Spatio-Temporal Decomposition, Clustering and Identification for Alert Detection in 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

Markus Latzel
Palomino System Innovations Inc.
Toronto, Ontario
M6G 1A8, Canada
markus@palominosys.com

27 July 2011

Abstract

In this work, we describe our research efforts at detecting alerts in event logs by analyzing the spatio-temporal partitions of a system log. Our research shows that these spatio-temporal partitions produce clusters, which can separate normal activity from anomalous activity, with a high accuracy. Therefore, a system, which can accurately identify these clusters into classes would provide an effective alert detection mechanism. While the steps of the framework described in this paper utilizes an entropy based approach for the clustering of the spatio-temporal partitions and heuristics for the identification of the resultant clusters, it is general enough to allow flexibility in the choice of methods used at each step of the framework.

1 Introduction

Faults and downtime events are routine in the world of large scale system management. Such faults can in some situations be fixed with a simple solution like a reboot, if there is adequate redundancy in the system. In more serious cases, quick diagnosis and repair are crucial to maintain uptime requirements and conform to service-level-agreements (SLAs). Information sources, such as

system logs (that contain event descriptions from the several interrelated components on the system), which can provide pointers to the failure root cause(s), are crucial at this point [2, 6]. Manual inspection of system logs proves to be unrealistic in large environments. Furthermore, recent adoption of virtualized cloud computing infrastructure causes systems logs to be multi-tiered, i.e. disjoint log information is gathered on multiple physical, and virtual appliances. Due to this fact, current research efforts have focused on the development of tools and techniques for the automatic analysis of system logs [15].

This work describes our research efforts at developing a Spatio-Temporal Alert Detection (STAD) framework for *alert* detection in system logs. We refer to *alerts* as events (or groups of events) in a system log that are symptomatic of failure or require the attention of an administrator. The framework consists of three main components/steps. Firstly, it decomposes the contents of a log spatio-temporally. Secondly it clusters the resultant decomposed units. If the clustering technique used in the second step is able to produce clusters that group different kinds of normal and anomalous activity together, then a third step is activated to separate the clusters that contain anomalous activity from those that contain normal activity, thus effectively detecting the *alerts* in the anomalous clusters. The framework can be implemented with limited user (system/network administrators) input and the methods used at each step of the framework can be chosen at the discretion of the user.

In our implementation, we utilize *nodehours* [10] as spatio-temporal units. Other spatio-temporal units can be chosen, the choice is purpose and system dependent. A nodehour is one hour of log information from a single node on the network. We utilize entropy based information content for clustering of the nodehours, which follows from entropy based alert detection [10, 9, 8]. The entropy based Nodeinfo alert detection is already deployed on production systems at Sandia National Laboratories (SNL) in Alburquerque, NM [10]. On the other hand, we utilize a heuristic approach for cluster identification.

Our research contributes to entropy based approaches to alert detection in two ways. Firstly, it allows alert detection in situations where the nodes on the network are dissimilar (an entropy based assumes similarity between the nodes on the network) and secondly it allows the determination of alerts without resort to the information content ranking of spatio-temporal partitions. The heuristics used in identifying the clusters, which do not directly relate to entropy based approaches, are also a contribution of this paper.

Our evaluation of this technique shows that we are able to detect 100% of all alerts with a false positive rate of 0.8% in the best case, while achieving a detection rate of 78% and a false positive rate of 5.4% on average. This evaluation was carried out using real-world log data from four high performance clusters (HPC), which are some of the fastest supercomputers in the world [11]. These logs are publicly available from the USENIX Computer Failure Data Repository (CFDR) and contain approximately 750 million log events in 81GB of textual files [17]. The alert messages in these system logs have been previously labelled by the system administrators at the institutions where these HPC systems are deployed i.e. Sandia National Laboratories and Lawrence Livermore

National Laboratories (LLNL) in Livermore, California. Thus, showing that the automatic discovery of these alerts is interesting and useful in a real-world application. Since these datasets are publicly available, our research results are also reproducible.

We compare our method with alert detection results from a version of Nodeinfo that uses message types as terms [9]. We will refer to this version as *NodeinfoPlus*. For this comparison, we determine the value of k required to achieve a similar detection rate as STAD and calculate the false positive rate at that point. The average false positive rate achieved by NodeinfoPlus across the datasets is approximately 25% compared to 5.4% for STAD. An analysis of variance (ANOVA) test (at 5% significance) carried between the false positive rates achieved by *NodeinfoPlus* and those achieved by STAD show a statistically significant difference in favor of STAD. These results provide a comparison between our work and a deployed alert detection mechanism.

The rest of this paper is organized as follows. We discuss previous work in Section 2. Section 3 discusses the steps of the STAD framework in detail. Section 4 and Section 5 discuss observed characteristics and assumptions about alert clusters and how these are used in deriving identification heuristics, respectively. In Section 6 we describe the methodology of our evaluations, whereas the results of those evaluations are discussed in Section 7. Finally, conclusions are drawn and the future work is discussed in Section 8.

2 Previous work

Perhaps the earliest work to recognize the importance of system log events to automatic system management and autonomic computing can be found in [16]. In this work, the author proposes a 3-tiered data driven approach to discovering knowledge in system logs. Other interesting approaches to the use of system logs in system management can be found in [5] and [12], in both works the authors propose frameworks that categorize system logs events into categories and the use of temporal information, statistical modeling and visualization to interpret and find relationships between the event categories.

Specifically, as it relates to alert detection in system management, approaches to alert detection vary from simple approaches that search system logs for message patterns, which are indicative of previously known failure conditions [13], to visualization techniques that aid the detection of alerts manually [5] and to more complex schemes that use computational techniques. Recent computational approaches include entropy based Nodeinfo [10], Principal Component Analysis (PCA) based detection [19], Principle Atom Recognition in Sets (PARIS) [1] and Finite State Auotmata (FSA) based detection [3].

