A Next Generation Entropy Based Framework for Alert Detection in System Logs

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

***Abstract*—Recent research efforts have highlighted capability of entropy based approaches in the automatic discovery of *alerts* in system logs. We refer to messages maybe of interest to an administrator as *alerts*. In the best case, they have been shown to detect all alerts at a false positive rate of 0%.**

In this work, we extend the recent research to present the detailed evaluations of three entropy based approaches on new datasets not utilized in previous papers. We also extend the approach with the introduction of a *Cluster Membership Anomaly* score. This extension of the approach is intended to reduce the false positive rates required to detect all alerts. Previous work has shown that false positive rates required for the detection of all alerts for an entropy based approach could be very high. The results show that the *Cluster Membership Anomaly* has value for the reduction of false positive rates.

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

I. Introduction

The need for the development of tools and techniques for the automatic analysis of system logs is well documented in literature [1]. Log analysis tasks are varied, our work however focuses on the task of *alert detection*. By *alert detection*, we refer to the task of identifying events (or group of events) in a system log that are symptomatic of failure or require the attention of an administrator. The automation of *alert detection* is important, not only because it contributes to system dependability but because it contributes to the larger goal of building autonomic computer systems, i.e. computer systems that are capable of self management [2].

Nodeinfo is an entropy based alert detection mechanism, which is currently in production use at Sandia National Laboratories [3]. Unlike previous automated approaches to anomaly detection in system logs [4] [5], Nodeinfo has been shown to work with an acceptable false positive rate of 0.05% [6], [3], while a variant of Nodeinfo has been shown to detect all alerts at a false positive rate of 0%. Entropy based approaches to alert detection work on the assumption that "Similar computers correctly executing similar code should have similar logs", as such *alerts* in system logs should *stand out* when logs from similar computers are compared. Entropy measures that measure randomness in data, are therefore suited for this kind of analysis.

In this work, we build on our previous success in the modification of the Nodeinfo framework [7], [8], [9]. We extend our evaluations of the different variants of the entropy based framework of Nodeinfo, through the use of several different datasets. The ability to replicate previous success on more datasets would indicate the robustness of the approach and reveal weaknesses if the results are contrary.

We also experiment with a new concept to alert detection, which we call *cluster membership alert detection*. This concept works on the assumption that alerts are generally infrequent in system logs. In some cases, predefined *alerts* maybe be programmed to report information into an event log periodically, e.g. a sensor that reports surrounding temperature values every x minutes or one that reports on the status of a link. Even in such cases, only a few of the reported *alerts* will provide information that will be of interest of the administrator. In the examples given above, an administrator may only be interested if the *alerts* report a temperature reading above a certain value or if the link is *down*, respectively. Hence our assumption of *alerts* being infrequent holds even in these cases.

If we can successfully cluster system log partitions based on the information content of events contained in those partitions, then we can assume that clusters that contain fewer log partitions are more likely to contain alerts. It is hoped that the introduction of this component to the framework would help to reduce the false positive rates required to detect all alerts. A problem that has been highlighted in previous work [9]. Based on the results of the evaluations of this concept, which show the efficacy of the approach, we propose a new entropy based alert detection framework, which includes the use of a *Cluster Membership Anomaly score*. We also discovered that contrary to our previous deductions, an entropy based approach to alert detection on groups of unrelated nodes could still produce reasonable results.

The rest of this paper is organized as follows. We discuss concepts important to understanding our work and previous work in Section 2. Section 3 discusses the proposed *cluster membership alert detection* technique. Section 4 discusses the methodology of the experiments we carried out to evaluate our proposed framework, whereas the results of those experiments are discussed in Section 5. Finally, conclusions are drawn and the future work is discussed in Section 6.

II. Background and Previous Work

A. Alert Detection in System Logs

A system log is a record of events that occur on a computer system, usually in temporal order. An event in a system log is not a homogenous entity, it comprises several information fields separated by a delimiting character, usually whitespace. Most events in general, report occurrences, which are of an *informative* nature i.e. non-critical events but sometimes they do report events, which are critical and may require the attention of an administrator. The task of identifying such critical events (or group of events) in a system log is what we refer to as *alert detection*.

A review of the literature shows that there are several previous attempts at automating the task of alert detection in system logs. They vary from simple approaches that search system logs for message patterns, which are indicative of previously known failure conditions [10], to visualization techniques that aid the detection of alerts manually [11] and to more complex schemes that use computational techniques like time periodicity of messages [12] or term weighting schemes [4].

