Towards Automated Log Parsing for Large-Scale Log Data Analysis

[Supplementary Report]

Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu, *Fellow, IEEE*

APPENDIX

This report is provided as supplementary material for our submission to TDSC. The content comprises three parts: parameter sensitivity, overview of existing log parsers, and PCA-based anomaly detection. Note that we use [Section xxx] along with the section title to point out the place where the corresponding supplementary content is referenced in the submitted main paper.

PARAMETER SENSITIVITY

POP has four parameters: GS, splitRel, splitAbs, maxDistance. We will explain these parameters one by one. All the sensitivity experiments are run on the 2k datasets, which are the datasets used to evaluate the accuracy of the parsers in Section 4.2.1. Similar to the parameter setting in our accuracy experiments, for dataset BGL, HPC, HDFS, and Zookeeper, we set GS to 0.6, splitRel to 0.1, splitAbs to 10, maxDistance to 0; for Proxifier, we set GS to 0.3, splitRel to 0.1, splitAbs to 5, maxDistance to 10. To study the impact of different parameters, we evaluate the accuracy of POP while varying the value of the studied parameter.

1) Impact of GS

According to our definition, GS decides whether a log group is complete in step 3. If a log group is complete, it will be sent to step 4 without further partitioning. Intuitively, GS is the threshold about the percentage of columns where all the tokens are the same in a log group. For example, if we set GS to 0.6, POP will regard log groups with more than 60% columns having the same tokens respectively as complete groups. For the log group in Fig. 1, two (i.e., the first and third) columns have the same tokens in that token position (i.e., "Send" and "file") respectively. Its GG (Group Goodness) is 2/3. GG (2/3) is larger than GS (0.6), so this log group is a complete group.

1.	Send	configuration	file
2.	Send	network	file
3.	Send	bootstrap	file
4.	Send	video	file

Fig. 1: Example Log Group

The sensitivity analysis for GS is demonstrated by Fig. 2. We can observe that the accuracy of POP is high for all datasets if we set GS in range [0.5, 0.6]. Intuitively, this means a log group is complete if more than half or 60% of its columns have the same tokens for the whole column. When GS is smaller, a log group is easier to get sent to step 4 without further partitioning, which may lower the accuracy because we may put log messages with different log events into the same log group. Thus we can observe the relatively lower accuracy for range [0, 0.3] on HPC and range [0, 0.2] on HDFS. When GS is larger, a log group has higher probability to go through partitioning process in step 3, which may lower the accuracy because we may put log messages with the same log event into different log groups. Thus we can observe the relatively lower accuracy in range [0.8, 1.0] on HPC, range [0.8, 1.0] on HDFS and range [0.9, 1.0] on Zookeeper. To tune GS for a new dataset, we could first set GS to 0.5 or 0.6. Then we evaluate the accuracy of POP on a sample dataset (e.g., 2k lines), which is often much smaller than the whole dataset. If the accuracy is not satisfactory, we can increase or decrease the GS by 0.1, and evaluate POP's accuracy on the sample dataset again. We can repeat this process until the accuracy becomes lower. Then we select the GS that achieves the highest accuracy on the sample dataset. According to our parameter tuning experiment (Section 4.2.2 in the revised manuscript), POP can achieve high accuracy when using the parameters tuned on small sampled data.

2) Impact of splitRel and splitAbs

Parameter splitRel and splitAbs are used to control the partitioning of the incomplete log groups in step 3 of POP. For each incomplete group, POP will try to split the log group based on the tokens in the split token position. If the tokens in the split token position are constants, the current group will be partitioned into several groups. Otherwise, the log group will be sent to step 4. splitAbs is the threshold about the number of unique tokens (i.e., AT) in a column (i.e., token position). For example, in Fig. 1, the first and third column have one unique token, while the second column has four unique tokens. splitRel is the threshold about the ratio between the number of unique tokens and the number of tokens (i.e., RT) in a column. For example, the

