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

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

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

ABSTRACT

In this work, we propose an approach based on analyzing the spatio-temporal partitions of a system log, generated by supercomputers consisting of several nodes, for alert detection without employing semantic analysis. In this case, "Spatial" refers to the source of the log event and "Temporal" refers to the time the log event was reported. Our research shows that these spatio-temporal partitions can be clustered to separate normal activity from anomalous activity, with high accuracy. Therefore, our proposed method provides an effective alert detection mechanism.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering—*algorithms*

Keywords

Network Control and Management, Event Log Mining, Fault Management, Systems Administration

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, which can provide pointers to the failure root cause(s), are crucial at this point [7]. 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 [14].

SAC'12, March 25-29, 2012, Riva del Garda, Italy.

This work describes our research efforts at developing a framework for *alert* detection in system logs, which we call Spatio-Temporal Alert Detection (STAD). 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 steps. Firstly, it decomposes the contents of a log spatio-temporally. "Spatial" refers to the source of the log event e.g. a node in the supercomputer, and "Temporal" refers to the time the log event was reported i.e. a timestamp. 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* [9] 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. The entropy based Nodeinfo alert detection is already deployed on production systems [9]. On the other hand, we utilize a rule-based approach for cluster identification. The main contributions of this research are: 1.) The basic assumption for entropy based alert detection is that *"Similar computers correctly executing similar code will have similar logs"* [9]. Our work goes beyond this by assuming that *"System logs events which are produced by similar spatial sources or produced during periods of similar system activity are likely to be similar"*. The new assumption allows us to extend the entropy-based approach to groups of dissimilar nodes. 2.) Our system allows the determination of alerts without resort to the information content ranking of spatio-temporal partitions. 3.) A novel feature set for the identification of the anomalous clusters. The rules employed for identification use these features.

Our evaluations of the proposed technique show 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 (recall) 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 [10]. These logs are publicly available from the USENIX Computer Failure Data Repository (CFDR) and contain approximately 750 million log events in 81GB of text files [15]. The alert states in these system logs have been previously labelled by the system administrators where these HPC systems are deployed [1].

We compare STAD and a version of $Nodeinfo$ [9]. Results show that the average false positive rate achieved by $Nodeinfo$ across the datasets is approximately 25% compared to 5.4% for STAD, the proposed system. An analysis of variance (ANOVA) test (at 5% significance) carried between the false positive rates achieved by *Nodeinfo* and those achieved by STAD show a statistically significant difference in favor of STAD.

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 the identification rules, respectively. Section 6 describes the methodology for evaluation and the results are presented in Section 7. Finally, conclusions are drawn and the future work is discussed in Section 8.

2. PREVIOUS WORK

Earlier work recognizes the importance of system log events to automatic system management and autonomic computing and proposes a 3-tiered data driven approach to discovering knowledge in system logs [14]. Other interesting approaches to the use of system logs in system management propose frameworks that classify system logs events into categories and the use of temporal information, statistical modeling and visualization to interpret and find relationships between the event categories [6, 11].

Specifically, approaches to alert detection vary from simple approaches that search system logs for message patterns, which are indicative of previously known failure conditions [12], to visualization techniques that aid the detection of alerts manually [6]. Recent computational approaches include entropy-based detection [9], automatic generation of classification rules [3], Principal Component Analysis (PCA) based detection [17], Principle Atom Recognition in Sets (PARIS) [1] and Finite State Auotmata (FSA) based detection [2]. Moreover, other recent work has attempted to improve on the Nodeinfo entropy based alert detection technique [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 works 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, it is assumed that the partitions on the top of the list are more likely to contain alerts. The STAD framework an entropy based approach for clustering spatio-temporal log partitions. The STAD framework is described in detail in the next section.

[1]A detailed description of these alert states and how they were identified can be found in [10]

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

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 that are described in the following.

3.1 Spatio-Temporal Decomposition of log events

System logs on large and complex systems would contain information from several components that make up the system. 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 [13].

Previous approaches to finding correlated events in logs include frequent itemset mining [16, 7], tracking of variables reported in message types [17] and the PARIS algorithm [1]. One of the major obstacles to finding correlated events in system logs is the fact that correlated events may not always follow each other in sequence in the system logs [16]. 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.

