Event Log Mining Tool for Large Scale HPC Systems

Ana Gainaru[1,3], Franck Cappello[1,2], Stefan Trausan-Matu[3], and Bill Kramer[1]

[1] University of Illinois at Urbana-Champaign, IL USA
[2] INRIA, France
[3] University Politehnica of Bucharest, Romania

Abstract. Event log files are the most common source of information for the characterization of events in large scale systems. However the large size of these files makes the task of manual analysing log messages to be difficult and error prone. This is the reason why recent research has been focusing on creating algorithms for automatically analysing these log files. In this paper we present a novel methodology for extracting templates that describe event formats from large datasets presenting an intuitive and user-friendly output to system administrators. Our algorithm is able to keep up with the rapidly changing environments by adapting the clusters to the incoming stream of events. For testing our tool, we have chosen 5 log files that have different formats and that challenge different aspects in the clustering task. The experiments show that our tool outperforms all other algorithms in all tested scenarios achieving an average precision and recall of 0.9, increasing the correct number of groups by a factor of 1.5 and decreasing the number of false positives and negatives by an average factor of 4.

1 Introduction

Event logs are a rich source of information for analysing the cause of failures in cluster systems. However the size of these files has continued to increase with the ever growing size of supercomputers, making the task of analysing log files a hard and error prone process when handled manually. The current way used by system administrators for searching through the log data is pattern matching, by comparing numerical thresholds or doing regular expression matching on vast numbers of log entries looking for each pattern of interest. However by using this method, only those faults that are already previously known to the domain expert can be detected. As a consequence, data mining algorithms have recently been explored for extracting interesting information from log data without the control of a human supervisor [15, 21, 14]. However the algorithms must be adapted since it has been found that traditional clustering methods are not working well when they are applied to high dimensional data [11].

As mentioned in [2], log files will change during the course of a system's lifetime due to many reasons, from software upgrades to minor configuration changes and so it is normal to encounter novel events as time passes by. This makes it difficult for the algorithms to learn patterns or models. The learned patterns may not be applicable for a long time so all analyzing techniques must be able to detect phase shifts in behavior. Current data mining algorithms [17, 19, 10, 4, 15, 14] have difficulties in dynamically updating the groups to cope with an incoming stream of novel events.

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6852, Part I, pp. 52–64, 2011.

In this paper we present HELO (Hierarchical Event Log Organizer) a novel unsupervised clustering engine that aims to accurately mine event type patterns from log files generated by large supercomputers. Our algorithm adapts data mining techniques to cope with the structure format of log files making the process computational efficient and accurate. Existing event pattern mining tools are not taking advantage of the characteristics that log files share or are unable to classify messages in an online manner. Our algorithm requires no prior knowledge or expectations as events are defined by their existence. This is critical when dealing with leading edge systems or with environments that change from system to system.

We have made experiments in order to compare our tool with two Apriori tools (Loghound [17], SLCT [19]), two other pattern extractors (IPLOM [10], MTE [4]), and an affinity propagation technique (StrAp [23]). HELO outperforms all other algorithms providing a better precision without an overhead in the computational cost. We will show that our tool increases the corrected classified messages by a factor of 1.5 and decreases the number of false positives and false negatives by an average factor of 4.

The rest of the paper is organized as follows: Section 2 provides related work and describes other mining algorithms that will be used as a comparison for the results obtained by HELO. In section 3 we present our classifier tool, highlighting its properties and characteristics. Section 4 presents the log files used for the experiment scenarios and section 5 shows several performance results being obtained in order to validate the proposed mining tool. Finally, in section 6 we provide conclusions and present future work.

2 Related Work

Indexing the information found in log files is an important task since analysing groups of related messages can find problems better than by looking at individual events [15, 21]. For example there are many anomalies that are indicated by incomplete message sequences. In general a change in the normal behaviour of the system is usually an indicator of a problem. Extracting templates and shaping this behaviour can greatly help systems in detecting or even predicting faults [16].