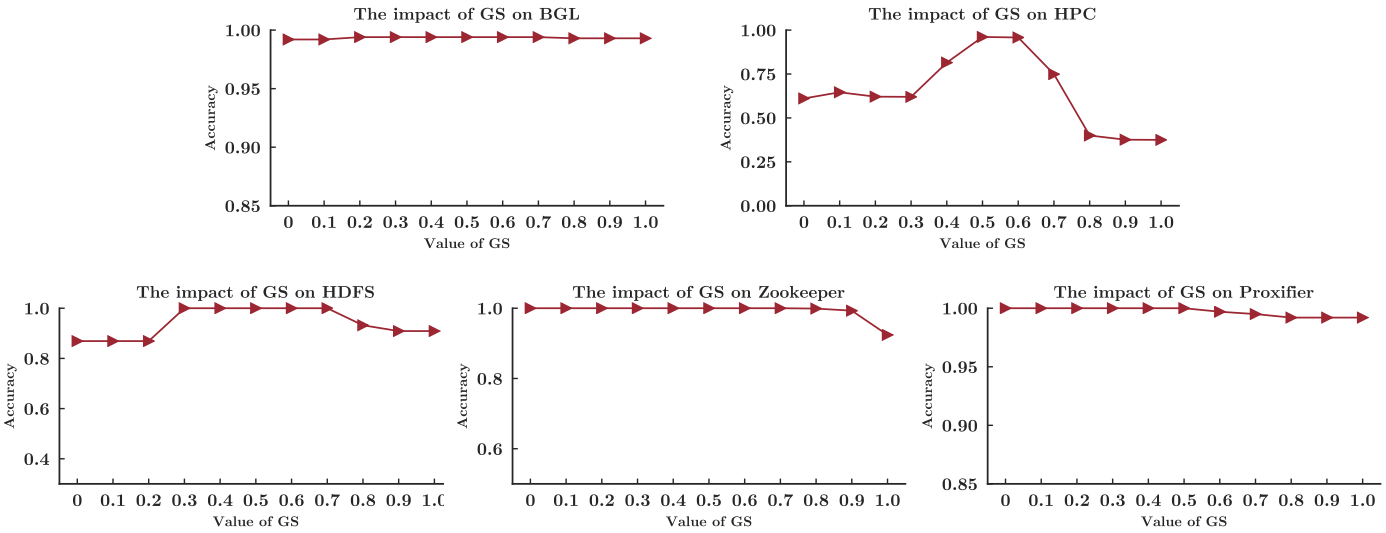

Fig. 2: Impact of GS

number of tokens for all the columns in Fig. 1 is four. If RT is larger than splitRel, and AT is larger than splitAbs, POP regards the tokens in the split token position as variables, because there are too many unique tokens. Otherwise, POP regards the tokens as constants.

We conduct the sensitivity analysis experiments for both splitRel and splitAbs. Since their results are very similar, we demonstrate the results for splitRel here.

From Fig. 3, we can observe that the accuracy of POP is insensitive to the value of splitRel. There are two main reasons. First, with a suitable (but easy to find) GS, we can already send some complete groups to step 4, which provides a lower bound for the accuracy. Second, splitRel and splitAbs control the partitioning of incomplete groups together in step 3. Thus with a suitable splitAbs, changing the value splitRel will not cause big accuracy change. We provide both splitRel and splitAbs to allow finer tuning by users. The accuracy of POP has observable change on BGL when varying the value of splitRel, and the accuracy peaks when splitRel is in range [0.1, 0.3]. This demonstrates the effectiveness of having both splitRel and splitAbs.

