# Text Mining and Its Applications

Shengyu Guo
School of Software Engineering
Tongji University
Shanghai, China
guo.shengyu@foxmail.com

Buyang Cao
School of Software Engineering
Tongji University
Shanghai, China
caobuyang@tongji.edu.cn

***Abstract*—As nowadays data centers are processing more jobs and collecting more data, the system status monitoring and analyzing functionality ensuring the availability, scalability and efficiency becomes more and more important. In order to build an automated status monitoring and alerting system， we need to group jobs performed at a data center upon jobs' characteristics. Since the job names are generated by system users at will, it is very hard to group them in order to monitor the job status efficiently. Thus we need to find some methods to sort out the system log, and help to group jobs that are beneficial for improving the accuracy and efficiency of the system analysis. This paper proposes a text mining algorithm and its application in grouping jobs for log analysis.**

***Keywords—Information Retrieval; Log Analysis; Text Mining; TF-IDF***

## I. Introduction

Hadoop system is very popular in big data applications nowadays. The scalability and efficiency of the system make it a perfect fit for large scale data processing. At the same time in order to meet the business demands from the real world, it is very critical to offer stable environment with high availability that in turn makes the system analysis and monitoring quite valuable [1][2]. To ensure the reasonable response time and in-time control of the system status, the operators of a data center need to check the system logs and extract useful information about the jobs running on the system. Unfortunately, as the data in the logs are unstructured and disordered, the analysis of the logs becomes a great challenge [3].

In this paper, we propose a data cleansing method for system logs, especially for grouping job names which are usually named at will by system users. The data cleansing component uses text mining technology to collect the job name information from the logs and sort out job names automatically. With the result of our cleansing system, the analysts are able to analyze the logs more conveniently and efficiently.

*Term Frequency-Inverse Document Frequency* (TF-IDF) is a method to measure the importance of one word to the document by numerical methods [4][5]. As a popular information retrieval technology, it is widely used as a weighting factor in text mining. TF-IDF is defined by

$$tf-idf_{i,j} = \frac{n_{i,\mathrm{j}}}{\sum_{i=1}^{J} n_{i,\mathrm{j}}} \times \log \frac{J}{\{\mathrm{j} \in \mathrm{J} : \mathrm{i} \in \mathrm{j}\}} \tag{1}$$

where $n_{i,j}$ stands for the occurrence of term $i$ in document $j$ and $J$ stands for the total number of documents .

As shown in (1), TF-IDF consists of two parts.

(1) Term frequency provides the information about how many times the term occurs in the document.

(2) Inverse document frequency provides the information how important this token is. For instance, stop words such as 'is' or 'the' exist in the document that are of little value but have a very high frequency will make negative effect on the extracted information. Jones [5] used the inverse document frequency to eliminate the stop word noise.

TF-IDF is constructed on the assumption that the high frequency terms (excluding those stop words with less value) make more sense to the semantics of the document. And common words like 'the' will be incorrectly emphasized rather than some more meaningful words.

The paper is organized as follows. In the second section we are describing typical Hadoop system log analyzing problem and presenting our textual mining idea to sort out log data. The third part presents the results of the experiment upon our algorithm and demonstrates the efficiency of our methods. And the paper is ended with some conclusions.

## II. System Description

### A. Overview

The target data of our job name cleansing component comes from the large system logs generated by Hadoop systems at the data center of one of the largest e-commerce companies of the world. The log file contains the names and users of the jobs to be performed as well as some system parameters such as CPU, memory, IO consumption, etc.

When a Hadoop system is being used, usually a user doesn't follow the job name convention completely or there is no such job naming standard at all. In this case it may cause troubles in analyzing system status and detecting abnormalities due to the difficulty in organizing/grouping/classifying jobs. If the same type of jobs could be clustered together based on job names and other related characteristics, or based upon analytical models, a system engineer would be able to figure out the running patterns for every kind (cluster) of jobs and then create rules for monitoring different types (clusters) of jobs and detecting the abnormal status of the operations more effectively. The engineer might stop abnormal jobs to ensure the stability and efficiency of the system [6]. To this end our data cleansing system is designed to clean and sort out the job names in the system logs to help the system analyses and failure detections.

In the following sections we are going to describe the job naming problems in the system logs and present our thoughts on job name tagging and some optimizing methodologies. The comprehensive relational analysis between job names and users who launch these jobs are conducted as well.