There is a considerable amount of papers that deal with message clustering: some use supervised learning and some unsupervised data mining techniques. All the supervised methods need a training phase that is quite expensive since it requires manually annotated events [7, 20]. Unsupervised techniques require only a few input parameters, the rest being done automatically by the algorithm. The most used unsupervised methods for extracting information from log files are the Apriori algorithm for frequent itemsets [17, 19, 18], Latent Semantic Indexing [9], event pattern [10, 4] and k-nearest neighbours [23]. Also, there are some studies that use the source code to extract error description format [6]. However there are systems that don't give access to the code that generates log files so these algorithms can not be used regardless of the system.

HELO differs from the other event pattern mining tools for several reasons. First, there are very few methods that classify events in an online matter and most of them have a major limitation: they are unable to dynamically update the clusters for novel events. StrAp [23] is able to create new groups for the outliers. However the tool was

design to cope with numerical data, having limitations in clustering log files. HELO was implemented so that it is able to adapt the initial templates in order to cope with any changes in the incoming stream of events. Also HELO uses an efficient splitting process that considers different priorities for different words according to their semantic meaning. All other tools partition the dataset only according to the syntactic form of message description. This a limitation when dealing with log files since symbols used in the message description indicate different types of components or registers.

In the next paragraphs we present 5 different algorithms that mine log files and cluster events based on the similarity between their descriptions. We discuss their methodology and limitations. For computing the accuracy of all algorithms the log files were manually labelled and classified after discussions with our faculty's tech support group.

Loghound [17] and SLCT [19] are Apriori-based tools designed for automatically discovering event cluster formats from log files by considering log messages as data points and then clustering them according to different density values. The size of log files generated by todays supercomputers has continue to grow, so since the set management part in the Apriori algorithm is costly for even a smaller number of patterns [8], analysing these logs is becoming a problem to these methods.

Iterative Partitioning Log Mining (IPLoM) [10] is an algorithm for mining clusters from event logs. The authors use 2 splitting steps by token count and token position and one step where the algorithm searches for bijections between tokens from different messages and splits the data accordingly. One limitation of this algorithm is the fact that all messages in one cluster must have the same length. Also another limitation in IPLoMs analysis is the syntactic depth of the mining process.

Streaming Affinity Propagation (StrAp) [23] is a clustering algorithm that extends Affinity Propagation to data streaming. In the first step, the tool finds the number of clusters that can be formed with the offline training set and then divides it by retaining the best items that represent each cluster. In the second step the rest of the input messages are treated as stream of data and the tool achieves online clustering by making new groups for the outliers and occasionally updating the exemplars from existing groups. The algorithm was implemented to cope with numerical input data, for example the duration of execution for each job. We implemented in StrAp, a Hamming distance metric for log messages [3] that is able to work with non-numeric values.

Message Template Extractor (MTE) is a component contained by the FDiag tool [4] that adds structure to the logs by extracting the messages template. The main idea used by the authors is that tokens in the English dictionary show the same patterns in different messages from the same type and that alpha-numerical values do not have the same property. The MTE extracts two template sets, one for constants and one for variables. The tool considers variables to be alpha-numerical tokens, i.e. words that contain letters, numbers in decimal or hex and symbols. However there are some cases when variables are also English normal words. For example, this is the case of filenames.

3 Methodology

Table 1 presents different event examples that illustrate the usual form of a log message. The first part of the description is defined by header information and the rest is

Table 1. Log message examples

Header	Message
[02:32:47][c1-0c1s5n0]	Added 8 subnets and 4 addresses to DB
[02:32:51][c3-0c0s2n2]	address parity check..0
[02:32:52][c3-0c0s2n2]	address parity check..1
[02:32:57][c1-0c1s5n0]	Added 10 subnets and 8 addresses to DB
[02:34:21][c2-0c1s4n1]	data TLB error interrupt

represented by the error message. In this study we only use the message description for classifying events for all the tools. However HELO can group messages after different criteria according to how much out of the header is included in the algorithm's input.

