Message Type Extraction Based Alert Detection in System Logs

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

October 29, 2009

Abstract

The task of alert detection in event logs, i.e. determining which events in the event log require action from an administrator, is very important in preventing or recovering from downtime events. The ability to do this automatically and accurately provides significant savings in time and cost of downtime events. In this work we combine message type extraction based alert detection with the entropy based approach of the Nodeinfo algorithm, which is in production use at Sandia National Laboratories, to significantly improve its performance. We show that with Message Type Indexing (MTI) and some modifications to the Nodeinfo framework, we can achieve an ∼99% reduction in the computational effort required for Nodeinfo and an F-Measure score of up to 100% in the identification of regions of the event log which contain alerts. Our work demonstrates a practical application of employing MTI on a real world data set using an alert detection framework that is currently in production use in a major government run national laboratory.

1 Introduction

Unscheduled downtime events are an unfortunate reality of any computer system in production. These downtime events come at a huge cost, both in personnel time required to troubleshoot but also in lost man hours if the downtime leads to a disruption of workflow. The ability to recover quickly when such events occur is therefore a major preoccupation of systems administrators. In such events, one of the major sources of information for system administrators are event logs. Event logs are a feature of most computer systems.

While event logs present a rich source of information, most systems administrators find them cumbersome to work with due to their typical large size and

complexity. Not all events in an event log are symptomatic of a failure. We refer events that are symptomatic of failure or require the attention of administrators as *alerts*. Unless administrators know exactly what they are looking for, locating alerts in an event log can be cumbersome when done manually, due to the large amount of data and the unstructured messages, which could be misinterpreted, in the event logs. The task of *alert detection* would therefore benefit from some amount of automation.

We can define the task of *alert detection* as the task of automatically finding those events, which are alerts, or regions in an event log which contain alerts[1]. *Nodeinfo* [1, 2] is one such alert detection algorithm. Unlike previous automated approaches to anomaly detection in event logs [3] [4], Nodeinfo has been shown to work with an acceptable false positive rate (FPR) of 0.05%. Utilizing the information entropy of message *terms* (a concatenation of a word and its token position within a message), Nodeinfo is able to detect the portions (*Nodehours*) of an event log, which are most likely to contain *alerts*.

Another concept closely related to the context of encoding of word and position pairs in log files, is the concept of *Message Types*. The work of Vaarandi was probably the first to show the importance of these word and position pairs, which we refer to as terms, in the task of automatically finding message types [5, 6]. Message types are semantic groupings of event messages which 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 for further automatic analysis. Unfortunately these message types are not always known *a priori*. Recent work using IPLOM (Iterative Partitioning Log Mining) [7], has demonstrated an effective way of finding these message types automatically.

In this work we attempt to integrate the concept of message types into the term based information entropy scoring mechanism of Nodeinfo. Using message types produced by IPLOM, we transform the messages in the BlueGene/L (BGL) dataset before they are input to Nodeinfo. The BGL dataset is one of five supercomputer logs, which have recently been made public [8]. By transforming the messages based on their message types, we aim to achieve:

- A reduction in computational complexity. Nodeinfo calculations will no longer need to be carried out using every term in the data.
- An improvement in accuracy by helping to mitigate undue contribution of message types with a large number of tokens to a Nodeinfo score.

We also propose modifications to the original Nodeinfo framework to improve the accuracy of alert detection and resolve an issue that we discovered with the score assignment mechanism of the original Nodeinfo framework. It is important to state that some of these changes made to the framework would not be possible without the use of the message type indexing (MTI) paradigm proposed in this paper. The results of our experiments show that it is possible to reduce the computational effort required for alert detection by upward of 99% using our MTI based approach. Moreover, it is possible to achieve an F-Measure accuracy

of up to 100% in the separation of the regions of the event log that contain alerts and those that do not (in one portion of the data), using our proposed modification to the Nodeinfo framework. This is a significant improvement on the 75% Precision at 50% Recall benchmark achieved using the original Nodeinfo framework [1].

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 to evaluate our proposed framework, whereas the results of those experiments are discussed in Section 4. Finally, conclusions are drawn and the future work is discussed in Section 5.

2 Background and Previous Work

2.1 Definitions

An event log E can be defined as a collection of lines of text reporting occurrences that occur on a computer system setup in temporal order. 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 decomposed into several fields, 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).

Each message (m_i) in 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 this approach is not reliable [8] 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].

