An Evaluation of Entropy Based Approaches to Alert Detection in High Performance Cluster Logs

Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada. B3H 1W5.
+1-902-494-3157
{makanju, zincir, eem}@cs.dal.ca

***Abstract*—Manual alert detection on modern high performance clusters (HPC) is cumbersome given their increasing complexity and size of their logs. The ability to automatically detect such alerts quickly and accurately with little or no human intervention is therefore desirable. The entropy-based approach of the Nodeinfo framework, which is in production use at Sandia National Laboratories, is one approach to automatic alert detection in HPC logs. In this work, we perform a comparative evaluation of three entropy based techniques, which are modifications to Nodeinfo. We evaluate these systems using three performance metrics, namely (i) Computational cost, (ii) detection accuracy, and (iii) false positive rate. Our results show that there is still room for improvement in entropy based approaches to the task of alert detection. We also show experimentally that it is possible to detect 100% of all alerts while maintaining an effective false positive rate of 0% using an entropy based approach. Our work suggests that entropy based approaches are viable for automatic alert detection in HPC and can improve the dependability of such systems if applied.**[1]

***Index Terms*—Algorithms; Modeling and Assessment; Network Operations Management Systems.**

I. Introduction

High Performance Clusters (HPCs) are known to be error prone, which is due in part to the complex nature of their architecture. The possibility of error is increased when many interdependent sub-systems and nodes need to co-operate and communicate to achieve the seamless synergy of a single system without any collisions. The ability to avoid or recover quickly from such errors or failures is therefore of interest in research that deals with the dependability, reliability and availability of such systems. Event logs are a feature of most modern computer systems and HPCs are no different. Event logs consist of lines (or events) that contain messages about the state of the system at a point in time. This fact makes them a rich source of information, but they unfortunately still remain an enigma to both manual and automatic analysis, due to size, complexity and unstructured content.

Not all events in an event log are symptomatic of a failure. We refer to events that are symptomatic of failure or require the attention of administrators as *alerts*. The task of *alert detection* in event logs can be defined as the task of finding those events, which are alerts, or regions in an event log that contain alerts [1]. Several approaches have been proposed for the task of alert detection in event logs, the most common being based on the semantics of the *terms* and the time properties of the messages. However, these have their limitations. *Nodeinfo* [1], [2] is an alert detection framework that uses an entropy based approach for alert detection. To the best of our knowledge *Nodeinfo* is the first log alert detection framework to utilize this approach. Nodeinfo takes into consideration the position of tokens within messages to provide structure for analyzing unstructured messages. It works on the assumption that similar computers correctly executing programs should have similar logs [1]. This implies that outliers can be detected by utilizing information-theoretic quantities based on the entropy of the terms in the event log. Nodeinfo has been shown to work with an acceptable false positive rate (FPR) of 0.05% [1], showing that an entropy based approach is feasible. However, we believe that there is still room for improvement with the framework as proposed, especially in regard to computational cost. This is the motivation for our proposed modifications and evaluations of these entropy based approaches.

In our work, we propose modifications to the Nodeinfo framework and evaluate the modified versions against the original Nodeinfo framework. In order to perform these evaluations, we have employed data from BlueGene/L (BGL), one of the fastest supercomputers in the world. [3], [4]. Our modifications add more context to the unstructured messages by introducing *Message Types*, which goes beyond that provided by simple word and position pairs utilized in Nodeinfo. Message types are semantic groupings of event messages that can be described using a textual template consisting of constant tokens and variable tokens. Knowledge of these message types is useful in imposing structure on the unstructured content of event logs, and can be useful for further automatic analysis. Unfortunately these message types are not always known *a priori*. Recent work using IPLoM (Iterative Partitioning Log

[1]This is a "Methodological and Technical" paper.

Mining) [5], has demonstrated an effective way of finding these message types automatically. Thus, IPLoM is utilized in this work to automatically find message types in the BGL data. These message types extracted by IPLoM are then used to transform the BGL data before input to Nodeinfo. Once message types are extracted, *Message Type Transformation (MTT)* techniques use the structure gleaned from the message types to transform the unstructured messages in the event logs before presenting to the framework. To the best of our knowledge, this paper presents the first work to modify the Nodeinfo framework and evaluate the modifications against the baseline framework.

Our intention with this evaluation of different approaches to entropy based alert detection in HPCs is to provide a reproducible benchmark on which other entropy based approaches can be evaluated and also shed more light on the advantages and disadvantages of the approach. In the following, we discuss concepts important to understanding our work and previous work in Section 2. Section 3 discusses the methodology of the experiments we carried out for our evaluations, whereas the results of those evaluations are discussed in Section 4. Finally, conclusions are drawn and future work is discussed in Section 5.

II. Background and Previous Work

A. Definitions

An event log E can be defined as a temporal sequence of lines of text reporting occurrences in a computer system. An example log file is shown in Figure 1. In the example, each line represents a separate event that occurred in the system that produced the log in question. Thus, an event log E is a record of a sequence of events e_1 to e_N, with N being the number of events contained in E. Each event e_i in E can further be decomposed into several fields, the exact nature and order of which would generally differ from event log to event log, but will generally consist of a timestamp (t_i), reporting computer or node (c_i), severity information (s_i) and a free form message (m_i). An example event message can be seen in Figure 2. The event consists of two broad fields i.e. the *"Header"* and the *"Message"*. The *"Header"* field consists of fields that to a large extent are sufficiently structured.

