Length Matters: Clustering System Log Messages using Length of Words

Keiichi Shima*

Abstract

The analysis techniques of system log messages (syslog messages) have a long history from when the syslog mechanism was invented. Typically, the analysis consists of two parts, one is a message template generation, and the other is finding something interesting using the messages classified by the inferred templates. It is important to generate better templates to achieve better, precise, or convincible analysis results. In this paper, we propose a classification methodology using the length of words of each message. Our method is suitable for online template generation because it does not require two-pass analysis to generate template messages, that is an important factor considering increasing amount of log messages produced by a large number of system components such as cloud infrastructure.

1 Introduction

The syslog mechanism and its protocols[6, 3] are widely deployed in various kinds of systems to collect system status messages from an informational level to a critical level in a standardized way. Since the original syslog protocol did not define any message body structure, the log messages are typically in free form text messages. The newer syslog protocol specification[3] tried to organize semantic structures in the body part of a message, however, not many programs respect the specification so far. Moreover, the decision on whether to use the newer format or not depends on vendors, software, or even individual programmers sometimes, we anyway have to handle both old and new message formats.

There are various approaches to infer log message templates. SLCT[9] is one of the basic approach to infer message templates without any prerequisite knowledge. SLCT is a two-pass template inferring method. In the first pass, it counts the number of words that appear in the entire log messages and find frequent words and its positions in a message. The more frequently a word appears, we can guess the word is likely a fixed keyword of the template message. For example, a message "`interface eth0 up`" is covered by the template "`{(interface, 0), (up, 2)}`" that means the keyword "`interface`" is at the position 0, and "`up`" is at the position 2.

LogCluster[10] is similar to SLCT but addressing shortcomings of SLCT. SLCT creates templates as a set of pairs of a word and its position. Because of this, SLCT is sensitive to the position of words. LogCluster allows variable length of parameters between fixed words. For example, "`interface eth0 up`" and "`interface HQ Link up`" are covered by one template "`interface *{1,2} up`", where "`*{1,2}`" means 1 or 2 wildcard words. Same as SLCT, LogCluster requires a two-pass processing to detect the list of frequent words.

Xu, el al. proposed a method using source code knowledge to infer message templates in [11]. This is useful when we know what kind of software are used in the target operation system. This approach requires preparation before classifying log messages. It also requires to update the inferred log template when software used in the target system is added or updated.

Kimura, et al. introduced a character class based

*IIJ Innovation Institute, Inc.

clustering method in their work[5]. They defined 5 classes of words each consists of only numbers, numbers and letters, symbols and letters, only letters, and only symbols respectively. The latter classes are considered more important than the former classes. The weight values for how much emphasize each class are pre-calculated based on a PA-I supervised leaning algorithm[2]. When comparing two messages, the ratio of the number of classes included in each message is used. The more the messages are similar, the ratio will get closer to 1.

SHISO[7] is another template generation method focusing on online processing. It calculates a property of words as a vector by counting types of characters included in a word such as capital alphabets, lower alphabets, numbers, marks, and so on. SHISO computes a Euclidean distance between words of two messages being compared and generates similarity index of two messages. If the index is smaller than the pre-defined threshold, SHISO infers the two are similar and makes a cluster.

2 Analysis of Messages in the Wild

2.1 Properties of Word Length in Messages

Since syslog messages are printed by programs, each message has a pre-formatted style. Figure1 shows some examples of system log messages.

As many previous works explained, a message comprises of two kinds of components, one is a fixed component and the other is a variable component. In the first line in Figure1, assuming that we know the head of the message contains date information and host information, `sshd`, `6845`, `vyatta`, and `41.190.192.158` are variable components, while `Invalid`, `user`, and `from` are fixed components. Many existing methods try to classify these components based on some predefined knowledge, such as frequency of appearance, ratio of character type, and so on. Our simple question was that do we really need to consider the property of each word.

Figure2 shows different examples of similar messages that should be clustered into the same group[1]. If we read these messages, we can easily create a cluster of "`postfix/cleanup[*]: *: message-id=*`" for the first group, and "`sshd[*]: Invalid user * from *`", where "`*`" means a variable component because we have knowledge of what is a process identifier, or what is an IP address to infer which parts are fixed and which are not. But even though we do not use such knowledge, here is another factor we can read from these examples, that is the length of each word. It is obvious that all the fixed components have the same word length in the message. Variable words have different word length, but tend to have similar length because they share the same context, such as a message identifier, an IP address, a process identifier, a host/user name, and so on.