Recent work has attempted to improve on the Nodeinfo entropy based alert detection technique [9, 8] by introducing the concept of message types into the framework and making modifications to its anomaly scoring mechanism. Entropy based alert detection in system logs work by assigning an information content score to spatio-temporal partitions of an event log. The information

content scores of the spatio-temporal partitions are calculated by exploiting the similarity between node sources. When the spatio-temporal partitions of the log are ranked based on their information content scores, we assume that the partitions on the top of the list are more likely to contain alerts. The STAD framework implementation described in this work utilizes an entropy bases approach for clustering spatio-temporal log partitions. The STAD framework is described in detail in the next section.

3 Spatio-Temporal Alert Detection

In this section, we detail the methods and techniques used in the STAD framework. The STAD framework has three main steps: 1) Spatio-Temporal Decomposition of log events, 2) Clustering of Spatio-Temporal partitions and 3) Anomaly based identification of clusters. We describe the steps in detail in the following sections.

3.1 Spatio-Temporal Decomposition of log events

System logs on large and complex systems where the manual inspection of system logs has become unrealistic, would contain information from several components that make up the system. For this reason, system logs should be good indicators of system state. However, a single reported event in the system log is unlikely to be a good indicator of system state. Strongly correlated events in the log are generally more interesting and are better indicators of system state [14].

Previous work in the log analysis have usually based their analysis on the correlated events in the log rather than single events. Previous approaches to find correlated events in logs include frequent itemset mining [18, 6], tracking of variables reported in message types [19] and the PARIS (Principal Atom Recognition in Sets) algorithm [1]. One of the major mitigations against finding correlated events in system logs is the fact that correlated events may not always follow each other in sequence in the system logs [18]. To this end, the approach utilized here is the decomposition of the event log spatio-temporally.

Events in system logs are typically not homogenous entities. Apart from the textual descriptions of the event, they also contain information about the reporting component (source), which could be a hardware and/or software component, and the occurrence time of the event (timestamp). By using the source and timestamp information to decompose the events in an event log such that each resultant unit of the event log contains only events from a single source over a unit of time, we increase the chance that the events reported in such units are correlated. Any combination of source and time information can be used for decomposing the contents of an event log spatio-temporally, however we utilize nodehours in this work [10].

3.2 Clustering of Spatio-Temporal partitions

The aim of this step of the process is to place the spatio-temporal partitions of the event log into partitions based on their similarity while minimizing the similarity between the eventual clusters. Any clustering technique could be used to achieve this aim, however, we utilize an information content based technique for this step. The technique leverages on previous work in the entropy-based approach to *alert* detection in system logs [10, 9, 8].

Before information content based clustering of nodehours can be carried out, the entropy based analysis of the contents of a log needs to be completed. So firstly, we assign entropy based information content scores to each *term* that appears in the free form message fields of the events in the log. *Terms* in our case would refer to the tokens in the free-form message field of a log event after Full Message Type Transformation (MTT) is applied[9]. Full MTT works by determining the message type of each free-form message in the system log, and replacing the free-form message with a single token. Message types are textual templates which abstract the free-form messages in event logs. Messages which belong to the same message type usually have the same semantic meaning. Unfortunately, message types are not always known apriori. Therefore in our work, we extract them automatically using the Iterative Partitioning Log Mining (IPLoM[1]) message type extraction algorithm [7].

We now describe how the entropy based analysis of an event log is completed. Let W be the set of unique terms in an system log, we can calculate the entropy based information content of each term using Eqs. 1 and 2. If we let S be the set of sources which the events in the log are attributed to, then matrix $\mathbf{X}$ represents a $|W| \times |S|$ matrix where $x_{w,s}$ is the count of the number of times term w appears in messages having s as source. The set of sources in S need to selected based on their similarity. In our evaluations, we decomposed log files firstly based on the functionality of the node sources in the log. In cases where the group of functional nodes were considered similar, S corresponds to the set of nodes which report events in the log partition. In cases where the group of functional nodes were considered dissimilar, the logs where further decomposed so that each log was produced by only a single node. For each log file produced by a single node, S would correspond to each 24 hour period in the log. Hence, the events are now attributed to a *temporal* source.

The output of this stage is a 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.

$$g_w = 1 + \frac{1}{\log_2(S)} \sum_{s=1}^{S} p_{w,s} \log_2(p_{w,s}) \tag{1}$$

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

Table 1: HPC 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

$$p_{w,s} = \frac{x_{w,s}}{\sum_{s=1}^{S} x_{w,s}} \tag{2}$$

An information content score can thus be assigned to each spatio-temporal partition of the event log. Let's define H_j^c as the j^{th} Nodehour for node c and H as the set of nodehours, we assign an information content score (ICS) to H_j^c using Eq. 3. In Eq. 3, $\mathbf{Z}$ is a $|W| \times |H|$ matrix, where $z_{w,j}^c$ effectively only records unique occurrences of terms in the event data i.e. $z_{w,j}^c$ is 1 if term w appears in nodehour H_j^c, 0 otherwise. Therefore, Eq. 3 assigns an ICS to a nodehour based on the magnitude of the vector of information content values of the terms contained in Nodehour H_j^c.

$$ICS(H_j^c) = \sqrt{\sum_{w=1}^{|W|} (g_w * z_{w,j}^c)^2} \tag{3}$$

The *terms* in a nodehour and the information content score for the nodehour thus become the features by which information content clustering is carried out. Information content clustering assigns each nodehour to a *cluster* or bin, which can be described using the tuple:
$(ICS, \text{“}MaxEntropyMsgType''\text{)}$, where ICS is a nodehour information content score value and $\text{“}MaxEntropyMsgType''$ is the *ID* of the message type with the maximum information content score value among all the message types that have instances in the nodehour. All nodehours with the same values for the tuple will end up in the same cluster.

