Analysis of Automated Log Template Generation Methodologies

Anoop Mudholkar [1], Varun Mokhashi [1], Deepak Nayak [1], Vaishnavi Annavarjula [2], and Mahesh Babu Jayaraman [3]

[1] School of Information Sciences, MAHE, Manipal, India
*anoopmudholkar1994@gmail.com
varunmkhs@gmail.com
meetdeepaknayak03@gmail.com
[2] Blekinge Institute of Technology, Karlskrona, Sweden
vaan15@student.bth.se
[3] Ericsson Research, Bangalore, India
mahesh.babu@ericsson.com

Abstract. Decision making and advanced diagnostics over log messages from operations point of view is an important and challenging area giving that these log messages do not adhere to a format, variable in length and essentially remains unstructured. Log analysis is a key function that deals with analyzing these messages to produce insights that help operate, understand, debug and manage the services, applications and/or functions to (i) *detect failures* (ii) *detect consistency issues* (iii) *deduce anomalous behaviors* and (iv) *continuous monitoring with prediction.*An infrastructure management practice relies on huge amounts of log messages collected from the devices such as servers, routers and switches in a Data center, Telecom, Datacom networks or IT operations. Entities emit log messages revealing state of the running system for management perspective.The growing scale and complexity of infrastructure makes it unrealistic and impractical to analyze log messages with manual or subject matter experts or even with expert systems.
Towards this emerged automated methods for log parsing, which carefully study and extract features of interest from these messages and produces message templates for applications to easily discover constituent properties.These methods are largely based on Clustering which is an unsupervised machine learning approach. Different parsers make use of clustering in different ways, in this work we study these techniques and compare its template generation capabilities. The generated templates so formed face challenges such as human readability, machine interpretability and inconsistent structures. In this paper, we present an analysis of existing log template generation methodologies quantiatively and qualitatively. We conclude with challanges that are still prevailing identifying the functional weaknesses using examples when templatizing the log message history.

Keywords: log analysis · template generation · unsupervised learning · clustering · machine learning

1 Introduction

Logging frameworks help developers inject logging related features through libraries, Application Programming Interface(API) or Software Development Kit(SDK) generated in mega bytes to giga bytes of dumps and hence I/O intensive. Choice of one over the other can significantly impact the performance of applications. Well known methods include Syslog [1], Windows EventLog [2], Log4j [3], SLF4J [4] and LOGBack [5].

Log message structures

- Traditional unstructured log
- Structured log

Structured logging gave greater control to developers by enabling to encode structure using SDK, parameter identification mechanisms and log message decorator frameworks. Libraries like message template [6], slog, NLog, etc provides single-line, multi-line, Java script object notation (JSON) formats. Structured logging benefited by reducing the need for regular expression based matching during log parsing and leading to solve leakage problems. Free form logging prevails and still continue to be in use due to its convenience and model free nature.

Traditional Log parsing feature extraction techniques

- Regular expression based parsing
- Rule based parsing and handling

Rigorous domain-agnostic processing methods like text analytics techniques were used. i.e like a counters accounting for number of times a certain log message, variable portion and set of messages that are repeated. These were inspired from bag of words model, n-gram model and also using techniques like transcationized log batching.

There are varieties of logs viz., Hadoop, Apache, sshd and other flavors include network, cloud infrastructure, standard-specific and product-specific logs. It becomes complex for the log parsers to perform domain-based feature extraction. Key-value notation based logging results in increase of the size of logs. Number of templates, rules or expressions that needs to be maintained in context of parsing adds significant book keeping requirements. Hence this remains a challenge for backward compatibility, freedom for log structure and flexibility during development.

1.1 Log Parsing Frameworks

Notable log parsing frameworks include liblognorm [7], logstash's grok filter [8] which essentially extracts structured data from unstructured logs using regular

expression and tagging based solutions. With large scale and converged infrastructure deployments, the problem of processing and analyzing log messages manually becomes intractable. Traditional methods does not meet the functional, scaling and flexibility requirements.

Hence evolved the new trend of machine learning based automated log parsing techniques. viz., Iterative Partitioning Log Mining (IPLOM) [9], Log Key Extraction (LKE) [10], Message Template Extraction (MTE) [11], LogSig [12], Simple Logfile Clustering Tool (SLCT) [13] and The LogCluster [14].