Figure3 shows examples of distribution of word length of some messages groups. As we can read from the figures, each syslog message has a unique pattern of distribution of length of each words. The length of the first position is usually fixed because a process name is printed here normally. The second position is process identifiers and it is usually 3 to 5 digits. The 7th position of Figure3(a) is a placeholder for IP addresses and contains either IPv4 or IPv6 address. Because IPv6 address can be printed shorter by eliminating zero fields, the position has wider range of length. The 9th position of Figure3(b) is a placeholder for host names. In our data, the median of the length of the position was 29 and almost fixed however, we saw some very short and long host names in the log.

We analyzed how much the set of length of words are correlated each other using the syslog message dataset #2 shown in Table 2. The dataset is a collection of messages gathered from hypervisors operated by the WIDE project[2]. The distribution of the number of words of each message (excluding date and host name information) is shown in Figure4.

The most popular message group was those whose number of words was 11. We then made 27 message templates by half-manual way to split the messages

[1]Note that some messages are folded in the middle of the message due to the limitation of page width.

[2]`http://www.wide.ad.jp/`

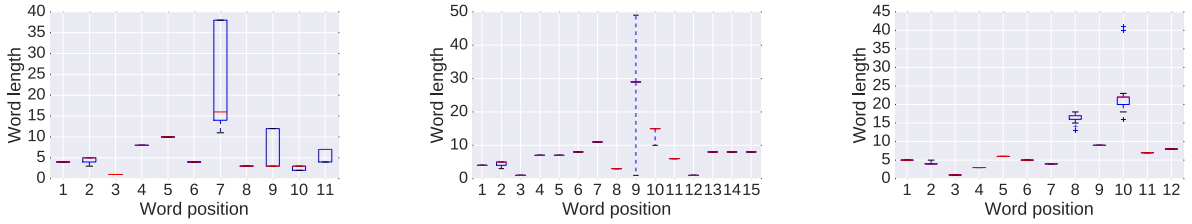
```
Oct  1 00:12:51 backup sshd[6854]: Invalid user vyatta from 41.190.192.158
Oct  1 00:12:51 backup sshd[6854]: input_userauth_request: Invalid user vyatta [preauth]
Oct  1 01:02:55 backup CRON[7069]: pam_unix(cron:session): session closed for user root
```

Figure 1: Examples of syslog messages.

```
Dec  1 00:05:01 vm1.example.com postfix/cleanup[2767]: 7EF561405E3:
message-id=<20151130150501.7EF561405E3@vm1.example.com>
Dec  1 00:10:01 vm1.example.com postfix/cleanup[3247]: 898FD1405E3:
message-id=<20151130151001.898FD1405E3@vm1.example.com>

Dec  1 00:27:27 backup sshd[15406]: Invalid user admin from 222.186.30.174
Dec  1 04:29:58 backup sshd[16287]: Invalid user a from 218.38.12.218
```

Figure 2: Examples of groups of syslog messages.



(a) `sshd * : Received disconnect from * 11: * * *` (79182 samples)

(b) `sshd * : reverse mapping checking getaddrinfo for * * failed - POSSIBLE BREAK-IN ATTEMPT!` (11926 samples)

(c) `named * : DNS format error from * resolving * invalid response` (878 samples)

Figure 3: Examples of word length distribution patterns of some syslog messages extracted from the messages on 1st October 2015 of the dataset #2 in Table 2.

as shown in Table 1. Figure5 shows the similarity matrix between two templates of Table 1. We found that it would be possible to distinguish two templates by measuring distance between them in most cases.

We also found that in some cases, the distance of two template messages is quite small even though the messages are completely different. We analyze such cases in Section 2.2.