A message description can be seen as constructed by variables and constants. Constant are words that keep their value in a group template and are represented by strings like "address" or "subnets" from our examples. These words carry crucial information since they describe the message type. Message variables like 8 from our message identify manipulated objects or states for the program. HELO is a hierarchical process that finds representations for all message types that exist in a log file by extracting constants and variables from message descriptions. The tool uses in the splitting process the fact that words formed by letters and not numbers or punctuation marks, have more chances of being constants in the final templates.

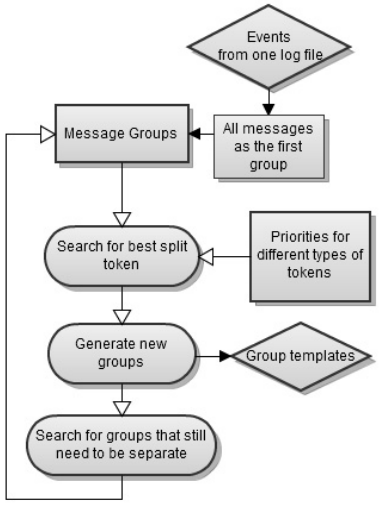
The methodology used by the tool has two different steps: an offline classification part where events found in log files are used to create the first template set by dividing them according to their description patterns and an online clustering part that classifies each new event and dynamically reshapes the previous found groups according to them.

3.1 Offline Clustering

The offline component of HELO deals with mining group patterns from log file messages. Basically the algorithm groups events considering their description in a 2 step hierarchical process. In the first step the algorithm searches for the best split column for each cluster and in the second step the clusters are divided correspondingly. A split column represents a word position in the message description that is used to divide the cluster into different groups. The methodology is described in Figure 1(a).

HELO starts with the whole unclustered log file as the first group and recursively partitions it until all groups have the cluster goodness over a specified threshold. The cluster goodness characterizes how similar all messages in one group are and is define as the percentage of common words in all events description over the average message length. This threshold can be provided by the user but is not mandatory for the execution of the tool. For all our experiments, we use the default value without trying to gain performance by tuning this parameter. This value has been set to 40% and was chosen after observing empirically that, for most of the log files, this value gives the best results.

The cluster goodness threshold is used to establish the generality of the final groups. If the threshold is low then there will be more words considered variables so the group generality increases. For example, with a lower threshold the tool generates a group



(a) Diagram

Algorithm 1 Find Split Position

Input: Partition M[] of messages
Output: Word position that can be used for the split

```
1:  for every message Maux in M do
2:      for every word_position in Maux do
3:          if Maux[word_position] is a hybrid word then
4:              Wrd = extract_hybrid(Maux[word_position])
5:          else if Maux[word_position] is a number then
6:              Wrd = number_value()
7:          else
8:              Wrd = Maux[word_position]
9:          end if
10:         if Wrd is unique for this word_position then
11:             appearance for this word = 1
12:         else
13:             increase appearance for this word with one
14:         end if
15:     end for
16: end for
17: get word_position where mean(words appearance) is
    max
18: return word_position
```

(b) Splitting process pseudocode

Fig. 1. Offline clustering methodology

with messages describing L2 cache errors, and with a higher threshold there will be two groups, one for L2 read errors and one for write.

We could not compare the real execution times obtained for all algorithms since they are implemented in different languages. We analysed only the theoretical complexity time between HELO and the other tool that obtained a good accuracy and precision for both online and offline test cases, StrAp. For the offline classification, StrAp measures the distance between any two messages; this means a time complexity of $O(N^2)$. HELO is an iterative process so uses different number of cycles for different log files. From all experiments, the clustering process is finished after 5 steps, but the worst case theoretical scenario takes $NlogN$ steps (if each message represents a unique template but their similarity is just under the threshold).