We also evaluate the impact of splitRel when splitAbs is 0 as in Fig. 4. We can observe that the accuracy of POP is consistent even we set splitAbs to 0. For BGL, HPC, HDFS, and Zookeeper, the accuracy is low when both splitRel and splitAbs is 0. Under this parameter setting, POP always regards the tokens in the split token position of the incomplete groups as variables. Thus, POP will send all the incomplete groups to step 4 without further partitioning. This lead to log groups containing log messages with different log events, which lowers the accuracy. For the parameter values in range [0.1, 1.0], POP consistently achieves high accuracy. To set a suitable splitRel (or splitAbs) value, we can first pick a reasonable value, for example 0.1 (or 10). Then we tune this value by evaluating the accuracy of POP on small sample datasets. The parameter value that has the highest accuracy can be used on the original dataset.

3) Impact of maxDistance

Parameter maxDistance is used in step 5 to merge the similar log groups based on the log events (i.e., log templates) extracted in step 4. POP calculates the Manhattan distance of log events in step 5 to merge log groups. Intuitively, maxDistance is the maximum number of different tokens allowed between the farthest two log events in two merging groups. In our paper, we set maxDistance to 0 for BGL, HPC, HDFS, and Zookeeper, where only two groups with the exactly same log event will be merged. Because POP can already achieve high accuracy, we do not use a larger maxDistance value. For Proxifier, if we set maxDistance to 0, the accuracy is 0.89. Besides, some log groups are over-parsed by step 3 on Proxifier. Thus, we set maxDistance to 10 to address over-parsing. We demonstrate the impact of maxDistance on accuracy in Fig. 5.

From Fig. 5, we can observe that the accuracy peaks when maxDistance is in range [4, 12]. When maxDistance is in range [0, 4), logs have been over-parsed, so the accuracy of POP is relatively low. When maxDistance is in range (12, 20], or is larger than 20, log groups that contain log messages with different log events are merged, which lowers the accuracy of POP. To find the suitable maxDistance, similar to other parameters, we can start with a value, for example, set maxDistance to 10. Then we tune this parameter by evaluating the accuracy of POP on a small sampled dataset. After finding the best parameter, we can directly apply it to the original dataset.

To pick a suitable value, we could first set the parameter to a reasonable value according to its physical meaning. Then we tune it on a small sample dataset by evaluating the resulting accuracy. After finding the best parameter, we can apply it to the original dataset. We add Section 4.5 in the revised manuscript to demonstrate the above parameter analysis results. The sensitivity analysis would greatly facilitate the parameter setting.