2.2 Properties of Positions of Words in Messages

The one of the primitive clustering methods of syslog messages is creating clusters based on the number of words in the messages. Since the messages are printed based on pre-defined styles, the resulting messages will have the same number of words usually. Because this clustering method is too primitive, we need to create sub-clusters using other information. In the previous subsection, we have mentioned that each syslog message cluster tends to have a similar series of word length values. This observation

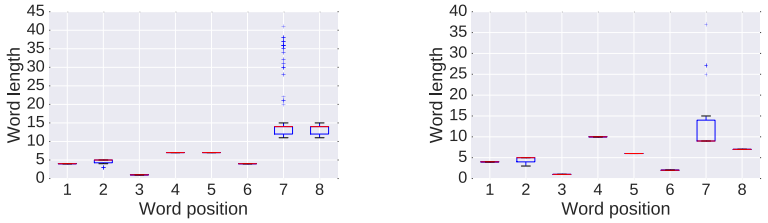
Table 1: Message Templates of Syslog Messages with 11 Words Extracted Half-Manually from the Messages on 1st October 2015 of the Dataset #2 in Table 2.

#	Template	# of msgs
0	40grub2: debug: parsing: if $linux_gfx_mode ! text ; then load_video; fi	5
1	40grub2: debug: parsing: menuentry Memory test memtest86+, serial console 115200 {	5
2	40grub2: appears to be an automatic reference taken from another menu.lst	14
3	CRON * : pam_unix cron:session : session closed for user *	8562
4	CRON * : root CMD cd / && run-parts --report /etc/cron.hourly	768
5	kernel: * Buffer I/O error on device loop0p1, logical block *	10
6	kernel: * INFO: task * blocked for more than 120 seconds.	12
7	kernel: * init: * main process * terminated with status *	18
8	kernel: 173315.040098 EXT4-fs vda1 : error count since last fsck: 3	1
9	kernel: * EXT4-fs vda1 : * error at time * *	2
10	kernel: * init: Failed to obtain startpar-bridge instance: Unknown parameter: INSTANCE	2
11	kernel: * systemd-logind 2127 : New session * of user *	1080
12	named * : client * view world: query cache * denied	3120
13	named * : error unexpected RCODE * resolving * : *	2224
14	ntpd * : Listen normally on * * * UDP 123	29
15	sshd * : Disconnecting: Too many authentication failures for * preauth	409
16	sshd * : User * not allowed because account is locked	1003
17	sshd * : Received disconnect from * 11: Bye Bye preauth	48299
18	sshd * : Received disconnect from * 11: disconnected by user	30883
19	sshd * : fatal: Write failed: Connection reset by peer preauth	21
20	sshd * : pam_unix sshd:session : session closed for user *	20525
21	su * : pam_unix su:session : session closed for user *	9
22	postfix/flush * : fatal: config variable inet_interfaces: host not found: *	5
23	postfix/master * : warning: process /usr/lib/postfix/flush pid * exit status 1	5
24	postfix/smtpd * : SSL_accept error from unknown * : lost connection	1
25	postfix/smtpd * : too many errors after DATA from unknown *	1
26	rpcbind: connect from * to dump : request from unauthorized host	14

helps to make sub-clusters in a cluster that has the same number of words in messages, however, there is some cases we need a different index other than word length values.

Figure6 shows the word length distribution patterns of two different syslog messages. In the later Section 3, we will discuss how to compare similarity between existing clusters and an incoming syslog message in detail, but in short, we use *Cosine Similarity* as a base idea for comparison. However, the two messages shown in Figure6 are quite similar in the sense of cosine similarity.

It is obvious for us to conclude that these two messages are different. The two messages in Figure6 only share the first word (and the third word ":") which is the name of the process that wrote the messages. The rest of the fixed components of the messages are completely different. So we also focus on the word positions of messages. If the words of two messages being compared doesn't have shared words in the same position, then we can think they should be clustered to different groups.



(a) `sshd * : refused connect from * *` (762 samples)

(b) `sshd * : Connection closed by * preauth` (16192 samples)

Figure 6: Examples of distribution patterns of word length of two completely different syslog messages where the distance between them becomes close.

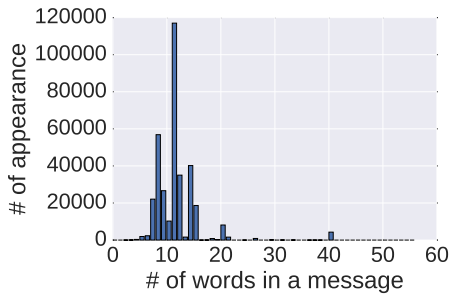


Figure 4: The distribution of the number of words of syslog messages on 1st October 2015 of the dataset #2 in Table 2.