More recent approaches to the task of automatic alert detection include Nodeinfo [3], Principal Component Analysis based alert detection [13] and Principle Atom Recognition in Sets (PARIS) [14]. In [13], Xu et al. propose a framework for detection of system problems through the mining of console logs. Using message types extracted directly from program source code, relevant features were extracted from system logs and processed using Principal Component Analysis. The Principal Component Analysis based analysis was able to identify outliers (alerts), which they showed corresponded with periods where faults were injected into the network. In [14], Aharon et al. first propose a sequential algorithm for the discovery of message types. They then propose a novel algorithm, Principle Atom Recognition in Sets (PARIS), which uses the message types discovered to identify message types that tend to occur together in the event stream. These correlated message types were then assumed to be indicative of *normal* operation in the system. Aharon et al. then propose an alert detection mechanism based on the monitoring and visualization of these correlated message types.

On the other hand, in [3], Oliner et al. proposed an entropy based approach to alert detection in system logs called Nodeinfo. Nodeinfo proceeds from the work of Liao [5] and Reuning[4] by using the more complex *"log.entropy"* term weighting scheme. Though it utilizes the concept of encoding token and position pairs as a means of capturing message context, Nodeinfo does not fully capture message context as it does not use message types. Nodeinfo does not assume that message types are known a priori, like in the work of Taerat et al. [15] or that a way of extracting them exists [13], [14], [16].

In our recent work, we have improved the state of the art as far as the Nodeinfo detection algorithm is concerned [7], [8], [9] by introducing the concept of message types into the framework and making modifications to its anomaly scoring mechanism. In our work, we do not assume that these message types are known, but we instead extract them automatically using the Iterative Partitioning Log Mining (IPLoM) message type extraction algorithm [16]. We utilize IPLoM for message type extraction because we do not assume access to source code as in the work of Xu et al. [13]. IPLoM had been shown to produce message types, which matched manually produced messages types closely. In addition, it is also capable of finding not only frequent patterns in the data but also infrequent ones. These are results, which improve on previous approaches to message type extraction with open source implementations like SLCT and Loghound [17]. In our recent work, we extracted message types automatically using IPLoM on data from one of the fastest supercomputers in the world [18] . These automatically extracted message types were shown to achieve 91% F-Measure accuracy based on micro-averaging, when they were compared to manually produced message types. The modifications made to Nodeinfo have been shown to improve on Nodeinfo by:

- Being able to process 100 times as many system log entries per time unit without a drop in the detection accuracy of the framework [7].
- Achieving an F-Measure detection accuracy of up to 100% leading to an effective false positive rate of 0% [8], [9].

This work extends the work carried out in [18], [7], [9] by extending our testing to more datasets and proposing additional approaches to reduce the false positive rates of the detection mechanism. This is explained in detail in section III. In the next section, we describe the Nodeinfo framework [6], [3] and its variants [7], [8], [9] in more detail. For the rest of this paper, we will refer to variants of Nodeinfo collectively as *NodeinfoPlus*.

B. Entropy Based Alert Detection

Nodeinfo and NodeinfoPlus are unsupervised entropy based system log alert detection frameworks [6], [3], [7], [8], [9]. They work based on the assumption that *similar computers correctly executing similar code should have similar logs* [6]. With this in mind, they calculate entropy based information content scores for the terms that appear in *similar* partitions of a system log, i.e partitions produced by similar nodes on the network. They then use these information content scores as indicators of *interestingness*.

Neither Nodeinfo nor NodeinfoPlus attempts to identify alerts on an event by event basis, they identify portions of the log referred to as nodehours [3], which are more likely to contain alerts than others. A nodehour is basically one hour of log information produced by a single node on the network.

Nodeinfo and NodeinfoPlus carry out their alert detection using a three step process:

- **Step-1:** Calculate an entropy based information content score for each term that appears in the system log.

- **Step-2:** Calculate the information content score for each nodehour based on the information content of the terms that occur in it.
- **Step-3:** Create a ranking of the nodehours based on their information content score to facilitate alert detection.

In the first step, Nodeinfo and NodeinfoPlus calculate entropy based information content scores for each individual *term* that appears in the free form message fields of an event. The difference for both frameworks however comes from how they define a *term*. With Nodeinfo each of the individual tokens in the free form message is converted to a term by concatenating it with a number corresponding to its ordinal position in the free form message. However, NodeinfoPlus creates terms with the use of message types. Using Message Type Transformation (MTT) [8], the message field of an event is transformed to produce more concise and structured representations of the messages. Three message transformations are proposed in [8], these are: Phrasal message type transformation, (ii) message type transformation with variables, and (iii) Full message type transformation. Full message type transformation can provide up to a 99% reduction in the number of unique terms found in a system log without reducing the detection accuracy of the framework [7], it is also utilized in this work. The computational and memory complexity of an entropy based approach to alert detection is largely affected by the number of the *terms* in the data. We therefore achieve a reduction of computational effort and memory requirements of NodeinfoPlus compared to Nodeinfo because of its reduced *term* set.

Full message type transformation simply replaces a message with a unique token representing its message type and ignores its variables completely. The intuition here is that variable tokens are not as important message types in the task of identifying alerts. The results presented in [7] shows that this hypothesis is plausible. If the analysis of message variables is necessary, then it should be done between message variables that occur within a single message type only and not between all the message variables that occur in a system log.