Table 1. Different Log Parsers

Parsing Method	Key Techniques Used
IPLoM	Feature extraction, Tokenization, Cluster formation, Templatization
LKE	Log keys clustering, Empirical rules, Log key templates
MTE	Predefined ASCII tokens, Constant generation, Templatization
LogSig	Unsupervised clustering, Word pair generation, Templatization
SLCT	Word vocabulary dictionary, Support value, Clustering
LogCluster	Input support value, Line patterns, Clustering

Other works like Semantic Query Federation and Linked-data paradigm are being adopted to ensure better detection over log data [15]. This is an interestingly new dimension for log analysis but still dependent on domain knowledge to codify the semantics. Unsupervised log signature extraction [16] by Thaler et. al, shows domain-agnostic method using auto-encoder neural network as an alternative to IPLoM and LogCluster. However, the extracted features are not human interpretable and remain an internal property within a black box model.

1.2 Log Management Frameworks

Log analysis is carried out either in centralized or de-centralized manner. Decentralized approach essentially distributes the log message processing functions to edges or jobs to shards and aggregates the processed results at a different level. A typical log management framework involves collect, process and analysis processes. Storage and cluster infrastructure are other important areas that these log management framework dwells with.

Most commonly used Log management frameworks in the industry are ELK Stack, Splunk, LogPacker, LogRhythm, LogScape, Loggly etc. ELK Stack is the most flexible open source framework which also recently has Machine Learning capabilities for log analysis. A brief summary of log management framework features is discussed below.

Log collection Large-scale systems continuously generate logs to record system states and runtime information, each comprising a time-stamp and a log

message indicating what has happened. These information can be used for multiple purposes like anomaly & network fraud detection etc. Storage, Indexing, buffering and Fast-retrieval capabilities are key system features enabled by the underlying platform.

Log processing Logs are dominantly unstructured, which contain free-form text. Processing it means cleaning, parsing, aggregating, filtering, converting and transforming such that applicable, relevant and important features are extracted into a structured and organized information.

Log Analysis & Visualization Log processing yields a database like structured information. This helps carry out generation of insights, derive key performance indicators through further processing and search/query over structured log information base. Analysis and visualization capabilities enable just-in-time generation of results (reports, summary, and alerts)and visualization (dashboards, graphical and interactive) for enhanced user experience.

Our observation is that these automated methods produce templates which are still not intuitive to human reading, supportive of machine interpretability and method consistency aspects. Machine interpretability in this context refers to the ability of the templating algorithm to identify the data type and form a schema of the logs. A detailed quantitative assessment is discussed in work by Pinjia et. al [17] which had more of use-case centric measures from respective demonstrator application such as anomaly detection [18] or from problem diagnostics [11] perspective or detect conditions [19]. No existing work has compared the quality of generated templates i.e. feature extraction for generating quality templates.This log template generation problem is also referred to as signature or log key extraction in literature.

The Log management frameworks are rule-based engines and are most appropriate to derive insights and perform analysis of the logs but are not useful in forming Log Templates

In this paper, we show examples of these functional weaknesses and impress upon the need for newer techniques. We present a brief study of algorithms in section 2, we discuss the quantitative & qualitative assessment of known log template generation methods and techniques in section 3 including the comparative analysis. Present conclusions in section 4.

2 Related Work

We are interested in the aspect of log parsing that extracts a group of event templates which are derived using clustering, partitioning and grouping techniques whereby logs can be mapped to template structures as part of subsequent analysis. More specifically, each log message can be parsed into an event template (constant part) with some specific parameters (variable part). A detailed analysis of domain-agnostic automatic feature extraction techniques are discussed below.

2.1 IPLoM

Iterative Partitioning Log Mining (IPLoM) [9] is a widely referred feature extraction techniques for log messages. First step groups messages by number of tokens into a cluster. Second step involves grouping by token position creating sub-clusters. Third step we search for bijective relationships between the set of unique tokens by searching for mapping relationships creating partitions. Final step is to extract the event template which involves verifying cardinality of the unique tokens. If the cardinality of the unique token values in a position is 1, then that token position becomes constant part in the template. Otherwise, variable token position is replaced with symbol "*". The results are very close to a regular expression structure differentiating constant and variable portions, which we believe is the key reason behind its broad adoption.

