Extracting Message Types from BlueGene/L's 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

***Abstract*—In this paper we present the results on extracting message types from the BlueGene/L supercomputer logs using the IPLoM (Iterative Partitioning Log Mining) algorithm. Previous work using IPLoM indicates that IPLoM shows promise as message type extraction algorithm. We compared the results of IPLoM against manually produced message types produced on the BlueGene/L data. To provide a baseline of performance we also perform similar experiments using SLCT (Simple Log File Clustering Tool). The results show that IPLoM improves the performance of SLCT by finding the infrequent patterns in the data and is also able to achieve an F-Measure result of 91% based on micro-average classification accuracy.**

I. Introduction

Event logs consist of several independent lines of text data which contain information about activity on a system. System administrators therefore turn to them frequently as part of daily routine or when they need to detect or diagnose malfunction or misuse. However, event logs can sometimes contain such large amounts of data that manual analysis becomes cumbersome, leading some system administrators to ignore them all together or to miss vital information in them when they do analyze them manually. This fact has made the development of tools and techniques for automatically analyzing the contents of event log files an important area of research.

A basic task in automatic analysis of log files is message type or event cluster extraction [1]. Extraction of message types makes it possible to abstract the contents of event logs and facilitates further analysis and the building of computational models. Message type descriptions are the templates on which the individual messages in any event log are built. 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".

These four log entries would form a cluster or event type in the event log and can be represented by the message type description (or line format):
*"Connection from * port *"*.
The wild-cards "*" represent message variables. The goal of message type mining is to find the message type representations of the message clusters that exist in a log file.

Previous work on inferring structure from ad-hoc data [3] falls short for the task of clustering event logs for a number of reasons, the most important being the assumptions made about the data. The algorithm outlined in [3] assumes that the data content is the product of a single process and therefore the events can all be described using one, possibly very complex, format. On the other hand the assumption when clustering event logs is that the data content is the product of several independent and heterogeneous processes, therefore each format must be learnt independently. Previous works that deal specifically with log clustering include tools like Loghound and SLCT (Simple Log File Clustering Tool) [2].

The shortcomings of Loghound and SLCT are two fold. Firstly, they both focus on finding only frequent message patterns in log data but not infrequent patterns. While this might suffice most times, it may sometimes be necessary to also find infrequent patterns for analysis. Infrequent patterns may be more interesting to find in applications such as anomaly detection. Secondly comes the issue of semantics. Patterns found by Loghound and SLCT are all valid but may not necessarily make sense to a human observer. This observation becomes relevant if the patterns found will be used in a visualization tool such as LogView [4]. It is therefore important to extend the work of tools like Loghound and SLCT by designing an algorithm that will allow the discovery of infrequent patterns and also patterns that are meaningful to a human observer. To this end, an algorithm such as IPLOM (*Iterative Partitioning Log Mining*) [5] may help. IPLoM has shown promise and this work extends the testing of IPLoM to supercomputers. The aim of this work is to see how well IPLOM scales to large and probably more complex data sets like those produced by BlueGene/L [6].

In this paper we present our work done in testing IPLoM on the BlueGene/L data set, a publicly available high performance computing log data set. The BlueGene/L supercomputer is a well known HPC machine designed by IBM and located

at Lawrence Livermore National Labs (LLNL) in Livermore, CA, USA. According to the Top-500 supercomputing site BlueGene/L ranked #4 in its list of the fastest supercomputers in the world [7]. We compare our results against manually produced message types and produce similar results using SLCT to provide a baseline of performance. The results show IPLoM improves the performance of SLCT by finding the infrequent patterns and is able to achieve an F-Measure of 91% based on micro-average classification accuracy.

The rest of this paper is organized as follows: section 2 discusses previous work in event type pattern mining and automatic analysis of BlueGene/L's data. Section 3 gives details of our novel algorithm and the methodology to evaluate its performance. Section 4 describes the results whereas section 5 presents the conclusion and the future work.

II. Background and Previous Work

A. Definitions

We begin this section by first defining some of the terminology used in this paper.