Each message (m_i) in the *"Message"* field of an event can be sub-divided into tokens, t_1 to t_p , where p is the number of tokens in the message, using a delimiter, which is usually whitespace. It should be noted that p varies from line to line. The message fields of events are generally unstructured. This is due to the fact that they are free form messages. This fact makes it difficult for the message field of events to be used in building models based on the content of event logs. Message type extraction or message type clustering is a way of building structured context into the unstructured message fields of events. Its goal is to find a set of textual templates, defined by constant tokens and variable tokens (wildcards), that abstract all the messages in an event log. Each message can be produced by one and only one template.

Not all events in an event log are indicative of a failure or require the consideration of an administrator. Events that do are what we refer to as *alerts*. The event field that is most likely to indicate if an event is an *alert* or not is the *severity* field. However, previous work has shown that reliance on the severity field is not reliable [3] due to ambiguous usage of terms by system programmers. This implies a need for more advanced techniques for *alert detection*. The task of *alert detection* can therefore be defined as the task of identifying actionable events in an event log or identifying portions of an event log, where these actionable events are likely to exist [2].

B. Alert (Anomaly) Detection in Event Logs

A review of the literature shows that there are several previous attempts at automating the task of alert (anomaly) detection in event logs. They vary from simple approaches that search event logs for message patterns that are indicative of previously known failure conditions [6], to visualization techniques that aid the quick detection of anomalies manually [7], to more complex schemes that use computational techniques like time periodicity of messages [8] or term weighting schemes [9].

A more recent computational approach to anomaly detection is Nodeinfo. Nodeinfo proceeds from the work of Liao [10] and Reuning [9] by using the more complex *"log.entropy"* term weighting scheme. Nodeinfo raises the bar of alert detection by achieving an operationally acceptable FPR of 0.05% at a Recall rate of 50% [1].

Though it utilizes the concept of encoding token and position pairs, Nodeinfo does not fully capture message context as it does not use message types, because it did not assume that message types are known a priori, like previous approaches that utilized message types [11]. Our work extends Nodeinfo by using the concept of message types. However, we do not assume that these message types are known, but we instead extract them automatically using the IPLoM message type extraction algorithm [5].

The goal of message type or event cluster extraction is to group the free form messages of system log events into semantically meaningful groups and produce textual templates (consisting of constant tokens and variable tokens) that represent all members of the group. Semantic groupings usually coincide with messages that are produced by the same print statement in the code of the underlying program that generates the message. The extraction of message types makes it possible to abstract the contents of event logs and facilitates further analysis and the building of computational models. For example, this line of code:
sprintf(message, Connection from %s port %d, ipaddress, portnumber);
in a *C* program could produce the following log entries:
"Connection from 192.168.10.6 port 25"
"Connection from 192.168.10.6 port 80"
"Connection from 192.168.10.7 port 25"
"Connection from 192.168.10.8 port 21".

```
2005-06-03-15.42.50.823719 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
2005-06-03-15.42.50.982731 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
2005-06-06-22.41.37.357738 R20-M0-NA-C:J15-U11 RAS KERNEL INFO generating core.3740
2005-06-06-22.41.37.392258 R20-M0-NA-C:J17-U11 RAS KERNEL INFO generating core.3612
2005-06-11-19.20.25.104537 R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt
2005-06-11-19.20.25.393590 R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt
2005-07-01-17.52.23.557949 R22-M0-NA-C:J05-U01 RAS KERNEL INFO 458720 double-hummer alignment exceptions
2005-07-01-17.52.23.584839 R22-M0-NA-C:J03-U01 RAS KERNEL INFO 458720 double-hummer alignment exceptions
```

Fig. 1. An example event log file. Each line represents an event.

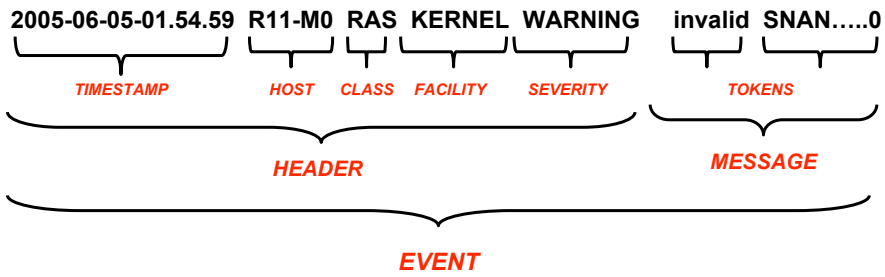


Fig. 2. An example system log event.

When message type extraction is applied, these four log entries would form a group (cluster) or message type, and can be represented by the message type description:
*"Connection from * port *"*.
The wildcards "*" represent message variables or variable tokens. The goal of message type extraction is to find the representations of the message types that exist in a log file. This is what the IPLoM algorithm attempts to do automatically. Detailed descriptions on how IPLoM works can be found in [12] and [5]. Evaluation of IPLoM shows that it has the capability of improving on the results of previous message type extraction/clustering algorithms by finding not only frequent patterns in the data, but also infrequent ones. In a recent evaluation of IPLoM with the BGL dataset, IPLoM was able to achieve an F-Measure result of 91% based on micro-average classification accuracy, when its results were compared with message types produced manually on the same data [13]. It is this set of automatically produced message types that are utilized in this work.