3.2 Splitting Process

In the first step of the process, the algorithm searches for the best split column for all clusters. Traditional data mining algorithms compute the information gain for each variable that could be used for the split and choose the one with the highest gain in accuracy. However, in our case, the best splitting position is the one that contains the maximum number of constant words. We consider that words with a high number of appearances on one position has more chances of being a constant, so HELO searches for the column where most unique words have a high appearance rate. This position corresponds to the column where the mean number of appearances for every unique word is maximum while still having enough words in order to be relevant to the analysed event dataset.

Table 2. Log template example

machine check interrupt (bit=0x1d): L2 dcache unit write read parity error
machine check interrupt (bit=0x10): L2 DCU read error
machine check interrupt (bit=d+): L2 * * * n+

For a better understanding we will consider the events presented in Table 1. From the fifth column the values are no longer relevant since the total number of messages that have words on that position is too small. The reason that this position is not considered is that the splitting process tries to obtain balanced groups in each step.

HELO considers that different type of words have different priorities dependent on their semantics. There are three types of considered words: English words, numeric values and hybrid tokens (words that are composed of letters, numbers and symbols of any kind). The lowest priority is for all-numeric values since the algorithm considers that these words have the most chances of becoming variables in the clusters. In the example from above, 8 and 10 from column 2 are decreasing the appearance mean for all unique words, so the split will not be done here. Hybrid values are represented by tokens like check..0 from our example. The algorithm extracts and considers only the English words incorporated in the hybrid token. For our example both check..0 and check..1 are considered as word check. If we reanalyze the example presented above, the first and the third columns could be chosen by the algorithm for the splitting position.

The creation of new groups has the exact semantic rules as the previous step. Since the numeric values have the least priority, all messages that contain any numeric value on the splitting position will be gathered in the same group. For our example, no matter which of the first three columns is chosen for the split, three groups will be generated, one with the first and the fourth message, one with the second and the third and one with the last message. For each created group, HELO computes the cluster goodness. If the value is under the chosen threshold, the group is sent to be divided again.

There are some cases where the partition step splits the groups by a column that is a good choice for the majority of the future groups but that divides some messages that should be together. After the splitting process is over and the final templates are created, HELO reanalyzes the clusters and merges group templates that are very similar. The default value for the threshold has been set to 80% since the group templates should be very similar in order to be considered one. In this last step templates are compared to one another so the time complexity is O(G^2). However, this is not a problem since the total number of templates is very small comparing to the whole dimension of the logs.

3.3 Output

When all clusters are stable, the algorithm identifies cluster description for each partition. A group template represents a line of text where variables are represented by different wildcards. HELO uses three types of wildcards: d+ represents numeric values, * represents any other single words, and n+ represents all columns of words that have a value for some of the messages and do not exist for others. In the example in Table 2 all three types of wildcards are illustrated.

The group templates describe all type of events that the system generates, in an intuitive way. The user-friendly group description generated by the tool could ease the work of system administrators to follow and understand errors from log files.

3.4 Online Clustering

The online clustering process deals with grouping messages as they are being generated by the system. Clustering tools must be able to change the group templates in order to manage novel messages that could appear. The input is given by the groups obtained with the offline process on the initial dataset. In HELO, each cluster needs to be represented by a description in the format described in the previous section and some statistics about the group.