2.2 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 which are indicative of previously known failure conditions [9], to visualization techniques that aid the quick detection of anomalies manually [10], to more complex schemes that use computational techniques like time periodicity of messages [11] or term weighting schemes [3].

A more recent computational approach to anomaly detection is Nodeinfo. Nodeinfo proceeds from the work of Liao [4] and Reuning[3] 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%. 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. This is most likely because message types are not always known *a priori*. Previous approaches to alert detection that have utilized message types have had to assume that they are known [12].

Our work extends Nodeinfo by using the concept of message types. We however do not assume that these message types are known, but we instead utilize message types that are extracted automatically using the IPLOM message type extraction algorithm [7]. Our work therefore demonstrates that a framework that is fully automatic from the extraction of the message types to the actual detection of the alerts in the event log is possible to achieve.

2.3 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 [2]. Therefore we define H_j^c as 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 into documents 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. To incorporate the encoding of token position into the framework, we introduce the concept of a *term*, 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 we create a *term* $w_j = t_j.j$. Now let W be the set of unique terms and let C be the count of nodes on the network. We compute a $|W| \times C$ matrix $\mathbf{X}$ where $x_{w,c}$ is the count of the number of times term w appears in messages having node c as source. We then 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. The output of Eq. 1 corresponds to *1* plus each term's Shannon information entropy [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.

The second step assigns a Nodeinfo score to each Nodehour based on the entropy of the terms contained in the Nodehour and how many times they appear. Let H be the set of all Nodehours, we define a $|W| \times |H|$ matrix $\mathbf{Y}$ 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.

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

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

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

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 come up lower in the ranking. For more details on the Nodeinfo framework please see [1, 2].

2.4 Message Types and Message Type Extraction

A basic task in automatic analysis of log files is message type or event cluster extraction [13]. The goal of message type or event cluster extraction is to group the free form messages of system log events into semantic groupings and produce textual templates (consisting of constant tokens and variable tokens) which represent all members of the cluster. Semantic groupings usually coincide with statements which 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".

When message type extraction is applied, these four log entries would form a cluster or event type and can be represented by the message type description:
*"Connection from * port *".*
The wild-cards "*" represent message variables or variable tokens. The goal of message type extraction is to find the message type representations of the message type clusters that exist in a log file.

Table 1: Log Data Statistics

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

As traditional clustering algorithms have been found to be unsuitable for message type extraction from event logs [5], specialized event log clustering algorithms have been developed. One such algorithm is IPLoM (Iterative Partitioning Log Mining) [7]. IPLoM works though a 4-Step process. In its first 3 steps IPLoM hierarchically partitions the messages in an event log into their respective clusters. In its 4th and final stage IPLoM generates templates for each of the clusters produced. Evaluation of IPLoM shows that it has the capability of improving on the results of previous message type extraction/clustering algorithms by nding not only frequent patterns in the data but also infrequent ones. . In addition, IPLOM can produce patterns, which can match patterns produced by a human more closely. 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 [14]. It is this set of automatically produced message types that we utilize in this work.

3 Methodology

3.1 BlueGene/L dataset

The BGL dataset utilized in our work is one of several supercomputer datasets [8] recently made available in the USENIX Computer Failure Data Repository [15]. The architecture of the BlueGene/L supercomputer on which the data is collected is detailed in [16]. In our recent work, we extracted message types automatically from these logs using IPLOM [14] . These automatically extracted message types were shown to achieve 91% F-Measure accuracy based on micro-average classification accuracy. It is these automatically generated message types that we utilize in transforming the message fields of the BGL data in this work. The characteristics of the BGL log are described in Table 1.

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, we separated the messages in the BGL data based on the functional roles of the nodes that produced them, leading to four categories i.e. *Compute, IO, Link and Other* categories. 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 2. We note that the *500,000* messages

Table 2: 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

recorded for the *Compute* nodes do not constitute all the messages generated by *Compute* nodes in the original data but just the first *500,000* messages. The actual number of messages from the *Compute* nodes is *4,153,008*. The sampling of *500,000* messages in this work was due to computational limitations of the original Nodeinfo algorithm.

3.2 Message Transformation Techniques

Using the message type templates extracted from the BGL event log by IPLoM, we transformed the messages in the event data so that we have a more concise representation. In the calculation of term based entropy 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 we could get equally interesting results if we are able to transform the messages such that we end up with fewer terms, which means less computation, while still maintaining the distribution of the terms across the nodes. In this work, this is what we aim to achieve with our message transformation techniques (MTT) on the BGL data using the message types extracted by IPLOM.