This simple *conceptual clustering* technique exploits the strong clustering of nodehours around single ICS values, which we observed in our experiments. We utilized this method to cluster the event logs from the HPC logs. The statistics of these event logs are shown in Table 1.

The graph in Fig. 1 shows a scatter plot of nodehours from the BGL-Link category. The BGL-Link category is one of the functional node categories derived from BGL HPC log in Table 1. Aside from the fact that the *alert* nodehours, i.e. nodehours that contain alert signatures have high ICS, we also notice a strong clustering of nodehours around single ICS values. 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 information content score value derived from the use of Eq. 3, 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. We therefore theorize that Nodehours with the same information content score value contain the same unique set of message types and information content scores, which occur frequently, represent some system behavior or characteristic

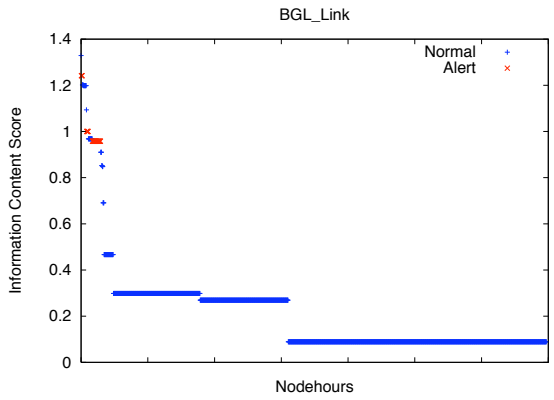


Figure 1: **BGL-Link Category:** Scatter Plot of nodehours (x-axis) vs. information content scores (y-axis). The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on information content score in the plot.

Our evaluations of the clusters formed from the nodehours of the functional node categories of the four HPC logs has shown these hypotheses to be plausible. These clusters created via information content clustering showed high internal cohesion and high external separation. We also observed that if the signature of an alert type could be found in one of the nodehours in a cluster then we could predict with 96% confidence that all the other nodehours in the cluster would also contain the signature for the alert type. With this observation, alert detection is reduced to the task of identifying the derived clusters based on the chance that they contain alert nodehours. This is what the third step of STAD framework attempts to achieve.

3.3 Anomaly based Identification of clusters

This step of the framework involves the separation of the clusters into two classes; an anomalous class and a normal class. Once a cluster is determined to be anomalous, all the nodehours that are part of that cluster are assumed to contain alerts. As with the other steps of the framework, any method of separation can be utilized to achieve this.

To carry out the separation of the clusters, we identified four important characteristics of alerts. These are the Bursty, Endemic, Epidemic and Near-Periodic properties. Using these properties, we derive three assumptions about alert clusters, which can be used in separating them from normal clusters. By alert cluster, we refer to clusters which contain a majority of nodehours with

alert signatures. These properties/assumptions and the identification heuristics implemented based on the assumptions are described in detail with real examples from the HPC logs in Table 1 in Section 4 and Section 5. The methods described for the identification of the clusters described in Section 5 have not been previously published and thus represent a contribution of this work.

4 Cluster Separation

In this section, we first describe the four alert characteristics, which we identified, see Section 4.1. Then, we propose three assumptions about alert clusters based on the identified characteristics, see Section 4.2. Finally, in Section 5, we describe the heuristics based on the assumptions as implemented in our evaluations.

4.1 Alert Characteristics

Four important alert characteristics are identified. They are described in detail below. We note that these alert properties are not mutually exclusive and are not intended to be exhaustive.

- **Bursty Property**: The bursty property occurs when we see a significant increase in the number of events over a period of time. An example of the bursty property is shown in Fig. 2. This figure shows the number of events (measured by size in bytes) produced by a single node (Ln30) on the Liberty high performance cluster on an hourly basis over a 24 hour period. We see a significant increase in the number of bytes produced in the 15th to 18th hours in the logs. Based on the labeling, there were at least 9 alert types active during this period, with the $R\_EXT\_CCISS, R\_EXT\_FS\_IO$ and R_EXT_INODE1 types being the major contributors to the burstiness experienced during this period. The number of bytes produced by this node drops to zero in the 19th hour right up to the 24th hour. This indicates that the alert that caused this burstiness, led to a failure of the node. This graph highlights the importance of alert detection in system management. The detection of the this alert in the 15th hour would have given administrators a 3 hour window within which they could have applied remedial action to prevent the failure of the node, which occurred in the 19th hour.
- **Endemic Property**: The endemic property occurs when a cluster shows sporadic activity over a period of time and also shows localized activity at each occurrence. The graph in Fig. 3 shows the activity of a cluster from the BGL log. This cluster is associated with the $KERNMC$ alert type. We can see that occurrences of this cluster type are sporadic and affect only one node at each occurrence.
- **Epidemic Property**: The epidemic property on the other hand occurs when a cluster shows sporadic activity over a period of time just like in

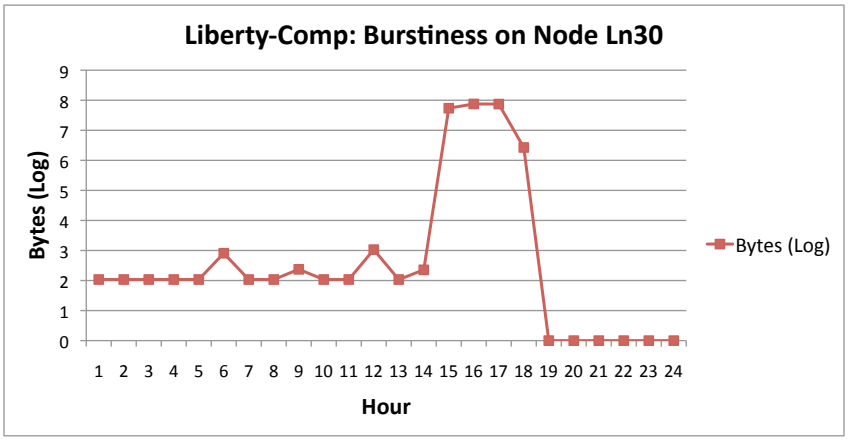


Figure 2: **Bursty Property:** This graph shows the size in Bytes (Log) of the events produced by a single node from the Liberty HPC at hourly intervals over a 24 hour period.

the endemic case but instead affects a relatively large number of nodes at each occurrence. The graph in Fig. 4 shows an example of another cluster from the BGL log, which shows the epidemic property. This cluster is associated with the *KERNREC* alert type. We can see that occurrences of this cluster type are sporadic and affect as many as 2,000 nodes at each occurrence. The activity though wide spread does not affect all nodes, the total number of nodes in the BGL event log category to which this cluster belongs contains 65,554 nodes.