## *B. Problem description*

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

As discussed above, there are certain complicated job names appearing in the Hadoop system logs. The following table lists some job name samples.

TABLE I. SAMPLE JOB NAMES

| No. | Class | Job Name |
|---|---|---|
| 1 | CalCoordinator- | CalCoordinator-0329-154707--136425752-step1 of 1 |
| | | CalCoordinator-0329-154844--1719221316-step1 of 1 |
| 2 | SimilarityFeaturesForConcepts | SimilarityFeaturesForConcepts$-0328-170950--560893234-step_1_of_5 |
| | | SimilarityFeaturesForConcepts$-0328-170950--560893234-step_2_of_5 |
| 3 | SNAPSHOT FILE GENERATOR | SNAPSHOT FILE GENERATOR input=/sys/edw//ngdf_ckbid_rltd/sequence/2014/03/26/00/,output=/sys/edw//ngdf_ckbid_rltd/snapshot/2014/03/26/00 metadata=NGDF_CKBID_RLTD.xml |
| | | SNAPSHOT FILE GENERATOR input=/sys/edw//ngdf_dtmrt_rltd/sequence/2014/03/26/00/,output=/sys/edw//ngdf_dtmrt_rltd/snapshot/2014/03/26/00 metadata=NGDF_DTMRT_RLTD.xml |
| 4 | AspectSupply | AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] |
| | | AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] |

The job name of the first job type 'CalCoordinator' is composed of the job functionality 'CalCoordinator', the timestamp, and the step information. As the job just has one step, the timestamp is the only distinguishable character for this kind of job names. The second kind of job called 'SimilarityFeaturesForConcepts' has the same job name structure. Since the job has 5 steps, the job names are different in timestamp and step number. The naming structure of 'SNAPSHOT FILE GENERATOR' is a bit more complex. The job name for this type of jobs consists of the job functionality, input, output file name and the metadata file information. As shown in the table, those three files are completely different in the samples. The job name for 'AspectSupply' is the most complicated one, since the job name has the job functionality, step information, and several running parameters. Furthermore, there are truncated words in the job name, like '…avail' in the first job name stands for some words ending with 'avail' and '…m' indicates some words having 'm' in the end.

While analyzing the system logs, the analyst needs to group the jobs based on their functions to get the job patterns that is vital for system status monitoring. Unfortunately it is very hard to group jobs upon the current system logs due to the name diversities even for the same type of jobs. Certain mechanism needs to be created in order to meet the requirements of system status monitoring and failure detection.

## *C. Regular expression replacement*

In order to sort out the job names, the first conceivable method could be to find the pattern in each kind of job names. For the jobs listed in table I, the pattern of 'CalCoordinator' can be 'CalCoordinator- [0-9]{4}-[0-9]{6}-[0-9]{9,10}-step1 of 1'. Based upon the pattern for a job name we can create regular expression replacing methods to clean the job name. For example, for the first job listed in table I we replace the timestamp and step information by symbol '~'. Thus the job name will be 'CalCoordinator-~-~' and all 'CalCoordinator' jobs can be grouped together. This method is very accurate but requires the potential name patterns for all jobs, which is almost impossible to establish in practice due to the large number of jobs.

The second way is to take the delimiters in the job names into account. For example, the job name for 'SimilarityFeaturesForConcepts' can be divided into job functionality and timestamp with step information by using delimiter'$'. If we create rules stating that the information after '$' should be replaced by '~' for this kind of job name, all 'SimilarityFeaturesForConcepts' jobs will be grouped together by being converted to 'SimilarityFeaturesForConcepts$~'. Albeit this method is relatively easy to implement, the accuracy is bad since we exclude the step information.