This and other modifications to Nodeinfo produce new frameworks that need to be evaluated for efficiency. This is what we attempt to do in this work. We utilize three criteria, namely;

- Computational Cost: While the Nodeinfo framework itself is not computationally complex, its computational cost can be very high due to the sizes of matrices that it computes (this is explained in detail in the next section). Any significant reduction in the sizes of these matrices reduces the computational cost of the framework. We measure our success in the reduction of computational cost of the framework to the extent that we can reduce the size of these matrices.
- Detection Accuracy: We measure the detection accuracy of the frameworks using their Precision, Recall and F-Measure scores. These are explained in more detail in Section III-D.
- False Positive Rate (FPR): To achieve a fair evaluation, we also include FPR scores for the frameworks. FPR scores evaluate Type-II errors, which are not fully evaluated using only Precision, Recall and F-Measure scores. FPRs are explained in more detail in Section III-D.

These criteria are intended to evaluate the efficiency of these frameworks in the task of alert detection. In the next section, we describe the baseline Nodeinfo framework.

C. The Nodeinfo Framework

Central to the Nodeinfo framework is the concept of a Nodehour. Given any event log E, a Nodehour can be defined as any grouping of lines produced by a single node (c) within a one hour interval in tune with wall clock time [1]. In this case, H_j^c denotes the j^th Nodehour for node c. For each line e_i in the event log, the reporting time and source node are determined by the timestamp (t_i) and reporting node (c_i) fields. Nodehours form the basis of the decomposition of an event log for analysis.

Nodeinfo bases its assessment of each Nodehour on the information content of the individual tokens, t_1 to t_p in the free form message (m_i) field of an event e_i, with regard to its source. To incorporate the encoding of token position into the framework, the concept of a *term* is introduced, which is formed by concatenating each token with a number corresponding to its ordinal position in the message. For each token t_j in message m_i a *term* $w_j = t_j.j$ is created. Now, let W be the set of unique terms and let C be the count of nodes on the network. A $|W| \times C$ matrix $\mathbf{X}$ is computed, where $x_{w,c}$ is the count of the number of times term w appears in messages having node c as source. It is then possible to use matrix $\mathbf{X}$ to compute vector $\mathbf{G}$ with cardinality $|W|$, where each element g_w of $\mathbf{G}$, is calculated using Eq. 1. The $p_{w,c}$ component of Eq. 1 is calculated using Eq. 2. $p_{w,c}$ is the

probability that term w is produced by node c. The output of Eq. 1 corresponds to *1* plus each term's Shannon information entropy over the nodes of the network [2]. Its value ranges between *0* and *1*, with *0* signifying low information content for the term and *1* signifying the highest information content possible. Terms with high information content are more likely to indicate conditions that are of interest to an administrator and could be alerts.

The second step assigns a Nodeinfo score to each Nodehour based on the information content (measured by g_w) of the terms contained in the Nodehour and how many times they appear. Let H be the set of all Nodehours, a $|W| \times |H|$ matrix $\mathbf{Y}$ is defined, where $y^c_{w,j}$ is the count of the number of times term w appears in Nodehour H^c_j. The Nodeinfo score for Nodehour H^c_j can then be calculated using Eq. 3. The $NodeInfo(H^c_j)$ value computed by Eq. 3 represents the magnitude of the vector of counts of terms contained in Nodehour H^c_j weighted by the information content of the term. The information content is determined by each term's value in vector $\mathbf{G}$.

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

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

$$NodeInfo(H^c_j) = \sqrt{\sum_{w=1}^{|W|} (g_w \log_2(y^c_{w,j}))^2} \tag{3}$$

A ranking of Nodehours based on their Nodeinfo scores can then be established. Nodehours with high Nodeinfo scores are then considered more likely to contain alerts than those that are lower in the ranking. For more details on the Nodeinfo framework please see, [1], [2].

III. Methodology

A. BlueGene/L dataset

The BGL dataset utilized in our work is one of several supercomputer datasets [3] recently made available in the USENIX Computer Failure Data Repository [4]. The alerts in these datasets are pre-labelled by domain experts, thus giving us a gold standard against which performances of different frameworks are evaluated. The architecture of the BlueGene/L supercomputer on which the data is collected is detailed in [14]. In our recent work, message types were extracted automatically from these logs using IPLoM [13] . These automatically extracted message types were shown to achieve 91% F-Measure accuracy based on micro-averaging. It is these automatically generated message types that we utilize to transform the message fields of the BGL data. The characteristics of the BGL log are described in Table I.

The Nodeinfo framework relies on the assumption that *"Similar computers correctly executing similar work should produce similar logs"* [1]. For this reason log events from similar nodes need to be analyzed together for the framework to work effectively. To this end, the messages in the BGL data was separated based on the functional roles of the nodes that produced them, leading to four categories i.e. *Compute, IO, Link and Other*. The *Other* node category is actually not a functional grouping of messages but consists of all messages that could either not be placed in any of the three other categories or has unknown source information. The data statistics of the resultant datasets based on functional groupings is detailed in Table II. We note that the *500,000* events recorded for the *Compute* nodes do not constitute all the events generated by *Compute* nodes in the original data but just the first *500,000* events. We extracted the first *500,000* events as opposed to a random sample to prevent a loss of temporal dependencies. The actual number of messages from the *Compute* nodes is *4,153,008*. The extraction of *500,000* events in this work was necessary due to computational and memory limitations of the original Nodeinfo framework. We note that the modified Nodeinfo frameworks can run using all *4,153,008* events in the dataset on our hardware, but are run with only *500,000* events since the original Nodeinfo was used as a baseline.

