

# Interactive Learning of Alert Signatures in High Performance Cluster System Logs

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

**Abstract**—The ability to automatically discover error conditions with little human input is a feature lacking in most modern computer systems and networks. However, with the ever increasing size and complexity of modern systems, such a feature will become a necessity in the not too distant future. Our work proposes a hybrid framework that allows High Performance Clusters (HPC) to detect error conditions in their logs. Through the use of anomaly detection, the system is able to detect portions of the log that are likely to contain errors (anomalies). Via visualization, human administrators can inspect these anomalies and assign labels to clusters that correlate with error conditions. The system can then learn a signature from the confirmed anomalies, which it uses to detect future occurrences of the error condition. Our evaluations show the system is able to generate simple and accurate signatures using very little data.

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

## I. INTRODUCTION

Modern computer systems and networks are increasing in size and complexity at an exponential rate. An observation of the progression towards the building of large scale data-centers to support cloud infrastructure and the drive to build bigger and larger High Performance Clusters (HPC) that can perform computations on the ever increasing amounts of data being collected, will confirm this assertion. For the first time all the top 10 HPC (supercomputers) in the world are capable of computing performance in the petaflop (quadrillion) range. The fastest supercomputer in the world, the K Computer located at the RIKEN Advanced Institute of Computational Science (AICS) in Japan is capable of 8 quadrillion calculations per second and has 68,544 nodes with 8 cores each, leading to a total of 548,352 cores [1]. It is fair to say that this trend is bound to continue.

The tasks required for the management of error and fault conditions on such systems can also be expected to reach a level of complexity where some degree of automation will be required to keep up with the pace. Self-healing in autonomic computing has set a goal for the engineering of systems that are capable of detecting and fixing their own error conditions. In this work, we propose a framework, which attempts to automatically detect error conditions (alerts) in HPC that

are detectable in system logs, one of the richest sources of information on most computer systems. As a framework that supports automatic detection of faults, our proposal falls within the ambit of self-healing systems in autonomic computing.

Our proposed framework is a hybrid system that combines anomaly detection, signature generation and visualization, while making use of a mix of supervised and unsupervised learning methods. It takes advantage of both signature and anomaly based approaches to alert detection by creating signatures from detected anomalies. The signatures are updated as new anomalies are detected. The visualization component of our system provides a window for Tier-1 operators, i.e. human administrators, to view results from the system and provide feedback. The system can in turn use this feedback to improve on its performance over time. Our framework is able to learn these alert signatures without the use of semantic analysis. The investigation of such an approach was one of our goals, as it increases the platform portability of the framework.

This framework differs from and improves on previous approaches due to its hybrid nature and its use of techniques that have low computational cost. Previous approaches have utilized either a signature-based [2] or anomaly-based [3] techniques but not both. Visualization techniques, if included, are not used interactively and are used only for the visualization of the results of analysis [4], [5].

We performed a number of experiments to test the detection capability of the alert signatures that can be learnt by the proposed system, using log data from 4 HPC machines i.e. *Blue-Genie/L*, *Liberty*, *Spirit* and *Thunderbird* [6]. These HPCs are each built by different manufacturers, i.e. IBM, Dell, Cray and HP, and are hosted at 2 different and reputable research laboratories, i.e. Sandia National Labs (SNL) and Lawrence Livermore National Labs (LLNL). This means that the data sources are varied, which adds to the credibility of our research findings. The alert states <sup>1</sup> in these log files have been pre-labelled by the administrators of the machines, giving ground truth for the evaluation. The results suggest that an overall detection rate of 88% at a false positive rate of 0% is achievable, even without the use of administrator feedback

<sup>1</sup>A detailed description of these alert states and how they were identified can be found in [7]

to improve the anomaly detection process. The results also suggest that for any error condition, an effective signature may be generated when only 10% of the exemplars that define an alert type in the log data are used in training. By using a hybrid anomaly detection and signature learning approach, our proposal provides a framework that has high accuracy and can adapt to an ever changing network environment.

The rest of this paper is organized as follows. Section 2 presents previous work. Section 3 introduces the proposed framework. Section 4 discusses its complexity and Section 5 details the methodology. Then, the results are presented in Section 6. Finally, conclusions are drawn and the future work is discussed in Section 7.

## II. BACKGROUND AND RELATED WORK

The goal of automatic alert detection is to create systems that are able, with limited input from a human operator, to identify their own error conditions/states. We differentiate here between errors as symptoms of a fault and the actual faults. Faults usually leave traces on systems before and after they occur. These traces manifest themselves in the form of errors (alerts) in the system. The goal here is the automatic identification of these error conditions that ultimately link to the actual fault conditions, thereby reducing troubleshooting time.

To accomplish this, the system would most likely utilize one of its many sources of continuous data streams. Examples of such data streams include metrics of system activity (memory, CPU, etc) and system logs. The work of Cohen et al. in [8] is a good example of an automatic alert detection mechanism that utilizes system metrics. In this work, the authors successfully demonstrate a method of using system metrics to define system states. Such observed states are clustered and indexed for similarity based retrieval to identify problems.