After having analyzed the shortcomings of the first two methods, we focus our attention on the elements in the job names that affect the grouping. By having studied large numbers of job names, we find the affected elements can be classified into timestamps, file paths, function parameters, step information, and truncated words. If we could invent some regular expression replacing methods to deal with each element, we might be able to manage cleaning the job names and grouping the same kind of jobs with fairly good accuracy. For timestamps, we create some regular expressions including '20[0-9]{2}\\-[0-9]{2}\\-[0-9]{2}' for all possible timestamps and replace them with '~'. We also apply some regular expressions with keywords like 'input', 'output' and 'metadata'(as shown in the third example in table I) to judge whether we encounter a file path like 'input=[0-9a-zA-

Z\s_/]{0,1024}’, ‘output=[0-9a-zA-Z\s_/]{0,1024}’ and ‘metadata=[0-9a-zA-Z\s_/]{0,1024}’. The step information can be treated in the similar way using regular expression replacing. Yet the truncated words such as the ‘…avail’ and ‘…m’ in the fourth example, are impossible to use regular expressions for the purpose of description and replacement. The replacing rules described here are easier to implement than the first method and the accuracy is much better than the second one. Nevertheless, we are still confronted with the issue of reading a lot of job names and creating rules for each affected element.

We summarize the three regular expression replacing methods in table II.

TABLE II. REGULAR EXPRESSION REPLACEMENT METHODS

| Method | Creation | Accuray | Generalization |
|---|---|---|---|
| Patterns for all job names | Hard | High | Low |
| Delimiters for separation | Easy | Low | Farily high |
| Patterns for elements | Fairly hard | Farily high | Farily high |

The patterns required by the first method are hard to recognize and they cannot be applied to new jobs. The second method sorts out the job name by using delimiters, but the accuracy cannot be guaranteed. The delimiting rules may be applied to new jobs because jobs often have similar delimiters to some degree. The third method tries to establish the patterns for all affected elements thus it requires fairly large amount of knowledge about the job names. The accuracy is fairly good and the replacing rules can be used for new jobs since job names have similar affected elements. As discussed above, there are no perfect ways to sort out job names based on regular expression replacement.

## *D. Job name tagging*

In order to solve the job name cleansing problem with more satisfactory results, we take some other information into consideration. In the system logs there are parameters about job processing and the submitter. As we all know, the user and the job are related to some degree. It is common that in a large system, one user submits similar jobs many times. And it is also a common practice that a small group of users is usually responsible for one kind of jobs. Therefore we may make an assumption that the job ‘topic’ can be predicted for one particular user by relating a user to a specific job name.

As discussed above the text mining technology is being employed to discover high-quality information in documents. Here we apply it by treating one token in a job name as the term and one particular user as a document. The token frequency and user frequency is calculated to get the weight for each token. As the result we choose the most important token(s) for each user as a tag library for this user, and at the end we use the tag library to generate the tags for each job name.

The job name tagging procedure is presented in Figure 1. As shown in Figure 1, the job names are cut into tokens based on the delimiters in the first step. We exclude all numeric tokens because in most cases they only inform timestamp and ID. The ‘step’ information appeared in the numeric token will be discussed later. After this first step we build one token libary for every user.

```
1)  Generate user token libraries from the job name library J and user library U
    For i:= 1 to size(J)
        Token_list := tokenize(J[i]);
        token_list := excluseNumericToken(token_list);
        user_token_lib.add(u, token_list)

2)  Calculate weight for each token
    For i:=1 to size(U)
        For j:= 1 to size(U[i].token_list)
            tk = calculateTF(U[i].token_list[j]);
            iuf = calculateIUF(U[i].token_list[j]);
            Weight = tk × iuf;
             user_token_lib.addWeight(U[i].token_list[j], weight);
3)  Select tags from the tokens
    If USE_SYSTEM_METHOD == TRUE
        For i:=1 to size(U)
            Tag_list = selectTag(user_token_lib.token_list[i],METHOD, par);
            User_tag_lib.add(U[i], tag_list);
    If USE_SELF_METHOD == TRUE
        selfDefinedSelectTag(user_token_lib);

4)  Tag all the job names
```

Fig. 1. Pseudocode for job name tagging

In the second step, for a user token library *j* and total *N* tokens we define the token frequency for token t as follows

$$tf_{t,u} = \frac{n_{t,u}}{\sum_{i=1}^{N} n_{i,u}} \qquad (2)$$

where $n_{k,u}$ is the number of occurring for token *k* in all the job names belong to user *u*. By utilizing this token frequency we are able to find the important tokens for each user. Some words such as ‘file’, ‘input’, ‘metadata’, ‘sys’ etc. are occurring quite often and they will be easily recognized as the important tokens. But the information provided by these common tokens is not valuable for cleansing job names. Therefore we need to conduct some statistics on all token libraries rather than only one token library. We define inverse user frequency for token t belonging to user u in token library U as