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. We utilize an information content based technique for this step [8]. Clustering using a traditional distance-based approach is not feasible. The features used in clustering will be the unique terms(words) that appear in the log. The number of unique words in a log can easily number in the millions, leading to the curse of dimensionality. The information content scores used in our technique are derived from the terms that appear in the log, after several steps of abstraction [8]. This simple *conceptual clustering* technique exploits the strong clustering of nodehours around information content values, which we observed in our experiments. Our work has shown that these scores provide an accurate means of finding the clusters without much computation.

We refer the reader to [8] for the details of information content clustering. The statistics of the datasets utilized to test the clustering method 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 information content scores (ICS), we also notice a strong clustering of nodehours around single ICS values. The ICS value is in a way a *hash* value for the set of unique message types in a given nodehour, so we can link a distinct information content score to one or more sets of unique message type combinations. We therefore hypothesize that nodehours with the same information content score value contain the same unique set of message types. Therefore information content scores, which occur frequently, can represent some system state.

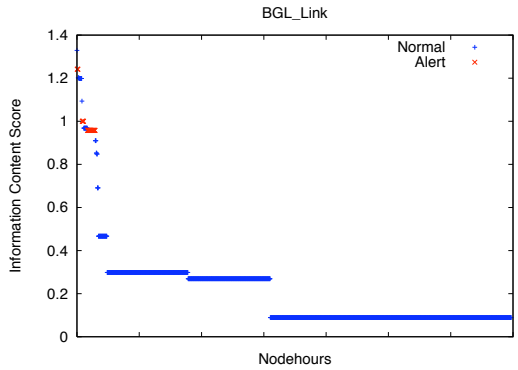


Figure 1: BGL-Link Category: Scatter Plot of nodehours (x-axis) vs. information content scores (y-axis). The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on information content score in the plot.

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. This can be observed in the Circos [5] visualization of the clustering of the BGL-Link nodehours shown in Fig. 2. The figure shows the mapping between the nodehours in the four alert clusters derived using our technique i.e. $1, 2, 3, 4$ and the nodehours known to belong to the five alert categories identified by system administrators in the log data. This visualization shows that all the derived clusters do not contain any $NORMAL$ nodehours and that a $1-1$ or $1-M$ mapping exists between the derived clusters and the alert categories i.e. $Cluseter1 \rightarrow LINKDISC$, $Cluster2 \rightarrow LINKPAP \,\&\, LINKIAP$, $Cluster3 \rightarrow MONPOW$ and $Cluster4 \rightarrow LINKBLL$. We note that the $LINKPAP$ and $LINKIAP$ alert categories are similar. With this observation, alert detection is reduced to the task of identifying the derived clusters based on the possibility that they contain alert nodehours. This is what the third step of the proposed 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.

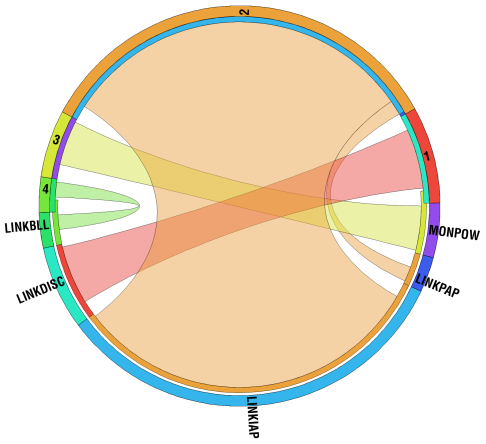


Figure 2: Circos-Table Plot showing the mapping between the administrator defined alert categories and the clusters produced using the information content clustering technique for the BGL-Link data set.

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 that contain a majority of nodehours with alert signatures.

4. CLUSTER SEPARATION

In this section, we first describe the four alert characteristics. Then, we propose three assumptions about alert clusters based on the identified characteristics. Finally, we describe the rules 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. 3. This figure shows the number of events (measured by size in bytes) produced by a single node (Ln30) on the Liberty HPC 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 and including 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 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 that occurred in the 19th hour.