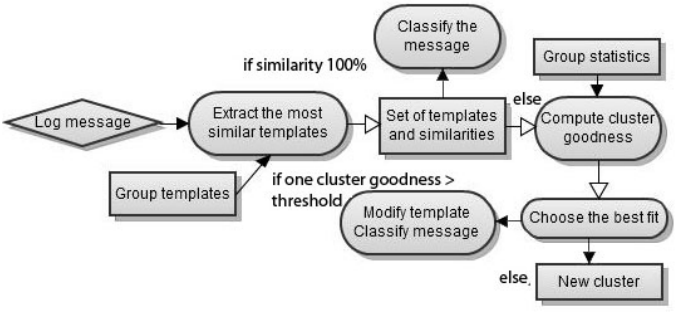


Fig. 2. Online clustering diagram

For each new message, the online component checks the description of the messages and retrieves the most appropriate group templates. If a message fits the exact description of a group (this means the group template does not need to be modified) then the search is over and we stamp the message with the template's group id. If the message does not have an exact match with any of the groups then we compute the cluster goodness for all the clusters retrieved before, after including the new message in each of them. For computing the goodness of the group if the message is inserted, we retrieve the average length of all messages in this cluster from the group statistics file. This information decides if including the new message decreases the cluster goodness under the threshold or not. If no cluster has the goodness over a specific threshold then a new group is formed. Else the group with the best cluster goodness will be chosen and the group template will be modified to accommodate the new message. The methodology for the online component of HELO is presented in figure 2.

4 Log Files

The logs we use for validating our tool are generated by five different supercomputers. Table 3 gives statistic information about the traces. All files except Mercury are downloaded from two websites: [5, 1]. Mercury logs are owned by the NCSA [13] and are not available to the public for privacy issues.

Table 3. Log data statistics

System	Messages	Time	Log type
BlueGene/L	4,747,963	6 months	event logs, login logs
Mercury	>10 million	3 months	event logs
PNNL	4,750	4 years	event logs
Cray XT4	3,170,514	3 months	event, syslog, console
LANL	433,490	9 years	cluster node outages

Table 4. Parameter values

Tool	Parameter	Value
IPLoM	File Support	0-0.5
Loghound	Support Th	0.01-0.1
SLCT	Support Th	0.01-0.1
StrAp	Offline support	$N/10^2$-$N/10^3$

We chose these 5 datasets because their diversity makes the analysis process more reliable: LANL [2] has a friendly format for all the tools under study; Cray XT4 has a very large amount of event patterns making the online classification less precise; Mercury has a very large amount of total messages, a few hundred thousand events per day, making it a good scalability testing scenario ; PNNL [22] has a large number of groups but most of them include a small amount of messages making it difficult for algorithms that take the frequencies of events into consideration to classify them; also BlueGene/L, Cray and Mercury put a lot of semantic problems.

We identified groups from each log file manually after discussions with people from the tech support group, and used this information to compute the performance of all the tools under study as an information retrieval task.

5 Results

We have made experiments in order to compare our tool with the other 5 algorithms using the traces presented in the previous section. The measures used are the classic evaluation units from information retrieval field: precision, recall and f-measure [12]. We define the main parameters used: A true positive is represented by a group template that is found by one tool and that it's also one of the annotated clusters; False negatives represent group templates that are not found by the tool when they should be and false positives are the templates that are found by the tool but are not part of the annotated clusters. Precision can be seen as a measure of exactness or fidelity and it represents the proportion of correct found templates to all the generated templates. Recall is computed as the proportion of true positives to all the messages from the manually annotated clusters and it represents a measure of completeness. F-measure is another information retrieval measuring unit that evenly weights precision and recall into a single value.

The output format is different for different tools. Even though Loghound, SLCT and IPLoM all compute the same type of groups with the one used by HELO but by only considering one type of wildcard, the rest of the tools have their own private cluster description. StrAp groups messages based on the computational distance between event description so the output is represented by an array of the same dimension as the log file with each line represented by a group id. In order to have a fair comparison between the output of all tools, we use the output format from IPLoM, but also compute the array of classification ids, like the one generated by Strap, for all other tools.

Each tool (except MTE) has different parameters that guide the output result. Parameters chosen to run each tool are the ones that give the best result and are shown in Table 4. HELO's parameters were left at their default values.

Due to the diversity of output formats we performed two types of offline comparisons. For the first sets of experiments, we computed precision and recall for the groups of messages found by all tools except StrAp. Since MTE eliminates all variables from the clusters description without placing some wildcard symbols in their place, we use for MTEs analysis a special manually computed group file that eliminates all variables from the group description. For the second sets of experiments we compute the percentage of corrected classified messages from the log file. These two scenarios shows how well the tools classify historic log files in an offline manner.