Our framework does not rely on system metrics but on system logs. To this end, Aharon et al. propose the PARIS (Principal Atom Recognition in Sets) algorithm [4]. This algorithm is able to detect *atoms*, i.e. sets of correlated message types, which are produced as part of a normal process or failure activity. Message types are textual templates, which can be used to semantically cluster the natural language descriptions found in log events. The authors then propose the monitoring of these atoms through visualization as a means of detecting failure in system logs. Xu et. al proposed a Principal Component Analysis (PCA) based framework for the detection of system problems through the analysis of console logs [5]. Fu et. al [9] propose alert detection by monitoring state transitions via Finite State Automata (FSA). Oliner et al. [10] propose a ranked based system that ranks log partitions using entropy-based information content. On the other hand Huang et. al. propose a system for the supervised automatic generation of classification rules for alert detection from logs. The supervised techniques that were evaluated include three associative classification techniques along with Naive Bayes classification and decision tree classification using C4.5. The framework we propose builds on previous work with the

Spatio-Temporal Alert Detection (STAD) framework for alert detection in system logs [11] and LogView [12], a method for visualizing system log clusters using Treemaps [13]. The STAD alert detection framework builds on previous work in entropy-based alert detection in system logs [3] and works by clustering system log partitions, which are decomposed spatio-temporally and separating the clusters into *normal* and *anomalous* categories [14]. STAD improves on previous entropy-based approaches by being able to detect anomalies without relying on a ranking and being able to work on log files even when the sources of events in the log are not similar. Evaluations comparing the accuracy of STAD to a previous entropy-based approach show a statistically significant difference in performance in favour of STAD [3].

This paper shows how STAD can be used for alert detection on production systems. By combining with a visualization technique, signatures of the detected anomalies can be generated and used on a production system to detect future alerts as they occur. In contrast to [4], [5], which use visualization simply for allowing the user to inspect the output, we propose that visualization can be used for interactive learning i.e. the human administrator uses the visualization to give the system feedback, which is then used by the system to improve on its models over time, as has been done in other domains [15].

It is important to note that the idea of using clustering for anomaly detection and learning signatures from labelled anomaly clusters has also been applied in the domain of intrusion detection. In [16], the authors summarise and separate the malicious activity from normal activity in a connection log by clustering the contents of the log using the Simple Log file Clustering algorithm (SLCT) [17] and the learning of attack signatures from the identified clusters based also on the iterative application of SLCT to a labelled training set of identified anomalous clusters. The authors argue that this approach can shorten the time required for the creation of attack signatures and also provides a framework that can adapt to novel attacks and ever changing attack behaviour.

## III. METHODOLOGY

The main idea behind our proposed framework is to use the content of historical logs to learn the signatures of previous alert conditions that may have manifested in the log, with the proviso that this be done with minimal human intervention. Once these signatures are learnt, they can be stored and applied online on production systems to detect future incidents of such alerts as they occur. Our proposed system utilizes anomaly detection, signature generation and visualization to achieve this goal. The proposed framework is therefore a hybrid signature and anomaly based system.

Signature based systems work by utilizing models of known and well defined behaviors of interest. They are usually very accurate but suffer from their inability to detect new behaviors of interest. Anomaly based systems, on the other hand, require less apriori knowledge of the behaviors the system aims to detect. They instead attempt to identify behaviors that differ from the *norm* and are therefore able to detect new behaviors

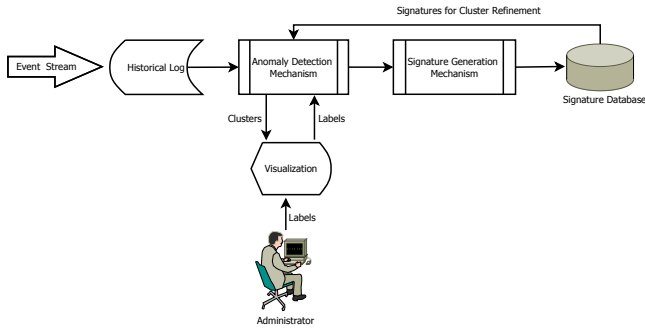


Fig. 1. Framework for Automatic Signature Generation from System Logs

of interest. However, they tend to be less accurate as legitimate behavior can sometimes differ from the *norm*. By utilizing concepts from both signature and anomaly based systems, our framework leverages on the strengths of both systems, by both being capable of detecting new types of alerts and being as accurate as a signature based system.

Due to the complexity of the problem, previous work asserts that in such situations human involvement is important to complement the automated system [15]. Thus, the visualization component provides the interactive component of the framework. It allows the human administrator to interact with the proposed system and provide feedback that will over time improve the models learnt by the system and keep them up-to-date. The diagram in Fig. 1 provides an overview of the phases of our proposed alert detection framework. These are:

- 1) Unsupervised anomaly detection through the clustering of system logs and the separation of normal and anomalous log clusters.
- 2) Feedback from the human operator in the form of identification and labeling of anomalous clusters.
- 3) Using the Feedback from the administrator is used for learning alert signatures and for cluster refinement.
- 4) Repeat Steps 1 - 3 as required.