TABLE I
Log Data Statistics

Start Data	Days	Size(GB)	Messages
2005-06-03	215	1.207	4,747,963

TABLE II
Functional Group Data Statistics

	# Events	# Nodes	# Nodehours	# Alert Nodehours
Compute	500,000	32,770	184,641	37,409
IO	400,923	1,024	219,722	83,973
Link	2,935	517	1,395	33
Other	191,096	2,167	13,666	59

B. Message Type Transformation

Using the message type templates extracted by IPLoM from the BGL event log, we transformed the messages in the event log to have a more concise representation.

In the calculation of the information content of terms as used in the Nodeinfo framework, what is most important is the distribution of the terms across the nodes on the network and not the terms themselves. This means that equally interesting results can be obtained if the messages are transformed into fewer terms. If this can be achieved, then it will result in less computation (because of fewer terms), while still maintaining the distribution of the terms across the nodes. In this work, we aim to investigate the effect of applying message transformation techniques (MTT) on the BGL data using the message types extracted by IPLoM.

Details of different approaches to message type transformation can be found in [15]. The MTT investigated in this work simply replaces a message with a token representing its message type and ignores its variables completely, see Figure 3. We call this full message type transformation and is referred to as MTT3 in the rest of this paper. Our intuition

here is that variable tokens would perhaps (in most cases) not add much value to the task of identifying alerts, instead the message types being more important. The MTT3 approach was designed to investigate this hypothesis.

C. Modifications to Nodeinfo framework

In this section, we describe the modifications that were made to the Nodeinfo framework in an attempt to simplify it and to improve the accuracy of the separation between the alert Nodehours and non-alert Nodehours. Our modifications involve the step that assigns a Nodeinfo score to each Nodehour, i.e. Eq. 3.

By employing MTT3, we transform the approach from a token based indexing one to an approach that is based on message type indexing. This way, individual message types can be easily mapped to alert categories. This means that what is important is the presence of a message type in a Nodehour rather than the number of times the message type appears in a Nodehour. With this intuition in mind, we define a new matrix **Z** analogous to matrix **Y** in the original framework, where $z^c_{w,j}$ is *1* when term w appears in Nodehour H^c_j and *0* otherwise. Matrix **Z** effectively only records unique occurrences of terms in the event data. Then, we now define a new equation for assigning a Nodeinfo score, Eq. 4.

$$NodeInfo(H^c_j) = \sqrt{\sum_{w=1}^{|W|}(g_w * z^c_{w,j})^2} \tag{4}$$

We can say that a Nodeinfo score assigns a value to a Nodehour that defines the degree of *oddity* of its contents. Intuitively, since we index based on message types, we can say that a Nodehour is only as *odd* as the *oddest* message type it contains. This is irrespective of the other message types it may contain. With this intuition, we define a new vector I^c_j for each Nodehour H^c_j, where $I^c_j[w]$ is equal to $g_w * z^c_{w,j}$. We can now assign a Nodeinfo score to Nodehour using Eq. 5, effectively the highest information content value over the terms reported during the Nodehour.

$$NodeInfo(H^c_j) = max_w(I^c_j[w]) \tag{5}$$

D. Experiments

We conducted four experiments using the BGL data. The only difference between the four experiments is the version of the framework applied. We describe these experiments in detail in the following.

- The first experiment, outlined in Figure 4, utilizes the original Nodeinfo framework as is on the BGL data. The data was not preprocessed using either IPLoM or MTT3. This experiment provides a baseline for evaluating the other results.
- The second experiment is intended to evaluate the effect of using MTT3 on the framework. In this experiment, the original Nodeinfo framework [2] was used without modifications, as outlined in Figure 5, but with MTT3 rather than *term* indexing of the original framework.
- In the third and fourth experiments, we evaluate the effect of the modifications (detailed in Section III-C) to the Nodeinfo framework. So, our process flow changes to what is shown in Figure 6. We utilize Eq. 4 in the third and Eq. 5 in the fourth experiment.

Fig. 4. Process flow for the first experiment.

Fig. 5. Process flow for the second experiment.

Fig. 6. Process flow for third and fourth experiments.

In all of the experiments, we utilized the *binary scoring* metric as defined in [2], which defines the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). With these values, we are able to calculate Precision, Recall, F-Measure and False Positive Rate (FPR) results using Eqs. 6, 7, 8 and 9, respectively.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

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

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

To produce the Precision-Recall graphs, which we present in Figure 7, we define R_k as the set of Nodehours formed by taking the top k Nodehours in a list of Nodehours sorted using their Nodeinfo scores at the end of an experiment. We vary the value of k from minimum to maximum and calculate Precision and Recall values for each value of k. This set of Precision and Recall pairs are then used to plot the trend for each experiment in the Precision-Recall plots. Therefore, the graphs in Figure 7 are discrete points, which are fitted to a curve.