2.2 LKE

Log key Extraction (LKE) [10] makes use of log key to deal with the unstructured data. Log key is defined as the common content among all log messages. Assuming that log messages printed by same statement are highly similar, clustering techniques are used to group log messages printed by the same statement together. Parameters in the form of URIs, IP address, special symbols may cause distortions while clustering, they are handled by writing empirical rules in the form of regular expressions and produces log key templates.

2.3 MTE

Message Template Extractor (MTE) component is part of FDiag - a diagnostic tool [11]. This method uses predefined ASCII token expression identifying the constant parts to arrive at templates. First it splits the raw logs based on respective individual days the messages were generated, then extract constant from the message bodies based on ASCII code message segment, merge the constants, find unique constants and represent it as a template and by computing the daily frequency of occurrence of the constant template identify the filtered templates. These are constant part based templates, a structure very specific to the FDiag system used for detecting correlation between any two log messages. This method enables construct episodes that will further support debugging. However may not be friendly to allow generation of templates.

2.4 LogSig

LogSig [12] is dependent on unsupervised clustering to derive the templates. It takes 'k'-clusters as input. First, it generates word pairs by extracting words preserving the order as part of the structure. It further iterates to determine clusters based on number of word pairs present and its frequencies. This step is repeated until all logs are identified with its cluster home. The formed k-clusters are then assessed for the most frequently occurring word pair order, and

is chosen as the log message template in that cluster. The key disadvantage to this method is to specify a 'k' that is appropriate for a given dataset. This method exhibits template fragmentation issue where multiple templates reported when 'k' is high. However reducing 'k' either is not supportive because the collapsed template structure is not representative of all the variants.

2.5 SLCT

Simple Logfile Clustering Tool (SLCT) [13] in its first step it builds a dictionary of word vocabulary that contains word frequency with respect to its position. In the next step, makes a second pass over the logs and constructs cluster candidates based on the most frequent words/phrases calling these as log templates and each log template is denominated as cluster candidate. Further, based on user configured minimum support value - 'N', its filters candidate clusters to arrive at final log templates. The candidate clusters that has less than 'N' logs are binned as outliers class.

2.6 LogCluster

The LogCluster [14] algorithm addresses the shortcomings of SLCT. With the assumption that there are 'n' lines and each line is a sequence of 'k' words. LogCluster takes support threshold 's' as input and divides log lines into clusters each with 's' line. Then it mines for line patterns with words and wild-cards where multiple log lines match to the pattern in that cluster. LogCluster finds patterns with support 's' or higher by identifying words which occur in at-least 's' event lines. But unlike IPLoM and SLCT, LogCluster considers each word agnostic to position in the log line.

3 Analysis of log parsers

In this section we carry out a detailed analysis of these template generation methods. The primary motive is to identify opportunities for improvement by identifying the prevailing algorithmic challenges that enhances applications and its usefulness the generated templates. We try to build on top of the detailed analysis by Pinje et. al [17] which enables log parser selection by appropriate parsing techniques and reiterated reuse of existing algorithms. It evaluated 4 log parsers using 5 different log message datasets with 6 insightful findings. However the previous works did not functionally validate the readability of templates and consistency issues with template generation methods.

In this paper, we present an evaluation based on readability and consistency criteria detailed further by selecting two candidate datasets.

From the parsing algorithms mentioned in Section 2, we have eliminated LKE, MTE and LogCluster due to the nature of generated templates. LKE depends on predefined rule based preprocessing that removes content beyond prefix of the message which we felt is doing too much change to the original

message and making this method farther to produce readable templates. MTE generates constant-templates which are seen as message keys rather than log template. LogCluster requires user to input parameters such as support threshold 's' which in-turn requires domain knowledge and understanding of dataset, which we felt is very expensive step.

3.1 Candidate datasets

This analysis makes use of (i) HDFS Dataset [20] referred as Dataset1, a subset of messages from Darkstar data based on Amazon EC2 cluster shown in table 2 and (ii) Dartmouth Dataset [21] referred as Dataset 2, a subset of syslog messages from the wireless network of Dartmouth college is shown in table 3. These input datasets have been pruned of the prefix portion for simplicity in respective table views.