$$iuf_t = \log\frac{M}{\{u \in U : t \in u\}} \qquad (3)$$

In this definition, M is the total number of users. U means the set for all user token libraries and {u∈U: t∈u} stands for the number of user libraries in which token t appears. The weight for the token, which we define as token frequency-inverse user frequency for token t of user u, is calculated by

$$tf - iuf_{t,u} = tf_{t,u} \times \mathrm{iuf}_t \qquad (4)$$

Now we have the weight for each token. In the next step (the third step) we need to select tags out of tokens based on the weight or other requirements for analyzing the system logs. The job name cleansing component provides the interface for analysts to define the selection method. If the user wants to choose tags by weight, the system will provide *THRESHOLD* method that selects tokens having weight larger than a predefined value. *NUMERIC_SORT* method selects tokens having the highest weight and *PROPORTION* method picks the tokens upon their ranks respectively. The system also provides the analyst with the interface for designing their own selection method.

After creating the tag library for every user, we will scan all job names in the system logs. If we find some words in the job name are in the tag library that belongs to a specific submitter, we will add these words to the tag result (the forth step). The whole tag result will be some tokens describing a job relatively accurately.

During the process of applying our job name tagging method, we found the algorithm can be improved further. Usually the first token of each job name has the higher value for job information. As shown in table I, all first words in the job names represent the functionality of these jobs. And it is a common case where the system user names the job with the functionality first. Hence our system provides an option for the analysts to choose whether they want to reserve the first token. It is easy to achieve it by putting the first token to the tag result in step 4 of the algorithm described above. If no tags are found in one job name, we will output some top ranked tokens in that job name as tag result.

As discussed in the previous section, the 'step' information of a job would be lost during tokenization because we omit all the numeric tokens. In order to retain this information, a dictionary of expressing the step information needs to be created. With this dictionary we can then take actions in step 4 to put the 'step' back to our tag result. There are also interfaces for analysts to define a valuable keyword dictionary and the contents of this dictionary will appear in the tag result.

## E. *Precondition test*

In this section we look back at the basis of the token frequency-inverse user frequency theory. TF-IDF is built up on the following assumption:

- The term with high frequency makes more sense to the semantics of the document.
- There are stop words in the document.
- Most of the documents are independent and focusing on different topics.

In order to validate the applicability of TF-IUF we need to confirm these three conditions are satisfied.

The higher frequency a token is, the more significant to the job name is. Whenever the user names a job, he will definitely include the functionality and meaningful keywords in the job name, which makes these keywords (and functionality name) appear more frequently. But we also need to consider some scenarios where a token is used by a large number of users frequently that is actually some kind of 'stop word' such as 'sys', 'file', 'user', etc. in the job names. Therefore these kinds of tokens are less meaningful comparing to the tokens with lower usage frequency. These verify the first two assumptions.

Below we use vector space model to observe the job name relationships among users [8][9][10]. After step 1 in Figure 1, we have obtained token libraries for all users. For certain user we represent his job name tokens as a vector $U_i = (w_{1,i}, w_{2,i}, ..., w_{T,i})$ . Each element of this vector corresponds a specific token occurring in job names collected for all users. And the weight for each token is calculated by TF-IUF (4). To find the relationship between tokens belonging to two users, we need to calculate the cosine of the angles $\theta$ between $U_i$ and $U_j$.