- **Near-Periodic Property**: The near-periodic property occurs when activity in a cluster occurs at almost regular intervals. The graph in Fig. 5 shows an example of a cluster exhibiting the near-periodic property. In this example from the Spirit event log, the cluster seems to occur almost on an hourly basis, over a period spanning about 4 weeks. This almost regular rate of occurrence and the frequency of occurrence is an indication of the near-periodic property. The cluster shown in Fig. 5 is linked to the *R_HDA_NR* and *R_HDA_STAT* alert types.

4.2 Alert Cluster Assumptions

In this section, we describe the properties, which we assume that alert clusters have. These assumptions are based on the identified alert characteristics and are the basis for cluster separation.

- Bursty, Endemic and Epidemic alert types show activity in relatively few time periods. This implies that the number of active periods for a cluster type should be a good indication of whether a cluster contains alerts or not, a cluster type that is active during relatively few time periods is

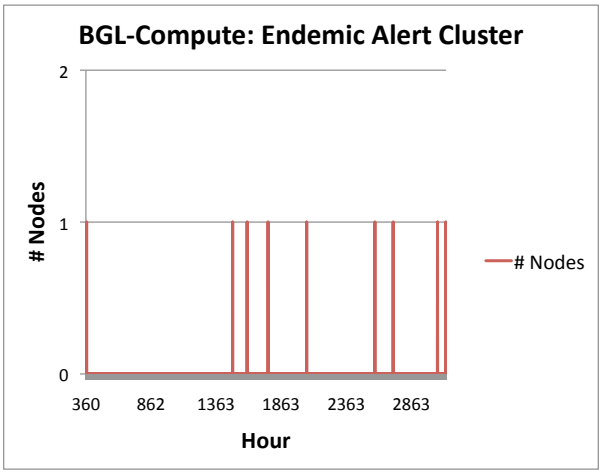


Figure 3: **Endemic Property:** The graph shows an example of a nodehour cluster which is known to be related to an alert type. The cluster type shows localized sporadic activity.

likely to contain alerts. This supports the assumption that alerts are usually infrequent in an event log. The number of active periods is also a better measure of frequency than the count of active nodes or the count of nodehours.

- Endemic alerts show localized activity each time they occur. Measuring localized activity in a cluster would therefore be a good indicator for an alert cluster. A cluster type, which show a high degree of localized activity each time it occurs is likely to contain alerts.

- Near-Periodic alert types have frequent occurrences and may or may not be localized. They therefore may not be captured by the heuristics above. They, however, have the property of having close to regular rates of occurrence. Measuring the periodicity of a cluster would therefore be a good indicator of this property. Clusters, which are relatively frequent and show periodic or *almost* periodic activity are therefore likely to contain alerts.

5 Identification Heuristics

We utilized the assumptions detailed above as a means of identifying the nodehour clusters which are derived from HPC logs detailed in Table 1. The details of the evaluations of the identification method are described in Section 6. However, in this section we describe the heuristics, which were implemented. There are three heuristics, one for each assumption. Before describing the heuristics, we first provide the following definitions.

- Let E be the event log, which we intend to analyze. Let each temporal

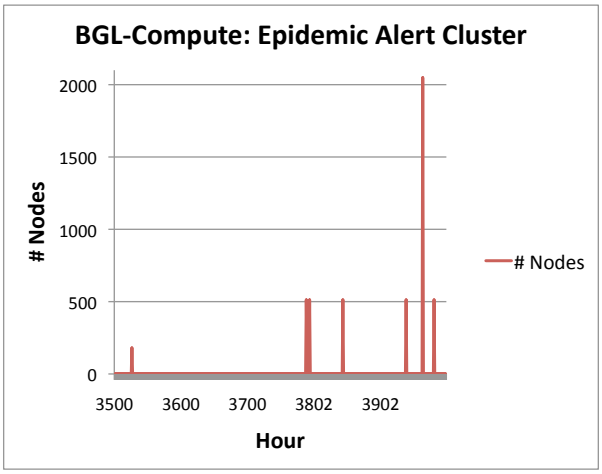


Figure 4: **Epidemic Property:** The graph shows an example of a nodehour cluster which is known to be related to an alert type. The cluster type shows sporadic activity that affects a relatively large number of nodes.

period spanned in E be assigned an ordinal number, n. The first hour is assigned a value of 1 and every subsequent hour is assigned a value of $n+1$ relative to its preceding hour, which would have a value of n.

- We define set $\mathbf{C}$ of spatio-temporal partition clusters derived from E, where $c_i \in \mathbf{C}$ is the ith element of $\mathbf{C}$.
- We define arrays $\mathbf{P}$ and $\mathbf{S}$, such that $\mathbf{P}[i]$ and $\mathbf{S}[i]$ are the counts of temporal periods and event sources reported in the nodehours in cluster c_i respectively.
- For each cluster c_i, we define array $\mathbf{Q}_i$ such that $\mathbf{Q}_i[j]$ is the ordinal number of jth temporal period of activity for cluster c_i.
- For each cluster c_i, we define array $\mathbf{R}_i$ such that $\mathbf{R}_i[j]$ is the count of the number of event sources reporting activity of type c_i during the jth temporal period of activity for cluster c_i.
- $|\mathbf{Q}_i| = |\mathbf{R}_i| = \mathbf{P}[i] = m$

5.1 Heuristic Definitions

- **First Heuristic:** This heuristic implements the first assumption about alert clusters.
  1. Let med_per = Median($\mathbf{P}$), ignoring values where $\mathbf{P}[i] = 1$.
  2. For cluster c_i, if $\mathbf{P}[i] < med\_per$, then c_i is considered an alert.