Table 2. HDFS Dataset

PacketResponder 1 for block blk_38865049064139660 terminating
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864
Received block blk_3587508140051953248 of size 67108864 from /10.251.42.84

Table 3. Dartmouth Dataset

Interface Dot11Radio0, Station 001360469694 001360469694 Reassociated KEY_MGMT[NONE]
Station 001360469694 Roamed to 004096ec1414
Interface Dot11Radio0, Deauthenticating Station 001360469694 Reason: Disassociated because sending station is leaving (or has left) BSS

3.2 Quantitative Analysis

We compare the parsers based on the templates generated for datasets namely HDFS(1k,2k,4k) and Dartmouth(5k,10k,20k) variants. The numbers in Table 4

have a lot to say with respect to functionality of each parser, in terms of number of templates, consistency and duplication.

Table 4. Dataset to #log templates generated

Dataset	Number of Logs	IPLoM	Logsig	SLCT
Dataset1	1k	12	31,27	16
Dataset1	2k	16	32,36	27
Dataset1	4k(doubled 2k)	16	35,38	161
Dataset2	5k	19	58,69	78
Dataset2	10k	19	44,43	164
Dataset2	20k	19	62,67	275

Number of templates generated From Figure 1 and 2, we can infer that the number of templates generated increases with the increase in number of log lines. While it is possible that there may be increase in log templates, however the differences in numbers between different parsers is large and it forced us to explore these variations individually.

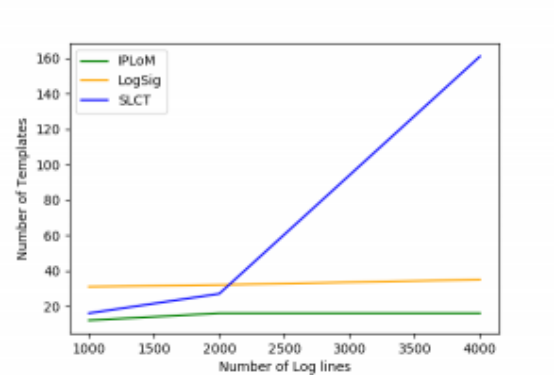

Fig. 1. HDFS

IPLoM Number of templates generated is small and it is almost constant throughout for all the test cases.

LogSig The numbers generated are not reliable since this method has consistency issue which is discussed in following section.

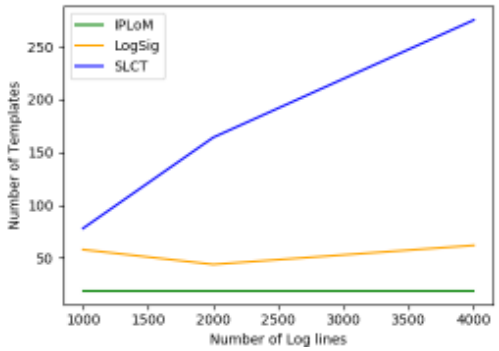

Fig. 2. Syslog

SLCT The curve is steeply increasing in number of templates. This was because SLCT uses a frequent word clustering approach which uses frequently occurring words to form the clusters. When more number of messages are given as input, it results in generating more number of templates which is not a desired behavior.

Across these parsers, except for IPLoM there is observable difference in the output and clearly indicating large variations that is not indicative of characteristics of the generated templates.

Consistency of the parsers In Figure 3, we observe a major issue with respect to consistency of LogSig. Although the templates for IPLoM, SLCT are consistent when run for more than single iteration, LogSig is inconsistent where the number of templates generated vary with every iteration.

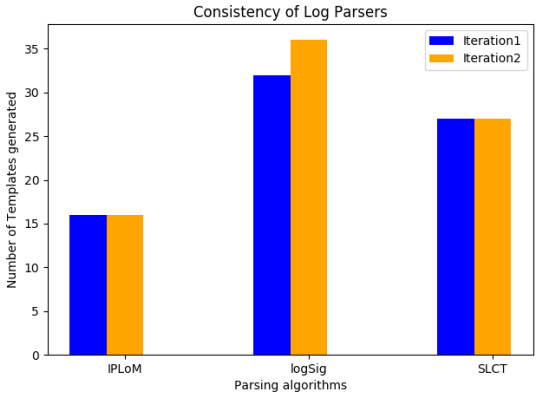

Fig. 3. Consistency of log parsers

Duplication of Log lines In Figure 4, we can observe how parsers react to duplicate log lines. Duplicates may occur when same log is registered multiple times either with different time stamp or as repeated messages.

