithm creates each *cluster* as a bin, which can be described using the tuple (ICS, "$MaxEntropyMsgType''$), where ICS is an information content score value and "$MaxEntropyMsgType''$ is the *ID* of the message type with the maximum entropy value among all the message types that have instances in the nodehour. Intuitively, it would be sufficient to use only the ICS as description for each bin. However, for the first hypothesis in Sec. III-A to be true, two nodehours with the same information content score should at the minimum have the same message type with maximum entropy, given that this message type would probably be the highest contributor to the information content score. Hence, the addition of $MaxEntropyMsgType$ to the description. All nodehours with the same values for the tuple will end up in the same cluster.

Below we list some of the properties of our proposed method that differentiates it from previous approaches:

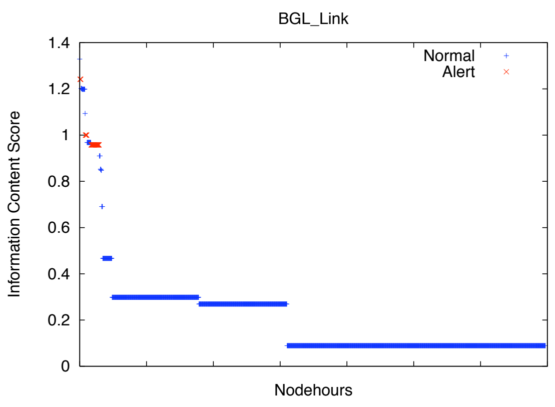
1) **Spatial Decomposition :** Previous work has shown that one of the major mitigations against finding correlated messages in system logs is the fact that correlated messages may not always follow each other in sequence in the system logs [20]. The entropy based information content approach of our method requires that the system log be decomposed spatio-temporally both at the point when entropy values are calculated for terms and the point where information content scores are assigned to nodehours. We note that nodehours are spatio-temporal partitions of the log data. This fact increases the chance that messages that follow each other temporally are likely to be correlated.
2) **Complexity :** The complexity of this approach is $O(n)$, where n is the number of spatio-temporal partitions, in our case nodehours.
3) **Interestingness :** Since each pattern belongs to a cluster that is partially defined by an information content score, we automatically have a means to evaluate the *interestingness* of the pattern that has been found without further analysis. By *interestingness*, we refer to likely hood that the pattern represents an occurrence which would require the attention of an administrator.
4) **Pattern Length :** The patterns will for the most part not contain sub-patterns of a larger pattern, thereby reducing the number of patterns found. This results because of the fact that the proposed method deals only with the entire message type combination found in a nodehour. Patterns mined from system log data could sometimes be just as large and complex as the log data itself, hence the production of only a small set of interpretable patterns is desirable[13].

Algorithm 1 This pseudo-code describes our method for clustering a set of nodehours.

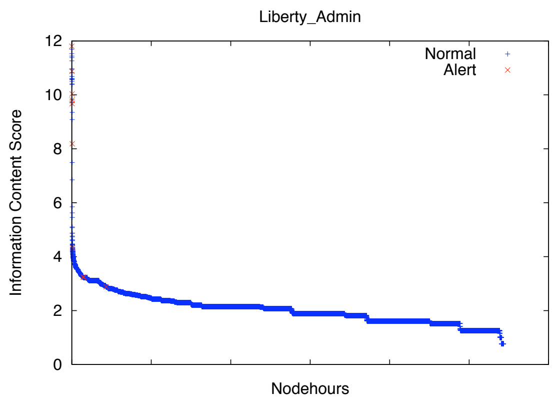
```
Input: Set $S$ of nodehours with associated information content (Nodeinfo) scores.
    Array $m_{max}$ which contains the message type with maximum entropy for each
    nodehour in $S$.
Output: Collection $S_1 \ldots Sn$ of clusters of $S$.
1: for each nodehour $s$ in $S$ do
2:     Determine the cluster $S_i$ that $s$ belongs to as combination of its information
       content score and its maximum entropy message type.
3:     if $S_i$ exists then
4:         $s \in S_i$ {Add $s$ to $S_i$}
5:     else
6:         Define a cluster $S_i$ of $S$, such that $|S_i| = \emptyset$
7:         $s \in S_i$ {Add $s$ to $S_i$}
8:     end if
9: end for
   {Each cluster produced will represent the set of nodehours which have the same
   information content score and the same maximum entropy message type.}
```

