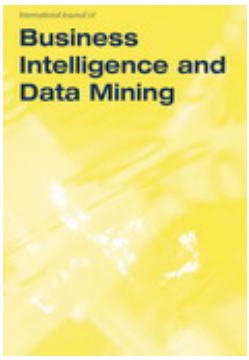


International Journal of Business Intelligence and Data Mining

ISSN online: 1743-8195 - ISSN print: 1743-8187
https://www.inderscience.com/ijbidm

Using unstructured logs generated in complex large-scale micro-service-based architecture for data analysis

Anukampa Behera, Sitesh Behera, Chhabi Rani Panigrahi, Tien-Hsiung Weng

DOI: 10.1504/IJBIDM.2022.10043252

Article History:
Received: 03 October 2020
Accepted: 03 August 2021
Published online: 30 November 2022

Using unstructured logs generated in complex large-scale micro-service-based architecture for data analysis

Anukampa Behera

Department of Computer Science and Engineering,
ITER,
S'O'A University,
Odisha, India
and
Department of Computer Science,
Rama Devi Women's University,
Odisha, India
Email: anukampa1@gmail.com

Sitesh Behera

Plivo Inc.,
Bengaluru, India
Email: sitesh.citzen@gmail.com

Chhabi Rani Panigrahi*

Department of Computer Science,
Rama Devi Women's University, India
Email: panigrahichhabi@gmail.com
*Corresponding author

Tien-Hsiung Weng

Department of Computer Science and Information Engineering,
Providence University,
Taichung 43301, Taiwan
Email: thweng@gm.pu.edu.tw

Abstract: With deployments of complicated or complex large-scale micro-service architectures the kind of data generated from all those systems makes a typical production infrastructure huge, complicated and difficult to manage. In this scenario, logs play a major role and can be considered as an important source of information in a large-scale secured environment. Till date, many researchers have contributed various methods towards conversion of unstructured logs to structured ones. However, post conversion, the dimension of the dataset generated increases many folds which are too complex for data analysis. In this paper, we have discussed techniques and methods to deal with extraction of all features from a produced structured log,

reducing N-dimensional features to fixed dimensions without compromising the quality of data in a cost-efficient manner that can be used for any further machine learning-based analysis.

Keywords: json data; micro services; data parsing; principal component analysis; PCA; multivariate data; unstructured data; tagged data; feature reduction; reverse indexed database; profiling.

Reference to this paper should be made as follows: Behera, A., Behera, S., Panigrahi, C.R. and Weng, T-H. (2023) 'Using unstructured logs generated in complex large-scale micro-service-based architecture for data analysis', *Int. J. Business Intelligence and Data Mining*, Vol. 22, Nos. 1/2, pp.248–263.

Biographical notes: Anukampa Behera received her MTech in Computer Science from the Biju Patnaik University of Technology, Odisha, India and currently pursuing her PhD in Computer Science at the Ramadevi Women's University University, Bhubaneswar, India. She is working as an Assistant Professor at the ITER, S'O'A Deemed to be University, Bhubaneswar, India and her research interests include security, anomaly detection, artificial intelligence implementation. In the current study, implementation of the modules with result analysis and overall systematic presentation of this research work has been conducted by her.

Sitesh Behera is currently working as a Senior DevOPs Manager in the Plivo Inc., India. He has 15+ years of experience in managing application hosted on large-scale infrastructures. His job involves, infrastructure design, development, management, capacity planning, automation of platform components, and problem solving using AI. He has a research interest in security and is an opensource enthusiast.

Chhabi Rani Panigrahi received her PhD in Computer Science and Engineering from the IIT Kharagpur, India. She is currently an Assistant Professor in the Department of Computer Science at the Rama Devi Women's University, Bhubaneswar, India. Prior to this, she was working as Assistant Professor in the Central University of Rajasthan, India. Her research interests include software testing, mobile cloud computing, and machine learning. She holds 20 years of teaching and research experience. She has published several international journals, conference papers, and books. She served as chairs and technical program committee member in several conferences of international repute.

Tien-Hsiung Weng is currently working as a Professor in the Department of Computer Science and Information Engineering, Providence University, Taichung City, Taiwan. He received his PhD in Computer Science from the University of Houston, Texas. His research interests include parallel computing, high performance computing, scientific computing, and machine learning.