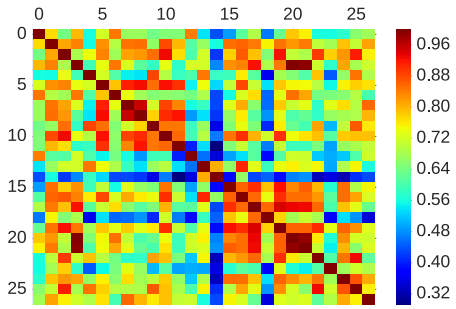


Figure 5: The similarity between two message templates shown in Table 1 using the cosine similarity over the vectors calculated by converting messages to vectors of the length of each word.

3 LenMa: Length Matters Clustering

Based on the observation in Section 2.1, we focus on the length of each words of the message as a similarity parameters of the message.

When clustering messages online, we need to compare the latest incoming message and existing clusters to decide which is the best cluster that should include the incoming message or create a new cluster if none of the existing clusters suites the message.

For the first line of the first group in Figure2, the length of words can be represented as a vector of

$$\begin{aligned}[][&len(\texttt{postfix/cleanup}),\\ &len(\texttt{2767}),\\ &len(\texttt{7EF561405E3}),\\ &len(\texttt{message-id}),\\ &len(\texttt{<201511...}(snip)\texttt{...example.com>})]\\ &= [15, 4, 11, 10, 44]\end{aligned}$$

which is the same vector calculated from the second line of the first group. For the second group, the vectors are $[4, 5, 7, 4, 5, 4, 13]$ and $[4, 5, 7, 4, 1, 4, 13]$. If the vectors of two messages are similar, then we can

guess that these messages are in a same cluster.

The similarity S_c is calculated from the word length vectors of the existing cluster and incoming new message as shown in (1) using the cosine similarity.

$$\begin{aligned} \boldsymbol{V}_c &= [v_{c,0}, v_{c,1}, ...v_{c,n}] \\ \boldsymbol{V} &= [v_0, v_1, ...v_n] \\ S_c &= CosineSimilarity(\boldsymbol{V}_c, \boldsymbol{V}) \\ &= \frac{\boldsymbol{V}_c \cdot \boldsymbol{V}}{|\boldsymbol{V}_c|\ |\boldsymbol{V}|} \\ &= \frac{\sum_{i=0}^{n} v_{c,i} v_i}{\sqrt{\sum_{i=0}^{n} v_{c,i}^2}\sqrt{\sum_{i=0}^{n} v_i^2}} \end{aligned} \quad (1)$$

where $\boldsymbol{V}_c$ and $\boldsymbol{V}$ are the word length vectors of the cluster c and incoming message respectively. $v_{c,i}$ and v_i are the length of ith word of the cluster c and incoming message.

Cluster vectors are updated whenever a new message is integrated to existing clusters. A new word length vector is calculated as shown in Algorithm 1. If the length of the ith word of the cluster and a new message is same, the value is kept unchanged, otherwise, the length value is updated with the new length value of the ith word of the new message.

Similarly, A new word vector which keeps a template string of the cluster is updated as shown in Algorithm 2. $\boldsymbol{W}_c$ and $\boldsymbol{W}$ are ordered set of words of a cluster and incoming message. For example, the top message of Figure1 can be represented as [`sshd`, `6854`, `Invalid`, `user`, `vyatta`, `from`, `41.190.192.158`]. For the case of a cluster, some of the words are replaced with a wildcard mark as variables such as [`sshd`, `*`, `Invalid`, `user`, `*`, `from`, `*`]. When updating the word vector $\boldsymbol{W}_c$, if the ith word of the cluster c is different from the ith word of the input word vector $\boldsymbol{W}$, the word is replaced by a wildcard mark, otherwise the value is kept unchanged.

When we receive a new log message, the following procedure is executed.