B. Evaluations

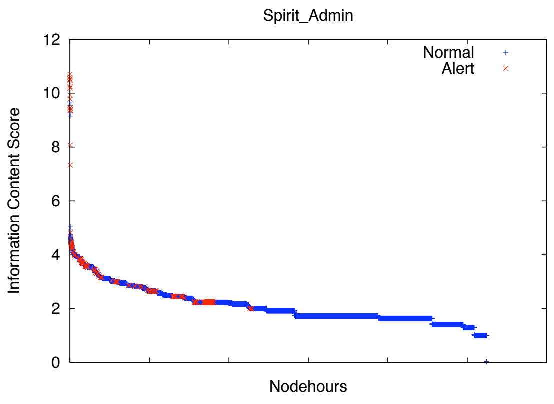
In this section, we describe the methods we used in evaluating the quality of the clusters formed by our proposed technique. It should be noted that since the information content



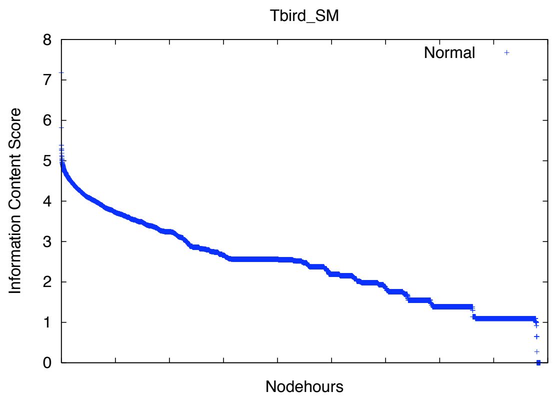
(a) BGL: Link Category



(b) Liberty: Admin Category



(c) Spirit: Admin Category



(d) Tbird: SM Category

Fig. 1. Scatter Plot of Nodehours (x-axis) vs. information content scores (y-axis) for select node functionality categories. The plot differentiates between alert nodehours and normal nodehours, the Tbird-SM category has no alert nodehours. Nodehours are sorted based on information content score in the plot.

score associated with each nodehour is a real number, we set a precision level of 10 decimal places when testing the equality of the information content scores in our evaluations. This is important for the reproducibility of our results. We also considered only clusters, which had > 9 nodehours in them, arguing that clusters with fewer nodehours might only have resulted by mere coincidence. Our experiments were carried out using all the fourteen datasets listed in Table II.

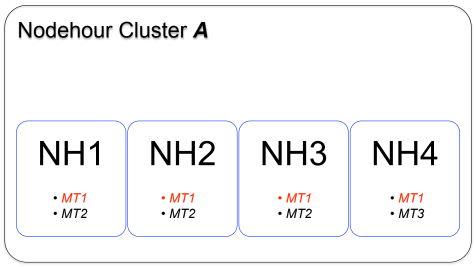
We evaluated the clusters formed at the end of our experiments using three measurements, namely i.e. cluster cohesion, cluster separation and cluster conceptual purity. Before we describe how we measured these concepts, we describe two important *meta-concepts* i.e. the distance between clusters and the cluster *centroid*.

The cluster centroid for each *cluster* is an n-tuple, which represents the message type combination that appears most frequently among the nodehours in the cluster. For example, in *Cluster A* in Fig. 2, the message type combination $(MT1, MT2)$ appears most frequently and therefore, is the *centroid* for this *cluster*. Similarly the message type combination $(MT2, MT3, MT4)$ is the *centroid* for *Cluster B*.