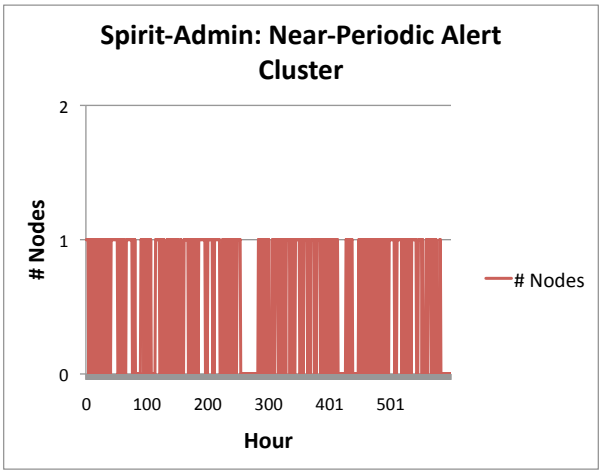


Figure 5: **Near-Periodic Property:** The graph shows an example of a nodehour cluster which is known to be related to an alert type. The cluster type shows very frequent activity compared to either the endemic or epidemic types and has almost periodic rate of occurrence.

Due to the pareto property, which is generally true for statistics involving system logs [18], several values in array $\mathbf{P}$ are $= 1$. Hence they are ignored in the calculation of med_per, if this is not done, then $med\_per = 1$ most of the time.

- **Second Heuristic:** This heuristic implements the second assumption about alert clusters.

  1. Calculate the average inverse node frequency INF_i for cluster c_i using Eq. 4.

$$INF_i = \frac{\sum_{j=1}^{m} \frac{1}{R_i[j]}}{m} \tag{4}$$

  2. Set an average inverse node frequency threshold INT.
  3. For cluster c_i, if $INF_i >= INT$, then then c_i is considered an alert. The average inverse node frequency metric, which is described in Eq. 4 attempts to provide a measure for localized activity. Its values are in the range $(0, 1]$ and the values close to one indicate localized activity within the cluster. The graph in Fig. 6 is a scatter plot of the alert clusters in the Liberty-Compute node category versus their average inverse node frequency values. The Liberty-Compute node category is one of the functional node categories from the Liberty HPC log detailed in Table 1. Fig. 6 shows that most of the alert clusters have an average inverse node frequency value of 1 and therefore are showing the endemic property. In our implementation, INT is set to 0.95.

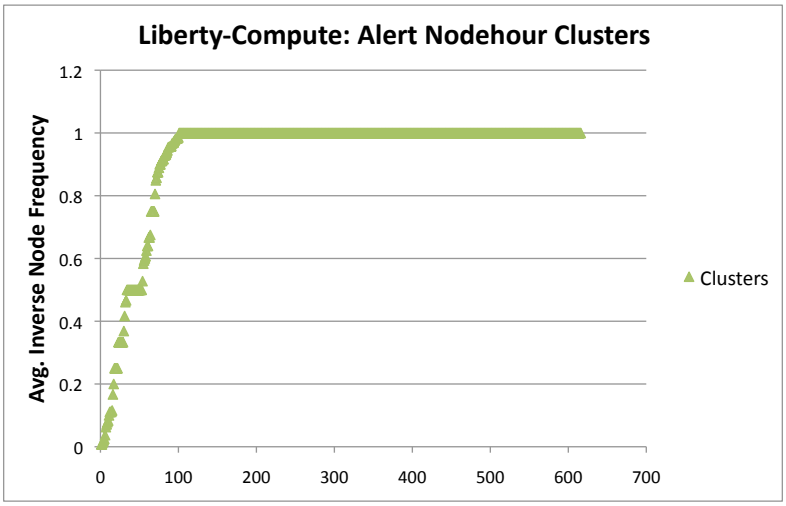


Figure 6: **Liberty-Compute Alert Clusters:** The graph shows a scatter plot of the alert clusters derived from the Liberty HPC log versus their inverse node frequency scores. The clusters are sorted based on their average inverse node frequency scores.

- **Third Heuristic:** This heuristic implements the third assumption about alert clusters.

  1. Calculate the mean time between activity μ_i for cluster c_i using Eqn. 5. μ_i represents the expected time between system activity of type c_i, if cluster c_i was periodic.

$$\mu_i = \frac{\sum_{j=1}^{m-1}(Q_i[j+1] - Q_i[j])}{m-1} \tag{5}$$

  2. Calculate the standard deviation STD_i from μ_i of intervals between system activity of type c_i using Eqn. 6. We assume that STD_i values close to 0 indicate near-periodic activity.

$$STD_i = \sqrt{\frac{\sum_{j=1}^{m-1}[(Q_i[j+1] - Q_i[j]) - \mu_i]}{m-1}} \tag{6}$$

  3. Set a standard deviation threshold STT.
  4. For cluster c_i, if $STD_i < STT$, then c_i is considered an alert.

Lets set the $norm\_per\_cnt_i$ (normalized period count) for any cluster c_i as $\frac{\mathbf{P}[i]}{Max(\mathbf{P})}$. The bar graph in Fig. 7 shows STD for the Spirit-Admin clusters with a $norm\_per\_cnt$ greater than 0.1. The Spirit-Admin node category is one of the functional node categories from the Spirit HPC log detailed in Table 1. Clusters with a $norm\_per\_cnt$ greater than 0.1 represent those

clusters with relatively high number of active periods. Cluster 16 with a STD value of approximately 2 is the only cluster that contains alert nodehours. The adjacent clusters i.e clusters 15 and 16 have STD values of approximately 1.8 and 4.0 respectively. This shows that a STD close to zero is a good indicator for an alert cluster. In our implementation, STT is set to 2.5.