To look deeper we took HDFS 2k dataset and doubled it to new dataset called HDFS 4K. The expectation is that the number of log templates generated is same as that of 2k variant. This is because doubling the dataset will not add a new template structurally.

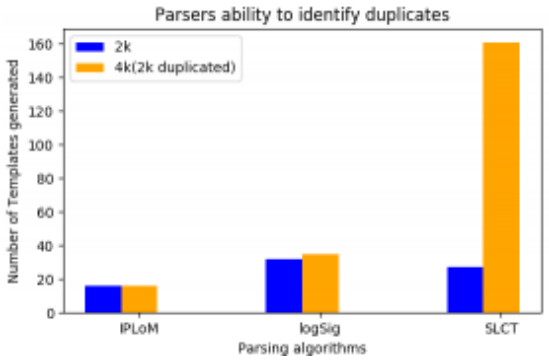

Fig. 4. Duplication response of Log Parsers

IPLoM Generates the same number of templates which highlights that this parser can handle repeating messages.

LogSig There were some duplicates that were insignificant. However, since this parser already exhibits inconsistency with number of generated templates, we skipped duplication check.

SLCT With 4K variant of the dataset, it only contributed to the internal support levels increasing i.e. there are more messages that are frequent with 4K variant compared to that of 2k. Hence, resulting in lot more templates which was not expected. This is a major drawback of SLCT parser.

Quantitative Analysis Summary: Neither the number of templates generated nor the consistency of parsers during iterations or the duplication of log lines, lead us to a conclusive understanding of the log parsers. This motivated and led us to perform Qualitative analysis which is described in the following section.

3.3 Qualitative Analysis

Qualitative analysis in this context refers to analyzing the templates generated by each of the parsers considered in detail. We have made an attempt to examine the templates with respect to certain characteristics such as fragmentation of

templates, handling of delimiters and detection of variables etc. This section shows an example set of templates that were generated and a discussion with its pros and cons.

Variable detection refers to the ability of the parser to identify variable segments in the message and represent them appropriately in the template either as wild-card or with an identifier name. Most the log template generation methods place a wildcard when it identifies a variable, this again leads to problems with human and machine interpretation.

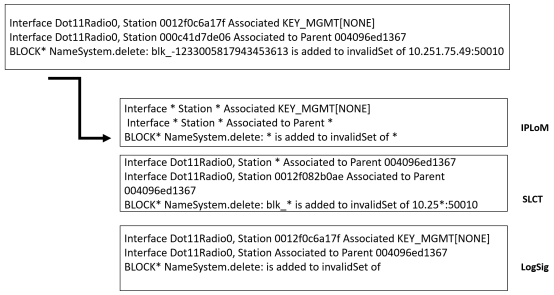

Fig. 5. Variable Detection

In Figure 5, we can observe variation with respect to variable detection in templates generated from different parsers.

IPLoM did excellent in identifying the variables (e.g. interface id, station id) and replaced these with wild-card to arrive at suitable templates.

SLCT results were a bit confusing, not all station identifiers were identified as variables as shown in the example.

LogSig did not recognize the variable portions expansively and this method produced multiple templates for same family of log messages as shown. Also, another observation is that, this parser does not place wild-card when deduced as variable which is a significant drawback when considering only the template structure aspect.

An ideal parser should identify the variable part and replace it with respective identifiers and its deduced type rather than wild-cards. Also, an ideal parser with variable detection would name this part potentially with pseudo names if not realistic like a domain expert would do. e.g. as shown below:

Interface (Interface ID:AplhaNumeric) Station (Station ID:AlphaNumeric) Associated KEYMGMT[NONE]

Delimiters Log messages are subtly embedded with set of delimiters to indicate separation between segments of the message or to separate the constant and variable parts whichever order they occur. Log parsers are expected to identify these correctly and templatize the messages accordingly.