Our first MTT is a phrasal message transformation technique, see Fig. 1. It breaks up the message using the position of the variable (wild-card) tokens in the message type template as delimiters. Each group of constant tokens is then treated as one term and replaced with a unique term throughout the log data. This MTT was designed to transform the message with minimal disruption to its original format. We call this technique MTT-1 in the rest of this paper. The second MTT transforms a message by first representing it using a unique term that represents its message type. It then appends its variable tokens to the transformation in the same order they appeared in the original message. The encoding of token positions is very important to the Nodeinfo framework, hence the importance of maintaining the order of the variable tokens in the original message format. We call this technique MTT-2 in the rest of the paper. The third MTT makes the most drastic changes to the data. It simply replaces a message with a token representing its message type and ignores its variables completely, see Fig. 3. Our intuition here is that variable tokens would perhaps in most cases not add much value to the task of identifying alerts, the message types being more important. This MTT was designed to investigate this hypothesis. We call this MTT-3 in the rest of this paper.

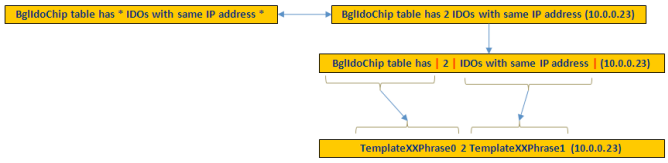


Figure 1: **Phrasal Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. Each phrase consisting of constant tokens is replaced with unique token. 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.

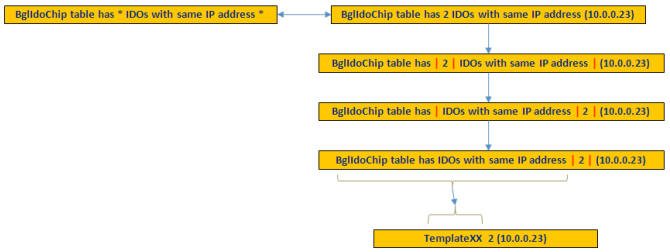


Figure 2: **Message Type Transformation with variables:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. In the final box on the right XX represents an ordinal number assigned to the message type and will thus change for different message types.

3.3 Modifications to Nodeinfo framework

In this section, we describe the modifications we made to the Nodeinfo framework to ameliorate the following problem we discovered with the framework and to improve the accuracy of the separation between the alert Nodehours and non-alert Nodehours. To understand the problem, consider a Nodehour that contains only one message. Every term in such a Nodehour will occur only once. Irrespective of the alert category of such a message, Eq. 3 will assign a Nodeinfo score of *0* to this Nodehour. This means that an alert Nodehour with only one message in it will be ranked low in a list of Nodehours sorted according to their Nodeinfo scores. We therefore made modifications to the Nodeinfo framework to address this problem.

Our modifications were only made to the step that assigns a Nodeinfo score to each Nodehour, i.e. Eq. 3. We modify Eq. 3 by removing the $\log_2$ component, giving us a new equation for assigning Nodeinfo scores to Nodehours, i.e. Eq. 4. Its is pertinent to mention that without the use of MTT, removing the $\log_2$ component would not make sense. The $\log_2$ component of Eq. 3 was

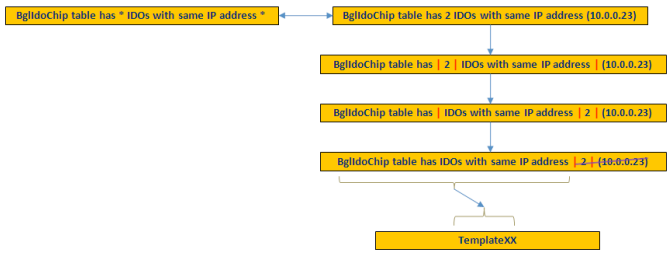


Figure 3: **Full Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. This method is identical to the message type transformation with variables except that the variable values are discarded in the final transformation. 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.

likely introduced by its designers to ameliorate a situation where a Nodehour consisting of a lot of low-entropy terms gets a high Nodeinfo score. However, such a problem is avoided when MTT is employed, since MTT reduces the number of terms in each message. Therefore, the alternative solution to this problem using MTTs, allows us to remove the $\log_2$ component.