In the last set of experiments we determine the percentage of correctly classified events in an online manner. Experiments are done only for HELO and StrAp since those are the only tools that can cluster messages in a data streaming scenario. The analysis will show how well the algorithms adapt to the overall changes in the system.

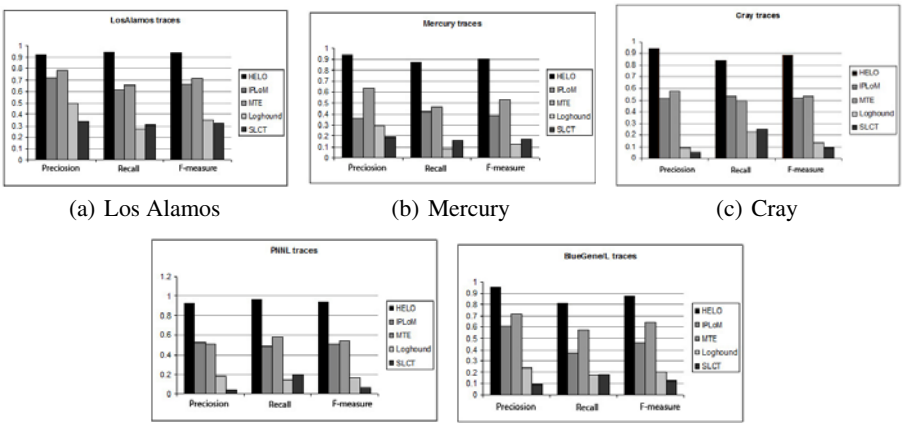
5.1 Offline

Figure 3 is showing the performance results obtained with all 5 tools for the input datasets. LosAlamos traces have the most user friendly format. Most of the generated messages are composed of English words, without having any messages represented by lists of registers. All tools obtained their best result for this dataset. Mercury and Cray generate many messages that represents the continuation of a previous message event (lists of memory locations or registers for example). This type of messages that have nothing to do with other event description, drastically decreases the performance of the mining tools. PNNL traces have the most consistency and syntactic problems for the event messages, so we can see the highest difference in performance (in both precision and recall) between our tool and the rest of the algorithm. One example of line descriptions that contain the same message but are written in different forms:
Corrective Measures SDE / DS2100 (upper) need to be replaced
Corrective Measures Upper DS2100 in need of Replacement

In general a decrease in the precision value indicates a high number of false positives. This usually means that the tool is generating more group templates than necessary, group templates that are not in the manual annotated file. One case, for example, is when the manual templates contain a lot of groups with messages of different length. For all log files, HELO keeps its precision value constant (HELO at around 0.92-0.93 with the next higher tool at around 0.67). The messages generated by all algorithms except HELO contain groups for each possible ending of the manual template. This results in an increase of the generated groups that are not manual templates.

A decrease in the recall value is influenced by a high number of false negatives. This means that there are many groups in the manual template file that are not generated by the tool. For all tools precision is mostly affected for PNNL and Mercury. The number of correct generated templates is decreasing for this log files basically because the systems produce more words with semantic problems than the rest.

The second sets of experiments computes the percentage of corrected classified messages from the log file. Figure 4 shows all results for all the analysed tools and for



(a) Los Alamos (b) Mercury (c) Cray

(d) PNNL (e) Blue Gene/L

Fig. 3. Comparing performance of HELO with the other 5 tools

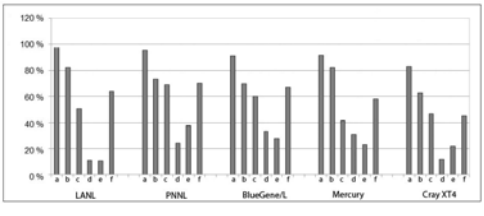


Fig. 4. Corrected classified messages a) HELO b) StrAp c) IPLoM d) Loghound e) SLCT f) MTE

all log files. The differences in results are due to the fact that log messages are not distributed uniformly over the group of messages. As expected LosAlamosl classifies messages with a higher confidence. On the other hand, the worst classification is obtained for Cray. The results for Cray can also be explained by the unfriendly form of messages generated by this supercomputer. The following message shows an example of an event description from which is hard to extract the relevant English word: $< ffffffff834c270 > : ptlrpc : lustre\_conn\_cnt + 80$