Precision as a measure of accuracy measures the ratio of relevant items retrieved in any set of retrieved items. In our case, relevant items would means alert nodehours, while the set of retrieved items would be R_k. On the other hand, Recall

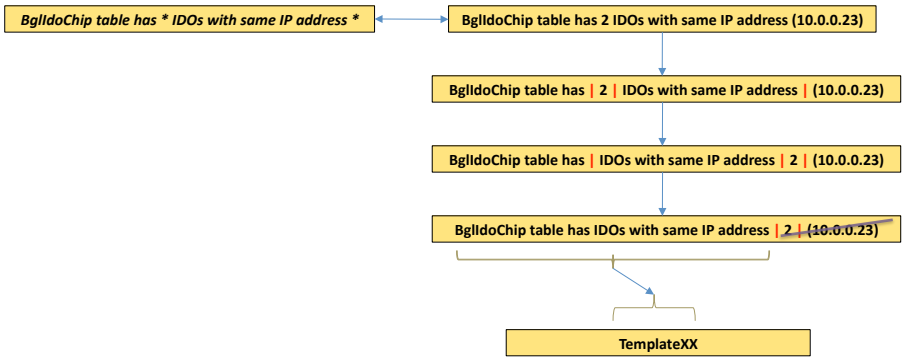


Fig. 3. **Full Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template that matches the message in the first box on the right. This method separates constant value tokens from variable value tokens and discards the variable value tokens in the final transformation. It then substitutes the constant value tokens with a string that represents the message type the event belongs to. In the final box on the right, *XX* represents an ordinal number assigned to the message type and will thus change for different message types.

as a measure of accuracy measures the ratio of relevant items retrieved to the entire set of relevant items. In our case, this would mean the ratio of the alert nodehours in R_k to the entire set of alert nodehours. Precision accuracy and Recall accuracy are usually antagonistic, i.e. its is difficult to improve one without degrading the other. To this end, the F-Measure accuracy score is used to provide a balanced measure of accuracy using both Precision and Recall. Precision, Recall and F-measure scores do not consider the count of TNs, as such they do not give a complete picture of accuracy. To this end, we also present evaluations utilising FPR scores. The FPR measures the ratio of non-relevant items retrieved to the entire set of non-relevant items. In our case, this would mean the ratio of the non-alert nodehours in R_k to the entire set of non-alert nodehours. Thus, the FPRs reported in our work are calculated, when the maximum Precision, Recall and F-measure are achieved. This way, we aim to provide a score of FPRs at the point, where the best performance is achieved using all our measures of accuracy. Since the maximum Recall always reaches 100%, the FPR measurement taken at that point can be susceptible to the effect of outliers, i.e. alert nodehours with very low Nodeinfo scores. To this end, we also present FPR scores measured at 90% Recall.

IV. Results

We will discuss the results of our experiments based on the three criteria we defined for evaluation, namely: (i) computational cost, (ii) detection accuracy and (iii) false positive rates (FPR). The computational effort required for Nodeinfo is to a large degree affected by the size of the $\mathbf{X}$ and $\mathbf{Y}$ matrices used in the computation. An effort to reduce the size of these matrices will result in faster computation. As the size of these matrices is affected by the cardinality of the set of unique terms W. A reduction in size of this set would reduce the computational effort. The results show an average reduction in the number of terms of approximately 99%, see Table III. This implies that we were theoretically able to improve the computational cost by approximately 99%. However, this result would only make sense, if we are still able to achieve equal or better performance with the faster computation. We will show this to be the case in the rest of this section. The results of Experiment-2 as defined in Section III-D are referred to as *Nodeinfo+MTT3* in the rest of this section, while results based on Experiment-3 and Experiment-4 are called *Nodeinfo+MTT3+Eq4* and *Nodeinfo+MTT3+Eq5* respectively.

When the results of *Nodeinfo* and *Nodeinfo+MTT3* are compared in Figure 7, we see that the *Nodeinfo+MTT3* framework achieved equal or better performance for all node categories, hereby showing that we do not sacrifice accuracy for speed. It is intuitive that for any event log E, the number of unique terms u and the number of message types m have a relationship $m \leq u$ and in most situations it is the case that $m \ll u$.

The results for all experiments are highlighted in Figures 7 and 8. For the FPRs, Figure 8 shows a general trend that suggest the FPRs taken at the maximum F-Measure, Precision and Recall are representative of average, best and worst case scenarios for the FPR score. Also, Figure 8(d) affirms our assertion that FPRs taken at 100% Recall can be susceptible to the effect of outliers. For instance the FPR required for 100% Recall for the *Compute* category using the *Nodeinfo+MTT3* framework drops from approximately 98% to 48%, (our investigation shows that this same result could have been achieved if the FPR was measured at 99% Recall). Much lower FPRs can can be achieved by reducing the Recall rate even further, this can be seen by observing the FPRs achieved at maximum Precision and F-Measure. The best case performance for FPRs was achieved with *Link nodes* using the *Nodeinfo+MTT3+Eq5* framework. In this scenario a FPR of 0% was achieved at all three checkpoints, this performance indicates perfect separation between the alert nodehours and the non-alert nodehours. The best case performance that highlights the effect of the modifications made to the framework have on FPR scores can be seen with the *Compute* nodes, where the FPR required for 100% Recall drops from approximately 98% for the baseline to approximately 4% for the *Nodeinfo+MTT3+Eq4* and *Nodeinfo+MTT3+Eq5* experiments. This result shows that