For the anomaly detection component of the system we use Spatio-Temporal Alert Detection (STAD) [11]. STAD is described in more detail in Section III-A. It works by clustering the spatio-temporal partitions of the log and separating the clusters into normal and anomalous categories. These categories would contain several different types of *normal* and *anomalous* behavior, hence the clusters would still require labels. These labels can be provided by the human administrator through a visualization system such as LogView [12], albeit with a modified hierarchical structure that would include spatio-temporal partitions and the clusters identified by the system.

Once these labels are acquired, the system can then take all alert clusters with the same label and generate a signature for them. We use a frequent itemset mining paradigm for signature generation, described in more detail in Section III-B.

The framework does not require the use of any specific mechanisms for carrying out its phases. The choice is left to the discretion of the user.

## A. Anomaly Detection

We use STAD as the anomaly detection mechanism. Evaluation of STAD shows that it is able, on average, to identify 78% of all alerts while maintaining a false positive rate of 5.4% [11]. These results represent a statistically significant improvement over the use of an entropy-based alert detection scheme as a baseline. These evaluations were performed using a set of logs from 4 HPC machines, which are available from the USENIX Computer Failure repository[6]. The alerts used as ground truth for the evaluations were labelled by the administrators of the machines. The statistics of these logs are listed in Table I. These logs are also used for the evaluations performed in this paper. The steps of the STAD mechanism are summarized below.

- **Message Type Transformation:** This involves the transformation of the natural language description of each line in the log into a unique token that represents the message type that produced the description.
- **Spatio-Temporal Decomposition:** Spatio-Temporal decomposition involves the partitioning of the contents of the log, so that each partition contains all events from a single source over a specified time period. In our work, we use the nodehour, which is one hour of log information from a single node[3].
- **Clustering:** This step requires the grouping of the nodehours, so that each group contains nodehours that are very similar to each other, while being very dissimilar from nodehours in other groups. The exact method used to achieve this can be freely chosen by the user. In our work, we utilize information content clustering (ICC) [14], which clusters the nodehours using the entropy based information content of the nodehours. The ICC technique helps to greatly simplify a task that would be very difficult using a traditional distance based clustering technique. Clustering of nodehours, in the most direct way, would use the individual terms (words) that appear in each nodehour. As the number of individual terms in the log can easily number in the millions, this leads to the curse of dimensionality. Information content scores are derived from the terms that occur in the nodehour through several abstraction steps [14] and provide a linear time approach to clustering nodehours. Clusters derived using ICC of the nodehours from the *BGL-Link* log are visualized using the Circos visualization technique [18] in Fig. 2. The *BGL-Link* log derived from the *BGL* log listed in Table I, by listing only events produced by nodes in *Link* category. 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 conditions identified by system administrators in the log data, i.e. *LINKBLL*, *LINKDISC*, *LINKIAP*, *LINKPAP*, *MONPOW*.
- **Anomaly Detection:** We assume that the resulting clusters from the previous step either contain a majority of normal activity or a majority of anomalous (alert)

TABLE I  
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

activity. The anomaly detection step serves to separate such clusters. Again, the exact method used is left to the discretion of the user. We propose the use of a heuristic approach that differentiates the clusters based on the number of time periods reported in the cluster, how localized the activity in the cluster is and the periodicity of activity in the cluster [11].

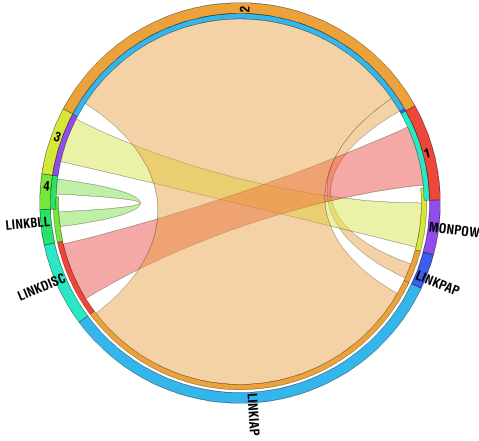


Fig. 2. **BGL-Link Category:** Circos-Table Plot showing the mapping between the administrator defined alert categories and the clusters produced using the information content clustering technique.

### B. Signature Generation

The input to the signature generation mechanism, see Fig. 3, consists of the nodehours that belong to the cluster(s) that have been identified by the human administrator as containing activity that relates to a particular alert type. This action would only require the administrator going over the clusters identified by the system as being anomalous. A cluster pruning step, as described in Algorithm 1, is then performed. During this step, the set of message types reported in each nodehour is pruned by iteratively computing the difference between the set of message types in the nodehour and the cluster centroids for each of the clusters identified as *normal* by STAD. Our method for choosing cluster centroids can be found in [14].

After pruning, frequent itemset mining is performed on the nodehours. Given a set of objects or items (called an item base),  $S$ , we define  $\mathbf{T}$  as a set of transactions defined over  $S$  such that for all  $T \in \mathbf{T}$ ,  $T \subseteq S$ . A subset (also called an itemset)  $S'$  of  $S$  is said to be *frequent* if the number of transactions in  $\mathbf{T}$  that are supersets of  $S'$  exceed a user-specified support threshold and is referred to as a *frequent itemset*. The goal of frequent itemset mining is to find all itemsets that occur in  $\mathbf{T}$  with a minimum support threshold.