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

With MTT (especially MTT-3) we transform the approach from a token based indexing one to an approach that is based on message type indexing (MTI). As 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 and not the number of times the message type appears in a Nodehour. With this intuition in mind, we define a new matrix $\mathbf{Z}$ analogous to matrix $\mathbf{Y}$ in the original framework, where $z_{w,j}^c$ is *1* when term w appears in Nodehour H_j^c and *0* otherwise. Matrix $\mathbf{Z}$ effectively only records unique occurrences of terms in the event data. We now define a new equation for assigning a Nodeinfo score i.e. Eq. 5.

$$NodeInfo(H_j^c) = \sqrt{\sum_{w=1}^{|W|}(g_w * z_{w,j}^c)^2} \tag{5}$$

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 with MTT, we can say that a Nodehour is only as *odd* as the *oddest* message type it contains, irrespective of the other message types it may contain. With this intuition, we define a new vector I_j^c for each Nodehour H_j^c, where $I_j^c[i]$ is

equal to $g_i * z_{i,c,j}$. We can now assign a Nodeinfo score to Nodehour using Eq. 6, effectively the highest entropy value of the terms reported during the Nodehour.

$$NodeInfo(H_j^c) = max_i(I_j^c[i]) \tag{6}$$

3.4 Experiments

Our experiments are twofold. In our first set of experiments we evaluate the effect of our message type transformation techniques on the accuracy of separation between alert Nodehours and normal Nodehours. An alert Nodehour is a Nodehour that contains at least one alert. In this set of experiments the original Nodeinfo framework [2] was used without modifications, as outlined in Fig. 4. To provide a baseline for performance, we also repeat this experiment without using IPLOM or MTTs.. In this case, achieving equal or better performance with the MTTs would be considered a success as we can make a claim to achieving this with less computational effort.

Figure 4: Process flow for first batch of experiments. The baseline for performance had the raw data input into Nodeinfo without preprocessing using IPLOM and MTTs.

In our second set of experiments, we evaluate the effect of our modifications (detailed in Section 3.3) to the original Nodeinfo framework. So our process flow changes to what is shown in Fig. 5. The baseline for performance in this case are the results from the first round of experiments where IPLOM and the MTTs are used along with the original Nodeinfo framework. Achieving better performance than with the original framework is imperative for success in this case. In both sets of 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 and F-Measure results using Eqs. 7, 8 and 9 respectively.

Figure 5: Process flow for second batch of experiments. The baseline for performance are the results from the first round of experiments where no modifications were made to Nodeinfo.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

Table 3: 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-1	10,397	0.85	7,205	2.52	560	6.51	11,634	1.36	2.81
MTT-2	10,368	1.13	7,176	2.91	554	7.51	11,562	1.98	3.38
MTT-3	86	99.18	48	99.35	12	98.00	92	99.22	98.94

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{9}$$

To produce the Precision-Recall graphs, which we present in Fig. 6 and Fig. 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 line for each experiment in the Precision-Recall plots.

4 Results

The primary motivation for the design of the message type transformations is to reduce the number of unique terms contained in the data. This reduction in number of terms in turn leads to a similar reduction in the computational effort required to compute the Nodeinfo scores. So we first report on the how these message type transformations affect the number of unique terms in the data. The results show an average percentage reduction in the number of terms of ∼3%, ∼3% and ∼99% for MTT-1, MTT-2 and MTT-3 respectively, see Table. 3. While there is reduction in the number of terms for all 3 types of transformations, only MTT-3 shows a reduction that would be considered significant. We believe that the reasons for the lack of significant reduction with MTT-1 and MTT-2 stems from the fact that they retain the variable tokens. The majority of tokens in log event data occur very infrequently and there is a strong correlation between these infrequent tokens and the variable tokens in message type templates. This finding agrees with properties of event data outlined in [5]. MTT-1 and MTT-2 only reduce the number of frequent tokens, which correlate with the constant tokens, hence we do not see a significant decrease in the total number of tokens. We note however that we set a minimum support threshold of 3 for the terms used with the original data and MTT-1 and MTT-2 i.e. we eliminated all terms that appeared less than 3 times throughout the data. Most tokens which occur infrequently in log data are usually variable tokens and are not likely to add much to the alert detection process. We however did not do this for MTT-3 since we had already explicitly discarded all variable tokens.