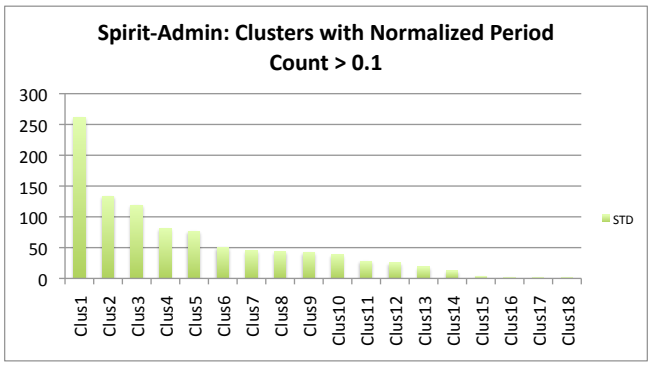


Figure 7: **Spirit Admin Clusters:** This graph shows a bar chart of clusters with a $norm\_per\_cnt > 0.1$ from Admin functional node category of the Spirit HPC log. Cluster 16 is the only cluster which is know to contain alerts.

Since the near-period property requires frequent occurrence, we only apply this heuristic to mid-size clusters, which is in respect to the count of active periods. We leave out clusters with large period counts as they are likely normal. With this in mind, we can set upper and lower bounds for the $norm\_per\_cnt$ and apply this heuristic only to clusters with a $norm\_per\_cnt$ value, which falls within these bounds. In our implementation, we set upper and lower bounds to 0.3 and 0.1 respectively. The values were set based on the pareto property of event logs, clusters with mid-sized period counts would likely fall within these bounds.

We know summarize the procedure for identifying a cluster c_i as either an alert or a normal cluster using the heuristics above. For any cluster c_i, if either of the heuristics above are true, it is set as an alert cluster and all the nodehours in it as set as alert nodehours. If all of the heuristics are false then c_i is considered along with all the nodehours in c_i. We note that the 2nd heuristic is not used, when dealing with a dissimilar node scenario. The average inverse node frequency metric has no value for distinguishing alert clusters in such a scenario.

6 Experiments

The evaluations involved 13 datasets. The 13 datasets are based on the functional node groups from the 4 HPC event logs listed in Table 1. Statistics

Table 2: System log Data Functional Grouping Statistics

	# Events	# Nodes	# Nodehours	# Msg-Types	% Alerts	Similar Nodes
BGL-Compute	4,153,009	65,554	1,581,845	399	4.2	Y
BGL-IO	400,923	1,024	219,722	49	38.22	Y
BGL-Link	2,935	517	1,395	13	2.37	Y
BGL-Other	191,096	2,167	13,666	97	0.43	Y
Liberty-Compute	200,940,735	236	1,748,865	481	0.29	Y
Liberty-Admin	52,211,676	2	27,162	601	0.04	N
Liberty-Other	12,416,820	6	44,447	510	0.22	N
Spirit-Compute	218,697,851	512	6,648,719	854	0.19	Y
Spirit-Admin	41,847,257	2	26,216	443	3.10	N
Spirit-Other	11,753,861	7	57,532	707	0.25	N
Tbird-Compute	155,403,254	4,514	14.520,204	1,262	0.17	Y
Tbird-Admin	15,306,749	20	100,740	627	0.02	N
Tbird-SM	19,109,810	2	8,859	597	0.00	N
Tbird-Other	21,392,379	1,319	626,030	1,387	0.02	Y

about these datasets are provided in Table 2. In the table *# Events* refers to the number of lines in the log, *# Nodes* refers to the number of nodes in the functional group, *# Nodehours* refers to the number of spatio-temporal partitions derived from the dataset via Nodehour decomposition, *# Msg-Types* refers to the number of message types found in the dataset using IPLoM, *% Alerts* refers to percentage of spatio-temporal partitions, which contain alert signatures based on the domain expert labeling in the logs, while *Similar Nodes* refers to processing methodology used for the dataset. A *Y* in the *Similar Nodes* column indicates that the dataset was processed under the assumption that the nodes in this dataset were sufficiently similar, while a *N* indicates that the nodes were assumed to be dissimilar.

The values for all parameters were set as described in Section 5, except in the case of the *med_per* parameter. Our implementation calls for the value of this parameter to set automatically. We found that the value automatically assigned to this parameter was always in the range $[3, 5]$. Based on this observation, we ran experiments for each dataset where the value of *med_per* was set manually to values in the range $[2, 6]$, in addition to experiments where its value was set automatically. We compare the manually set evaluations against the auto-tune evaluations in the Result section.

The evaluation metrics used for the experiments are Recall (Detection) and False Positive Rates. These metrics are calculated using Eqs. 7 and 8, respectively. The values for the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) used in these equations were derived using the *binary scoring* metric as defined in [10]. In a dissimilar node scenario, processing is performed on a node-by-node basis, but during the evaluation the TPs, FPs, TNs and FNs are summed to provide a single value for Recall and False Positive Rates for the functional group.

To provide a baseline comparison, we compare our method against NodeinfoPlus [9]. NodeinfoPlus uses a rank based mechanism for alert detection, for this reason we only compare false positive results. We determine the value

of k required to the achieve a similar detection rate as those achieved in our experiments for the Top_k separation of nodehours. We then calculate the false positive rate as this point and compare to the false positive rates achieved using STAD.