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

The second step assigns a score to each nodehour based on the information content of the terms contained in the nodehour. With Nodeinfo this score represents the magnitude of the vector of information content values of the terms contained in nodehour H_j^c weighted by the frequency counts of the terms. Let H be the set of all nodehours, a $|W| \times |H|$ matrix $\mathbf{Y}$ is defined, where $y_{w,j}^c$ is the count of the number of times term w appears in nodehour H_j^c. The Nodeinfo score for nodehour H_j^c can then be calculated using Eq. 3. The modifications made to this step of the framework produces two variants of NodeinfoPlus, we will refer to these variants as *NodeinfoPlus_Uniq* and *NodeinfoPlus_Max* in the rest of this paper.

For *NodeinfoPlus_Uniq* and *NodeinfoPlus_Max*, we define a new matrix $\mathbf{Z}$ analogous to matrix $\mathbf{Y}$, where $z_{w,j}^c$ effectively only records unique occurrences of terms in the event data. *NodeinfoPlus_Uniq* now defines a new equation for assigning a information content score to a nodehour, i.e. Eq. 4. *NodeinfoPlus_Max* assigns an information content score to each nodehour using Eq. 5. Results highlighted in [8], [9] have shown that *NodeinfoPlus_Uniq* and *NodeinfoPlus_Max* contribute significantly to improve alert detection accuracy, in one case achieving an F-Measure score of 100%.

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

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

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

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

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

The third step of the computation proceeds in very much the same way for both systems. After each nodehour is assigned an information content score, a ranking of nodehours based on their information content scores is established. Nodehours with high information content scores are then considered more likely to contain alerts than those that come up lower in the ranking. In summary the major differences between Nodeinfo and NodeinfoPlus are twofold:

- The use of message types as *terms* instead of word and token position pairs, i.e. Message Type Indexing (MTI).
- Variations in the mechanism for assigning a Nodeinfo (information content) score to a nodehour.

III. Cluster Membership Alert Detection

In this section, we introduce a new component of the alert detection mechanism of our framework. We call this component *cluster membership alert detection*, which produces what we call a *cluster membership anomaly score*, which measures the possibility that a nodehour contains an alert. The intuition behind this is based on the fact that alerts are usually infrequent in a system log. Therefore, if we are able to cluster nodehours in a way that captures the probability that a nodehour might contain an alert or not, then we can say that nodehours that belong to clusters with very few members

are more likely to contain alerts than those that belong to clusters with a larger number of members. Our implementation is described below.

Let S be the set of all nodehours, we can separate S into n clusters, $S_1 \dots Sn$, such that $S_1 \cup S_2 \cup \dots \cup S_n = S$ and $S_i \cap S_j = \emptyset \forall i, j$. The disjunctive rule follows from the fact that every event in the system log contains information about its source node and the time it was produced, hence an event cannot belong to more than one nodehour. We also state that each nodehour H_j^c in S is associated with an information content score ni_j^c calculated using the methods described in Section II-B. We define the following terms:

- Max_{ICS}**:** This is the maximum information content score associated with any nodehour in S.
- Max_{CNT}**:** $Max_{CNT} = max(|S_i|)\, for\, i = 1 \dots n$, this is effectively the cardinality of the subset (cluster) of S with the most number of elements.
- Min_{CNT}**:** $Min_{CNT} = min(|S_i|)\, for\, i = 1 \dots n$, this is effectively the cardinality of the subset (cluster) of S with the least number of elements.
- NH_{CNT}**:** This is a value associated with each nodehour. For a nodehour $H_j^c \in S_i$, then $NH_{CNT}(H_j^c) = |S_i|$, this is effectively the cardinality of the subset (cluster) of S that nodehour H_j^c is a member of.

We now define the cluster member anomaly score (CMS) for each nodehour using Eq. 6. The $(Max_{CNT} - NH_{CNT})/(Max_{CNT} - Min_{CNT})$ component of Eq. 6 produces a value that represents the normalized distance (based on cardinality) of the cluster a nodehour belongs to from the cluster with the most number of elements. This value is then squared, to ensure that only the clusters with the smallest number of elements have large distance values. The distance values produced are in the range $[0, 1]$ and are intended to be used in conjunction with the information content scores, so we multiply the distance value by Max_{ICS} to ensure that the cluster membership anomaly scores produced have the same scale as the information content scores.

$$CMS(H_j^c) = Max_{ICS} * \left(\frac{Max_{CNT} - NH_{CNT}}{Max_{CNT} - Min_{CNT}} \right)^2 \quad (6)$$

We can now have a new anomaly score for each nodehour, which is the sum of the information content score and the cluster membership anomaly score as defined in Eq. 7, w_1 and w_2, are weights, which can be used to adjust the contribution of the information content score and cluster membership score to the final anomaly score of a nodehour.

$$AS(H_j^c) = (w_1 * NodeInfo(H_j^c)) + (w_2 * CMS(H_j^c)) \quad (7)$$

The question that may now be asked is how to cluster S to ensure that clusters produced are somehow representative of the nodehours containing alerts? To provide a proof of concept and test this idea, we design a simple clustering technique, which clusters nodehours based on the entropy of the message types they contain. The steps of this clustering technique are given below in Algorithm 1.