- **Event Log:** A text based audit trail of events that occur within the system or application processes on a computer system.
- **Event:** An independent line of text within an event log which details a single occurrence on the system. An event typically contains not only a *message* but other other fields of information like a *Date*, *Source* and *Tag* e.g as defined in the `syslog` RFC (Request for Comment) [8]. For message type extraction we are only interested in the *message* field of the event. This is why events are sometimes referred to in literature as messages or transactions.
- **Token:** A single word delimited by white space within the *message* field of an event.
- **Event Size:** The number of individual tokens in the message field of an event.
- **Event Cluster/Message Type:** These are *message* fields of entries within an event log produced by the same print statement.
- **Cluster Description/Message Type Description/Line Format:** A textual template containing wild-cards which represents all members of an event cluster.
- **Constant Token:** A token within the *message* field of an event which is not represented by a wild-card value in its associated message type description.
- **Variable Token:** A token within the *message* field of an event which is represented by a wild-card value in its associated message type description.

B. Previous Work

Data clustering as a technique in data mining or machine learning is a process whereby entities are sorted into groups called clusters, where members of each cluster are similar to each other and dissimilar from members of other groups. Clustering can be useful in the interpretation and classification of large data-sets, which may be overwhelming to analyze manually. Clustering therefore can be a useful first step in the automatic analysis of event logs.

If each textual line in an event log is considered a data point and its individual words considered attributes, then the clustering task reduces to one in which similar log messages are grouped together. For example the log entry *Command has completed successfully* can be considered a 4-dimensional data point with the following attributes *"Command", "has", "completed", "successfully"*. However, as stated in [9], traditional clustering algorithms are not suitable for event logs for the following reasons:

1) The event lines do not have a fixed number of attributes.
2) The data point attributes i.e. the individual words or tokens on each line, are categorical. Most conventional clustering algorithms are designed for numerical attributes.
3) Traditional clustering algorithms also tend to ignore the order of attributes. In event logs the attribute order is important.

For these reasons several algorithms and techniques for automatic clustering and/or categorization of log files have been developed. Moreover, some researchers have also attempted to use techniques designed for pattern discovery in other types of textual data to the task of clustering event logs. Algorithms like SLCT [9] and Loghound [10] and IPLoM [5] are algorithms, which were designed specifically for automatically clustering log files, and discovering event formats. While Teiresias, a bio-informatics pattern discovery algorithm developed by IBM [11] was applied to the task of event log clustering [1]. Recent work which compared IPLoM, SLCT, Loghound and Teiresias [5] suggests that IPLoM is the most accurate algorithm at clustering event logs to match human output, achieving an average F-Measure result of 78% compared to the closest algorithm SLCT which achieved 10%. We therefore test the BlueGene/L data set against IPLoM and SLCT in our experiments.

The BlueGene/L data set is described in [6] is one of five supercomputer logs which have been recently made available to the research community. For this reason its logs have been used in recent works on the analysis of supercomputer logs. An Adaptive Semantic Filter algorithm was designed for failure analysis in BlueGene/L logs in [12]. While message types were not used in that study, the fact that the authors developed a semantic filter as a replacement for their previous work which used Spatio-Temporal filtering [13], [14] highlights the importance of semantic analysis of the contents of log files. Clustering log files to find message types is a form of semantic analysis as message types represent semantic groups of the messages in a log file. Time To Interrupt (TTI) estimation was carried out in [15]. This work utilized message types that were generated manually, this again highlights the importance of tools and techniques that can find these message types automatically to be developed.

III. Methodology

In this section we describe the BlueGene/L data set, give an overview of IPLoM and then we describe in detail our

TABLE I
LOG DATA STATISTICS

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

methodology for testing its performance and that of SLCT against the BlueGene/L data set.

A. BlueGene/L dataset

The BlueGene/L data set is one of several failure data sets available in the USENIX Computer Failure Data Repository (CFDR) [16]. A detailed description of the BlueGene/L computer architecture can be found in [13].

The characteristics of the BlueGene/L log are described in Table I. Each line in the log data contains an actual event from the BlueGene/L logs plus four additional fields which are added to ease parsing. The four fields represent alert category (a "-" meaning no alert category), the Unix time-stamp, date and the name of the device that generated the event. After these four fields comes the actual event as reported in the logs, the event consists of six fields, the first field represents the time-stamp, next comes the message source, the event type (the mechanism through which the event is reported), the event facility (the reporting component), the severity and the free form event description or message. In our work we are only interested in the event description or message field of the logs.