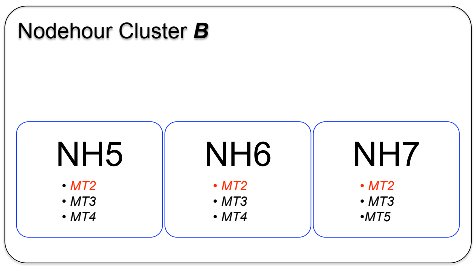
The distance between two nodehours is defined using the standard function for finding the distance between objects defined by nominal variables [9], Eq. 6, where p represents the number of variables common to the objects and m represents the number of variables for which both objects match. For example, in *Cluster A* in Fig. 2, the message types common to $NH1$ and $NH4$ would be $(MT1, MT2, MT3)$, hence in this case $p = 3$ and the message types that match between them are $(MT1, MT2)$, and therefore, in this case $m = 2$. Thus, the distance between $NH1$ and $NH4$ would be 0.33. This way, distances between pairs of nodehours and *cluster centroids* can be calculated by:

$$d(i, j) = \frac{p - m}{p} \tag{6}$$

- **Cluster cohesion :** This measures the degree of similarity of the members of a cluster. We would want the members of a cluster to be very similar. This is measured using the *Fwithin* statistic as defined in Eq. 7. It represents the standard deviation within a cluster, using the *centroid* as mean. In Eq. 7, μ_x represents the *centroid* of *cluster X*, while x_i represents the ith nodehour in *cluster X*.
- **Cluster separation :** This measures the degree of similar-



(a) Cluster A



(b) Cluster B

Fig. 2. The figure above shows two example nodehour clusters formed using Algorithm 1. The boxes enclosed represent nodehours that belong to the cluster and the bulleted lists represent the set of unique message types that appear in the nodehour. In each bulleted list the message type highlighted in *red* represents the message type with maximum entropy which will be common to all nodehours in a cluster according to Algorithm 1.

ity between different clusters. We would expect clusters to be very dissimilar. This is measured using the *Distance* statistic as defined in Eq. 8. It represents the distance between two clusters X and Y using Eq. 6.

- **Conceptual Purity :** We assume that the proposed method produces *conceptual clusters*, it would therefore be interesting to see if the clusters formed could be linked to concepts and if so, with what level of confidence. We attempt to measure the degree to which the clusters formed meet this criterion using the *Conceptual Purity* measure. The datasets used in our experiments provide us with ground truth with respect to the *alert concepts* that exist in these datasets, in the form of the alert categories assigned to each event in the log [28]. To this end, we measure the conceptual purity of the *alert clusters*, i.e. clusters that contain a majority of alert nodehours. The process defined in Algorithm 2 determines the ratio of nodehours in an alert cluster, which contains the signature for an alert category. It measures the degree to which the alert nodehours in a cluster can be linked to an alert category. A value of 1 for this ratio implies that all the alert nodehours in a cluster can be linked to the same alert category. Therefore the cluster can conceptually be linked to the alert category with 100% confidence. For example, suppose that alert category β can be linked to $MT2$ i.e. $MT2$ is a signature for the error represented by β. In *Cluster A* in Fig. 2, $MT2$ appears in 3 out of 4 nodehours, hence the alert category to alert cluster ratio for *Cluster A* with respect to alert category β is $\frac{3}{4}$. We can therefore repeat this process for all alert categories whose signatures appear in the nodehours of *Cluster A* and do the same for all the clusters. The average of these values represents the conceptual purity with respect to the alert categories defined in the system log.