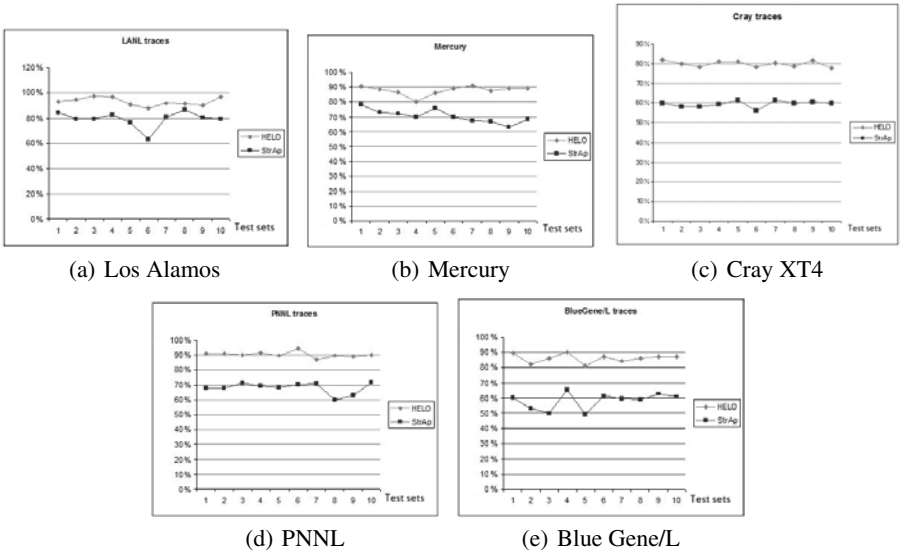
5.2 Online

In the last set of experiments, we compare the results of HELOs online component with the ones obtained by StrAp. For this set of experiments we use the 10-fold cross validation. We divide each log into 10 equal sets and then use one part for a training process of offline classification and the rest of 9 sets for online clustering using the groups found as input. We use the same method 10 times switching each time the set for the training phase. The same manually annotated files as for the offline process are used and we compute for both algorithms the number of corrected classified messages. Figure 5 shows results for both tools for all ten training sets and for each log files. In most of the cases, the two graphs follow the same curves. The different values for the

percentage of correct classified messages by one tool are given by the characteristics of events from each training set. If the training set has many new and different events from the ones found in the training set, it is likely that the value will decrease and if the training set contains all events that are in other sets than the tool will obtain the best classification, very close to the clustering obtained by the offline component.

In general, the performance follows the shape of the offline one. The shifts in the two graphs can be explained by the different methodology used by the tools. If the training set has a lot of semantic problems the distance between the two graphs will be higher. On the other hand, StrAp regroups the clusters when the number of messages that do not belong to existing clusters exceeds a threshold so StrAps performance will increase in the case of many clustered messages with different lengths.

The overall values are lower than for the offline components because usually the online classification algorithms focus on finding the best local solution for each message and not the overall best clustering result. However the results are still very good.



(a) Los Alamos (b) Mercury (c) Cray XT4

(d) PNNL (e) Blue Gene/L

Fig. 5. Comparing performance of the online component of HELO with StrAp

6 Conclusion and Future Work

In this paper we introduced HELO, a message pattern mining tool for log files generated by large scale supercomputers. We developed two components: 1) an offline clustering process that finds group templates with a high precision for events gathered in log files for long periods of time and 2) an online classification algorithm that groups new events adapting the templates to the changes in the underlying distribution. Current approaches like pattern matching are no longer efficient due to the shear mass of data to go into further analysis such as correlative analysis and forecasting.