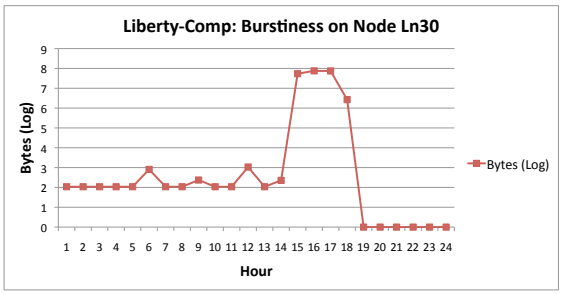


Figure 3: Bursty Property: Events produced by a single node at hourly intervals over a 24 hour period.

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. 4 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.

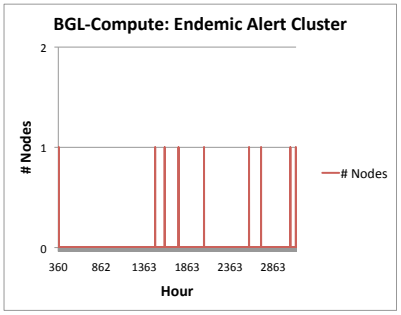


Figure 4: Endemic Property: Localized sporadic activity.

Epidemic Property: The epidemic property on the other hand occurs when a cluster shows sporadic activity over a period of time just like in the endemic case but instead affects a relatively large number of nodes at each occurrence. The graph in Fig. 5 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.

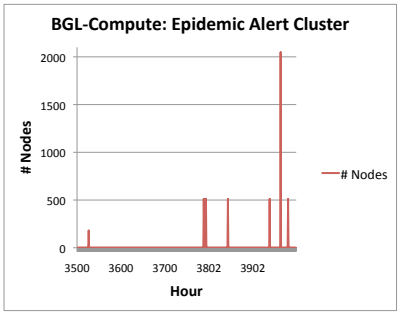


Figure 5: Epidemic Property: Sporadic activity that affects a relatively large number of nodes.

Near-Periodic Property: The near-periodic property occurs when activity in a cluster occurs at almost regular intervals. The graph in Fig. 6 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. 6 is linked to the *R_HDA_NR* and *R_HDA_STAT* alert types.

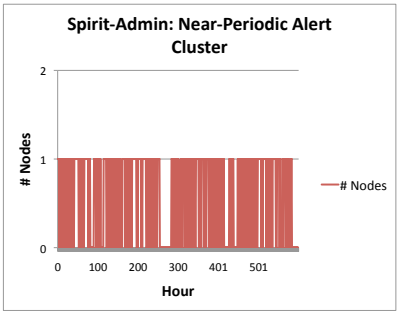


Figure 6: Near-Periodic Property: The graph shows very frequent activity compared to either the endemic or epidemic types and has an almost periodic rate of occurrence.

4.2 Alert Cluster Assumptions

Based on the aforementioned alert characteristics, we make the following assumptions for cluster separation:

1. 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 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.

2. 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 that shows a high degree of localized activity each time it occurs is likely to contain alerts.

3. Near-Periodic alert types have frequent occurrences and may or may not be localized. They therefore may not be captured by the rules 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 RULES

We utilized the assumptions detailed above as a means of identifying the alert nodehour clusters. To this end, there are three rules, one for each assumption. Before describing the rules, we first provide the following definitions.

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

First Rule - Active Periods: This rule implements the first assumption about alert clusters.

1. Let $med\_per = \text{Median}(\mathbf{P})$, 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.

Due to the Pareto property, which is generally true for statistics involving system logs [16], 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 Rule - Localization: This rule implements the second assumption about alert clusters.

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

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

2. Set an average inverse node frequency threshold INT.
3. For cluster c_i, if $INF_i >= INT$, then c_i is considered an alert.

   The average inverse node frequency metric, which is described in Eq. 1 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. 7 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. 7 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.

Third Rule - Periodicity: This rule implements the third assumption about alert clusters.

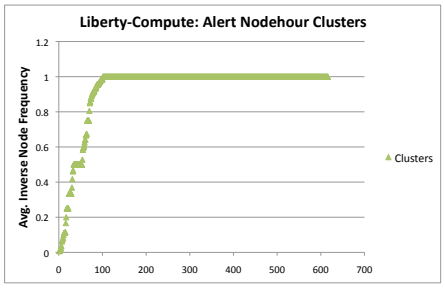


Figure 7: Liberty-Compute Alert Clusters: Scatter plot of the alert clusters versus their inverse node frequency scores. The clusters are sorted based on their average inverse node frequency scores.