TABLE III
PERCENTAGE REDUCTION IN # OF TERMS

	Compute		IO		Link		Other		Avg.
	#Terms	% Red.	#Terms	% Red.	#Terms	% Red.	#Terms	% Red.	
Original	10,486	0.00	7,391	0.00	599	0.00	11,795	0.00	0.00
MTT-3	86	99.18	48	99.35	12	98.00	92	99.22	98.94

the modifications made to the framework in Eqs. 4 and 5 do make a positive difference to the framework. Overall the FPR results show that the frameworks work with an acceptable level of accuracy for the *Compute* nodes and the *Link* nodes. However, with the *IO* and *Other* nodes, this is not the case. Having said this, we note that useful FPRs are achieved for the *Other* nodes at maximum F-Measure and Precision, and for the *IO* nodes at only maximum Precision. The FPRs at 90% and 100% Recall show that high FPRs achieved here are not due to outliers. Moreover, and for the *IO* nodes the FPR at maximum F-Measure are also unsatisfactory. We believe that the reasons behind this phenomena are the following. Closely allied to FPRs is the number k of top nodehours that need to be considered to achieve 100% Recall. These results are also provided in Figure 8(e). They follow a similar trend as those seen with the FPRs required for 100% Recall.

To understand the reasons for the poor FPR performance with the *Other* and *IO* node categories, we look at the Precision-Recall plots in Figure 7. In the case of the *Other* nodes, the Precision-Recall plots show no significant difference between the results for all experiments, implying that the changes made to the framework were of no effect to the baseline. We also note that none of the data representations achieves a Precision rate higher than 2%. We believe that the poor performance with this category is due to the fact that this is not a true functional grouping of nodes. This reaffirms the *Similar Nodes, Similar Work, Similar Logs* hypothesis highlighted in [1], which is essential to the intuition behind the Nodeinfo framework. However, with the *IO* nodes, we also notice no significant difference between the results for all the experiments, though we did achieve usable Precision rates over 80%. Our investigations on this node category reveal an interesting artifact, which may not be obvious from the data. We found that *6* out of the *17* alert categories associated with *IO* nodes (which accounts for approximately 80% of all alerts in *IO* node data) showed a close correlation to *4* message types. However, these message types all have entropy based information content values, which are less than *0.1*. The range of information content values for event terms is 0-1, with *0* implying the lowest possible information content. Therefore, an information content value of *0.1* would be considered very low and bordering on being completely *normal*. This makes it difficult to achieve high Precision at high Recall rates, as alert Nodehours containing these alerts will be ranked very low amongst the other Nodehours.

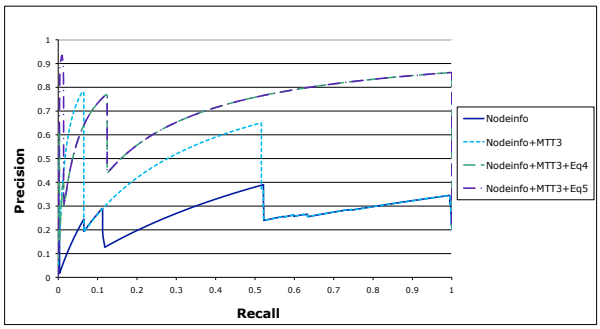
In our opinion, this observation could be due to either one of two reasons. First, this could mean that certain types of errors occur on the *IO* nodes at the same frequency throughout the network, or it could mean that certain errors are not generated autonomously by the individual *IO* nodes but by an *IO* subsystem, which controls the *IO* nodes. These errors generated by the subsystem are then likely *sensed* by all *IO* nodes, leading to a near equal frequency of occurrence across the nodes. We believe that the latter explanation may be more likely, because in such cases, it is very likely that very low entropy based information content values will be seen.

On the other hand, with the *Compute* and *Link* nodes, when we consider detection accuracy, as represented by the Precision-Recall plots in Figure 7, we notice visible significant changes in the results. The most significant is with the *Link* nodes, where the results achieve 100% Precision and 100% Recall with the framework that uses Eq. 5. This then results in the effective FPR of 0% achieved at all checkpoints. With the *Compute* nodes, the frameworks based on Eq. 4 and Eq. 5 show that the best performance achieves approximately 100% Recall at approximately 87% Precision in both cases. The results suggest that the most promise is held by the frameworks based on Eq. 4 and Eq. 5, further tests are required to determine the better one.

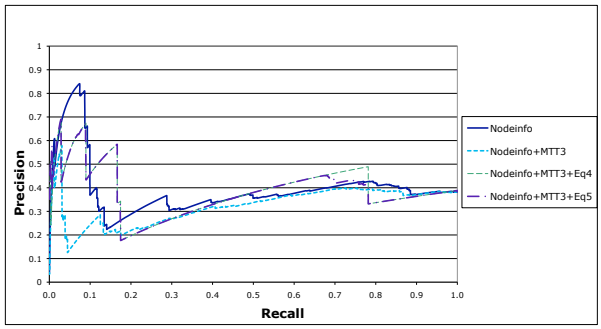
V. CONCLUSION AND FUTURE WORK

In this work we evaluated three different entropy-based approaches to alert detection in HPC logs on the BGL data. In these evaluations, we employed the Nodeinfo Framework [2] as a baseline. Nodeinfo is an alert detection framework that is currently in production use at Sandia National Laboratories. The results show much promise for entropy based approaches and also shows their limitations. The major limitation as seen with *Other* node category is that useful results may only be achievable if there is a way to place the nodes into categories based on their similarities. We also note that we must be certain that we use the primary source of all messages in evaluating entropy values. In the case of the *IO* nodes we were not able to significantly increase accuracy for high recall rates for this reason. It may be necessary in this case to release sub-system information for the data as possible sources of the events in the log. This will form part of our future work.