The IPLoM algorithm is designed as a log data clustering algorithm. It works by iteratively partitioning a set of log messages used as training exemplars. At each step of the partitioning process the resultant partitions come closer to containing only log messages which are produced by the same line format. At the end of the partitioning process the algorithm attempts to discover the line formats that produced the lines in each partition. These discovered partitions and line formats are the output of the algorithm.

An outline of the four steps of the algorithm is given in Fig. 2. The algorithm is designed to discover all possible line formats in the initial set of log messages. As it may be sometimes required to find only line formats that have a support that exceeds a certain threshold, the *File Support Threshold* is incorporated into the algorithm, which plays a similar role to the support threshold used in other log data clustering algorithms like SLCT and Loghound. By removing the partitions that fall below the threshold value at the end of each partitioning step, we are able to produce only message type descriptions that meet the desired support threshold at the end of the algorithm.

A more detailed description of the IPLoM algorithm can be found in [5]. We however give a summary of each step of the algorithm in the following sub-sections.

B. Step 1: Partition by event size.

The intuition for the first step of IPLoM's partitioning process is that log messages that have the same message type description are likely to have the same event size. For this reason IPLoM's first step uses the event size heuristic to partition the log messages. So IPLoM's first step is to simply group messages with the same event size before they are processed further. By partitioning our data first by event size we are taking advantage of the property of most cluster instances of having the same event size, therefore the resultant partitions of this heuristic are likely to contain the instances of the different clusters which have the same event size.

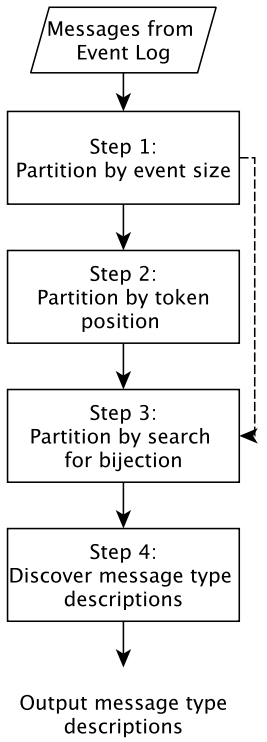

Fig. 2. Overview of IPLoM processing steps.

C. Step 2: Partition by token position.

This step of the algorithm works on the assumption that the token position with the least number of variables (unique words) is likely to contain words which are constant in that position of the message type descriptions that produced them. Our heuristic is therefore to find the token position with the least number of unique values and further split each partition using the unique values in this token position i.e. each resultant partition will contain only one of those unique values in the token position discovered.