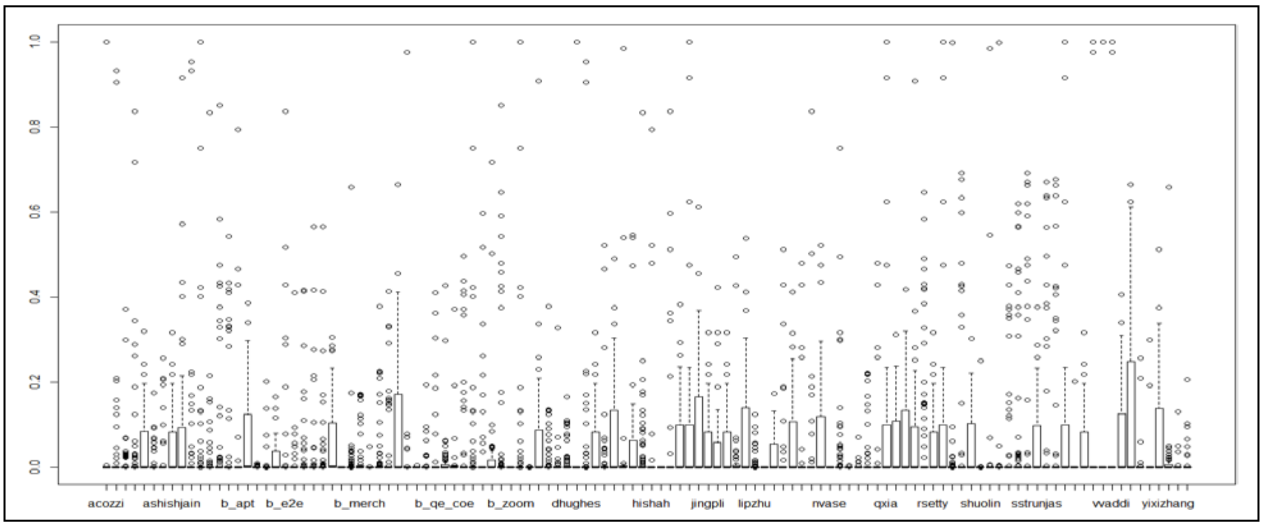

Fig. 2. Boxplot of job name relevance among users

If the cosine $\theta$ between the two vectors equals to 1, it means that the tokens are all the same [11]. On the other hand, if the value of cos is 0, it means the tokens are not related at all. We have computed the relevance among all users for the dataset we collected and the result is shown in Figure 2. The X axis represents the users and the Y axis is for the cosine relevance between one user and all the others. As shown in the boxplot, the relevance is below 0.2 for most of the cases. This outcome indicates the job names of different users are significantly different. We can conclude that the job name of individual user is almost unique indicating the third assumption mentioned above: most documents are focusing on different topics.

In summary TF-IUF is applicable for our cases.

## III. Computational Experiments

In this section we present three computational experiments on the job name cleansing. In the first one we use one of the regular expression replacing methods to sort out the job names. For the second experiment the text mining method is employed. The third experiment is conducted upon the improved name tagging methodology.

### A. *Experiment of Regular Expression Replacing*

We use the third method called 'patterns for elements' which is shown in table II to clean the job names in the first experiment. After having studied a lot of job names, we create some regular expression replacement rules for affected elements. Some sample rules are illustrated below.

- For timestamp, we replace' 20[0-9]{2}\\/[0-9]{2}\\/[0-9]{2}\\/[0-9]{2}\\/[0-9]{2}' with symbol '~'.
- For ID we replace 'yosemite-eod.[(A-Za-z0-9]{10}' with symbol '~'.
- For file name we replace '\\/[a-z0-9]{8}\\-[a-z0-9]{4}\\-[a-z0-9]{4}\\-[a-z0-9]{4}\\-[a-z0-9]{12}' with symbol '~'.

The result for the system logs collected in one week is shown in Figure 3.

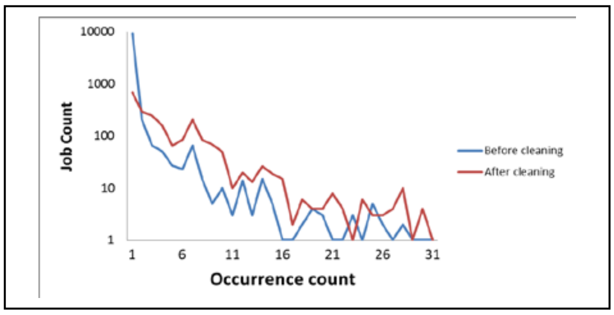

Fig. 3. Logarithmic Scale of Job Name types with the same Occurrence Number (Regular expression Replacing Method)

Based on Figure 3 we are able to find out that the job names occurring less frequently dropped dramatically. There is an increase for job names occurring 7 times or more because many daily jobs for this week were grouped together. Note that the job names occurring more than 32 times are not included in this figure.

TABLE III. Statistics of experiment 1

| | Job Name Types | Job names occurring once |
|---|---|---|
| Before Replacing | 10914 | 10291 |
| After replacing | 2188 | 697 |
| Reduced | 80.0% | 93.2% |

Table III gives us some statistical insights on the cleansing. We can find the big drop of the job types and one-time executed job, which means that the method has impact on majority of job names in the system log. The sample results of the job name replacements of table IV are presented below.