**Algorithm 1** The cluster pruning pseudo code.

**Input:** The nodehours that need to be pruned  $A$ . The cluster centroids for all the nodehour clusters identified as *normal* by STAD,  $B$ .

**Output:** The pruned nodehours

```

1: for each  $A$  as  $A$  do
2:    $\{A$  will contain a set of message types. $\}$ 
3:   for each  $B$  as  $B$  do
4:      $\{B$  will also contain a set of message types. $\}$ 
5:      $A = A \setminus B$   $\{$ Compute set difference $\}$ 
6:   end for
7: end for

```

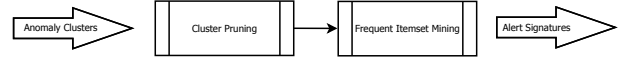


Fig. 3. Signature Creation Mechanism: This figure shows the steps in the signature creation phase of the framework.

The apriori algorithm is a classical algorithm for frequent itemset and association rule mining [19] and can be used here.

We apply the frequent itemset mining paradigm to the problem of alert signature generation by supposing that message types are items and a nodehour is a transaction. Hence the set of message types found in a system log is the item base and the transaction database is the set of nodehours found in the log. We theorize that for any set of related transactions (nodehour cluster) in the transaction database (system log), the set of frequent itemsets that occur in the transaction cluster would be an effective signature for identifying future occurrences of that transaction type. If a transaction cluster is related to an alert type, then the frequent itemsets mined from such a cluster would form a signature for that alert. Given a nodehour cluster  $C$  that contains some sort of alert behavior, the set of frequent itemsets derived from  $C$  i.e.  $C^f$  will contain the signature for that alert behavior. The alert signature defined by  $C^f$  is triggered, if a message type that is contained in any of the itemsets in  $C^f$  is found in a nodehour.

Once the signatures are generated, they can be stored in a database and used to detect future alerts on a production system, Fig. 4. This database will of course be updated as more up-to-date signatures are discovered. The log parsing module would parse the log event stream, searching for any events (or log partitions) that match any of the signatures in the database. If a match is found, an alarm is raised. The alarms can then be passed directly to the administrator for action. Thus, we have a system that can automatically search its own logs for symptoms of failure and notify the administrator. Unlike other systems that do this using signatures that are produced and managed manually, our proposed system produces the signatures itself, while only relying on the administrator for class labels.

### C. Visualization, Feedback and Cluster Refinement

The anomaly detection capability of STAD depends on its ability to properly cluster spatio-temporal partitions of a log file. The feedback received by the anomaly detection mechanism from the visualization system and the signature generation system can be used to refine the results of clustering

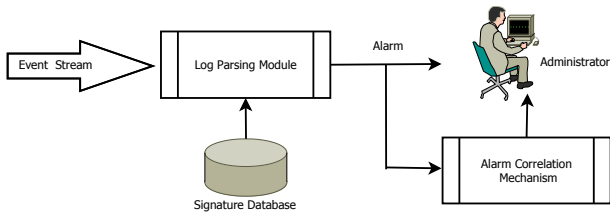


Fig. 4. Online Alert Detection: This shows how the alert signatures produced by our system can be used for online alert detection on a production system for more accuracy. In this case, the feedback from the visualization will be provided by human administrators in the form of labels for identified clusters and the category for a label, i.e. normal or anomalous. While feedback from the signature generation system will be the generated alert signatures.

Nodehour decomposition is done arbitrarily: , it aims to ensure that the resulting spatio-temporal partitions contain correlated messages types that define a single system state. However, a nodehour decomposition may separate correlated message types or may combine correlated message types from different states. As an example of how the system can use feedback for cluster refinement, consider the case where a nodehour combines correlated message types from two different *normal* states, which we will call *A* and *B*. Such a nodehour will not be clustered into either of the normal clusters and will likely be classified as *anomalous*. The information that anomaly detection mechanism receives via the feedback loop can be used to resolve this scenario.

If one of the normal states present in the nodehour, i.e. *A*, had previously been confirmed through feedback by the administrator, then the system would have stored this feedback by saving the centroid for that cluster in its database. The system can thus search that nodehour for message type(s) that define the centroid of *A*. If, as we will assume in this example, the system finds the message types in the nodehour, an *IF* clustering scenario can then be created by re-clustering the nodehour without considering the message types that are part of the cluster centroid of *A*. This action will show that the cluster also belongs to *B*. The system therefore now knows that the nodehour is actually a *soft* cluster that belongs partially to *A* and *B*. It can thus eliminate the cluster that the nodehour belongs to as an anomalous cluster and in effect can reduce its FPs. This is just one example of how feedback can be used for cluster refinement, other scenarios exist.

#### IV. COMPLEXITY

For our proposed framework to be usable in an online environment, its time complexity must allow quick processing of the content of very large logs, which is the trend in today's network systems.