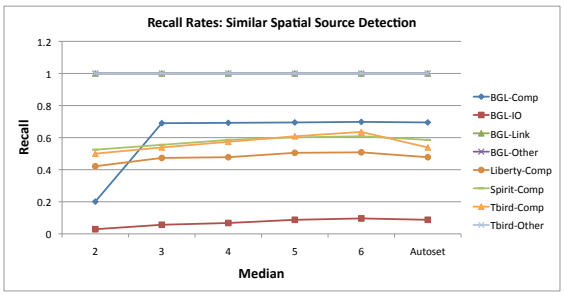
$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

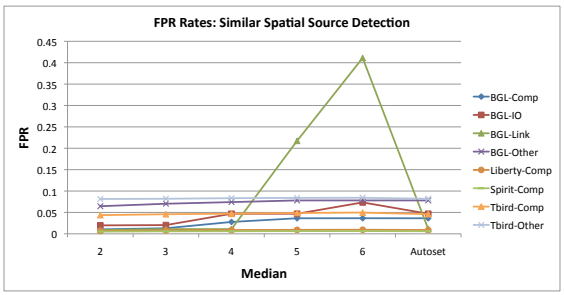
7 Results

For the similar node categories, depending on the value for the *med_per* parameter, we were able to achieve above 50% detection, with a single digit false positive rate for all node categories except the BGL-IO category, see Figs. 8a and 8b. With the *BGL-IO* node category only about 8% detection was achieved. The reasons for this performance are explained in previous work [8]. In this node category, approximately 80% of the alert events are closely correlated to message type signatures that have entropy based information content values, which are less than *0.1*. This indicates an almost equal rate occurrence across nodes. This observation is due to the fact that certain error types in this category are not generated by the individual nodes but by an *IO* subsystem, such errors are then sensed and reported by all nodes. This means that these errors are attributed to the wrong *source* for the entropy based analysis. The high alert nodehour ratio of this node category of 38.22% further emphasizes the fact that the alerts in this node category are unusual. If the results for this node category are adjusted by ignoring the alerts that show this property, the detection rate goes up to approximately 60%. We also note that we were able to achieve 100% detection in three node categories i.e. *BGL-Link* , *BGL-Other* and *Tbird-Other*, irrespective of the value of *med_per*.

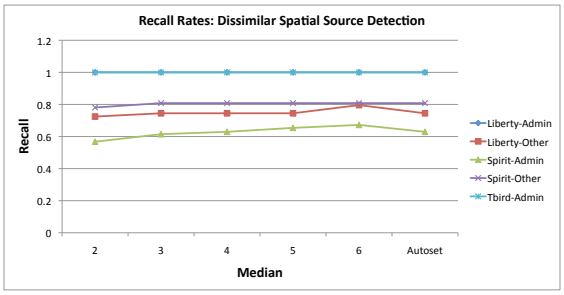
For the dissimilar node categories we were also able to achieve above 60% detection for all node categories depending on the value of the *med_per* parameter. Also achieving 100% detection in two node categories, *Liberty-Admin* and *Tbird-Admin*, irrespective of the value of *med_per*. What is however interesting here is that while we note that setting the *med_per* to 2 gives us single digit false positive rates for 4 out 5 node categories, the false positive rates here tend to be on the order of about 1.5 times larger than those experienced with similar node detection. While these false positive rates are higher, they do improve on the baseline.



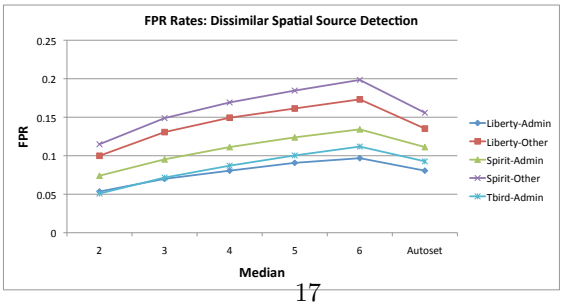
(a) Recall Rate - Similar Nodes



(b) False Positive Rate - Similar Nodes



(c) Recall Rate - Dissimilar Nodes



(d) False Positive Rate - Dissimilar Nodes

Figure 8: This figure shows recall and false positive rates for evaluations of STAD on the datasets listed in Table 2.

Table 3: ANOVA Test Summary

Treatment	**F**	**P-Value**	**F crit**
FPR-Baseline vs. FPR-Autoset	5.036	0.034	4.259
FPR-BestCase vs. FPR-Autoset	0.003	0.957	4.259
DR-BestCase vs. DR-Autoset	0.817	0.375	4.259

A summary of the results of the evaluation is given in Fig. 9. In this graph, we select the best case result for each dataset, from the experiments where the value of the *med_per* parameter was manually set and compare it with the result where the value of *med_per* was set automatically. We also show a baseline false positive rate result using NodeinfoPlus. In choosing the best case, a balance between a high detection and low false positives was considered. The graph shows that the overall best case was achieved with the *BGL-Link* category with 100% detection at a false positive rate of 0.8%. The average detection (across node categories) was 78% and 77% for the manual experiments and auto-tuned experiments, respectively, while the average false positive rate was 5.4%, 6.9% and 25% for the manual experiments, auto-tuned and *NodeinfoPlus* experiments, respectively. An ANOVA test carried out at 5% significance indicates that there is no statistically significant difference between the results achieved by setting the value of *med_per* manually and those achieved by setting it automatically. A similar test between the baseline false positive rates achieved by *NodeinfoPlus* against those achieved by the auto-tuned STAD results show a statistically significant difference. We note that even if there was no statistically significant difference between the results, the fact that STAD achieves similar results without the the manual determination of k for Top_k analysis remains a contribution of our method. In the determination of alerts through the information content ranking of spatio-temporal partitions by a Top_k analysis, it may be difficult to choose a value for k, which can be used on all datasets. The ANOVA results are summarized in Table 3.