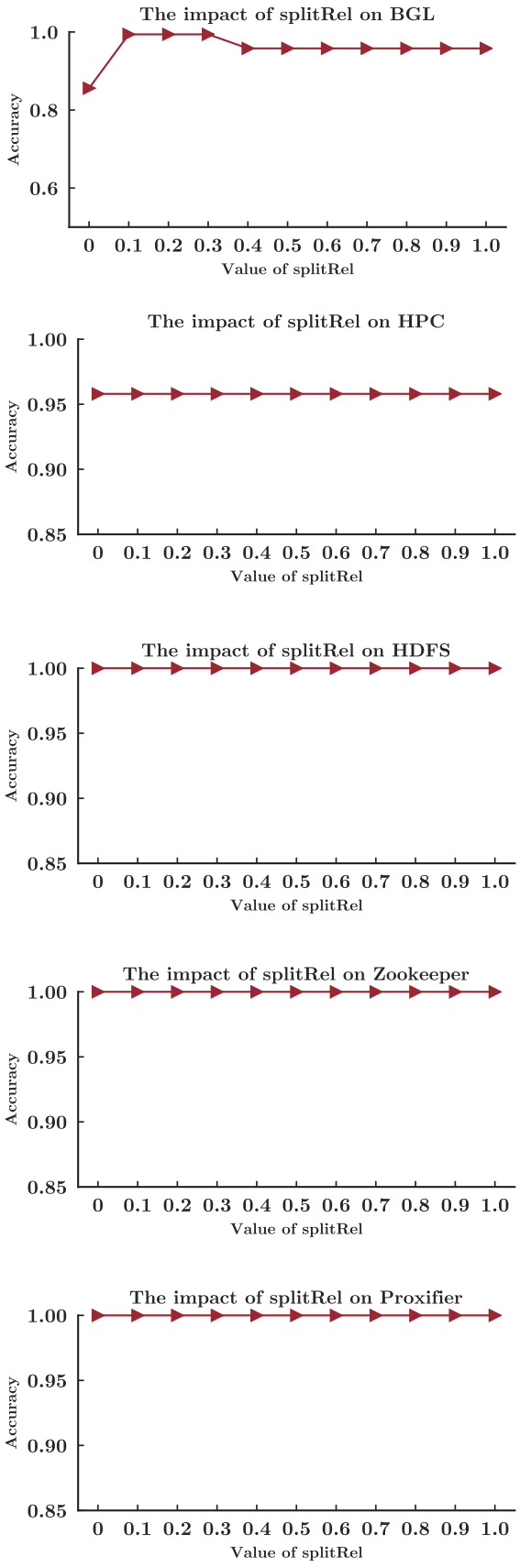

Fig. 3: Impact of splitRel

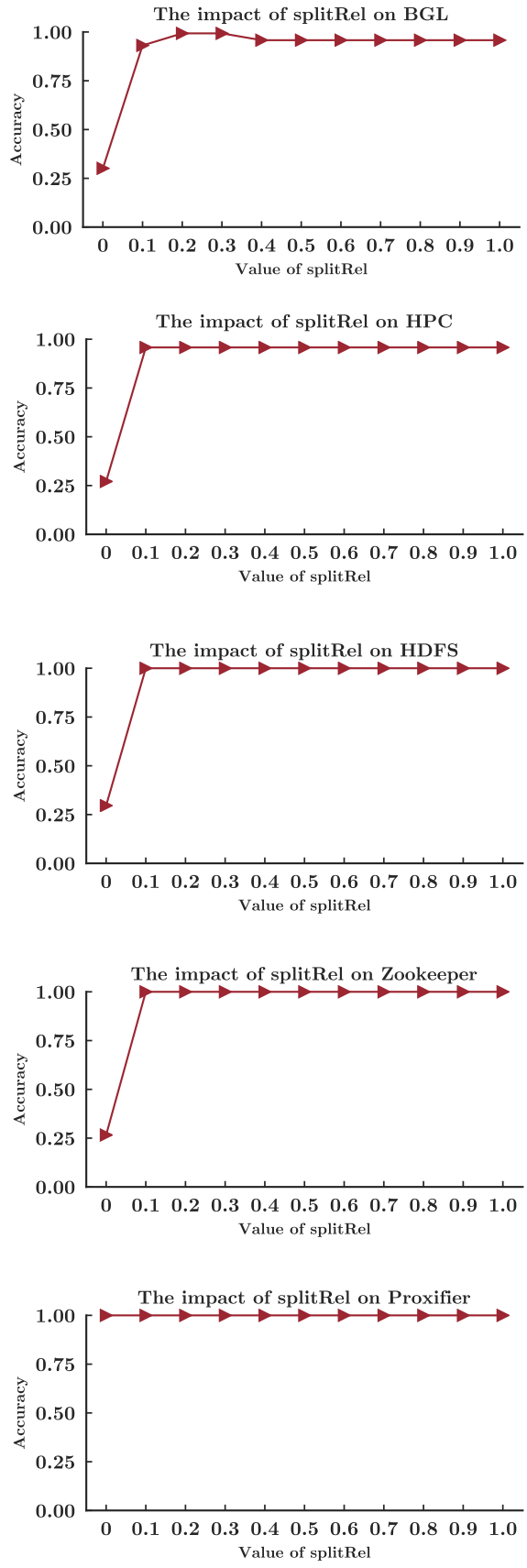

Fig. 4: Impact of splitRel (splitAbs=0)