At the anomaly detection phase, the significant operations are message type transformation and clustering. The message type transformation is a linear time operation, with respect to the number of events in the logs. However, this is a preprocessing step that can be performed in advance. The ICC requires the calculation of a number of entropy-based values. Let  $W$  represent the number of terms in the log,  $C$

the number of reporting sources and  $H$  the number of spatio-temporal partitions. ICC can be performed after calculations are performed on two matrices of size  $W \times C$  and  $W \times H$ . The time complexity is therefore of the order of  $W \times C$  and  $W \times H$ . However, previous work has shown that the use of message types to represent the terms in the log, i.e  $W$ , significantly reduces the dimensionality of the problem by an order of about 100 [3]. The actual clustering step that comes after these calculations is a linear time computation with respect to  $H$ .

Let  $n$  represent the number of items in an item base. The time and space complexity for mining the set of frequent itemsets from a transaction database defined over the item base is theoretically of the order of  $\sum_{k=1}^n \binom{n}{k}$  i.e. the number of possible itemsets that can be generated from the items in the item base. However, in practice, the size of the largest candidate itemset is more likely bounded by the size the largest transaction in the transaction database, which is usually a lot less than the number of items in the itembase. The use of the classical apriori algorithm helps to reduce this complexity further by reducing the number of candidate itemsets that need to be generated. There are also several published approaches using techniques like sampling, partitioning and transaction reduction, that can improve the efficiency of the process even further[20].

On the basis of this discussion, we argue that the methods used all scale gracefully with the size of the logs.

#### V. EXPERIMENTS

Our experiments involved simulating the alert generation mechanism of our framework using historical log data and then using the signatures to detect alerts in nodehours as they occur. The goal is to measure the detection accuracy of the generated signatures. We make the following assumptions:

- The signatures are static, i.e. once they are created they do not change.
- The anomaly detection portion of the framework works perfectly, i.e. it is able to perfectly separate *normal* and *anomalous* clusters.
- The administrator is able to accurately label all alert clusters shown to him/her by the system. As these labels are the basis on which the system decides what alert clusters to mine signatures form is based, the accuracy of this labeling will have an impact on the result.

The visualization component of the system allows the human operator to interact with the system and provide labels for signature generation and cluster refinement. Since in these experiments we do not perform cluster refinement and assume that all labels have been provided accurately, we do not evaluate the visualization phase in this paper.

Each experiment was done for only a single alert type using 13 datasets derived from the HPC logs mentioned earlier in Table I. Splitting the logs into these datasets was performed to ensure that the anomaly detection mechanism can effectively produce clusters that can be separated into *anomalous* and *normal* categories. Statistics about these datasets are provided in Table II. In the table *# Events* refers to the number of lines

in the log, *# Nodes* refers to the number of nodes reporting events into the log, *# Nodehours* refers to the number of spatio-temporal partitions derived from the dataset, *% Alerts* refers to percentage of nodehour that contain alert signatures, *# Alert Types* refers to number of alert types identified by domain experts in the log, while *Similar Nodes* refers to anomaly detection methodology used for the dataset. A *Y* in the *Similar Nodes* column indicates that anomaly detection was carried out under the assumption that the nodes in this dataset were sufficiently similar, while a *N* indicates that the nodes were assumed to be dissimilar.

We split each log file into five separate training and testing pairs for each alert type, see Table II. These training and testing pairs were not of equal size. The split point for the training file was determined by the time at which 10%, 20%, 30%, 40% and 50% of all nodehours of the alert type had occurred. All log events occurring after each of these points were used to test the signatures found. Each run involved the execution of our proposed framework on a test file, which is assumed to be a historical log, to learn the signature for an alert type. The generated signature(s) are then applied to the test set to detect nodehours that contain the alert type. We can then determine the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), which can then be used to calculate the detection rate (DR) and false positive rate (FPR) as defined in Eq. 1 and Eq. 2 respectively. This setup implies that a total of 980 experiment runs were carried out.

$$DR = \frac{TP}{TP + FN} \quad (1)$$

$$FPR = \frac{FP}{FP + TN} \quad (2)$$

We utilized an efficient open-source implementation of the apriori algorithm<sup>2</sup> in all our runs. The support threshold was set to 50% for all runs. It is not uncommon for frequent itemset mining to generate a large number of itemsets, which are sometimes redundant. For this reason we also generate only *closed* frequent itemsets. Frequent itemsets that have no superset with the same support value are referred to as *closed*.

## VI. RESULTS

We present results on the detection accuracy of the signatures and the nature of the signatures produced by the system.

The FPRs achieved during the experiments showed very little variation. The FPR achieved was approximately 0% for all runs. These FPRs are operationally acceptable and supports the assertion that our goal of converting detected anomalies into signatures as a means of reducing false positives was achieved.