TABLE I
LOG DATA STATISTICS

System	# Days	Size(GB)	Messages
Blue-Gene/L (BGL)	215	1.21	4,747,963
Liberty	315	22.82	265,569,231
Spirit	558	30.29	272,298,969

Algorithm 1 This pseudo-code describes our method for clustering the nodehours in S.

Input: Set S of nodehours with associated information content (Nodeinfo) scores.
Set MT of all message types found in the system log E defined by S.
Output: Collection $S_1 \dots Sn$ of clusters of S.

```
for each message type in MT do
    Define a cluster S_i of S, such that |S_i| = ∅
end for
for each nodehour H_j^c ∈ S do
    if message type with maximum entropy in H_j^c is mt_i then
        H_j^c ∈ S_i {Add H_j^c to S_i}
    end if
end for
for each cluster S_i of S do
    if |S_i| = ∅ then
        Delete S_i {Discard all empty clusters}
    end if
end for
```

IV. METHODOLOGY

A. Datasets

The three datasets utilized in our work are part of a set of high performance computing (HPC) system logs, which were made publicly available in the USENIX Computer Failure Data Repository [19]. These datasets are well suited for the evaluations, which we undertake for the following reasons:

- The events in these datasets have been previously labelled as alerts and non-alerts by domain experts. This gives us ground truth with which to compare the results of automatic analysis.
- The fact that the datasets are all publicly available adds to reproducibility of the results presented here.
- Previous work in the development and evaluation of Nodeinfo has used these datasets [6], [3].

Details of the hardware architecture, configuration, characteristics, log collection methods and alert identification policies of these datasets and the systems that produced them can be found in [20]. A summary of the characteristics of the datasets is provided in Table I.

Compared to our previous publications, the additional data sets considered in this work (i.e., Liberty and Spirit) are much larger in size and cover a longer time period, but they contain fewer alerts [20]. These characteristics make it more difficult to detect alerts in them. Their addition therefore supports a more robust evaluation.

Both the Nodeinfo and NodeinfoPlus frameworks rely on the assumption that *"Similar computers correctly executing similar work should produce similar logs"* [6]. For this reason, log events from similar nodes need to be analyzed together for

TABLE II
FUNCTIONAL GROUP DATA STATISTICS

	# Events	# Nodes	# Nodehours	% Alerts
BGL-Compute	4,153,009	32,770	1,581,845	4.42
BGL-IO	400,923	1,024	219,722	38.22
BGL-Link	2,935	517	1,395	2.37
BGL-Other	191,096	2,167	13,666	0.43
Liberty-Compute	200,940,735	236	1,748,865	0.29
Liberty-Admin	52,211,676	2	27,162	0.04
Liberty-Other	12,416,820	6	44,447	0.22
Spirit-Compute	218,697,851	512	6,648,719	0.19
Spirit-Admin	41,847,257	2	26,216	3.10
Spirit-Other	11,753,861	7	57,532	0.25

the framework to work effectively. To this end, we separated the messages in the datasets based on the functional roles of the nodes that produced them, leading to ten categories. Four categories for BGL i.e. *Compute, IO, Link and Other*; and three categories for Liberty and Spirit, i.e. *Compute, Admin and Other*, for each dataset. The three *Other* node categories are actually not functional groupings of messages but consist of all messages that could either not be placed in any of the other categories (of each dataset) or had unknown source information. The data statistics of the resultant datasets based on functional groupings are detailed in Table II.

B. Experiments

The experiments, which we report in this work, are designed to investigate three questions.

1) **Which of the NodeinfoPlus_Uniq and NodeinfoPlus_Max frameworks is more robust?** Previous work has shown the NodeinfoPlus_Uniq and NodeinfoPlus_Max to be capable of producing more accurate results than Nodeinfo with 99% less computation [7], [8], [9]. No clear winner was however established, we carried out further tests in a bid to determine the more robust of the two frameworks.
2) **How does the information loss caused by the use of the Z matrix in NodeinfoPlus, as opposed to the Y matrix used in Nodeinfo affect the alert detection capabilities of NodeinfoPlus?** The **Z** matrix effectively only records unique occurrences of terms in the event data as opposed to the **Y** matrix which records the frequency of term occurrences. It is our opinion that measuring the occurrence frequency of terms will most likely only be beneficial to the task of alert detection, when we are detecting bursty alert types.
3) **How does the cluster membership anomaly score helps to reduce the false positive rates of the alert detection mechanism of NodeinfoPlus.** We evaluate our experiments to show the contribution of this score with experimentally measured metrics.