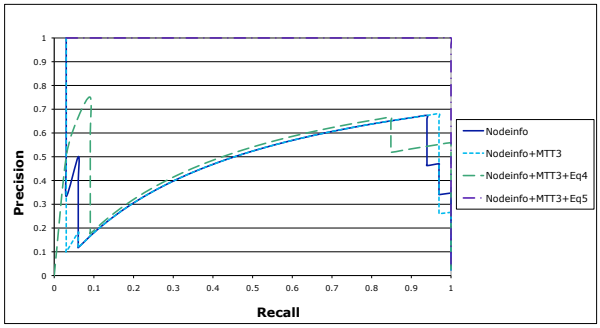
Table IV summarizes our evaluations for *Compute* and *Link* categories using *Nodeinfo* as a baseline. In Table IV the *Computational Cost* column shows the percent reduction in computational cost w.r.t. baseline, the *Accuracy* column shows the average maximum F-Measure achieved, while the *FPR* column shows the average FPR achieved at the maximum F-Measure. The evaluations show that we can achieve equal or better performance, and much less computation time with the entropy based approaches when we form term indexes based



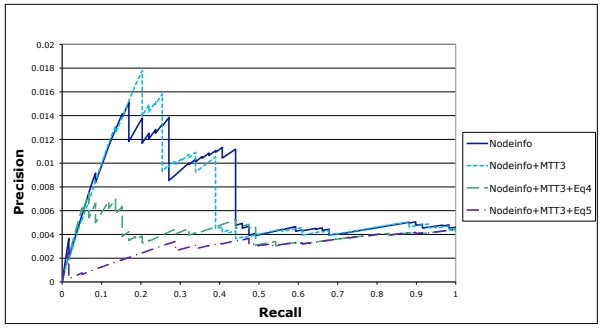
(a) Compute



(b) IO



(c) Link



(d) Other

Fig. 7. Precision-Recall plots for all node categories. Results for Experiment-1 are labelled Nodeinfo, results for Experiment-2 are labelled Nodeinfo+MTT3, results for Experiment-3 are labelled Nodeinfo+MTT3+Eq4 while results for Experiment-4 are labelled Nodeinfo+MTT3+Eq5.

on message types. We also showed that the upper-bound on the performance of an entropy based approach is effectively 100% precision at 100% recall leading to an effective FPR of 0%. This result shows much promise for the application of entropy based approaches to the task of alert detection in HPCs and shows that there is still a lot of room for improvement. Our evaluations suggest that best promise is held by the frameworks based on Eqs. 4 and 5. On the other hand, while information on the frequency of terms is lost when the Nodeinfo framework is modified using Eqs. 4 and 5, results show that such information loss does not seem to affect accuracy. We theorize that this may be due to the fact that messages linked to alerts do not appear frequently and measuring only their presence in a Nodehour is most times sufficient to identify an alert nodehour. On the other hand measuring only the presence of normal messages seems to affect the performance positively by reducing the impact of normal messages on the Nodeinfo score, since they almost always occur frequently. The log_2 component of Eq. 3 also addresses the impact of frequent normal message terms, but we believe our approach addresses the problem better. It is our opinion that measuring the occurrence frequency of terms will only be beneficial to the task of alert detection, when we are detecting bursty alert types. We also note that measuring only the occurrence of terms in nodehours, which for MTT3 are message types, represents an implicit way of carrying out automatic grouping of time-correlated messages.

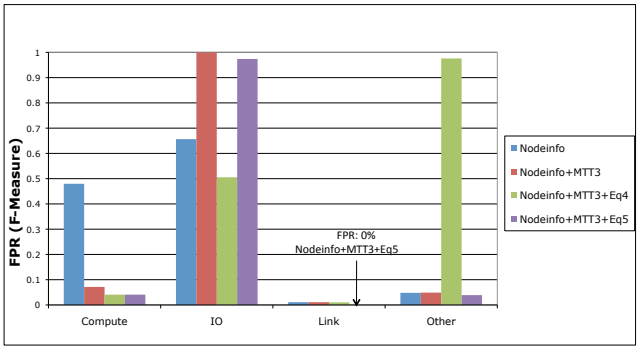
TABLE IV
SUMMARY OF EVALUATION

	Computational Cost	Accuracy	FPR
Nodeinfo	Baseline	0.65	0.25
Nodeinfo+MTT3	98.6%	0.69	0.04
Nodeinfo+MTT3+Eq4	98.6%	0.84	0.03
Nodeinfo+MTT3+Eq5	98.6%	0.97	0.02

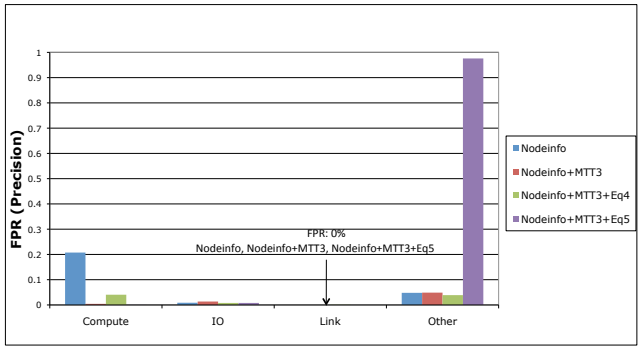
Future work will involve further evaluation of entropy based alert detection on more real world datasets. We also intend to investigate further the reasons for the relatively poorer performance of the frameworks on the *IO* nodes. The findings of such an investigation could be applied to make entropy based alert detection more effective.