An average DR of 88% was achieved for all runs. However, we state that a DR close to 100% can be expected for most of the signatures produced. Fig. 5 shows the DR distribution for the runs for each log file. It can be seen that the average DR

would be much higher if the few *outliers* are not considered. Our investigations show that some of the outlier DR values were due to the quality of clusters produced or due to the distribution of signature(s) across time. In cases where the alert state was not the majority behavior in a cluster, the signature learnt would not relate to the alert and hence would produce a DR close to zero. This situation can however be mitigated by reducing the support threshold used in signature generation. In cases where signature(s) for an alert were not evenly distributed in time, it was possible to only partially learn the signature(s) for an alert. Leading to less than perfect detection, this situation is particularly pronounced in datasets 4, 9, 11, 13 in Fig. 5. In a real life implementation, the signatures would not be static, as is the case in our experiments.

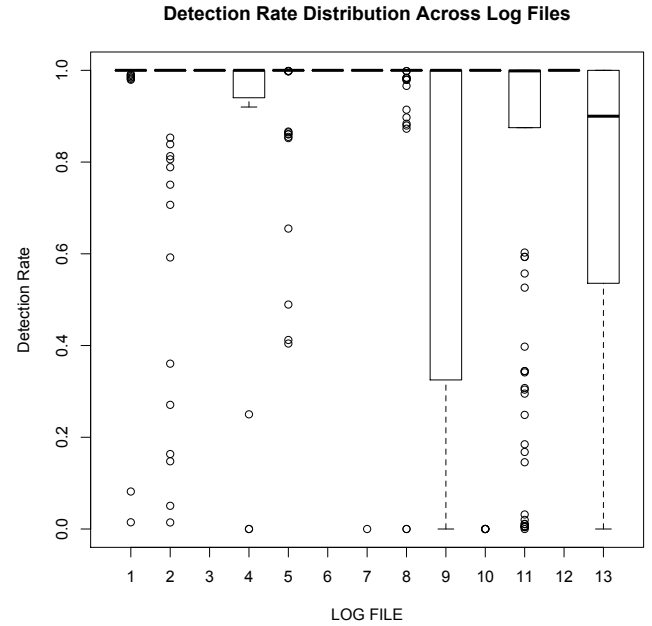


Fig. 5. Detection rates for the different HPC logs. Log files are numbered in the same order as shown in Table II

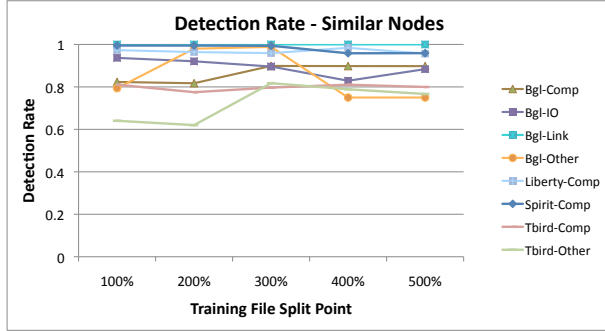
The graphs in Fig. 6 show how the average DR changes as we change the size of the training file. The entropy-based calculations used for clustering the nodehours differed based on whether the nodes were considered *Similar* or *Dissimilar*. The results show that while increase in DR performance was noticed for 3 log files i.e. *Bgl-Comp*, *Tbird-Other* and *Liberty-Other*, as the size of the training file was increased, there seemed to be no correlation between the size of the training file and the DR performance. This is confirmed by an Analysis of Variance (ANOVA) test performed at 5% significance, which is summarized in Table III. The ANOVA results show no statistically significant difference between the FPR and DRs achieved when the training file size is changed. These results suggest that alert signatures can be learnt when as little as 10% of the exemplars for an alert type are present in the log.

We discuss the nature of the signatures produced using three factors; their length, their support value and the number of signatures generated for each alert type. An alert signature, as

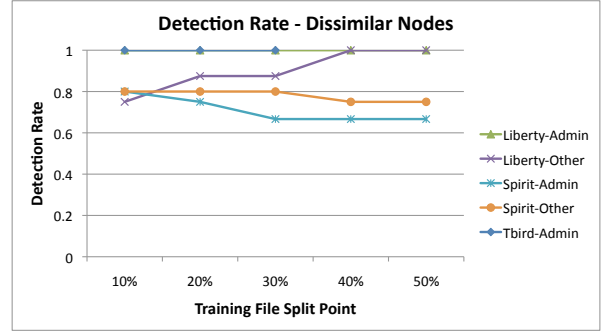
<sup>2</sup>Downloadable from <http://www.borgelt.net/apriori.html>

TABLE II  
SYSTEM LOG DATA FUNCTIONAL GROUPING STATISTICS

	# Events	# Nodes	# Nodehours	% Alerts	# Alert Types	Similar Nodes
<b>1: BGL-Compute</b>	4,153,009	65,554	1,581,845	4.2	18	Y
<b>2: BGL-IO</b>	400,923	1,024	219,722	38.22	17	Y
<b>3: BGL-Link</b>	2,935	517	1,395	2.37	5	Y
<b>4: BGL-Other</b>	191,096	2,167	13,666	0.43	7	Y
<b>5: Liberty-Compute</b>	200,940,735	236	1,748,865	0.29	17	Y
<b>6: Liberty-Admin</b>	52,211,676	2	27,162	0.04	3	N
<b>7: Liberty-Other</b>	12,416,820	6	44,447	0.22	7	N
<b>8: Spirit-Compute</b>	218,697,851	512	6,648,719	0.19	29	Y
<b>9: Spirit-Admin</b>	41,847,257	2	26,216	3.10	3	N
<b>10: Spirit-Other</b>	11,753,861	7	57,532	0.25	11	N
<b>11: Tbird-Compute</b>	155,403,254	4,514	14,520,204	0.17	31	Y
<b>12: Tbird-Admin</b>	15,306,749	20	100,740	0.02	7	N
<b>13: Tbird-Other</b>	21,392,379	1,319	626,030	0.02	10	Y