$$Fwithin(X) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d(\mu_x, x_i)} \tag{7}$$

$$Distance(X,Y) = d(\mu_x, \mu_y) \tag{8}$$

Algorithm 2 This pseudo-code describes our method for determining the degree to which the signature of an alert category can be linked to an alert cluster.

```
Input: Set $M_{alert}$ of messages types that can be linked to alert category
    Cluster $S_{alert}$ whose conceptual purity with respect to the alert category we want
    to determine.
Output: Alert Category to Alert Cluster Ratio. [Range [0,1]]
 1: $ratio\_sum = 0$
 2: $count = 0$
 3: Determine the number $n$ of nodehours in $S_{alert}$
 4: for each message type $m$ in $M_{alert}$ do
 5:   Determine the number $x$ of nodehours in $S_{alert}$ that contain $m$
 6:   if $x > 0$ then
 7:     $temp\_ratio = \frac{x}{n}$
 8:     $ratio\_sum = ratio\_sum + temp\_ratio$
 9:     $count++$
10:   end if
11: end for
12: $cluster\_ratio = \frac{ratio\_sum}{count}$
13: $Return(cluster\_ratio)$
```

V. Results

The results generally show that by using our proposed information content based clustering method; we were able to produce well formed and conceptually meaningful clusters. The statistics of the clusters formed are presented in Table III. We only considered clusters of size $>= 10$, the number of clusters formed with this restriction is given in the *"# Clusters ($>=$ 10)"* column, the total number of clusters formed irrespective of size is given in column *"Total Clusters"*. The *"% Nodehours"* column gives the percentage, which belong to clusters if only clusters of size $>= 10$ are considered. The results show that on average, we are able to cluster ~92% of nodehours using this approach.

In the following sections, we discuss our results on the *goodness* of the clusters formed using (i) internal measures (cohesion and separation) and (ii) ground truth (alert categories). Finally, we discuss how the results validate our hypotheses.

A. Internal Measures

The results also demonstrate that on average the clusters formed show an average Fwithin measure of ~0.01 indicating tightly formed clusters and an average *Distance* measure of ~0.82 indicating that the clusters are well separated. Details

TABLE III
CLUSTER STATISTICS

	# Clusters (>= 10)	% Nodehours	Total Clusters
BGL-Compute	135	99.96	446
BGL-IO	62	99.91	113
BGL-Link	6	97.71	17
BGL-Other	37	97.64	177
Liberty-Compute	412	99.74	3061
Liberty-Admin	202	87.82	2,069
Liberty-Other	345	88.86	5,326
Spirit-Compute	615	99.82	9,247
Spirit-Admin	259	86.53	2,118
Spirit-Other	495	78.46	8,641
Tbird-Compute	5,389	99.01	110,201
Tbird-Admin	557	94.01	3,492
Tbird-SM	78	68.62	2,000
Tbird-Other	556	97.65	8,653

TABLE V
ALERT NODEHOUR CLUSTER CONCEPTUAL PURITY

	# Categories	# Alert Clusters	Conceptual Purity Ratio
BGL-Compute	15	95	1.00
BGL-IO	15	41	1.00
BGL-Link	5	3	0.90
BGL-Other	6	6	0.90
Liberty-Compute	19	57	1.00
Liberty-Admin	3	0	N/A
Liberty-Other	9	1	1.0
Spirit-Compute	29	114	1.00
Spirit-Admin	3	9	0.86
Spirit-Other	6	2	1.00
Tbird-Compute	31	273	1.00
Tbird-Admin	7	0	N/A
Tbird-SM	0	0	N/A
Tbird-Other	10	0	N/A

TABLE IV
AVERAGE FWITHIN AND *Distance* FOR NODEHOUR CLUSTERS

	Avg. FWithin	Avg. *Distance*
BGL-Compute	0.000	0.963
BGL-IO	0.000	0.965
BGL-Link	0.000	0.914
BGL-Other	0.027	0.927
Liberty-Compute	0.000	0.748
Liberty-Admin	0.034	0.741
Liberty-Other	0.001	0.576
Spirit-Compute	0.003	0.784
Spirit-Admin	0.059	0.786
Spirit-Other	0.000	0.672
Tbird-Compute	0.000	0.896
Tbird-Admin	0.001	0.783
Tbird-SM	0.001	0.774
Tbird-Other	0.003	0.903

of the Fwithin and *Distance* results for each of the datasets can be found in Table IV.