1 Introduction

Logs generated from various systems are considered as one of the prime mechanism for the purpose of analysis and studying the behaviour of an IT infrastructure. The generated logs are regarded to be loaded with abundant information regarding the various events generated from different subsystems including network switch, service clusters, firewalls, office laptops, database, IPS/IDS solutions, etc. The data generated is huge in volume. Now, it is a humongous task to prepare the data for any kind of analysis as majority of the log portion are basically unstructured and sometimes semi-structured (Tovarňák and Pitner, 2019). For any automated processing data needs to be structured, normalised as well as suitable for visualisation. Hence various methods have evolved and proposed in the past decade to prepare the data for analytics. The journey starts with parsing the data and converting them to tuple format with later addition of count vectors (Lou et al., 2010). These works have enabled us to primarily convert an unstructured data to a structured format, thus creating a formatted dataset for us to attempt any kind of data analytics on the available data.

However, we still have another problem with the data volume being very high. Hence there have been some proposals to extract relevant fields out of the entire logline and maintain the quality of data by introducing some rules of extraction (Breier and Branišová, 2015). With this approach we get reduced volume of data and relevant fields which are needed for data analytics.

Having reached this stage we have identified that though we have relevant fields and reduced volume, we end up with very high dimensional multivariate data. Analysing multivariate data is complex, resource as well as time consuming. Hence, we would like to propose a solution on how to reduce the dimensionality of data to only three dimensions without losing the functionality of elements, retaining the quality of the data. This approach will reduce the complexity involved in data analytics and becomes very efficient method while dealing with real time systems.

The paper is organised as follows: Section 2 lists selected works done till date in the field of unstructured log analysis. Section 3 describes the suggested method elaborately and emphasises on the importance of data pre-processing. Experimental setups and results are presented in Section 4. Final conclusion is given on Section 5.

2 Related work

In this section, various methods proposed for using unstructured log towards the purpose of data analysis are discussed.

Xu et al. emphasised that, even a simple algorithm can achieve better results by improvising the pre-processing methods. For experimentation purpose authors used software monitoring system generated console log. Log parsing mechanism was used for processing unstructured logs (Xu et al., 2009). Lou et al. used unstructured console log for the purpose of anomaly detection after processing the log through log parser there by converting them to tuple form. Then similar messages were grouped together and uniquely represented using a unique count vector (Lou et al., 2010). Breier and Branišová used transaction blocks collected from several log sources. They created rules for identifying each category of deviations and used a unique identifier for spatial recognition of data. Towards the reduction of log analysis time, Hadoop technology

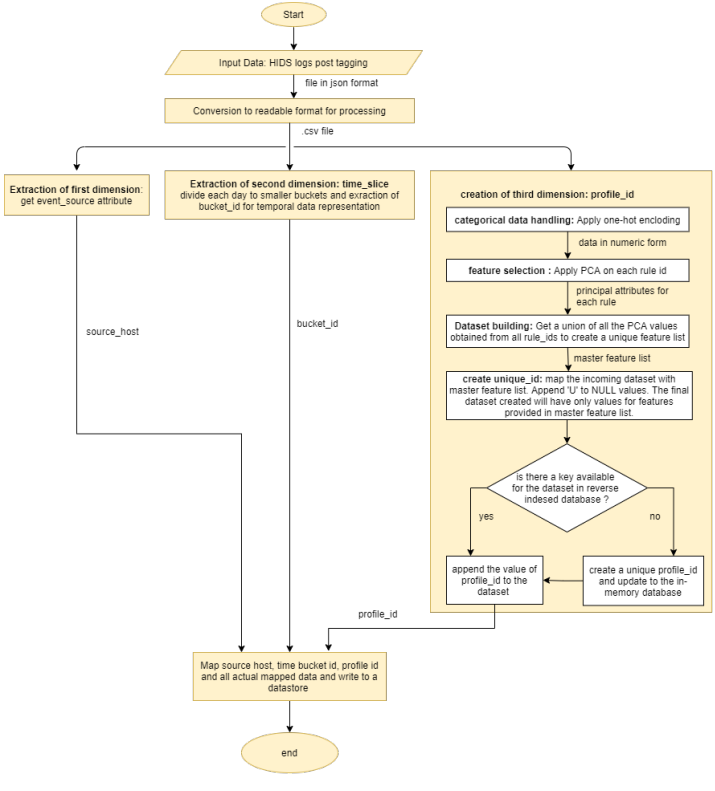
was recommended (Breier and Branišová, 2015). To cater the need for handling ad hoc queries in large in-memory logs, Tandon et al. introduced hardware accelerator namely HAWK. It used hardware pipeline for scanning a constant chunk of input data for further processing. The multiple characters input were examined in parallel within a span of single accelerator clock cycle (Tandon et al., 2016). Various other methods like log-line tokenisation, logs-key sequences, parsing based on attribute behaviour, etc. are used by several researchers for handling unstructured data. But, most of these works were based on standard datasets available. Tovarňák and Pitner suggested to convert the high velocity unstructured logs generated from IT infrastructure to streams of structured event objects (Tovarňák and Pitner, 2019).