TABLE IV. Experiment 1 result samples

| Before replacing | After replacing |
|---|---|
| CalCoordinator-0329-154707--136425752-step1 of 1 | CalCoordinator-~step1 of 1 |
| CalCoordinator-0329-154844--1719221316-step1 of 1 | CalCoordinator-~step1 of 1 |
| SimilarityFeaturesForConcepts$-0328-170950--560893234-step_1_of_5 | SimilarityFeaturesForConcepts$-~step_1_of_5 |
| SimilarityFeaturesForConcepts$-0328-170950--560893234-step_2_of_5 | SimilarityFeaturesForConcepts$-~step_2_of_5 |
| AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] | AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] |
| AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] | AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] |

The experiment results for 'CalCoordinator' and 'SimilarityFeaturesForConcepts' are quite good. But for 'AspectSupply' the rules are not functioning well because its structure is rare and hard to create patterns.

### B. *Experiment of Job Name Tagging*

In this experiment we apply the algorithm introduced in Figure 4 to the same system log used in experiment 1. The tags are select by THRESHOLD method, where the value is set to be 0.02. The result is shown in Figure 4.

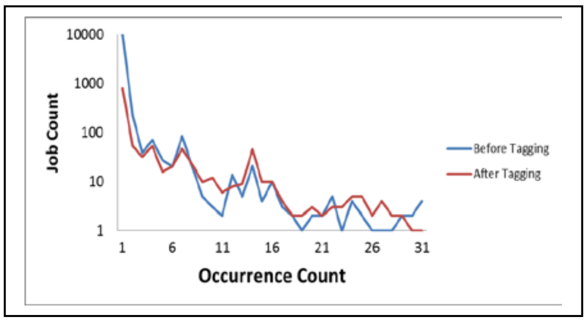

Fig. 4. Logarithmic Scale of Job Name types with the same Occurrence Number (Tagging Method)

We can see that the job names occurring less than 10 times have dropped significantly. And in table V we present the statistical information as we did for the first experiment. Comparing to the replacing method, the tagging method generates fewer job types and jobs that occur just once.

TABLE V. STATISTICS OF THE EXPERIMENT 2

| | Job Name Types | Job names occurring once |
|---|---|---|
| Before Replacing | 10914 | 10291 |
| After replacing | 1263 | 786 |
| Reduced | 88.4% | 92.4% |

The sample results of name tagging are shown in table VI. For 'CalCoordinator' we are unable to get the step information of the job. The step information for 'SimilarityFeatures-ForConcepts' can be recognized. And the results for 'AspectSupply' are unfortunately blank because no tokens have weight that is bigger than the threshold. We will address this issue in the next experiment.

TABLE VI. EXPERIMENT 2 RESULT SAMPLE

| Before replacing | After replacing |
|---|---|
| CalCoordinator-0329-154707--136425752-step1 of 1 | CalCoordinator step1 of |
| CalCoordinator-0329-154844--1719221316-step1 of 1 | CalCoordinator step1 of |
| SimilarityFeaturesForConcepts$-0328-170950--560893234-step_1_of_5 | SimilarityFeaturesForConcepts step_1_of_5 |
| SimilarityFeaturesForConcepts$-0328-170950--560893234-step_2_of_5 | SimilarityFeaturesForConcepts step_2_of_5 |
| AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] | AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] |
| AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] | AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] |

## C. *Experiment of Improved Job Name Tagging*

To avoid the blank tag encountered in the previous experiment, the first methodology might be to reduce the weight threshold. Yet the less meaningful tokens would also be recognized as tags. Therefore we decide to reserve the first token of each job name and apply a step information pattern library while tagging, which is discussed in the last paragraph of subsection D in second section for improvement.

In this experiment we apply the same tagging function and threshold to the same data used in the previous two experiments. However, we always retain the first token in each job name. The result is shown in Figure 5 and Table VII. The job names occurring only once drops significantly and the method has reduced more job types than the one employed in the second experiment.