ACKNOWLEDGEMENTS

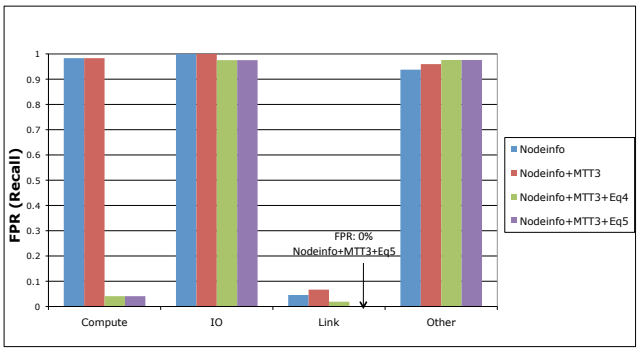
The authors would like to thank Jon Stearley of Sandia National laboratory for his help while re-engineering the Nodeinfo framework and in preparing this manuscript. This research is supported by a Natural Science and Engineering Research Council of Canada (NSERC) Strategic Project Grant. This work is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



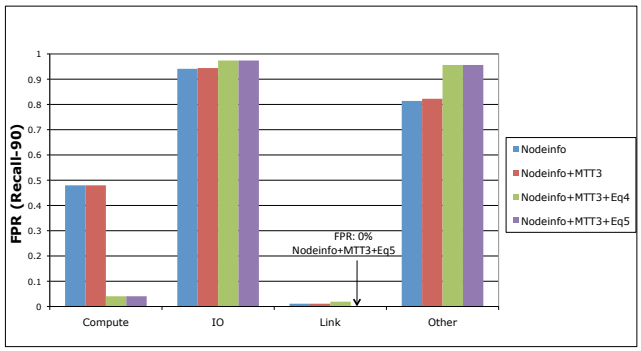
(a) Effective False Positive Rate (FPR) at Maximum F-Measure



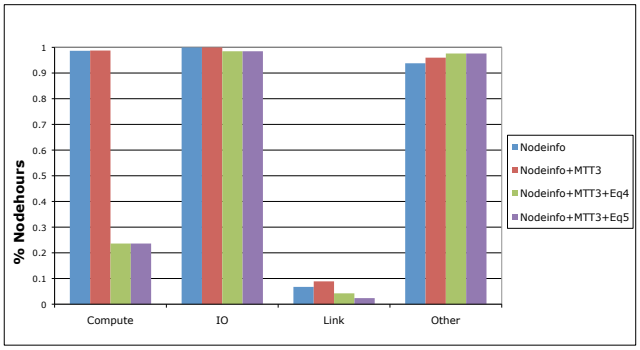
(b) Effective False Positive Rate (FPR) at Maximum Precision



(c) Effective False Positive Rate (FPR) at 100% Recall



(d) Effective False Positive Rate (FPR) at 90% Recall



(e) Top Percentile of Nodehours considered to achieve 100% Recall

Fig. 8. Results for Experiment-1 are labelled as Nodeinfo, results for Experiment-2 are labelled as Nodeinfo+MTT3, results for Experiment-3 are labelled as Nodeinfo+MTT3+Eq4 while results for Experiment-4 are labelled as Nodeinfo+MTT3+Eq5. These results are for all node categories using all versions of the alert detection framework.

REFERENCES

[1] J. Stearley and A. Oliner, "Bad words: Finding faults in spirit's syslogs," in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, may 2008, pp. 765 –770.

[2] 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.

[3] 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.

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

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

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

[7] 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 2008*, October 2008, pp. 99 – 108.

[8] S. Ma and J. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," in *Proceedings of the 16th International Conference on Data Engineering*, 2000, pp. 205–214.

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

[10] Y. Liao and V. Vemuri, "Using Text Categorization Techniques for Intrusion Detection," in *11th USENIX Security Symposium*, August 2002, pp. 51–59.

[11] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, S. Scott, and C. Engelmann, "Blue Gene/L Log Analysis and Time to Interrupt Estimation," in *International Conference on Availability, Reliability and Security*, 2009, pp. 73–180.

[12] A. Makanju, N. Zincir-Heywood, and E. E. Milios, "Iplom: Iterative Partitioning Log Mining," Dalhousie University, Tech. Rep. CS-2009-07, March 2009.

[13] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Extracting Message Types from BlueGene/L's Logs," in *Proceedings of the SOSP Workshop on the Analysis of System Logs (WASL)*, 2009.

[14] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta, "Filtering Failure Logs for a BlueGene/L Prototype," in *International Conference on Dependable Systems and Networks.*, July 2005 2005, pp. 476–485.

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