In light of the first question above, we add the Liberty and Spirit datasets [20] in addition to the BGL dataset used in previous work, so as to provide more robust evaluations that will show a clear winner. We do note two major differences between the results presented here and those presented in [7], [8], [9]. Firstly, in previous works, the message types were extracted from the entire event log, before functional decomposition. However in this work, the message types were extracted after functional decomposition, i.e. message types were extracted from each of the node category logs independently. This slight difference in order is more logical. This change may not make much of a difference to the results but is reported to ensure the reproducibility of our results. Secondly, the results presented for the BGL *Compute* node category were not carried out on a subset of the data but on the entire dataset. In these set of experiments, w_1 and w_2 in Eq. 7 are set to 1 and 0, respectively.

To be able to answer the second question, we add a new evaluation technique to the experiments carried out above. A high event inter-arrival rate (or *burstiness*) in any system log is usually an indication of a faulty condition [6]. Event *burstiness* should usually lead to a system log with an unusually large size. A very simple technique called *Bytes* uses this knowledge for alert detection [6], [3]. *Bytes* simply ranks nodehours based on the number of bytes of data that they contain, i.e. nodehours with a high byte count are more likely to contain alerts. Previous work has already shown that Nodeinfo outperforms *Bytes* in the task of alert detection [3]. If we are able to show that Nodeinfo only significantly outperforms NodeinfoPlus when *Bytes* is able to detect alerts effectively, it will add strength to our argument. It will also help to validate our hypothesis that collecting statistics about the frequency of message terms, as is done with the Nodeinfo framework, is perhaps only beneficial when alerts, which have *bursty* signatures, are detected.

After we have established the more robust framework between NodeinfoPlus_Uniq and NodeinfoPlus_Max, we now boost the nodehour scores of that framework using the cluster membership anomaly score to answer the third question. In these set of experiments, w_1 and w_2 in Eq. 7 are set to 1 and 1, respectively. We then compare the evaluations of both frameworks to ascertain the contribution of the cluster membership anomaly score. Any improvement in performance will provide a proof of concept for the idea of using clustering to determine the existence of alerts.

For all our experiments, we utilized the *binary scoring* metric as defined in [3], 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 results using Eqs. 8, 9, 10 and 11, respectively.

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{10}$$

$$FPR = \frac{FP}{FP + TN} \tag{11}$$

If we define R as an array of the nodehours in S sorted using their anomaly scores at the end of an experiment and R_k as the set of nodehours formed by taking the top k nodehours in R, then we can vary the value of k from minimum to maximum and calculate Precision and Recall values for each value of k. This is essentially the way we evaluated our experiments to generate the Precision-Recall graphs, which we present in Figure 1. In our evaluations, we assume that R_k should contain the set of alerts and $\sim R_k$ should contain non-alerts.

Precision measures the ratio of alert nodehours to non-alert nodehours in R_k. While Recall measures the ratio of alert nodehours in R_k to the complete set of nodehours in R. Precision and Recall are usually negatively correlated, i.e. in a non-trivial problem, it is 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. Definitions of and detailed discussion about Precision, Recall and F-Measure can be found in these works: [21], [22]. While an example of the use Precision and Recall in evaluation of experiments in the retrieval of failure signatures in systems management can be found in [23].

Precision, Recall and F-measure scores do not consider Type-II errors, as such they do not give a complete picture of accuracy. To this end, we also present evaluations utilizing false positive rate scores. The false positive rate measures the ratio of non-alert nodehours in R_k to the complete set of non-alert nodehours in R. The false positive rates in our work are reported at the point when maximum Recall is achieved and is therefore representative of the worst case false positive rate, which can be encountered with the frameworks. Much lower false positive rates are possible, albeit at the expense of not been able to detect all alerts. Since the maximum Recall always reaches 100%, the false positive rate measurement taken at that point can sometimes be susceptible to the effect of outliers, i.e. alert nodehours with very low Nodeinfo scores. To this end, we also present false positive scores measured at 99% Recall to evaluate experiments in the general case. The false positive rate at 99% Recall implies that the false positive rate measured at the value of k where 99% of alerts are included in R_k. However, we present false positive rates measured at 100% Recall for experiments, which evaluate the contribution of *cluster membership alert detection*. Likewise, the false positive rate at 100% Recall implies that the false positive rate measured at the value of k where 100% of alerts are included in R_k.