How to use the high-velocity as well as very high-dimension data generated from todays complicated micro-service architecture, cost effectively is still a trending research topic. Our proposed method is a contribution towards achieving the above said requirement, which is described in next section.

3 Proposed method

This section describes the steps involved in collection of real life data to handling generated high volume of instance elaborately. A detailed flow chart of the entire process is given in Figure 1.

Figure 1 Flow chart of the proposed method (see online version for colours)

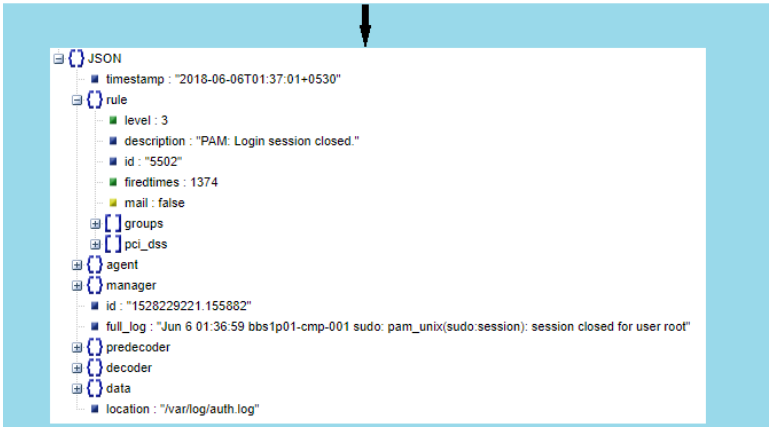


3.1 Data collection

Till date it is found that the best approach to get a structured and relevant data out of the logs is to implement host-based intrusion detection system (HIDS) and to use its output instead of working from the scratch with the raw data. HIDS is a holistic system that can monitor and analyse the internals of a computing system as well as the network package on its network interfaces (Wang and Zhu, 2017). A HIDS solution does all the works proposed by various researchers till date starting from collection of all logs from various sources, parsing them, adding counter vectors and attach them with rule IDs; thus converting all unstructured data to structured and tagged data. In Figure 2, a sample tagged output from a HIDS server in json format is shown which is structured as well as tagged.

Figure 2 Structured and tagged HIDS output in the json format (see online version for colours)


```
{"timestamp":"2018-06-06T01:37:01+0530","rule":{"level":3,"description":"PAM: Login session
closed.","id":"5502","firedtimes":1374,"mail":false,"groups":
["pam","syslog","gpg13_7.8","gpg13_7.9"],"pci_dss":["10.2.5"]},"agent":{"id":"016","name":"bbs1p01-
cmp-001","ip":"10.1.249.1"},"manager":
{"name":"BAVDCU16POSSEC1"},"id":"1528229221.155882","full_log":"Jun  6 01:36:59 bbs1p01-cmp-001 sudo:
pam_unix(sudo:session): session closed for user root","predecoder":
{"program_name":"sudo","timestamp":"Jun  6 01:36:59","hostname":"bbs1p01-cmp-001"},"decoder":
{"parent":"pam","name":"pam"},"data":{"dstuser":"root"},"location":"/var/log/auth.log"}
```

3.2 Conversion to readable format for processing

As data available in json format is not suitable for further processing, as the next step it must be converted to a more formatted and readable csv form.

Now this converted dataset available in .csv format is N-dimensional data which we have reduced to three dimensions in the following steps without losing any functionality and successfully retaining the data quality.