B. Ground Truth

If the clusters formed have some sort of conceptual meaning then they should for the most part be able to separate alert behavior from normal behavior. This means that if at least one nodehour in a cluster is indicative of alert behavior then most of the other nodehours in the cluster should also be indicative of alert behavior. We present results, which highlight this for the BGL-Compute, Liberty-Compute, Spirit-Compute and Tbird-Compute categories in Fig. 3. The results of this graph indicate that our clustering method was able to separate normal nodehours from alert nodehours to a good degree. Note that there are other clusters not shown in the graphs in Fig 3, which consist entirely of normal nodehours.

The ground truth provided does not just give us the ability to differentiate alert messages from normal messages. It also provides alert categories that allow us to differentiate different kinds of alerts. The conceptual purity ratio attempts to measure the degree to which the clusters are able to differentiate these alert categories, i.e. if one of the nodehours in a cluster contains the signature for a specific alert category to what degree do the other nodehours also show the signature for the alert category. The results show an average conceptual purity ratio of ~0.96 with regard to the alert categories. Details are provided in Table V. The first two columns of this table present the number of alert categories (this is for the entire dataset and not just those found in the alert clusters) and the number of alert clusters respectively.

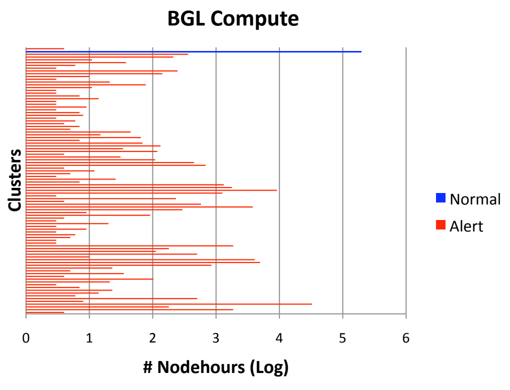
Since we only considered clusters of size $>= 10$, some of our datasets did not have any alert clusters of size $>= 10$, hence we had no results for these datasets for conceptual purity. The results also show that the relationship between alert categories and alert clusters did not follow a one to one correspondence in some instances. The signature of an alert category could be linked to several alert clusters, while an alert cluster could be linked with more than one alert category. Cases where more than one alert category is linked to a cluster probably indicate the existence of correlated alert categories that sometimes occur together and an alert category being linked to different alert clusters could indicate different background activity at the point when the error occurs. This background activity could indicate different causes for the same error. Overall, this highlights the complex interactions that go on in the system with events showing different correlated behavior at different times. It is therefore an advantage that our approach can potentially differentiate these different temporally dependent behaviors.

We are only able to measure the degree to which the clusters formed mirror alert behavior, due to the fact that we only had ground truth for alert behavior. It is however likely that results for alert behavior could be extended to normal behavior.

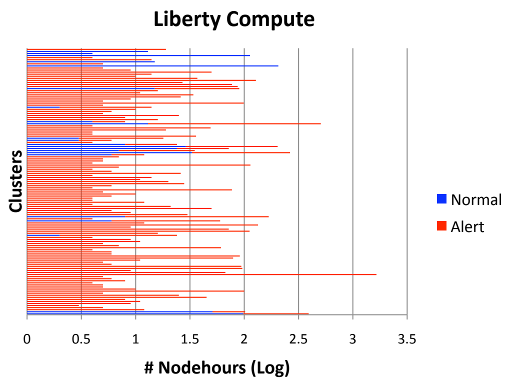
C. Hypotheses

These results overall help to validate the hypotheses stated in Sec. III-A.