1. Calculate the mean time between activity μ_i for cluster c_i using Eqn. 2. μ_i represents the expected time between system activity of type c_i, if cluster c_i was periodic.

$$\mu_i = \frac{\sum_{j=1}^{m-1}(Q_i[j+1] - Q_i[j])}{m-1} \tag{2}$$

2. Calculate the standard deviation STD_i from μ_i of intervals between system activity of type c_i using Eqn. 3. We assume that STD_i values close to 0 indicate near-periodic activity.

$$STD_i = \sqrt{\frac{\sum_{j=1}^{m-1}[(Q_i[j+1] - Q_i[j]) - \mu_i]}{m-1}} \tag{3}$$

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

Let us 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. 8 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.

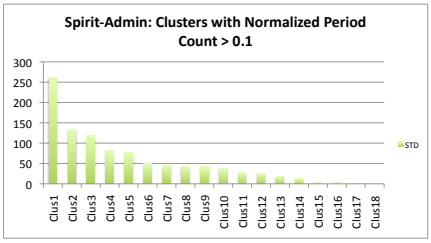


Figure 8: Spirit Admin Clusters: Bar chart of clusters with a $norm\_per\_cnt > 0.1$. Cluster 16 is the only cluster that is known to contain alerts.

Since the near-period property requires frequent occurrences, we only apply this rule 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 rule 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 can now summarize the procedure for identifying a cluster c_i as either an alert or a normal cluster using the rules above. For any cluster c_i, if either of the rules above is true, it is set as an alert cluster and all the nodehours in it are set as alert nodehours. If all of the rules are false then c_i is considered alert free along with all the nodehours in c_i. We note that the 2nd rule 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 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 an *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 be 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. 4 and 5, 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 [9]. 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.

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

7. RESULTS

For the similar node categories, we note that we were able to achieve 100% recall in three node categories, *BGL-Link*, *BGL-Other* and *Tbird-Other*, irrespective of the value of *med_per*. Also, depending on the value of the *med_per* parameter, we were able to achieve above 50% recall, with a single digit false positive rate for all node categories except the BGL-IO category, see Figs. 9a and 9b. With the *BGL-IO* node category only about 8% recall was achieved. The reason for this performance is that 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 of 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 (38.22%) of this node category 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 recall rate goes up to approximately 60%.

For the dissimilar node categories we were able to achieve 100% recall in two node categories, *Liberty-Admin* and *Tbird-Admin*, irrespective of the value of *med_per*. We were also able to achieve above 60% recall for all node categories depending on the value setting of the *med_per* parameter. What is however interesting here is that while we note that setting the *med_per* = 2 gives us single digit false positive rates for 4 out of 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. However, they are still better than what NodeInfo achieves on the same data sets.

A summary of the results of the evaluation is given in Fig. 10. 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 Nodeinfo. In choosing the best case, a balance between a high recall and low false positives was considered. The graph shows that the overall best case was achieved with the *BGL-Link* category with 100% recall at a false positive rate of 0.8%. The average recall (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 *Nodeinfo* 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 *Nodeinfo* against those achieved by the auto-tuned STAD results show a statistically significant difference. Statistical tests show that STAD achieves

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

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

similar results without the manual determination of k for Top_k analysis, as employed by NodeInfo. This is a significant achievement 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.

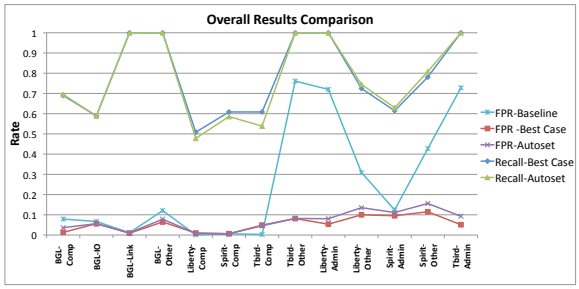


Figure 10: The best case results for the experiments where the *med_per* parameter is set manually and the results where its value is set automatically.

8. CONCLUSION AND FUTURE WORK

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

The FPRs achieved by our experiments are not uncommon with anomaly detection systems [4]. The FPRs achieved are sufficiently low to potentially support 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.