3.3 Extraction of first dimension – source host

For any data analysis the source of event is a compulsory factor to be known. Thus the first dimension that we retrieve from the obtained dataset is the ‘source host’ from where the event is originated.

3.4 Extraction of second dimension – time bucket

In this step, we take care of large volume of instances those are being produced by the various logs that we have considered. So, we divide the entire 24 hours logs to smaller sized buckets taken on a reduced time-stamp. Then each bucket must be marked with a unique identifier which can be passed to the model per event. Here every instance or sample is taken at a particular time. This instance belongs to a particular user and each sample has multiple features. The number is selected on temporal basis, i.e., we select the instances to create a subset based on a particular time period and this subset which is ready to be passed to any analytics model. Thus the subset of the dataset are all sample till a particular time and belong to all users who have tried to access the system in that stipulated time period. We mathematically represent the notation for a single sample which is taken at a certain time belonging to a certain user as follows:

Let $X_{n,t}$ denote a sample (feature vector) taken at a time t, belonging to an user n.

Let $D_{t<\tau\tau_m<t<\tau_n} \subset D = \{X_{n,t}\}|\tau_m < t < \tau_n$ denote the subset of all samples taken with a time τ_m and τ_n.

Let D denote the entire dataset that we selected after PCA.

A single sample for user n within a time period t can be represented as

$$X_{n,t} \in D \tag{1}$$

3.5 Obtaining the third dimension by handling large number of features challenge – profile ID

After obtaining the two necessary features, now the remaining of the N-dimensional dataset obtained needs to be processed towards the goal of obtaining the third dimensions. It involves several sub processes as listed below.

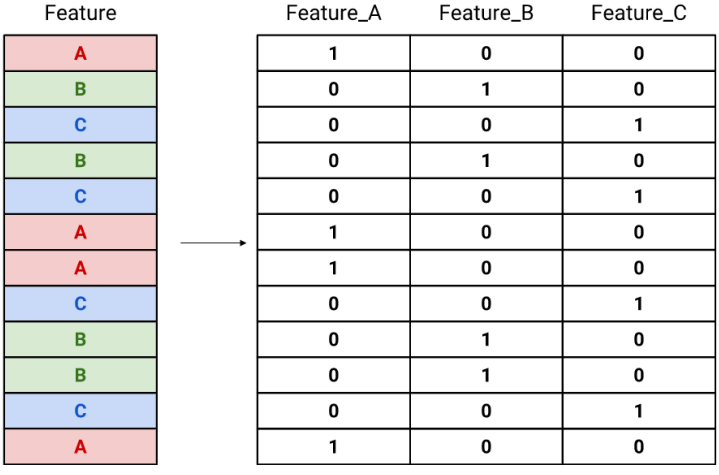
3.5.1 Conversion of categorical data to numeric data

Now, we get an output file which records a very high-dimensional data with most data being categorical and derived. Categorical data refers to variables that contain labelled or string values instead of numeric values. The number of unique variations for values present in the domain is often limited to a fixed set. These variables are often called nominal, like 'protocol' variable with the values: 'ICMP', 'TCP' and 'UDP'. If the values in the variable have natural relationship with each other they can be called ordinal. For example, a 'rank' variable having values like 'first' and 'second'. As many of the machine learning algorithms cannot work directly on labelled data. So we convert these categorical data to numerical data. An illustration of one-hot encoding mechanism is shown in Figure 3.

Now as the attribute acts as the as input variables we need to apply some encoding scheme to do the conversion, for which we are using one-hot encoding technique (Brownlee, 2017). It is a coding scheme that is used very commonly. In this scheme each level of the categorical variable is compared to a fixed reference level. In one hot encoding a single variable with x observations and p unique values are transformed to p binary columns with x instances each, where each instance depict participant (1) or non-participant (0) value of the dichotomous binary variable (Potdar et al., 2017). In Figure 3 we have illustrated the one-hot-encoding applied on the 'feature' attribute in a

sample dataset. In the 'feature' variable there are three categories as 'A', 'B' and 'C'. So it creates three binary variables 'Feature_A', 'Feature_B' and 'Feature_C' which act as dummy variables.