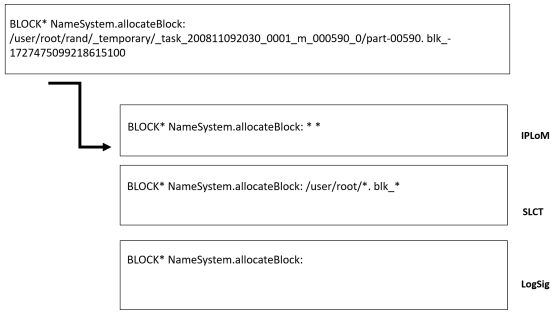

Fig. 6. Delimiters

In Figure 6, we can observe different interpretation of DELIMITERS by respective parsers.

IPLoM has interpreted two variable parts separated by delimiter correctly and placed wild-card for these portions in the generated template fittingly.

SLCT comparatively has done a better job by identifying the constant prefixes within the variable part and replaced the remaining portion with wild-cards.

LogSig parser observably seems to drop portions of messages in the generated templates. This is because, this parser does not place wild-card and hence the generated template is not reflective of possible variables and its delimiters that separate them.

Fragmentation can be referred to as generation of multiple templates for similar log lines. These generated templates may be representing the log message line either completely or partially or in duplicates. Partial template case arises when parser generated structure is not fully representative of the log message line or when it is compounded with other problems as discussed elsewhere in this paper. Fragmentation further leads to the main challenge mentioned earlier, human readability and machine interpretability because multiple templates of same kind leads to ambiguity.

In Figure 7, we can observe FRAGMENTATION problem with almost all the parsers.

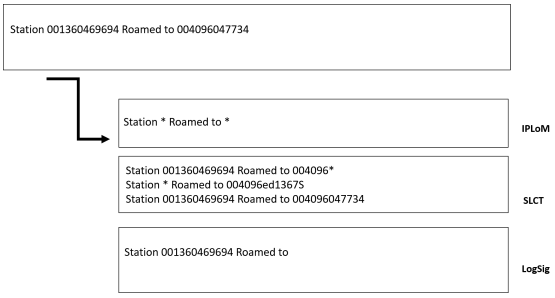

Fig. 7. Fragmentation

IPLoM in general produces very good templates by detecting and replacing variables with wild-cards. However, we observe scarce conditions where a variable part in a message that typically flips between fixed set of values (e.g. associated and disassociated) are not identified as variables, which is discussed in detail in Enumeration detection problem section as in figure 8.

SLCT produces templates with wild-cards for parts of the variable portions in the message which leads to multiple templates. This fragmentation occurs because, this parser algorithm is inherently depending on support threshold which fragments templates in countless circumstances.

LogSig on the other hand is skipping certain segments of the log line which is more severe problem when compared to SLCT. This issue is similar to delimiters issue too.

Enumeration refers to identifying variants that are flipping between certain set of values. Unlike an IP address or a station id type of variable, enumeration type can be associated to when its value swings between examples like "up, down", "associated, disassociated", and etc. These shifts in values are consistently within a globally derivable dataset for the given input log message history. Not dealing with enumerated value type only leads to fragmentation i.e. multiple independent templates. Similar to Fragmentation, Enumeration also leads to problems with respect to consistent human and machine interpretation.

In Figure 8, we can observe ENUMERATION problem. The words *Associated* and *Reassociated* in the input log message history are not translated to variable in any of the log parser methods. This is because these values being the only option at that position and also not varying unlike a variable value these words easily cross the support levels measured within the algorithm and hence considered as constants. Subsequently, these methods end up creating two or multiple different templates for the log message history.

Pre-processing refers to cleaning the raw log messages. Identifying and removing segments of a log line like time-stamp, host-name, port number and facility

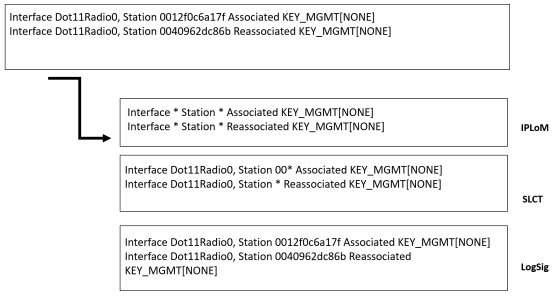

Fig. 8. Enumeration

in case of Syslog & time-stamp and prefix pattern in case of HDFS logs. Most of the parsers have the functionality for pre-processing inbuilt but human expertise is needed to identify these segments that needs removal, before applying the log parsing algorithm. Here we wanted to evaluate how the existing log parsers perform when cleaning is not undertaken. The primary motive is to establish and measure how much these algorithms are domain agnostic and sensitive to those structures and how generalized these algorithms are to automatically learn such features.