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 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 system 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 data mining based tool such as STAD would prove useful in the correlation of events across multiple tiers making log analysis efforts easier. The extension to alert detection to dissimilar nodes provides the possibility of using this framework for alert detection on distributed systems. Future work will involve the semi-supervised association of alerts to faults i.e. converting anomalies to fault signatures.

Acknowledgements

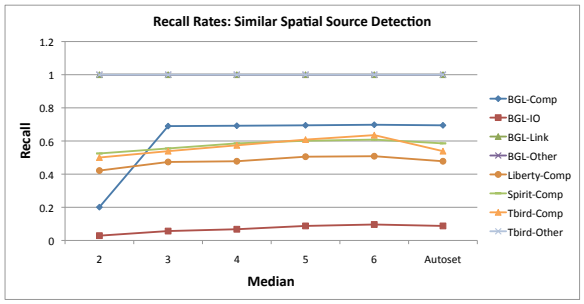
This research is supported by a Natural Science and Engineering Research Council of Canada (NSERC) Strategic Project Grant. This work is conducted as part of the Dalhousie NIMS Lab at http://www.cs.dal.ca/projectx/.

9. REFERENCES

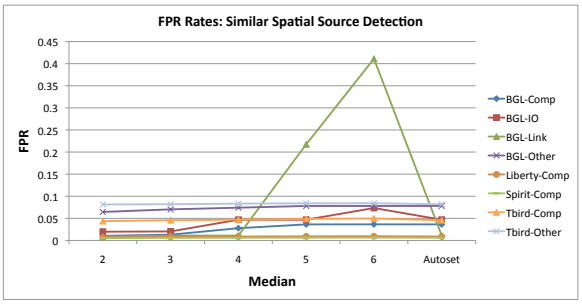
[1] M. Aharon, G. Barash, I. Cohen, and E. Mordechai. One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs. *Lecture Notes in Computer Science*, 5781/2009:227–243, 2009.

[2] 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.

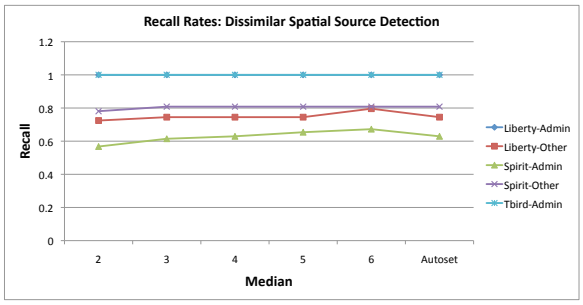
[3] L. Huang, X. Ke, K. Wong, and S. Mankovskii. Symptom-based problem determination using log data abstraction. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 313–326, New York, NY, USA, 2010. ACM.



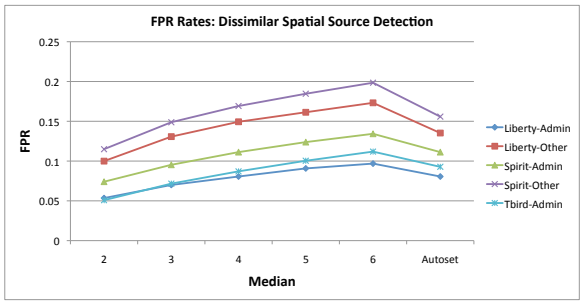
(a) Recall Rate - Similar Nodes



(b) False Positive Rate - Similar Nodes



(c) Recall Rate - Dissimilar Nodes



(d) False Positive Rate - Dissimilar Nodes

Figure 9: **This figure shows recall and false positive rates for evaluations of STAD on the datasets listed in Table 2.**

[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] M. I. Krzywinski, J. E. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 2009.

[6] 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.

[7] 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.

[8] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. System State Discovery via Information Content Clustering of System Logs. In *Proceedings of the 2011 International Conference on Availability, Reliability and Security, ARES 2011. Vienna, Austria.*, August 2011.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] USENIX. USENIX - The Computer Failure Data Repository. Published to the web. Last Accessed October 2011.

[16] 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.

[17] W. Xu. *Detecting Large Scale System Problems by Mining Console Logs*. PhD thesis, University of California, Berkeley, 2010.