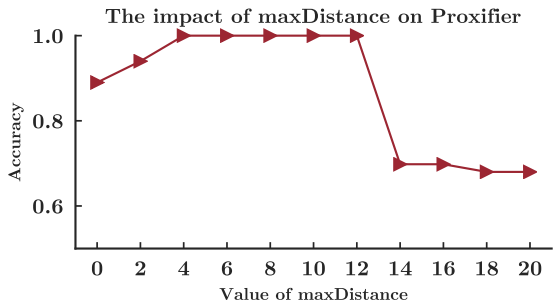

Fig. 5: Impact of maxDistance

Overview of Existing Log Parsers

Log parsing has been widely studied in recent years. Among all the log parsers, we choose four representative ones, which are in widespread use for log mining tasks. With the main focus on evaluations of these log parsing methods, we only provide brief reviews of them; the details can be found in the corresponding references.

1) SLCT

SLCT (Simple Logfile Clustering Tool) [1] is, to the best of our knowledge, the first work on automated log parsing. The work also released an open-source log parsing tool, which has been widely used in log mining tasks, such as event log mining [2], symptom-based problem determination [3], and network alert classification [4]. Inspired by algorithms of association rule mining, SLCT works as a three-step procedure with two passes over log messages: *1) Word vocabulary construction.* It makes a pass over the data and builds a vocabulary of word frequency and position. *2) Cluster candidates construction*. It makes another pass to construct cluster candidates using the word vocabulary. *3) Log template generation.* Clusters with enough log messages are selected from candidates. Then, the log messages in each cluster can be combined to generate a log template, while remaining log messages are placed into an outlier cluster.

2) IPLoM

IPLoM (Iterative Partitioning Log Mining) [5] is a log parsing method based on some heuristics that are specially designed according to the characteristics of log messages. This method has also been used by a set of log mining studies (e.g., alert detection [6], event log analysis [7], and event summarization [8]). Specifically, IPLoM performs log parsing through a three-step hierarchical partitioning process before template generation: *1) Partition by log length.* Log messages are partitioned into different clusters according to different lengths. *2) Partition by token position.* For each partition, words at different positions are counted. Then the position with the least number of unique words is used to split the log messages. *3) Partition by search for mapping.* Further partition is performed on clusters by searching for mapping relationships between the set of unique tokens in two token positions selected using a heuristic criterion. *4) Log template generation.* Similar to SLCT, the final step is to generate log templates from every cluster.

3) LKE

LKE (Log Key Extraction) [9] is a log parsing method developed by Microsoft, and has been applied in a set of tasks on log analysis [9], [10]. LKE utilizes both clustering algorithms and heuristic rules for log parsing: *1) Log clustering.* Raw log messages are first clustered by using hierarchical clustering algorithms with a customized weighted edit distance metric. *2) Cluster splitting.* A splitting step based on heuristic rules is performed to further split the clusters. *3) Log template generation.* The final step is to generate log templates from every cluster, similar to SLCT and IPLoM.

4) LogSig

LogSig [11] is a more recent log parsing method, which has been validated in [12]. LogSig works in three steps: *1) Word pair generation.* Each log message is converted to a set of word pairs to encode both the word and its position information. *2) Log Clustering.* Based on the word pairs, a potential value is calculated for each log message to decide which cluster the log message potentially belongs to. After a number of iterations, the log messages can be clustered. *3) Log template generation.* In each cluster, the log messages are leveraged to generate a log template.

PCA-based Anomaly Detection

The input of the anomaly detection task is a text file, each line of which is a raw log message recording an event occurring on a block in HDFS. In this step, log parsing method is adopted to figure out two things. One is all the event types appearing in the input file. The other is the events associated with each block, which distinguished by block ID. These two are exactly in the two output files of our log parser modules. We emphasize that the parsing output is not specific to anomaly detection, but also suitable for other log mining tasks.