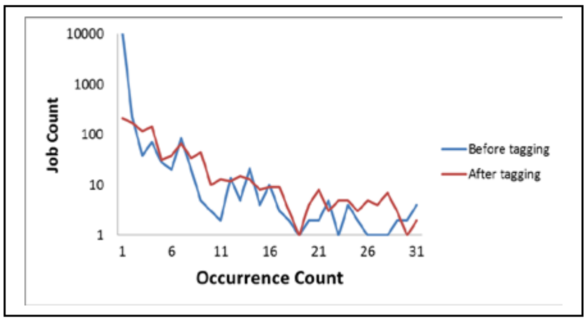

Fig. 5. Logarithmic Scale of Job Name types with the same Occurrence Number (Improved Tagging Method)

TABLE VII. STATISTICS OF THE EXPERIMENT 3

| | Job Name Types | Job names occurring once |
|---|---|---|
| Before Replacing | 10914 | 10291 |
| After replacing | 1116 | 208 |
| Reduced | 89.8% | 98.0% |

Some samples obtained in this experiment are presented in table VIII. Based on the result one is able to recognize that the entire step information is kept and the first token of 'AspectSupply' has been treated as the tag. Although the result is not perfect, for example the step of job 'AspectSupply' is lost, the achieved results are still quite valuable for cleaning job names to prepare the data for system status analysis.

TABLE VIII. EXPERIMENT 3 RESULT SAMPLE

| Before replacing | After replacing |
|---|---|
| CalCoordinator-0329-154707--136425752-step1 of 1 | CalCoordinator step1 of 1 |
| CalCoordinator-0329-154844--1719221316-step1 of 1 | CalCoordinator step1 of 1 |
| AspectSupply[(1/7) ...avail', 'wacko_yn', 'lstg_status_id', 'auction_format']]1][listings/97347/]] | AspectSupply |
| AspectSupply[(2/7) ...m', 'aspct_vlu_nm', 'ns_type_cd', 'aspct_src']]1][aspects_condition/28381/]] | AspectSupply |

## IV. CONCLUSIONS

In this paper, we propose a new text-mining based approach for job name cleansing which is quite worthwhile for system log analysis. Compared to the manual regular expression replacing method, the job name tagging proposed in this paper is running automatically and is able to achieve the goal of cleansing job names reasonably. Some improvement thoughts have been prototyped according to the characteristics and semantics of the job name structures in the system logs. Various computational experiments demonstrate the effectiveness of our methodology. The future researches will focus more on the further improvements of the method to make the name tagging results more valuable and dependable.

## ACKNOWLEDGMENT

Research for this paper is partially supported by CIUC2014004

## REFERENCES

[1] Singh K, Kaur R. Hadoop: Addressing challenges of Big Data[C]//Advance Computing Conference (IACC), 2014 IEEE International. IEEE, 2014: 686-689.

[2] Casolari S, Colajanni M, Tosi S. Detecting behavioral variations in system resources of large data centers[C]//Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on. IEEE, 2011: 371-378.

[3] Fu Q, Lou J G, Wang Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis[C]//Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. IEEE, 2009: 149-158.

[4] Aizawa A. (2003) “An information-theoretic perspective of tf–idf measures”. Information Processing & Management, 39(1): 45-65.

[5] Jones K S. (1972) “A statistical interpretation of term specificity and its application in retrieval”. Journal of Documentation, 28(1): 11-21.

[6] Kambatla K, Pathak A, Pucha H. (2009) “Towards optimizing Hadoop provisioning in the cloud”. ln: Proc. of the First Workshop on Hot Topics in Cloud Computing(2009).

[7] Zhang W, Yoshida T, Tang X. (2011) “A comparative study of TF* IDF, LSI and multi-words for text classification”. Expert Systems with Applications, 38(3): 2758-2765.

[8] Christopher D. Manning (2008) “Scoring, term weighting, and the vector space model”. In: Introduction to Information Retrieval: 100-123. Cambridge: Cambridge University Press

[9] Lee, D.L.; Huei Chuang; Seamons, K. (1997) "Document ranking and the vector-space model" , Software, IEEE , vol.14, no.2, pp.67,75, Mar/Apr 1997

[10] Salton G, Wong A, Yang C S. A. (1975) “Vector space model for automatic indexing”. Communications of the ACM, 18(11): 613-620.

[11] E. Garcia, "Cosine Similarity and Term Weight Tutorial," http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html, 2010.