V. Results

Before we discuss the results of our experiments in detail, we highlight points, which are important for the interpretation of the results presented. Firstly, all the results presented here using any variation of the NodeinfoPlus framework are produced with 99% less computation and memory on average. As this result was already shown in previous work, we do not give detailed results in this paper [7]. Secondly, low (below 50%) Precision scores in the Precision-Recall plots presented in Fig. 1 do not necessarily mean poor results. These results have to be evaluated in conjunction with alert ratios presented in Table. II. We notice from Table. II, that in most cases alert nodehours form less than 1% of all nodehours. So if for instance, in one functional grouping, we have alert nodehours forming only 1% of all nodehours, achieving a Precision score of 10% could actually be *good*, as this performance is in the order 10 times better than what would be expected from a simple random sampling of nodehours. The results in Fig. 2 show the false positive rates achieved at 99% Recall for all node categories. These false positive rates are representative of the worst case false positive rates required to detect all alert nodehours in the data. Lower false positive rates are possible at the expense of not detecting all alerts. Let us now analyze the results of our experiments in more detail.

Our first conclusion from the results is that NodeinfoPlus_Uniq is the most robust framework amongst the frameworks evaluated. This is inspite of the fact that NodeinfoPlus_Max produces the best overall result (100% F-Measure) on the *BGL-Link* category. The false positive rates results for the *Liberty-Admin* and *Spirit-Admin* node categories in Fig. 2 show NodeinfoPlus_Max with a false positive rate, which is significantly poorer than both NodeinfoPlus_Uniq and Nodeinfo. These results support the assumption that NodeinfoPlus_Uniq is a better framework and this answers the first question we asked in Section IV-B. The effect of faults in a system log are most times not restricted to any single event in the log, NodeinfoPlus_Max takes the *greedy* approach of evaluating nodehours based on a single event. This approach can achieve very good results as seen with the *BGL-Link* category but is generally not robust.

Secondly, the results presented for the *Liberty-Compute* and *Spirit-Compute* categories in Fig. 2, strongly support our second hypothesis. In these categories, Nodeinfo performs just as well as NodeinfoPlus for lower Recall rates. With the *Liberty-Compute* category, it even outperforms both NodeinfoPlus frameworks for Recall rates below 55% and achieves a Precision score of $\sim$75% but its performance falls steeply for Recall rates above 55%. It is pertinent to mention that *Bytes* also produced good results in these categories, meaning that a good portion of the alerts in these categories have *bursty* signatures. We also notice how similar the trend lines for Nodeinfo and *Bytes* are with the *Liberty-Compute* category.

Thirdly, the false positive rate at 100% Recall for the BGL node categories in Fig. 3 highlight the contribution of the cluster membership anomaly score. As stated earlier, false positive rates measured at 100% Recall are sometimes susceptible to the effect of outliers. In Fig. 1, it can be seen that at this point the entropy based approaches do not differ significantly from the simple *Bytes* technique. Thus, reducing the justification for their use when the detection of *all* alerts is required. The results shown in Fig. 3 show how the introduction of the cluster membership anomaly score helps to reduce the effect of these outliers by reducing the effective false positive rates experienced at 100% Recall. The most significant is with the *BGL-Compute* and *BGL-IO* categories.

Fourthly, looking generally at the results presented in Fig. 1

and Fig. 2, we see that the NodeinfoPlus frameworks generally outperform the Nodeinfo framework, especially for Recall rates below 50%. We also notice the high false positive rates (for all frameworks) produced on *BGL-IO* node category. This can explained by the fact that certain alert types in the *BGL-IO* category can be reported by nodes in the category even when the fault is not local to the node [9]. This causes these alerts to appear *normal* as they have an equal rate of occurrence across the nodes in the category. The unusually high alert ratio of this node category of 38.22% further emphasizes the fact that the alerts in this node category are unusual. We also note that contrary to the results presented in previous work, an entropy based approach can still show high performance with non-functional groupings of nodes. This can be seen from the results of the NodeinfoPlus_Uniq framework on the *Other* node categories in Fig. 1. It is possible that the extraction of messages types after the *Other* node categories have been grouped together may have contributed to this success.

VI. Conclusion and Future Work

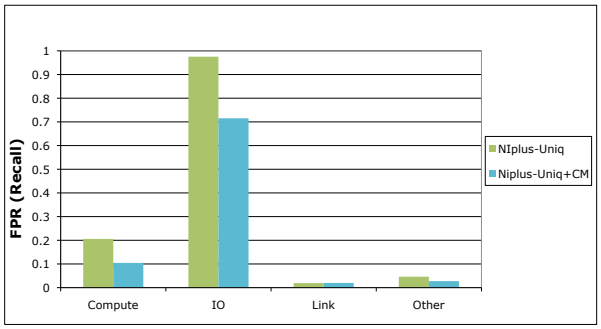
In this work, we present our findings on the evaluation of three variants of the Nodeinfo alert detection mechanism using new datasets not used in previous evaluations. We also propose the introduction of a *Cluster Membership Anomaly score* to the framework to reduce the false positive rates required to detect all alerts.