(a) Similar Nodes



(b) Dissimilar Nodes

Fig. 6. Detection rates for the different HPC logs, showing variation in performance for different sizes of training files

TABLE III  
ANOVA TEST SUMMARY

Treatment	F	P-Value	F crit
FPR	0.874	0.485	2.525
DR	0.162	0.957	2.531

defined by this work, consists of a set of message types, which disjunctively or conjunctively can be used to detect an alert condition in a log file partition. The *length* of a signature is therefore the number of message types that define it. We are of the opinion that shorter signatures are not only simpler but better defined i.e. compact. So we deem shorter signatures to be preferable than longer ones. The graph in Fig. 7 (b) shows the length distribution of the signatures for each of the HPC machines. The results show an overall median length of 2 and mean length of 6. Since minimum signature length is 1, these results show that the signatures produced by our method are relatively simple and compact.

We generate our signatures as frequent itemsets from node-hour clusters. Therefore each signature has a support rate and since we can have more than one frequent itemset, it is possible to have more than one signature for an alert type. While it can be debated, it is safe to say that signatures with higher support rates are likely to be better detectors than those with lower support rates. We therefore report on the support of the signatures produced during our experiments. The graph in Fig. 7 (a) shows the support distribution for the signatures. The results show an overall median support rate of 100% and mean support rate of 94.45%. These support rates are very high and attest to the quality of the signatures produced. The results are also not surprising given the high internal cohesion rates

TABLE IV  
NUMBER OF SIGNATURES PER ALERT TYPE

	# Alert Types	Median	Max
<b>BGL-Compute</b>	18	1.0	6.0
<b>BGL-IO</b>	17	1.0	2.0
<b>BGL-Link</b>	5	1.0	2.0
<b>BGL-Other</b>	7	1.0	2.0
<b>Liberty-Compute</b>	17	2.0	5.0
<b>Liberty-Admin</b>	3	2.0	4.0
<b>Liberty-Other</b>	7	1.0	9.0
<b>Spirit-Compute</b>	29	2.0	33.0
<b>Spirit-Admin</b>	3	1.0	2.0
<b>Spirit-Other</b>	11	1.0	3.0
<b>Tbird-Compute</b>	31	1.0	36.0
<b>Tbird-Admin</b>	7	1.0	5.0
<b>Tbird-Other</b>	10	1.0	2.0

reported from our assessment of the clusters produced using ICC [14]. Generating a few (ideally one) signature for each alert type is also desirable. The data in Table IV reports on the median and maximum number of signatures created for each alert type. The median number of signatures was the minimum (1) for all the log files except *Liberty-Compute*, *Liberty-Admin* and *Spirit-Comp* where the median was 2.

## VII. CONCLUSION AND FUTURE WORK

We propose a hybrid framework for alert detection in system logs in this work. The proposed framework combines the advantages of anomaly-based and signature-based detection. By also including an interactive visualization, the framework hopes to provide a window by which human administrators can provide feedback to the system, which it uses to generate signatures and improve on its anomaly detection capability over time.