We tested HELO against the performance of other tools used for the same task, that are using different methods. In our experiments, we used five different logs generated by different systems that have been prior manually annotated. Results clearly show that HELO outperforms other algorithms for all offline and online tasks having a precision and recall average of over 0.9 without having an overhead in the execution time.

The extracted group templates are used to describe events generated by the supercomputers and to future characterize the overall behaviour of fault and failures in the system. It is important to have a high precision for this mining step since in the future we intend to use the groups to analyse temporal and spatial characteristics as well as correlations between events. HELO not only has a high accuracy but also presents to system administrators the description of each group making it easier for a human interaction in the process of cluster reorganization before the analyser step.

References

[1] Archive, F.T., `http://fta.inria.fr` (accessed on 2010)
[2] Schroeder, G.G.B.: A large-scale study of failures in high-performance computing systems. In: IEEE DSN 2006, pp. 249–258 (June 2006)
[3] Bookstein, A., all: Generalized hamming distance. Information Retrieval Journal 5(4), 353–375 (2002)
[4] Chuah, E., et al.: Diagnosing the root-cause of failures from cluster log files (2010)
[5] T. computer failure data repository, `http://cfdr.usenix.org` (accessed on 2010)
[6] Fu, Q.: all. Execution anomaly detection in distributed systems through unstructured log analysis. In: ICDM, pp. 149–158 (December 2009)
[7] Fu, S., Xu, C.-Z.: Exploring event correlation for failure prediction in coalitions of clusters. In: Proceedings of the ACM/IEEE Conference on Supercomputing (November 2007)
[8] Han, J., et al.: Mining frequent patterns without candidate generation. In: ACM SIGMOD, pp. 1–12 (May 2000)
[9] Lan, Z., all: Toward automated anomaly identification in large-scale systems. IEEE Trans. on Parallel and Distributed Systems 21(2), 174–187 (2010)
[10] Makanju, A., et al: Clustering event logs using iterative partitioning. In: 15th ACM SIGKDD, pp. 1255–1264 (2009)
[11] McCallum, A., all: Efficient clustering of high-dimensional data sets with application to reference matching. In: ACM SIGKDD, pp. 169–178 (August 2000)
[12] Mitra, M., Chaudhuri, B.: Information retrieval from documents: A survey. Information Retrieval Journal 2(2-3), 141–163 (2000)
[13] NCSA, `http://www.ncsa.illinois.edu` (accessed on 2010)
[14] Pang, W., et al.: Mining logs files for data-driven system management. ACM SIGKDD 7, 44–51 (2005)
[15] Park, Geist, A.: System log pre-processing to improve failure prediction. In: DSN 2009, pp. 572–577 (2009)
[16] Salfner, F., et al.: A survey of online failure prediction methods. ACM Computing Surveys 42(3) (March 2010)
[17] Stearley, J.: Towards informatic analysis of syslogs. In: IEEE Conference on Cluster Computing (September 2004)
[18] Stearley, J.: Towards informatic analysis of syslogs. In: IEEE International Conference on Cluster Computing, vol. 5, pp. 309–318 (2004)
[19] Vaarandi, R.: Mining event logs with slct and loghound. In: IEEE NOMS 2008, pp. 1071–1074 (April 2008)

[20] Wei Peng, S.M., Li, T.: Mining logs files for data driven system management. ACM SIGKDD 7, 44–51 (2005)
[21] Xue, Z., et al.: A survey on failure prediction of large-scale server clusters. In: ACIS SNPD 2007, pp. 733–738 (June 2007)
[22] Zarza, G., et al.: Fault-tolerant routing for multiple permanent and non-permanent faults in hpc systems. In: PDPTA 2010 (July 2010)
[23] Zhang, X., Furtlehner, C., Sebag, M.: Data streaming with affinity propagation. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 628–643. Springer, Heidelberg (2008)