Based on our evaluations we were able to determine the following. Firstly, we determined that the NodeinfoPlus_Uniq is generally more robust than the NodeinfoPlus_Max framework. Secondly, we determined that the use of the $\mathbf{Z}$ matrix in NodeinfoPlus, as opposed to the $\mathbf{Y}$ matrix used in Nodeinfo limits the ability for NodeinfoPlus to detect alerts, which have bursty signatures. Thirdly, we also determined that the use of the proposed cluster membership anomaly score reduces the false positive rate required for the detection of 100% of all alerts. Lastly, we determined that an entropy based approach can show good performance when a non-functional grouping of nodes is analyzed.

While the Nodeinfo framework may seem better when it comes to detecting *bursty* alerts, it is not as generalized as NodeinfoPlus_Uniq. Thus, the signatures that faults introduce into system logs are so varied that no one alert detection mechanism can detect them all. Based on our results, we suggest that the future direction of alert detection mechanisms in system logs should consist of not one (probably complex) mechanism, which is able to detect all types of alerts, but several (probably simple) detection mechanisms, which are specialized toward detecting certain alert signatures.

Future work will involve the analysis of the clusters formed by the purposed clustering technique. Perhaps a more sophisticated approach to clustering of nodehours could improve the results, too.

While we test our framework using data from HPCs, it is possible for the proposed framework to be generalized for use on any system log, with a little modification if necessary.



(a) BGL

Fig. 3. This figure shows the false positive rates at 100% Recall for the BGL node categories. comparing the NodeinfoPlus framework based on Eq. 4. Results titled *NIplus-Uniq* and *NIplus-Uniq+CM* are for false positive rates evaluated before and after boosting using the cluster membership anomaly score respectively.

Acknowledgements

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

References

[1] J. Stearley, "Towards Informatic Analysis of Syslogs," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, 2004, pp. 309–318.

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

[3] 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.

[4] 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.

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

[6] 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.

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

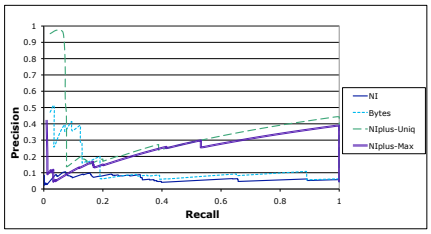
[8] A. Makanju, N. Zincir-Heywood, and E. E. Milios, "Message type extraction based alert detection in system logs," Dalhousie University, Tech. Rep. CS-2009-08, March 2009.

[9] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "An Evaluation of Entropy Based Approaches to Alert Detection in High Performance Cluster Logs," in *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST)*, September 2010.

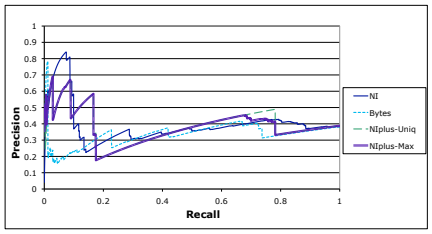
[10] J. E. Prewett, "Analyzing Cluster Log Files using Logsurfer," in *Proceedings of the 4th Annual Conference on Linux Clusters*, 2003.

[11] 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.

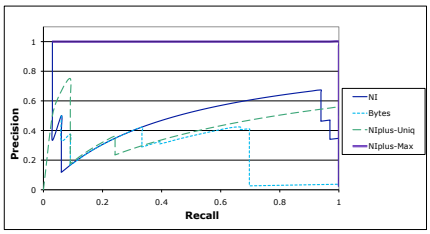
[12] S. Ma and J. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," in *Proceedings of the 16th International Conference on Data Engineering*, 2000, pp. 205–214.