It may work out that the token position with least number of variables actually contains words that are variables. To mitigate this problem we introduce a *partition support threshold* into the algorithm, it is essentially a threshold that controls backtracking. When IPLoM completes the partitioning of a group of messages at its second step, it checks to find those newly created partitions that have instances that fall below the *partition support threshold* in respect to the original group of messages. Those partitions with instances that fall below the threshold are regrouped into one partition.

```
- 1118394102 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.42.286586 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL guaranteed instruction cache block touch.0
- 1118394102 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.42.468969 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL guaranteed data cache block touch........1
- 1118394102 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.42.763059 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL force load/store alignment..............0
- 1118394102 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.42.943318 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL icache prefetch depth...................0
- 1118394103 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.43.150959 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL icache prefetch threshold................0
- 1118394103 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.43.351206 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL general purpose registers:
- 1118394103 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.43.526568 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 0:00000002 1:1fea8640 2:leeeeeee 3:fe9bc000
- 1118394103 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.43.816703 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 4:00439430 5:ffff96e0 6:fffbccf0 7:0f554c48
- 1118394104 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.44.018605 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 8:0f4c4108 9:0d563fc0 10:0d7679b8 11:05681130
- 1118394104 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.44.285613 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 12:0d7007b8 13:leeeeeee 14:00000015 15:fe9bc000
- 1118394104 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.44.460384 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 16:00364380 17:00440000 18:05fb0d70 19:05f6da60
- 1118394104 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.44.661736 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 20:004d96ac 21:0d6323c0 22:07bd4840 23:0567a810
- 1118394104 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.44.836451 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 24:09218840 25:00000001 26:004e0000 27:0a154900
- 1118394105 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.45.010366 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL 28:0007d000 29:0b798900 30:00067200 31:00090b40
- 1118394105 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.45.347698 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL special purpose registers:
- 1118394105 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.45.526659 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL lr:00118244 cr:42424844 xer:20000002 ctr:00000001
KERNRTSP 1118394105 2005.06.10 R10-M0-N1-C:J14-U11 2005-06-10-02.01.45.845479 R10-M0-N1-C:J14-U11 RAS KERNEL FATAL rts panic! - stopping execution
- 1118394112 2005.06.10 R16-M1-N2-C:J17-U01 2005-06-10-02.01.52.670449 R16-M1-N2-C:J17-U01 RAS KERNEL INFO 255 ddr errors(s) detected and corrected on rank 0, symbol 2, bit 4
- 1118394112 2005.06.10 R17-M0-NA-C:J13-U01 2005-06-10-02.01.52.696028 R17-M0-NA-C:J13-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 22, bit 6
- 1118394112 2005.06.10 R10-M0-N7-C:J17-U01 2005-06-10-02.01.52.713289 R10-M0-N7-C:J17-U01 RAS KERNEL INFO 30 ddr errors(s) detected and corrected on rank 0, symbol 14, bit 7
- 1118394112 2005.06.10 R17-M1-N7-C:J11-U11 2005-06-10-02.01.52.739543 R17-M1-N7-C:J11-U11 RAS KERNEL INFO 7 ddr errors(s) detected and corrected on rank 0, symbol 7, bit 2
- 1118394112 2005.06.10 R07-M0-N6-C:J11-U01 2005-06-10-02.01.52.799196 R07-M0-N6-C:J11-U01 RAS KERNEL INFO 1 ddr errors(s) detected and corrected on rank 0, symbol 18, bit 0
- 1118394112 2005.06.10 R01-M0-N7-C:J07-U01 2005-06-10-02.01.52.842376 R01-M0-N7-C:J07-U01 RAS KERNEL INFO 2 ddr errors(s) detected and corrected on rank 0, symbol 6, bit 1
- 1118394112 2005.06.10 R03-M0-NB-C:J10-U11 2005-06-10-02.01.52.867680 R03-M0-NB-C:J10-U11 RAS KERNEL INFO 2 ddr errors(s) detected and corrected on rank 0, symbol 12, bit 0
```

Fig. 1. Sample events from the BlueGene/L data set. In our work we are only interested in clustering the portion of the events after the severity field.

D. Step 3: Partition by search for bijection

The third step of IPLoM's partitioning process is the most complex. It works by searching for bijective relationships between the set of unique tokens in two token positions, which are selected using a heuristic. We live the possibility that Step-2 of the partitioning process may be skipped, so the heuristic is different for partitions that have gone through Step-2 and does that have not. For partitions coming direct from Step-1 the heuristic is to select the token positions with the least number of unique tokens, while the heuristic for partitions coming from Step-2 is to select the first two token positions with the most frequently occurring event size value greater than 1. It should be noted that partitions that are formed as result of regrouping using the *partition support threshold* at Step-2 are treated like they are coming from Step-1 and not Step-2. A bijective function is a *1-1* relation that is both injective and surjective. When a bijection exists between two elements in the sets of tokens, this usually implies that a strong relationship exists between them and log messages that have these token values in the corresponding token positions are separated into a new partition.

To avoid unnecessary partitioning at this step a *cluster goodness threshold* is introduced. Partitions coming from Step-2 which have a cluster goodness value (which is calculated as the ratio of the number of token positions with only one unique value over the event size of the partition) above the threshold are not partitioned any further.

E. Step 4: Discover message type descriptions (line formats) from each partition.

In this step of the algorithm partitioning is complete and we assume that each partition represents a cluster i.e. every log message in the partition has the same message type description. This is done by counting the number of unique tokens in each token position of a partition. If a token position has only one unique value then it is considered a constant value in the message type description, if it has more than one unique value then it is considered a variable.

F. Experiments

In order to evaluate the performance of IPLoM in finding the message types in the BlueGene/L data set, we produced

```
ciod: Missing or invalid fields on line * of node map file *
CE sym * at * mask *
dbcr0=* dbsr=* ccr0=*
idoproxy communication failure: socket closed
NFS Mount failed on * slept * seconds, retrying *
```

Fig. 3. Examples of the message type descriptions produced manually.

message type descriptions from the data manually[1], examples of the message type descriptions found manually are shown in Figure 3. We then produced message types on the same data using IPLoM algorithm and compare against the manual message types found as gold standard. To provide a baseline for performance we also compare message types produced by SLCT against the manual message types. As described in Section III-A, only the portion of the log after the severity field were used in both manual and automatic analysis.