The results of running the original Nodeinfo framework on the original and transformed data is provided in Fig. 6. While the results differ for each set of

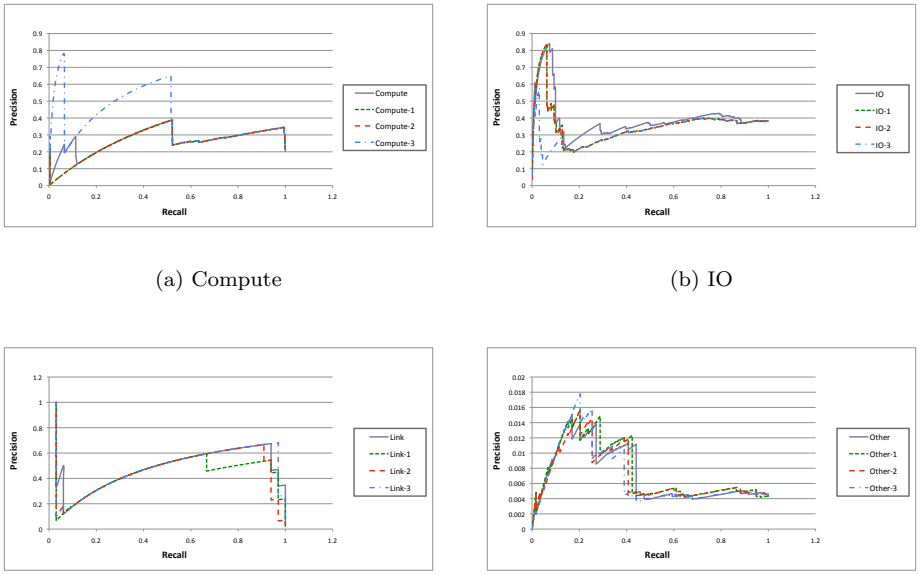
nodes, our overall conclusion from the results is that it is possible to achieve a similar if not better result by using the proposed message type transformations.

In the case of the *Compute* nodes, the results show exactly the same performance for all the data representations for Recall rates above ∼50%. However MTT-3 seems to outperform the other representations at Recall rates lower than ∼50%, achieving ∼50% Recall at ∼65% Precision. This is in spite of the ∼99% reduction in its term list. This result advances the notion of a correlation between message types and alert types. However, with the *IO* nodes we do not notice this trend, we can only state here that there is no visible significant difference in the results for each data representation, we still count this a success as we achieved similar results with less computational effort.

For the *Other* nodes we also notice no visible significant difference between the results, Fig. 6. We do however note that results achieved here are of no use as none of the data representations achieves a Precision rate higher than 2%. This observation reaffirms the *Similar Nodes, Similar Work, Similar Logs* statement highlighted in [1]. On the other hand, the *Link* nodes also show interesting results. As expected, we see no significant difference between the results for each data representation. We however noticed that a majority of the Nodehours in the *Link* data group contained only *1* message, hence these Nodehours ended up with a Nodeinfo score of *0*. We theorize that the performance on this data group could have been significantly better if the singularity problem discussed in Section 3.3 is solved, because this data group is severely affected by the singularity problem. This result highlights the need for modifications to be made to the original Nodeinfo framework [2].

Since our previous set of results show that we can achieve similar or better performance using all message transformation techniques, we only test our modified Nodeinfo assignment equations using MTT-3 as it is the only one that showed a significant decrease in the number of tokens, using the results for the original data. This allows us to achieve similar if not better performance with less computational cost. So in this new set of experiments we only employed MTT-3 transformation and compared it to the above results, the new results are highlighted in Fig. 7. For the *Other* nodes as expected, we again got less than desirable results for all frameworks. The *IO* nodes however show no change in results from the baseline even with the changes applied to the framework. The reasons behind this will be discussed later.

On the other hand, with the *Compute* and *Link* nodes, we notice visible significant changes in the results, the most significant of course being with the *Link* nodes where the results improve impressively to achieve 100% Precision and 100% Recall with the framework that uses Eq. 6. The frameworks using Eq. 3 and Eq. 4 show similar results. We believe this is due to the fact that the matrices $\mathbf{Y}$ and $\mathbf{Z}$ utilized by the frameworks, respectively, are almost the same in this scenario, as most terms occur only once in most of the Nodehours. With the *Compute* nodes the frameworks based on Eq. 5 and Eq. 6 show the best performance achieving ∼100% Recall at ∼87% Precision. The framework based on Eq. 4 however does not do so well, achieving a best case performance of ∼50% Recall at ∼55% Precision which is lower than the baseline, , Fig.7. The