Figure 3 One hot encoding illustration (see online version for colours)



Feature
A
B
C
B
C
A
A
C
B
B
C
A

Feature_A	Feature_B	Feature_C
1	0	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	0
1	0	0
0	0	1
0	1	0
0	1	0
0	0	1
1	0	0

3.5.2 Feature selection

In this step, from a very large number of features we select a few features using principal component analysis (PCA) based on rule IDs. PCA is an unsupervised linear transformation technique. It is generally applied on high-dimensional data. Using PCA, we find the correlation existing between the features, through which we can find pattern. Using PCA the direction on which we get maximum variance is found and project it to a new subspace that has either equal to or less number of features than the original dataset was containing. Suppose we have (m, n) dimensional dataset where m is the number of samples and n is the number of features. So using PCA, we create a (m, p) dimension transformation matrix M such that it will allow mapping of a sample vector 'a' into a new p-dimensional subspace of features where $p < n$. Thus we transform (m, n) dimension to (m, p) dimension, i.e., with a feature set of p. When we arrange these features with descending order of variance, we get the first principal component with highest importance. Instead of dropping any features PCA transforms them linearly and creates new feature set (Raj, 2019). But as it is very sensitive to the range of data, we must scale the data before passing them to PCA. For scaling and to find the relevant features, we used the following steps as given below.

Step 1 The n-dimensional dataset was standardised.

Step 2 Co-variance matrix for n-dimension was drawn. Now covariance matrix can be found by finding covariance between pair of features using the following formula (https://education.howthemarketworks.com) to form the matrix.

$$cov(x, y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X}))(Y_i - \overline{Y})}{n - 1} \tag{2}$$

to form the matrix (https://math.stackexchange.com)

$$\sum = \begin{pmatrix} var(X) & cov(X,Y) & cov(X,Z) \\ cov(X,Y) & var(Y) & cov(Y,Z) \\ cov(X,Z) & cov(Y,Z) & var(Z) \end{pmatrix} \tag{3}$$

Step 3 The eigenvalue and eigenvector were found from the above matrix.

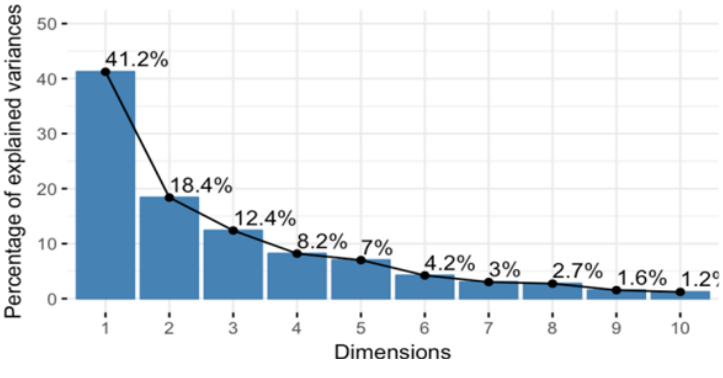
Step 4 Then the top p eigenvalues were found and then the corresponding eigenvectors were chosen.

Step 5 The projection of matrix M were constructed using top P eigenvectors.

Step 6 A scree plot to determine the number of principal components was drawn. A scree plot represents the plot of eigenvalues ordered from largest to the smallest. We determined a point, beyond which the remaining eigenvalues were all relatively small and of comparable size. This point determined the number of components (Vidal et al., 2016).

For example in Figure 4, we might want to stop at the fifth principal component. 87% of the information (variances) contained in the data are retained by the first five principal components. The number of component is determined at the point, now applying PCA based on the rule IDs generated by HIDS; we select required number of features.

Figure 4 A sample scree plot (see online version for colours)



3.5.3 Creation of a common format

We take a union of the extracted PCA features of all rule IDs and create a common list of features which will fit to all the input data. This gives a common format which can be further grouped to find common features and frequency. For instance, rule ID 1 has features 1, 3, 4, 5 and rule ID 2 has features 1, 2, 3, 4. So we take a union of all these rule IDs. Here for a particular instance, in whichever field it has value we put the value and where values are not available, i.e., NULL we are replacing it by 'U' – 'undefined'. So the output is in a common format where certain fields are populated and some are U. This becomes a common structure where we have a common PCA value and are grouped to a common format.

3.5.4 Identifying patterns and uniquely represent them

Here, if we observed that majority of the features represent footprint of a user where each instance is unique. So we keep them together as a pattern, use a random key generator to generate a key which is alpha-numeric.