1. Create a word length vector and word vector of the new message.

Algorithm 1 Update a word length vector

```
procedure UpdateWordLengthVector(V_c, V)
    for all v_{c,i} ∈ V_c, v_i ∈ V(i ← 1 ··· |V_c|) do
        if v_{c,i} ≠ v_i then
            v_{c,i} ← v_i
        end if
    end for
end procedure
```

Algorithm 2 Update a word vector

```
procedure UpdateWordVector(W_c, W)
    for all w_{c,i} ∈ W_c, w_i ∈ W(i ← 1 ··· |W_c|) do
        if w_{c,i} ≠ w_i then
            w_{c,i} ← *
        end if
    end for
end procedure
```

2. Calculate a similarity score between the new message and each cluster which has the same number of words.

3. If none of the cluster has a similarity value larger than the threshold T_c, a new cluster with the new message is created and returned.

4. Update the most similar cluster with the new message using the algorithms defined in Algorithm 1 and Algorithm 2.

5. Return the most similar cluster.

Algorithm 3 shows the above procedure. As we discussed in Section 2.2, we use the cosine similarity score to compare the new message and existing clusters, however it is not enough to determine a proper cluster in some cases. We need to judge if the fixed parts of the cluster template are similar enough to the fixed parts of the new message. To achieve this goal, we introduced a positional similarity index S_p based on the number of shared words at the same positions. The S_p is calculated as shown in (2).

$$S_p = |\{w_{c,i} = w_i(w_{c,i} \in \boldsymbol{W}_c, w_i \in \boldsymbol{W})\}| \quad (2)$$

Table 2: Syslog Datasets

Dataset #	Content of dataset
1	Public Security Log Sharing Site[1]
2	Hypervisor cluster operated by the WIDE project
3	Server cluster of our laboratory

where $\boldsymbol{W}_c$ and $\boldsymbol{W}$ are the word vectors of the cluster and incoming message, i is the position of the word in a template or a message.

When comparing a cluster template and incoming message, we consider how many shared words are there in the same positions. If the number is smaller than the pre-defined threshold T_p, the message is considered as out of the cluster.

Algorithm 3 Find or Create a cluster

```
C ← ∅
procedure FindOrCreateCluster(message)
    V ← CreateWordLengthVector(message)
    W ← CreateWordVector(message)
    Cand ← ∅
    for all [V_c, W_c] in C do
        if Similarity(V_c, V, W_c, W) > T_c then
            Cand ← Cand ∩ [V, W]
        end if
    end for
    if Cand = ∅ then
        C ← C ∩ [V, W]
        return [V, W]
    end if
    [V_c, W_c] ← HighestSimilarity(Cand)
    UpdateWordLengthVector(V_c, V)
    UpdateWordVector(W_c, W)
    return [V_c, W_c]
end procedure
```

Algorithm 4 Calculate similarity

```
procedure Similarity(V_c, V, W_c, W)
    if |V_c| ≠ |V| then
        return 0
    end if
    if W_c matches W then
        return 1
    end if
    S_c ← CosineSimilarity(V_c, V)
    S_p ← |{w_{c,i} = w_i(w_{c,i} ∈ W_c, w_i ∈ W)}|
    if S_p < T_p then
        return 0
    end if
    return S_c
end procedure
```

4 Implementation

We have implemented the proposed algorithm in Python and applied it with three different syslog datasets shown in Table 2. The dataset #1 is a public data provided by Chuvakin[1]. The dataset #2 is a set of syslog messages collected at the hypervisor cluster operated by the WIDE project. The dataset #3 is a set of syslog messages collected from servers of our laboratory including several hypervisors and service hosts such as web servers.

We introduced one heuristic approach assuming the beginning of a message includes a date string, a host name, and a process name that is syslog specific. Although RFCs says that the standard syslog message starts with the date information followed by a host name, a process name, and other components, we understand not all the syslog messages doesn't strictly follow the recommended message format. From our experience however, the first 3 components (date, host name, and process name) share the same context in most cases.

The processing cost of inferring template formats linearly depends on the number of inferred templates. As shown in the algorithm, the value of the threshold affects the final number of inferred templates. The smaller threshold generates less number of templates that contain more wildcard marks. Once the number of templates becomes stable, the algorithm can

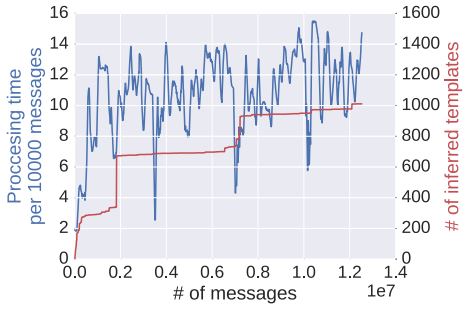


Figure 7: A processing time convergence graph measured with the messages in October 2015 of the dataset #2.

process messages within a certain fixed rate. Figure7 shows the processing time of messages in October 2015 of the dataset #2. The blue line indicates the time required to process 10000 messages and the red line shows the number of inferred templates. The processing time increases as the number of templates increases, however the number of the templates becomes stable once we have found most of them and the processing time also becomes stable.