Parsed results are used to generate an event count matrix Y, which will be fed into the anomaly detection model. In the event count matrix, each row represents a block, while each column indicates one event type. The value in cell $Y_{i,j}$ records how many times event j occurs on block i. We could generate Y with one pass through the parsed results. Instead of directly detecting anomaly on Y, TF-IDF [13], which is a well-established heuristic in information retrieval, is adopted to preprocess this matrix. Intuitively, TF-IDF is to give lower weights to common event types, which are less likely to contribute to the anomaly detection process.

Anomaly detection is to find out suspicious blocks that may indicate problems (e.g., HDFS namenode not updated after deleting a block). The model used in our case study in Section 4.4 is Principle Component Analysis (PCA) [14], which is a statistical model that captures patterns in high-dimensional data by selecting representative coordinates (principle components).

PCA is used in this problem because principle components can represent most frequent patterns of events associated with blocks, which is called normal space S_d. Specifically, the first k principle components are selected to form S_d, while the remaining $n - k$ dimensions form S_a (anomaly space), where n is the number of columns (total

number of event type) of the matrix. In this task, each row in the event count matrix is a vector y associated with a block. The intuition of anomaly is the vector whose end point is far away from normal space. The "distance" could be formalized by squared prediction error $SPE \equiv ||y_a||^2$, where y_a is the projection of y on S_a. y_a is calculated by $y_a = (I - PP^T)y$, where $P = [v_1, v_2, ..., v_k]$. A block is marked as anomaly if its corresponding y satisfies:

$$SPE = ||y_a||^2 > Q_\alpha,$$

where Q_α is a threshold providing $(1 - \alpha)$ confidence level. For Q_α, we choose $\alpha = 0.001$ as in the original paper [14].

REFERENCES

[1] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IPOM'03: Proc. of the 3rd Workshop on IP Operations and Management*, 2003.

[2] ——, "Mining event logs with slct and loghound," in *NOMS'08: Proc. of the IEEE/IFIP Network Operations and Management Symposium*, 2008.

[3] L. Huang, X. Ke, K. Wong, and S. Mankovskii, "Symptom-based problem determination using log data abstraction," in *CASCON'10 Proc. of the Conference of the Center for Advanced Studies on Collaborative Research*, 2010, pp. 313–326.

[4] R. Vaarandi and K. Podis, "Network ids alert classification with frequent itemset mining and data clustering," in *CNSM'10: Proc. of the Conference on Network and Service Management*, 2010.

[5] A. Makanju, A. Zincir-Heywood, and E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 24, pp. 1921–1936, 2012.

[6] ——, "Fast entropy based alert detection in super computer logs," in *DSN-W'10: Proc. of International Conference on Dependable Systems and Networks Workshops*, 2010, pp. 52–58.

[7] ——, "Investigating event log analysis with minimum apriori information," in *IM'13: Prof. of International Symposium on Integrated Network Management*, 2013, pp. 962–968.

[8] Y. Jiang, C. Perng, and T. Li, "Meta: Multi-resolution framework for event summarization," in *SDM'14: Proc. of the SIAM International Conference on Data Mining*, 2014, pp. 605–613.

[9] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM'09: Proc. of International Conference on Data Mining*, 2009.

[10] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *ATC'10: Proc. of the USENIX Annual Technical Conference*, 2010.

[11] L. Tang, T. Li, and C. Perng, "LogSig: generating system events from raw textual logs," in *CIKM'11: Proc. of ACM International Conference on Information and Knowledge Management*, 2011.

[12] L. Tang, T. Li, L. Shang, F. Pinel, and G. Grabarnik, "An integrated framework for optimizing automatic monitoring systems in large it infrastructures," in *KDD'13: Proc. of International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 1249–1257.

[13] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrival," Cornell, Tech. Rep., 1987.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordon, "Detecting large-scale system problems by mining console logs," in *SOSP'09: Proc. of the ACM Symposium on Operating Systems Principles*, 2009.