Our performance evaluation metrics are Recall, Precision and F-Measure which are described by Eqs. (1), (2) and (3) respectively. The terms TP, FP and FN in the equations are the number of True Positives, False Positives and False Negatives respectively. Their values are derived by comparing the set of manually produced message type descriptions to the set of retrieved formats produced by each algorithm. In our evaluation a message type description is still considered an FP even if matches a manually produced message type description to some degree, the match has to be exact for it to be considered a TP.

For completeness we evaluated our Precision, Recall and F-Measure values using three different methods. In the first two methods we evaluated the results of the algorithms as a classification problem. Using the manually produced event types as classes we evaluated how effectively the automatically produced classification matched the manually produced labels. It should be noted that using the manually produced message types as classes leads to a scenario where none of the class members will be mis-classified if an automatic algorithm gets the message type description right and where the class members will all be mis-classified if the automatic algorithm does not get the message type description. This classification evaluation produced Micro-average and Macro-

[1]Full list of manual descriptions are available for download from http://torch.cs.dal.ca/~makanju/iplom

average results. These results are referred to as *"Micro"* and *"Macro"* in the results section. In the third method the manually produced message type descriptions are compared against the automatically produced message type descriptions. This evaluation method is called "IR" in the results section. We however believe that what we called the "IR" method evaluation satisfies our goals better as it tests the goodness of the clusters produced. The next section gives more details about the results of our experiments.

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3)$$

IV. Results

In our experiments we tested SLCT and IPLoM on the BlueGene/L data set. The parameter values used in running the algorithms are provided in Tables II and III. SLCT was chosen as a baseline for performance as it showed the best performance behind IPLoM in previous work [5] and the support threshold provided is based on an absolute line count and not a percentage value. The *file support threshold* for IPLoM was set to its lowest possible value which is its default value i.e. a line count of 1. IPLoM's other values were set as specified in previous work [5], except in the case of the *partition support threshold* which was set to 0.1%. We suggest as good practice to always set the partition support threshold so it can mitigate errors when they occur. Its value was set very low in this case due to the large number of exemplars in our data $4.7m$, this implies that the value of the *partition support threshold* should be set such that it scales to size of the data set but should normally not exceed 5% .

TABLE II
SLCT Parameters

Parameter	Value
Support Threshold (-s) (Absolute)	20
Seed (-i)	5

TABLE III
IPLoM Parameters

Parameter	Value
File Support Threshold (Absolute)	1
Partition Support Threshold	0.001
Lower Bound	0.1
Upper Bound	0.9
Cluster Goodness Threshold	0.34

The number of message types found by algorithms compared to those found manually are shown in Table IV. Already we can see an advantage of IPLoM, while it does not produce the exact number of message types produced manually, the number of types it produces falls within a relatively close range to those produced manually. The Precision, Recall and F-Measure results of comparing these message types listed in Table V. We see IPLoM producing better F-Measure scores for all our evaluation methods.

TABLE IV
Message Types Produced on BlueGene/L Data

Manual	SLCT	IPLoM
394	33,343	332

TABLE V
Cluster Description performance of algorithms

PRECISION PERFORMANCE		
	SLCT	IPLoM
Micro	0.24	0.91
Macro	0.29	0.50
IR	0.00	0.60
RECALL PERFORMANCE		
	SLCT	IPLoM
Micro	0.24	0.91
Macro	0.29	0.50
IR	0.29	0.50
F-MEASURE PERFORMANCE		
	SLCT	IPLoM
Micro	0.24	0.91
Macro	0.29	0.50
IR	0.01	0.54

Due to the problem of insufficient information in data as highlighted in Figure 4, it is pertinent to highlight that results in Table V do not show the complete picture of IPLoM's performance, as in certain cases IPLoM is able to produce the right partitioning of a message type but is not able to come up with the right cluster description. Fig. 5 show the F-Measure results for all our evaluation methods.