We applied our algorithm and SHISO algorithm to cluster the datasets shown in Table 2. The result is shown in Table 3. We found a lot of kernel messages that appear when a system boots up in the dataset #2 and #3. In Figure7, such template messages are found at 2 points, the first point is the point around 200 million messages are processed, and the second point is where 700 million message are processed. We found a lot of kernel boot messages in the inferred templates. These messages appear only once while the system is running but generates a lot of different patterns that increases the total number of templates. The values in parentheses are the number of templates that are not related to kernel boot messages. This indicates that the online template mining mechanisms work well even for the infrequent messages, however cleansing of raw messages may be required to avoid unwanted template generation.

[3]SHISO proposes a second level clustering that merges templates that contains similar set of words regardless the # of words, however, in this paper we compare the result of the first level clustering result.

Table 3: Clustering Results

Dataset #	**# of templates found**	
	SHISO[3]	**LenMa**
1 (`log/secure*` only)	29	26
2	1093 (446)	1075 (404)
3	1113 (361)	891 (302)

The prototype code is available at GitHub[4].

5 Using Clustered Syslog Messages for Analysis

We tried to find unique syslog message patterns using the messages clustered by LenMa. We clustered all the messages using the algorithm and made message groups for every minute. The dataset used for this grouping is the dataset #3 and the period is from October to December 2015. Each group has its own distribution pattern of templates, however, many of them are similar each other. We counted the number of appearance of templates of each group, clustered them using the χ^2 test, and finally achieved 25 message group clusters out of 132480 (= 60 minutes $\times$ 24 hours $\times$ 92 days) groups. The frequently observed patterns are shown in Figure8, and the counts are 16235 and 115299 times respectively. Since each group is one-minute-long, almost 91 days out of 92 days match these patterns.

There were some unique patterns found from the result. Figure9(a) shows that there were not common ssh incoming activities. Figure9(b) was observed when one of the nodes in the target node group rebooted.

There are many approaches to detect anomalies or cluster messages based-on templates ([11, 5, 4, 8]). Our template mining technology can be used with these detection/clustering methods.