A set of attributes K is a super key for a relation r if r cannot contain two distinct tuples $t1$ and $t2$ such that $t1[k] = t2[k]$. K is a key for r if k is minimal super key (Martinez-Mosquera et al., 2020).

So these patterns along with the key are maintained in a reverse-indexed database. Thus whenever this pattern is found we tag it with the key and store it in the N-column database where all the data are maintained. Thus we can say each key represents a profile, hence we are able to represent all the features related to the footprint of a user as a single column; hence reducing the number of features drastically without anyway compromising the quality of data. A reverse-indexed NoSQL database is used to store the profile with an ID as key. Here the reverse-indexed database is used because, the data input that we receive is in the form of a pattern, for which key needs to be searched. And it is just the reverse of the process we use in a regular database management system. Now the reason behind choosing a NoSQL database is, as the data volume is huge we need a system which is capable of retrieving information very fast as well must be able to scale at the same rate as the data volume becomes more and more. NoSQL systems are distributed databases designed to cater to the demands of huge volume of data. In the management and analysis of massive amounts of data it requires the system to be highly scalable and fault-tolerant. NoSQL databases are coded in many distinct programming languages and are generally available as open-source software (Aniceto et al., 2015). NoSQL systems are also sometimes called 'not only SQL' to emphasise that they may support SQL-like query languages, or sit alongside SQL databases in polyglot persistent architectures (Fowler, 2012; Rouse, 2017).

3.6 Storage of signatures

Next, the signatures created per event type are sent to an in-memory data structure for storing signature reference and a signature history. We are using in-memory database because unlike database management system which store data primarily in the disks or SSDs, an in-memory database stores data primarily on memory. These are a type of non-relational database. As for any data retrieval, they do not need to access disks, the response time in in-memory databases are minimal (Kabakus and Kara, 2017).

3.6.1 Final outcome

Hence in each time bucket we are representing the host, the footprint key, the number of times each pattern is fired and the timestamp. So by using the above methods we can use unstructured logs generated in complex large-scale micro-service-based architecture for further data analysis and also are able to handle such high-dimensional data generated efficiently without anyway losing any data that we receive originally.

4 Experimental setup

In this section, the detailed experimental setup along with the results obtained is presented and is shown as in Figure 5.

Figure 5 Experimental setup (see online version for colours)