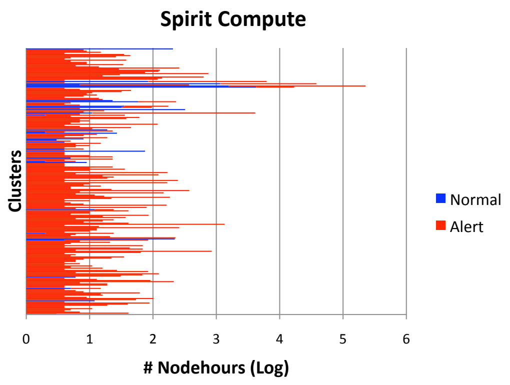
- **Nodehours with the same information content score (based on Eq. 4), could contain the same unique set of message types :** The Fwithin results i.e. Eq. 7 validate this hypothesis.
- **Information content scores, which occur frequently represent nodehours, which contain strongly correlated message types and could represent some system behavior or characteristic :** The *Distance* i.e. Eq. 7 and concept purity results validate this hypothesis.



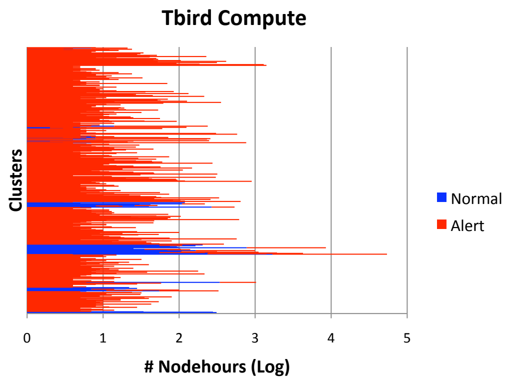
(a) BGL



(b) Liberty



(c) Spirit



(d) Tbird

Fig. 3. Stacked Bar Graphs for select node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represents the distribution of alert nodehours (red) to normal nodehours (blue).

VI. Discussion

In this section, we discuss the implications of our results and our ideas about how this method can be generalized.

We discussed in Section III-A the constraints on the distribution of message types across nodehours, which allow our method to work. These constraints have to be met for the proposed method to work. These datasets are from four different HPC systems that are different in configuration, installation location and usage. Despite this, the constraints applied to them all. This gives a strong support that these constraints are generalizable to HPC systems and might even be to other systems that are similar such as Data-centers.

The first constraint can be achieved by the utilization of sufficiently large datasets i.e. a data set that gives us sufficient bins (spatio-temporal partitions) of the system log. The second constraint on the other hand, likely results due to the Pareto sampling of messages and the choice of the time window for the spatio-temporal partitions used. Since the Pareto property is a characteristic of the log data, it is generalizable to most system logs. Therefore, meeting the second constraint would only require careful selection of an appropriate time window for the spatio-temporal partitions. A one hour time window (from a nodehour) was utilized in our evaluations and was able to generalize for the 14 datasets used. Previous work states that correlated messages in system logs could occur within time windows of between 1 second to 1 day [12]. A more generalized approach would be to vary this time window with the inter-arrival rate of messages in the system log. However, once patterns have been extracted from the system log, they can be applied without recourse to the time window that is used to extract them.

For the proposed method to work, it is necessary for the events in a system log to be separable by source and time. This is required for the information content calculations. This method cannot be generalizable for a system log where source and time information is not available.

Is this method be generalized to an system log, which contained only messages for a single node or computer? Our method is generalizable to fit this scenario because it is possible to leverage other system log fields to act as *sources* for the events in the log. Reporting processes or applications could be utilized if known; there is also the possibility of using time slices. In the use of time slices, we decompose the events, so events produced during similar time slices (days of the week, off-peak, on-peak, weekdays, weekends etc) can be compared. The time slice should be chosen in a way that