[4] `https://github.com/keiichishima/templateminer`

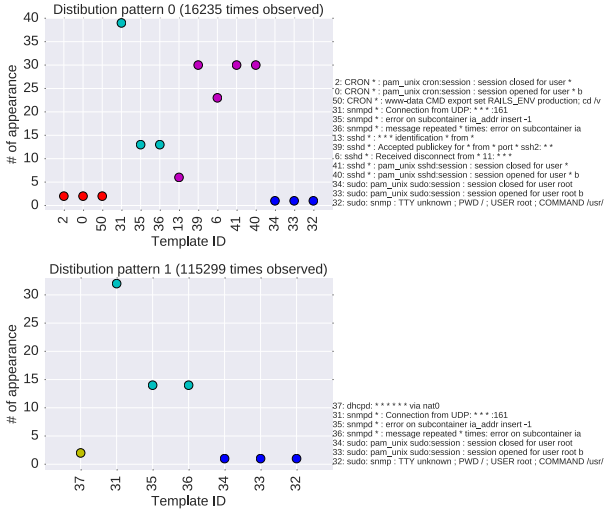


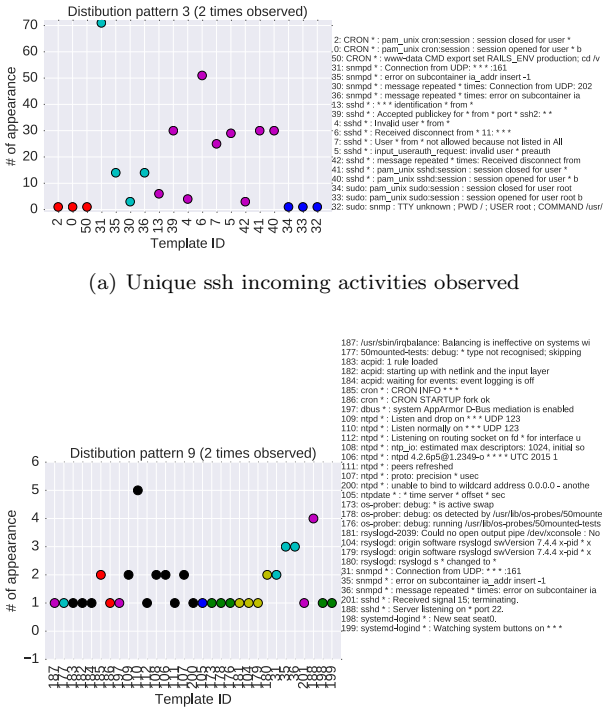
Figure 8: Frequently observed message group patterns where each color represents a process name group

6 Remaining Issues

In this section we discuss issues of our proposed method. Some of the issues discussed here are not specific to our proposal only, but are applied to online template mining methods in general.

The proposed algorithm doesn't take into account frequency of appearance of words. This causes state messages invisible from output. For example, `interface eth0 up` and `interface eth1 down` may generate a template such as `interface * *`. However, we may want two different templates `interface * up` and `interface * down` in some cases. Because we are focusing on online realtime template generation, it is difficult to predict a specific word is going to be a stable word like `up` and `down` or not.

How to determine the threshold value is an important factor in the method. If the value is too loose, the algorithm will generate more specific templates that will separate messages of same meaning to different groups. In this paper, we used 0.9 as T_c and 3 as T_p to cluster three different message sources (Table 2) achieved from different administrative groups.



(a) Unique ssh incoming activities observed

(b) Node rebooting observed

Figure 9: Examples of unique message group patterns observed

We could achieve the similar number of templates as SHISO could infer with the threshold values when applied to the standard Linux server syslog messages, however the proper values may be different when applied to other kinds of dataset.

7 Conclusion

In this paper, we have proposed a new clustering method for inferring system log message templates using the length of each words of messages. Many existing template mining approaches try to characterize words of messages by their character types, ratio of character types. Our proposal comes from the question that do we really need to investigate the word property so close. We have focused on the length of each word and found each message template has a

unique sequence of word length that can be used to cluster messages.

The proposed method is designed for use of online (one-pass) template mining. The two-pass methods usually generate better templates by surveying frequency of words to detect if a specific word is a fixed word or a variable word, however they require time before clustering and has difficulty to adapt dynamic changes of message trends. Recently many systems are implemented with dynamic components such as open source software. Such components are updated continuously, and even replaced with a different components providing the same functionality. In such a situation, adapting upgraded components and/or new components are important. Our proposed mechanism could produce similar number of templates as past works with less complicity of mining processing.

References

[1] A. Chuvakin. Public Security Log Sharing Site. `http://log-sharing.dreamhosters.com/`, 2010. accessed Jun 2016.

[2] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online Passive-Aggresive Algorithm. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.

[3] R. Gerhards. *The Syslog Protocol*. IETF, March 2009. RFC5424.

[4] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto. Spatio-temporal Factorization of Log Data for Understanding Network Events. In *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications (INFOCOM'14)*, pages 610–618, April 2014.

[5] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi. Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs. In *Proceedings of the 11th International Conference on Network and Service Management (CNSM 2015)*, pages 8–14, Nov 2015.

[6] C. Lonvick. *The BSD syslog Protocol*. IETF, August 2001. RFC3164.

[7] M. Mizutani. Incremental Mining of System Log Format. In *Proceedings of the IEEE International Conference on Services Computing (SCC 2013)*, pages 595–602, 2013.

[8] Q. Tongqing, Z. Ge, and D. Pei. What Happened in my Network? Mining Network Events from Router Syslogs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC'10)*, pages 472–484, 2010.

[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)*, pages 119–126, 2003.

[10] R. Vaarandi and M. Pihelgas. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In *Proceedings of the 11th International Conference on Network and Service Management (CNSM 2015)*, pages 1–7, 2015.

[11] 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 (SOSP'09)*, volume 4, pages 117–132, 2009.