(a) Compute

(b) IO

(c) Link

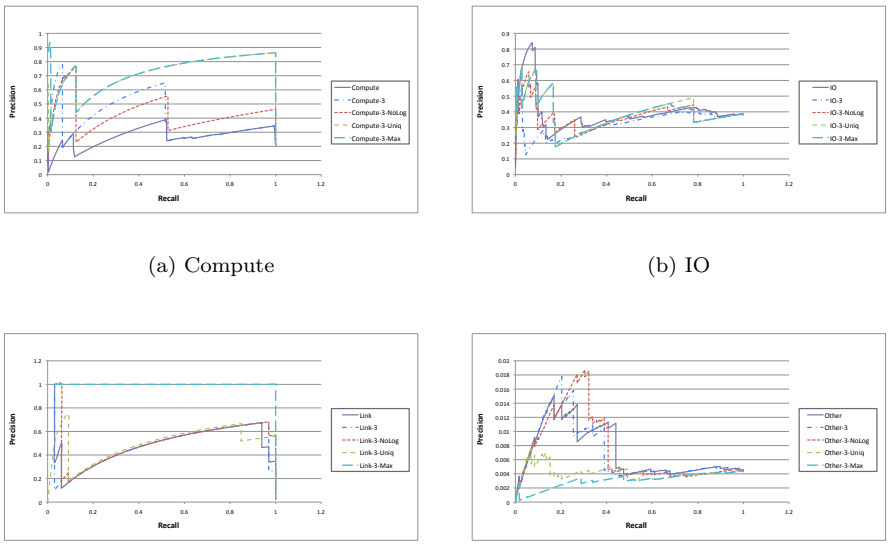
(d) Other

Figure 6: Precision-Recall plots for all node categories using the original Nodeinfo framework with different message type transformations. Results with suffix *1* are transformed using MTT-1, with suffix *2* using MTT-2 and with suffix *3* using MTT-3.

results suggest that the most promise is held by the frameworks based on Eq. 5 and Eq. 6, further tests are required to determine which is the better one.

A question that arises from the results of the second set of experiments is: Why do we not achieve similarly impressive results with the *IO* nodes as we did with the *Compute* and *Link* nodes? Our investigations reveal an interesting artifact, which may not be obvious from the data. We found that *9* out of the *17* alert categories associated with *IO* nodes (which account for ∼60% of all alerts in *IO* node data) showed a close correlation to *5* message types. These message types however all have entropy values, which are less than *0.1*. This would be considered very low and bordering on not being *anomalous*. This makes it difficult to achieve high recall rate at high precision, 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 *broadcast* to all *IO* nodes, leading to a near equal frequency of occurrence across the nodes. We believe that the latter explanation may be more likely.



(a) Compute

(b) IO

(c) Link

(d) Other

Figure 7: Precision-Recall plots for all node categories using MTT-3 and the original message format with original and modified Nodeinfo frameworks. Results with suffix *NoLog* are done using Eq. 4, with suffix *Uniq* using Eq. 5 and with suffix *Max* using Eq. 6.

5 Conclusion and Future Work

In this work we succeeded in demonstrating a practical application of using message type extraction as the basis for automatic log analysis, in this case alert detection. Using the message types extracted by the IPLOM algorithm, we modify the framework of the Nodeinfo algorithm for alert detection in system logs. Nodeinfo is an alert detection algorithm which is currently in production use at Sandia National Laboratories.

By introducing Message Type Indexing (MTI) after Message Type Transformation (MTT) based on the message types produced by IPLOM, we show that we can reduce the size of the term vector utilized in the Nodeinfo framework by up to ~99% leading to an equal reduction in computational effort. Our results also seem to suggest that variable tokens within message types do not add much context to the task of alert detection. Using the modifications a MTI based framework affords, we also show that we can achieve an F-measure score of up to 100% in the identification of regions (*Link* nodes group of the dataset) of the event log which contain alerts.

Apart from showing the improvements that our new proposed framework can provide for alert detection in system logs, this work demonstrates practically the leverage that can be achieved by building models on event log data based on the message types they contain. With the introduction of algorithms like IPLoM, automatically extracting accurate message type descriptions from log data has

become a possibility.

Future work will involve further testing of our framework on more real world datasets and investigating further the reasons for the relatively poorer performance of our framework on the *IO* nodes. For the latter investigation it may be necessary for the information in our data set to be revised to include sub-systems as possible sources of events.

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 and A. Oliner, "Bad Words: Finding Faults In Spirit's Syslogs," in *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID '08.*, 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] 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.

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

[5] R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns from Event Logs," in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, 2003, pp. 119–126.

[6] ——, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," in *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems (LNCS)*, vol. 3283, 2004, pp. 293–308.

[7] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering Event Logs Using Iterative Partitioning," in *15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2009).*, July 2009.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.