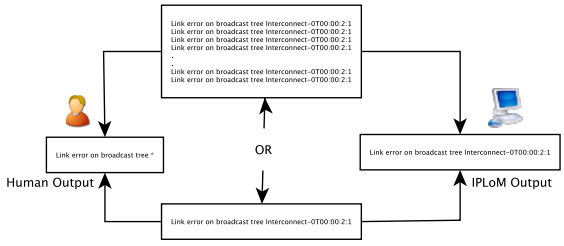

Fig. 4. Example: Insufficient Information in Data. When a message type contains exact exemplars or only one exemplar in the data, it is possible for IPLoM to get the right partitioning of the data but end up with the wrong message type description.

We would also like to highlight the fact that while in some cases the message types and partitions produced by IPLoM might not match those produced manually they are still meaningful clusters. Consider this message type produced by IPLoM on the data:

```
"ciod: Error reading message prefix on CioStream socket to
* * * * *"
```

This message type did not match any of those produced manually, however looking at the manual types we discover two types which are similar:

```
"ciod: Error reading message prefix on CioStream socket to
* Link has been severed"
```

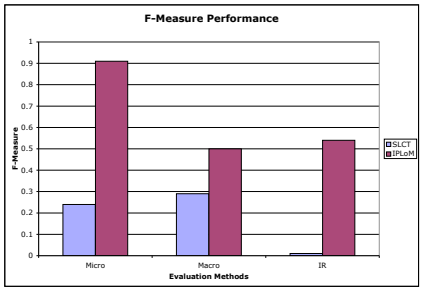

Fig. 5. Comparing Algorithm F-Measure performance across evaluation methods.

```
"ciod: Error reading message prefix on CioStream socket to
 * Connection reset by peer"
```

It is clear that IPLoM produced a message type that encompasses these two manually produced message types. While it does not match them exactly, it still forms a meaningful cluster at a higher level of abstraction compared to the manual types.

V. Conclusion and Future Work

Due to the size and complexity of sources of information used by system administrators, it has become imperative to find ways to manage these sources of information automatically. Application logs are one of such sources of information.

In this work, we present a novel message type extraction and partitioning algorithm, IPLoM on the BlueGene/L data set. We validate our results by comparing them to message types produced manually and produce similar results using SLCT to provide a baseline for performance. The results show IPLoM improves the performance of SLCT by not just finding the frequent patterns but also by finding the infrequent patterns, this fact is highlighted by the fact that it achieved an F-Measure of 91% based on Micro-average classification accuracy, improving on the performance of SLCT which had 24% accuracy. From the results we ascertain that IPLoM produces cluster descriptions that can match human output very closely and can scale to large and complex data sets like those produced by BlueGene/L.

Message types are fundamental units in any application log file. Determining what message types can be produced by an application accurately and efficiently is therefore a fundamental step in the automatic analysis of log files. Message types, once determined, provide groups for categorizing and summarizing log data, which simplifies further processing steps like visualization or mathematical modeling. Specific case studies on how message types can be applied to misuse detection and flow analysis can be found in [2].

Future work with IPLoM will involve using the information derived from the results of IPLoM in other automatic log analysis tasks and optimization of its parameters to improve its accuracy.

Acknowledgements

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

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] 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.

[3] K. Fisher, D. Walker, K. Q. Zhu, and P. White, "From dirt to shovels: Fully automatic tool generation from ad hoc data," in *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York, NY, USA: ACM, 2008, pp. 421–434.

[4] 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.

[5] 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.

[6] 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.

[7] TOP500.Org, "Top500 supercomputing sites," Published to the web. http://www.top500.org/. Last Accessed June 2009. [Online]. Available: http://www.top500.org/

[8] C. Lonvick, "The BSD Syslog Protocol," RFC3164, August 2001.

[9] 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.

[10] ——, "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.

[11] I. Rigoutsos and A. Floratos, "Combinatorial Pattern Discovery in Biologcal Sequences: The Teiresias Algorithm," in *BioInformatics*, vol. 14. Oxford University Press, 1998, pp. 55–67.

[12] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "An adaptive semantic filter for blue gene/l failure log analysis," in *IEEE International Parallel and Distributed Processing Symposium.*, March 2007, pp. 1–8.

[13] 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.

[14] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "BlueGene/L Failure Analysis and Prediction Models," in *International Conference on Dependable Systems and Networks.*, Philadelphia, PA., June 2006, pp. 425–434.

[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] "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