TABLE VI
CLUSTER MEASUREMENTS FOR A RANDOM SINGLE NODE

	Avg. Fwithin	Avg. *Distance*	Conceptual Purity
BGL-Compute	0.00	1.00	1.00
Liberty-Compute	0.00	0.741	1.00
Spirit-Compute	0.00	0.734	N/A
Tbird-Compute	0.00	0.766	N/A

ensures that the time slices where similar activity is performed. We provide proof of concept results for this in Table VI. This table shows the results for running the proposed method on events produced by a randomly selected node form each of the *.Compute datasets. In this case, we set C in Eq. 1 to be the set of 24-hour periods in the data. The matrix $\mathbf{X}$ then represents a $|W| \times |C|$ matrix where $x_{w,c}$ is the count of the number of times term w was produced during a 24-hour period c.

The proposed approach ignores the order in which messages occur in a nodehour in determining patterns. It could be interesting to investigate how taking this into consideration could affect the results. However, since clocks among several computers are usually not well synchronized to the level of precision that is required for such analysis, the correct ordering of the events is not guaranteed [20], [6]. Therefore checking to see if the ordering matters maybe futile.

VII. CONCLUSIONS AND FUTURE WORK

In this work, we demonstrate how the grouping of spatio-temporal partitions of a system log based on the entropy-based information content can lead to the formation of conceptual clusters. We evaluate our method using 14 datasets derived from the functional decomposition of system logs of four HPC systems. The results show that the clusters are well formed i.e. having high internal cohesion (Fwithin) and high external separation (*Distance*). We also show that with a high level of confidence we could *conceptually* map the clusters to different alert categories. While our results are only tested on alert behavior, the proposed approach could be extended to normal behavior.

The well formed nature of the clusters produced by the proposed method lends credence to the assertion that these clusters potentially mirror the behavioral characteristics of the system. We have ground truth at a low level but no ground truth at the level of system behavior, it is therefore impossible to fully test this assertion. However, the metrics we have used to evaluate our results, using low level ground truth and internal measures, provide sufficient evidence that these clusters are useable for describing the state of a system at a point in time. Hence our results have practical applications for enhancing the self-awareness and self-monitoring capabilities of a system. Characteristics, which are important for an autonomic system [4].

Future work will entail testing of our approach on non-HPC system logs and tests to determine an appropriate method for choosing a time window for the spatio-temporal partitions used in the approach. The analysis of the clusters produced by our method would also be interesting. This would be a bid to further validate them as indicators of system state and the building of models of state transition. Since our method is able to separate alert behavior from normal behavior, it will be interesting to see how information from the clusters can be used to improve the task of alert detection [5], [8].

ACKNOWLEDGEMENTS

The authors would like to thank Jon Stearley of Sandia National laboratory for his help while re-engineering the Node-info framework. This research is supported by a Natural Science and Engineering Research Council of Canada (NSERC) Strategic Project Grant. This work is conducted as part of the Dalhousie NIMS Lab at http://www.cs.dal.ca/projectx/.

REFERENCES

[1] J. O. Kephart and D. M. Chess, “The Vision of Autonomic Computing,” *Computer, Monthly publication of the IEEE Computer Society*, vol. 36, pp. 41– 50, June 2003.

[2] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, “Capturing, indexing, clustering, and retrieving system history,” in *Proceedings of the twentieth ACM symposium on Operating systems principles*, ser. SOSP ’05. New York, NY, USA: ACM, 2005, pp. 105–118. [Online]. Available: http://doi.acm.org/10.1145/1095810.1095821

[3] C. Lim, N. Singh, and S. Yajnik, “A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems,” in *Proceedings of The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008)*, June 2008.

[4] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, “Fulfilling the Vision of Autonomic Computing,” *Computer, Monthly publication of the IEEE Computer Society*, vol. 43, no. 1, pp. 35–41, January 2010.

[5] A. Oliner, A. Aiken, and J. Stearley, “Alert Detection in System Logs,” in *Proceedings of the International Conference on Data Mining (ICDM). Pisa, Italy.* Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 959–964.