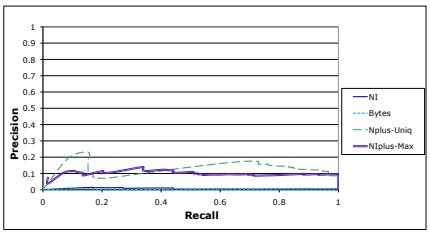
(a) BGL-Compute



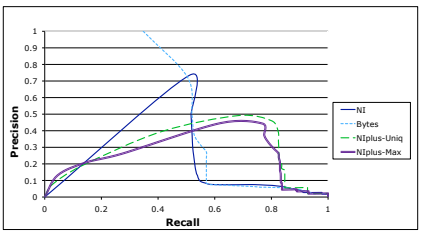
(b) BGL-IO



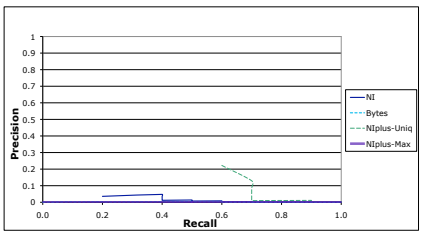
(c) BGL-Link



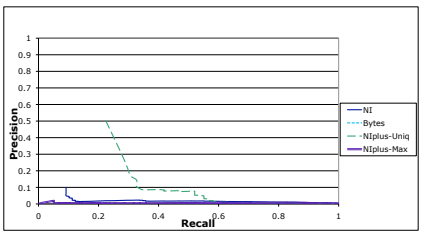
(d) BGL-Other



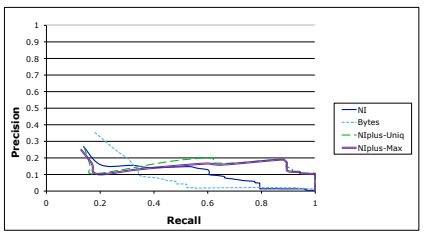
(e) Liberty-Compute



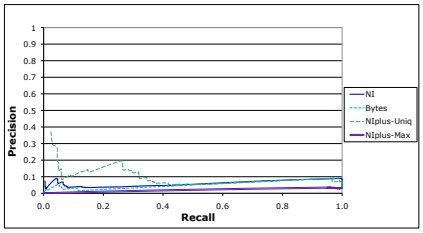
(f) Liberty-Admin



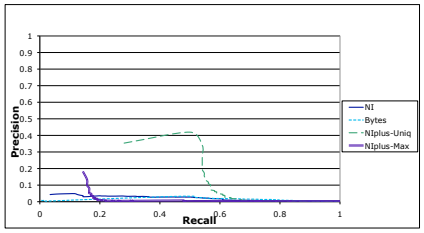
(g) Liberty-Other



(h) Spirit-Compute



(i) Spirit-Admin



(j) Spirit-Other

Fig. 1. This figure shows the Precision-Recall plots for all node categories comparing the various versions of the Nodeinfo framework and alert detection using *Bytes*. Results titled *NI* are for the original Nodeinfo framework, while results titled *NIplus-Uniq* and *NIplus-Max* are for the NodeinfoPlus frameworks based on Eq. 4 and Eq. 5 respectively.

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

[14] 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.

[15] 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.

[16] 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.

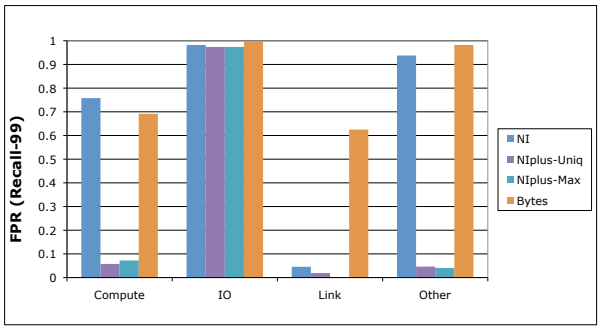
[17] R. Vaarandi, "Mining Event Logs with SLCT and Loghound," in *Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium*, April 2008, pp. 1071–1074.

[18] 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.

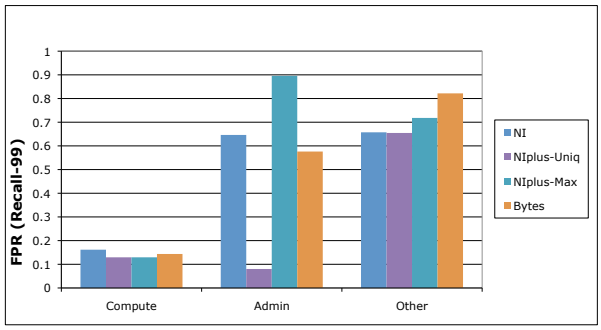
[19] "Usenix - the computer failure data repository," Last Accessed June 2009. [Online]. Available: http://cfdr.usenix.org/data.html

[20] 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.

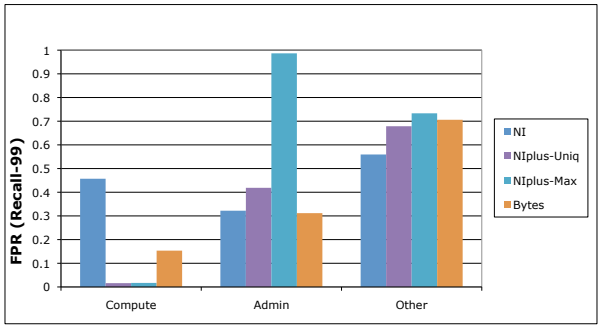
[21] C. J. V. RIJSBERGEN, *Information Retrieval*, 2nd, Ed. Butterworths, London, 1979.



(a) BGL



(b) Liberty



(c) Spirit

Fig. 2. This figure shows the false positive rates at 99% Recall for all node categories comparing the various versions of the Nodeinfo framework and alert detection using *Bytes*. Results titled *NI* are for the original Nodeinfo framework, while results titled *NIplus-Uniq* and *NIplus-Max* are for the NodeinfoPlus frameworks based on Eq. 4 and Eq. 5 respectively.

[22] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, pp. 1–47, March 2002. [Online]. Available: http://doi.acm.org/10.1145/505282.505283

[23] 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