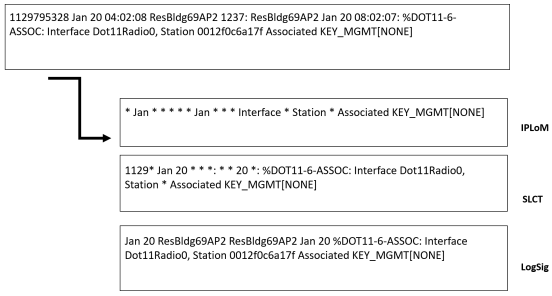

Fig. 9. Pre-processing

Figure 9 shows templates created without performing the expected PRE-PROCESSING. Two key observations were made across all parsers. First, the number of templates created increased significantly and Secondly, part of the raw log lines where pre-processing is normally expected were not templatized properly. IPLoM algorithm is the one that gave best results very close match as when generated after cleaning. However this resulted in too many wild-card replacements and fragmentation of templates because month related parts in the log message like *Jan*, *Feb*, etc. were not deduced as variables.

4 Conclusion & Future Work

We highlighted results from additional quantitative measurements (over and above the already reported performance figures in earlier work [17]) for the extracted templates. It was conclusive and evident that the generated templates were not easily human readable and also machine interpretable. Also, in some cases our measurements indicated sufficient variability and inconsistencies in number of generated templates. In this paper, we presented a discussion of observations, viz., what makes these algorithms unique and its respective challenges from the structure of generated templates.

Open challenges that remains in log template generation

- Fragmented templates
- Enumerations not captured
- Inappropriate delimiter handling
- Data types not inferred

In summary, there is a need for new algorithms that are domain agnostic, preprocessing free and ability to extract features without human or subject matter expert intervention for all possible formats. Towards this, in our future work, we are further studying tree-based clustering, text analytics and pattern mining algorithms to overcome these challenges.

References

1. Syslog, "Syslog — Wikipedia, the free encyclopedia," 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Syslog
2. Microsoft, *Windows Events : Simple Logging Facade for Java*, 2018. [Online]. Available: https://docs.microsoft.com/en-in/windows/desktop/Events/windows-events
3. Apache, *Apache Log4j 2 : a next generation logging framework for Java applications*, 1999-2018. [Online]. Available: https://logging.apache.org/log4j/2.x/
4. SLF4J, *SLF4J : Simple Logging Facade for Java*, 2004-2017. [Online]. Available: https://www.slf4j.org/
5. LOGBack, *LOGBack : Proposed successor framework for log4j*, 2018. [Online]. Available: https://logback.qos.ch/
6. M. Templates, "Message Templates : A language neutral specification for capturing and rendering structured log events," 2018. [Online]. Available: https://messagetemplates.org/
7. liblognorm, "liblognorm : Making sense out of syslog data into a well understood interim structured format," 2018. [Online]. Available: http://www.liblognorm.com/news/liblognorm-2-0-5-released/
8. Elastic, "Grok : A pattern based log parsing framework to create structured data from unstructured," 2018. [Online]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html
9. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.

10. Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 2009, pp. 149–158.
11. E. Chuah, S.-h. Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, "Diagnosing the root-causes of failures from cluster log files," in *High Performance Computing (HiPC), 2010 International Conference on*. IEEE, 2010, pp. 1–10.
12. L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 785–794.
13. R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*. IEEE, 2003, pp. 119–126.
14. R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 1–7.
15. K. Kurniawan, "Semantic query federation for scalable security log analysis."
16. S. Thaler, V. Menkovski, and M. Petkovic, "Unsupervised signature extraction from forensic logs," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 305–316.
17. P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *DSN'16: Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.
18. D. K. Ahirwar, S. K. Saxena, and M. Sisodia, "Anomaly detection by naïve bayes & rbf network," *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, vol. 1, no. 1, pp. pp–14, 2012.
19. W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. Citeseer, 2010, pp. 37–46.
20. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 117–132.
21. D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, "CRAWDAD dataset dartmouth/campus (v. 2009-09-09)," Downloaded from https://crawdad.org/dartmouth/campus/20090909, Sep. 2009.