[6] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” in *SOSP ’09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems principles*. New York, NY, USA: ACM, 2009, pp. 117–132.

[7] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, “Fast Entropy Based Alert Detection in Supercomputer Logs,” in *Proceedings of the DSN Workshop on Proactive Failure Avoidance, Recovery and Maintenance (PFARM)*, June 2010.

[8] ——, “An Evaluation of Entropy Based Approaches to Alert Detection in High Performance Cluster Logs,” in *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST)*, September 2010.

[9] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[10] 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.*, July 2005 2005, pp. 476–485.

[11] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward, “Dependency-aware fault diagnosis with metric-correlation models in enterprise software systems,” in *Proceedings of the 6th International Conference on Network and Service Management (CNSM)*, October 2010, pp. 137 – 141.

[12] T. Li, F. Liang, S. Ma, and W. Peng, “An Integrated Framework on Mining Log Files for Computing System Management,” in *Proceedings of of ACM KDD 2005*, 2005, pp. 776–781.

[13] W. Peng, C. Perng, T. Li, and H. Wang, “Event summarization for system management,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD ’07. New York, NY, USA: ACM, 2007, pp. 1028–1032. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281305

[14] R. Vaarandi, “Mining Event Logs with SLCT and Loghound,” in *Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium*, April 2008, pp. 1071–1074.

[15] M. Aharon, G. Barash, I. Cohen, and E. Mordechai, "One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs," *Lecture Notes in Computer Science*, vol. 5781/2009, pp. 227–243, 2009.

[16] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer, "Path-based faliure and evolution management," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*. Berkeley, CA, USA: USENIX Association, 2004, pp. 23–23. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251175.1251198

[17] M. Jiang, M. Munawar, T. Reidemeister, and P. Ward, "Automatic fault detection and diagnosis in complex software systems by information-theoretic monitoring," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, July 2009, pp. 285 –294.

[18] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering Event Logs Using Iterative Partitioning," in *Proceedings of the 15th ACM Conference on Knowledge Discovery in Data.*, July 2009, pp. 1255–1264.

[19] A. Makanju, N. Zincir-Heywood, and E. E. Milios, "Message type extraction based alert detection in system logs," Dalhousie University, Tech. Rep. CS-2009-08, March 2009.

[20] R. Vaarandi, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," in *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems (LNCS)*, vol. 3283, 2004, pp. 293–308.

[21] B. Topol, "Automating Problem Determination: A First Step Toward Self Healing Computing Systems," IBM White Paper, October 2003. [Online]. Available: http://www-106.ibm.com/developerworks/autonomic/library/ac-summary/ac-prob.html

[22] J. E. Cook, Z. Du, C. Liu, and A. L. Wolf, "Discovering models of behavior for concurrent workflows," *Comput. Ind.*, vol. 53, pp. 297–319, April 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=982250.982255

[23] D. Thoenen, J. Riosa, and J. Hellerstein, "Event relationship networks: a framework for action oriented analysis in event management," in *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, 2001, pp. 593 –606.

[24] C.-S. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein, "Data-driven validation, completion and construction of event relationship networks," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 729–734. [Online]. Available: http://doi.acm.org/10.1145/956750.956848

[25] J. Stearley and A. Oliner, "Bad Words: Finding Faults in Spirit's Syslogs," in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, May 2008, pp. 765 –770.

[26] J. Reuning, "Applying Term Weight Techniques to Event Log Analysis for Intrusion Detection," Master's thesis, University of North Carolina at Chapel Hill, July 2004.

[27] "Usenix - the computer failure data repository," Last Accessed June 2009. [Online]. Available: http://cfdr.usenix.org/data.html

[28] A. Oliner and J. Stearley, "What Supercomputers say: A Study of Five System Logs." in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007 (DSN '07)*, June 2007, pp. 575–584.

[29] R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns from Event Logs," in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, 2003, pp. 119–126.