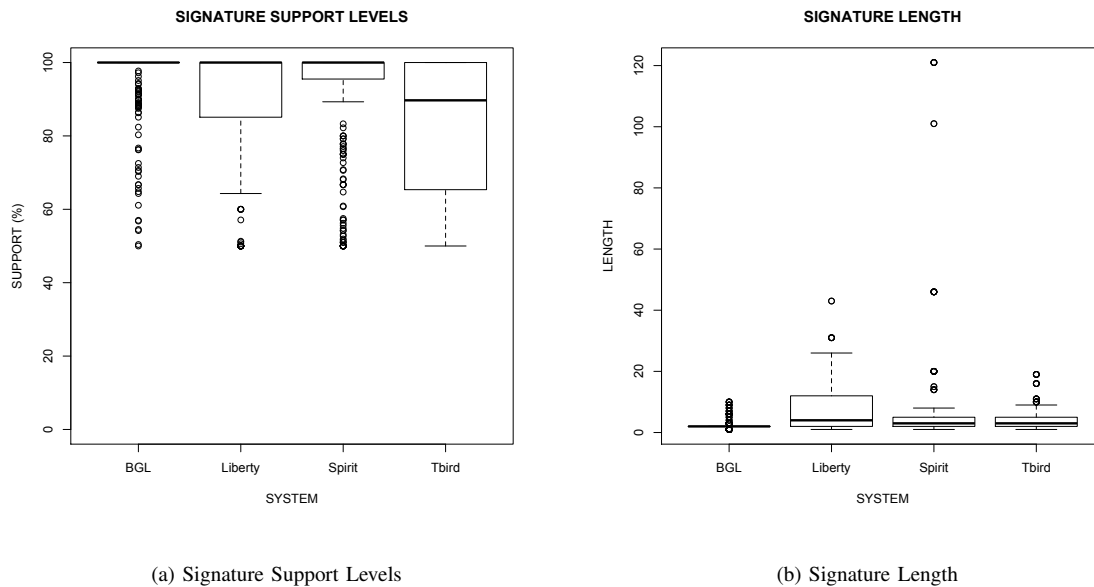


Fig. 7. Boxplots showing distributions for (a) Signature support levels (b) Signature length i.e. number of message types that define a signature.

Our evaluations suggest that effective signatures can be learnt with minimal amounts of data. The signatures learnt are also not overly complex and since they are composed of sets of message types, they are human readable. The signatures were also found to be relatively accurate, able to achieve a DR of 88% on average while maintaining an operational acceptable FPR of approximately 0%. Achieving high DRs with no FPs is dependent on accurate message type extraction and clustering. The signatures created by our system can be created in real-time and can be brought online immediately.

The framework also provides a lot of flexibility in the choice of techniques used during each of its phases. None of the techniques used in our evaluation is tied to the framework. They can all be selected at the discretion of the user. While we use nodehours during our evaluations, it is possible to use spatio-temporal partitions with lower levels of granularity e.g. nodeminutes. This is especially useful for real-time alert detection, when quick discovery is important. Signatures once learnt can be applied to a spatio-temporal partition at any level of granularity desired.

Future work will involve testing of the framework on logs collected on non-HPC systems. A user study involving a prototype of the framework would also be useful. Such a study would provide valuable design choices that would improve the visualization component of the framework and also show how well the system is able to improve itself over time.

#### ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their in-depth comments, which have helped to improve this manuscript. 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] TOP500.Org, "Top500 supercomputing sites," Published to the web. Last Accessed, October 2011. [Online]. Available: <http://www.top500.org/>
- [2] 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*, ser. CASCON '10. New York, NY, USA: ACM, 2010, pp. 313–326. [Online]. Available: <http://doi.acm.org/10.1145/1923947.1923979>
- [3] 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)*, ser. DSNW '10. Washington, DC, USA: IEEE Computer Society, June 2010, pp. 52–58. [Online]. Available: <http://dx.doi.org/10.1109/DSNW.2010.5542621>
- [4] M. Aharon, G. Barash, I. Cohen, and E. Mordechai, "One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs," *Lecture Notes in Computer Science*, vol. 5781/2009, pp. 227–243, 2009.
- [5] W. Xu, "Detecting Large Scale System Problems by Mining Console Logs," Ph.D. dissertation, University of California, Berkeley, 2010.
- [6] USENIX, "USENIX - The Computer Failure Data Repository," Published to the web. Last Accessed October 2011. [Online]. Available: <http://cfd.usenix.org/data.html>
- [7] A. Oliner and J. Stearley, "What Supercomputers say: A Study of Five System Logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007 (DSN '07)*, June 2007, pp. 575–584.
- [8] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *Proceedings of the twentieth ACM symposium on Operating systems principles*, ser. SOSP '05. New York, NY, USA: ACM, 2005, pp. 105–118. [Online]. Available: <http://doi.acm.org/10.1145/1095810.1095821>
- [9] 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*, December 2009, pp. 149–158.
- [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. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 959–964.
- [11] A. Makanju, A. Zincir-Heywood, and E. Milios, "Spatio-Temporal Decomposition, Clustering and Identification for Alert Detection in



- System Logs,” in *Proceedings of the 27th ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, March 2012.
- [12] A. Makanju, S. Brooks, N. Zincir-Heywood, and E. E. Milios, “Logview: Visualizing Event Log Clusters,” in *Proceedings of Sixth Annual Conference on Privacy, Security and Trust (PST)*, October 2008, pp. 99 – 108.
- [13] B. Schneiderman, “Tree Visualization with Tree-Maps: A 2-D space filling approach,” in *ACM Transactions on Graphics.*, vol. 2, no. 1, 1992, pp. 92–99.
- [14] 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.
- [15] S. Amershi, B. Lee, A. Kapoor, R. Mahajan, and B. Christian, “Human-Guided Machine Learning for Fast and Accurate Network Alarm Triage,” in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011, Barcelona, Spain)*, July 2011, pp. 2564–2569.
- [16] G. R. Hendry and S. J. Yang, “Intrusion Signature Creation via Clustering Anomalies,” in *SPIE Conference on Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008*, vol. 6973, 2008, pp. 69730C–69730C12.
- [17] R. Vaarandi, “A Data Clustering Algorithm for Mining Patterns from Event Logs,” in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, 2003, pp. 119–126.
- [18] 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. [Online]. Available: <http://genome.cshlp.org/content/early/2009/06/15/gr.092759.109.abstract>
- [19] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 12–15 1994, pp. 487–499.
- [20] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.