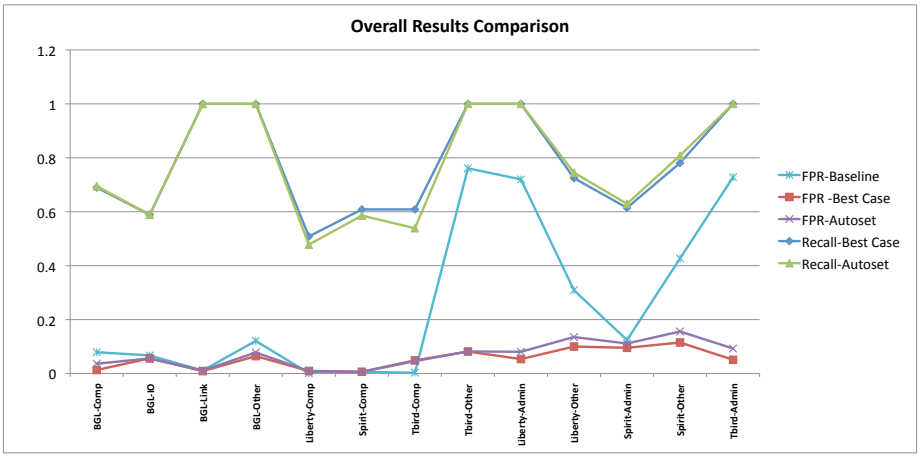


Figure 9: This graph shows the best case results for the experiments where the *med_per* parameter is set manually and the results where its value is set automatically. It also shows the false positive rate achieved by NodeinfoPlus for a similar detection as the autoset experiments. The results for the *BGL-IO* category are adjusted.

8 Conclusion and Future Work

In this work, we evaluated a Spatio-Temporal alert detection framework on the system logs of 4 HPCs. The results show that we could on average detect 78% of all alerts while maintaining a false positive rate of of 5.4%. In the best case, 100% detection at a false positive rate of 0.8% was achieved.

The FPRs achieved by our experiments are not uncommon with anomaly detection systems, where having relatively high FPRs is well documented [4]. It is our opinion that the FPRs achieved in our experiments are low enough to assist a semi-supervised approach. This would involve an administrator going over the detected alerts to document root causes and signatures for actual alerts and flagging signatures for the FPs. The system can therefore use such information for future detection by searching for and reporting known alert signatures thus suppressing future FPs. Such an approach will, over time, lead to the reduction of the FPs to much lower levels. This approach has been used successfully with intrusion detection alarms [4].

Perhaps the most important lesson learnt from our experiments is that it is possible (with an appreciable level of accuracy), once the right properties are identified, to separate clusters of spatio-temporal partitions of event logs that may contain anomalous activity from those that contain normal activity. The anomaly detection heuristics developed in this work are intended to show that separation of the clusters is possible. It is possible to provide further automation by using these properties as training features for a machine learning algorithm. The separation produced from such automatic classification, might increase the accuracy further.

The framework can easily be deployed with little or no user input. The only significant user input is the definition of the similarity categories used in the entropy based calculations. The methods involved in the framework are computationally inexpensive and easy to understand. As reported by our industry partner to this project, recent adoption of virtualized cloud computing infrastructure causes systems logs to be multi-tiered. Thus, manual analysis of these logs proves to be difficult. System administrators of cloud computing systems are desperately in need of tools to make sifting through large volumes of inter-related log data an easier task. A tool such as STAD would prove useful in the correlation of events across multiple tiers making log analysis efforts easier.

The major contributions of this work are: (i) the proposed framework for identification of alert clusters, (ii) the ability to determine the alerts without resorting to a ranking, and (iii) extension of the scope of an entropy based alert detection mechanism to allow detection on a group of dissimilar nodes. The last contribution therefore requires an update of the basic assumption for entropy based alert detection, i.e. *"Similar computers correctly executing similar code will have similar logs"*. The updated basic assumption would thus be: *"System logs events which are produced by similar spatial sources or produced during periods of similar system activity are likely to be similar"*.

Future work will involve the semi-supervised association of alerts to faults (converting anomalies to fault signatures), which should lead to reduction in false positives. Improving the various sub-tasks of the framework i.e message type extraction, spatio-temporal decomposition, nodehour clustering to improve the final output of the framework. Also the use of the identified cluster properties to produce features to train an automatic classifier for the clusters.

Acknowledgements

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] 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*, 5781/2009:227–243, 2009.

[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*, SOSP '05, pages 105–118, New York, NY, USA, 2005. ACM.

[3] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Data Mining,*

2009. ICDM '09. Ninth IEEE International Conference on, pages 149 – 158, December 2009.

[4] K. Julisch. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security*, 6(4):443–471, November 2003.

[5] 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*, pages 776–781, 2005.

[6] 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.

[7] 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*, pages 1255–1264, July 2009.

[8] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. 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] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. Fast Entropy Based Alert Detection in Super Computer Logs. In *Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, DSNW '10, pages 52–58, Washington, DC, USA, June 2010. IEEE Computer Society.

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

[11] 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)*, pages 575–584, June 2007.

[12] 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*, KDD '07, pages 1028–1032, New York, NY, USA, 2007. ACM.

[13] J. E. Prewett. Analyzing Cluster Log Files using Logsurfer. In *Proceedings of the 4th Annual Conference on Linux Clusters*, 2003.

[14] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset. Analyzing System Logs: A New View of What's Important. In *Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, pages 6:1–6:7, Berkeley, CA, USA, 2007. USENIX Association.

[15] J. Stearley. Towards Informatic Analysis of Syslogs. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 309–318, 2004.

[16] R. Sterritt. Towards autonomic computing: effective event management. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 40–47, Dec. 2002.

[17] USENIX. USENIX - The Computer Failure Data Repository. Last Accessed June 2009.

[18] 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)*, volume 3283, pages 293–308, 2004.

[19] 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*, pages 117–132, New York, NY, USA, 2009. ACM.