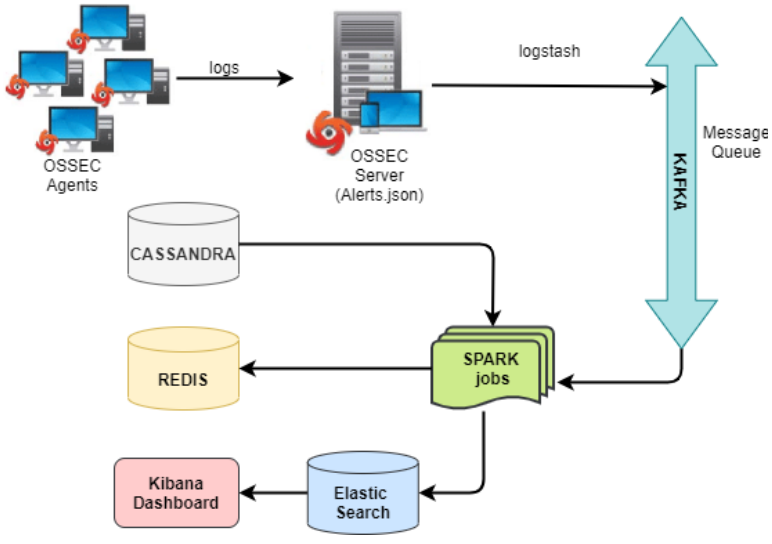


Figure 6 Sample data received from OSSEC in *json*


```
{"timestamp":"2018-06-06T23:59:24+0530","rule":{"level":12,"description":"System runn
,"linuxkernel","service_availability","gpg13_4.12"],"pci_dss":["10.6.1"]},"agent":{"i
ll_log":"Jun  6 23:59:22 rd1p01-pfm-003 kernel: [17009095.437101] Memory cgroup out o
  6 23:59:22","hostname":"rd1p01-pfm-003"},"decoder":{"name":"kernel"},"location":"/v
{"timestamp":"2018-06-06T23:59:25+0530","rule":{"level":3,"description":"Successful s
13_7.13"],"pci_dss":["10.2.5","10.2.2"]},"agent":{"id":"803","name":"rd1p01-cmp-011",
-011 sudo:     root : TTY=unknown ; PWD=/var/lib/collectd ; USER=root ; COMMAND=/sbin
p-011"},"decoder":{"parent":"sudo","name":"sudo"},"data":{"srcuser":"root","dstuser":
auth.log"}
{"timestamp":"2018-06-06T23:59:25+0530","rule":{"level":3,"description":"PAM: Login s
","gpg13_7.9"],"pci_dss":["10.2.5"]},"agent":{"id":"803","name":"rd1p01-cmp-011","ip"
 sudo: pam_unix(sudo:session): session opened for user root by (uid=0)","predecoder":
"},"data":{"dstuser":"root","uid":"0"},"location":"/var/log/auth.log"}
{"timestamp":"2018-06-06T23:59:25+0530","rule":{"level":3,"description":"PAM: Login s
["10.2.5"]},"agent":{"id":"803","name":"rd1p01-cmp-011","ip":"10.1.249.11"},"manager"
ion): session closed for user root","predecoder":{"program_name":"sudo","timestamp":"
on":"/var/log/auth.log"}
{"timestamp":"2018-06-06T23:59:25+0530","rule":{"level":12,"description":"System runn
,"linuxkernel","service_availability","gpg13_4.12"],"pci_dss":["10.6.1"]},"agent":{"i
ll_log":"Jun  6 23:59:23 rd1p01-pfm-003 kernel: [17009096.349784] Memory cgroup out o
un  6 23:59:23","hostname":"rd1p01-pfm-003"},"decoder":{"name":"kernel"},"location":"
{"timestamp":"2018-06-06T23:59:25+0530","rule":{"level":5,"description":"Systemd: Ser
cal","systemd","gpg13_4.3"]},"agent":{"id":"870","name":"RD113APP6773166","ip":"10.1.
md[1]: consul.service: Unit entered failed state.","predecoder":{"program_name":"syst
{"timestamp":"2018-06-06T23:59:26+0530","rule":{"level":3,"description":"Successful s
13_7.13"],"pci_dss":["10.2.5","10.2.2"]},"agent":{"id":"678","name":"rd1p01-cmp-010",
-010 sudo:     root : TTY=unknown ; PWD=/var/lib/collectd ; USER=root ; COMMAND=/sbin
p-010"},"decoder":{"parent":"sudo","name":"sudo"},"data":{"srcuser":"root","dstuser":
auth.log"}
{"timestamp":"2018-06-06T23:59:26+0530","rule":{"level":3,"description":"PAM: Login s
","gpg13_7.9"],"pci_dss":["10.2.5"]},"agent":{"id":"678","name":"rd1p01-cmp-010","ip"
 sudo: pam_unix(sudo:session): session opened for user root by (uid=0)","predecoder":
```

4.1 Data collection

For our experiment, we have used Open Source HIDS Security (OSSEC), the HIDS server installed for Rama Devi Women's University. The reason behind choosing OSSEC is that, it produces a structure file that complied with all the prerequisites needed for our work – instead of storing all logs, it stores only alerts those are generated based on some rule ID (Jain and Trivedi, 2016). In Figure 6, we have provided a part of the json file that we received from OSSEC server. OSSEC being open-source, we write our own rules and rule-IDs. For that we used OSSEC HIDS to collect all the system, authentication, router, switch, firewall activation logs from various sources. OSSEC converts all such unstructured data to structured, tagged data with appropriate alert-IDs and level-IDs and produces a formatted output of raw logs which we received in json format.

4.2 Data conversion

We designed an API for conversion of json data to .csv format so that the data becomes more readable and ready for further processing. In Figure 7, we have given a part of the .csv file that we generated from the json file received from OSSEC.

Figure 7 Data after conversion to *.csv* (see online version for colours)

C	D	E	F	G	H	I
3 Successful sudo to ROOT executed	5402	9	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	9	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	10	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	10	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	11	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	11	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	12	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	12	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	13	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	13	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	14	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	14	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	15	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	15	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	16	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	16	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	17	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	17	FALSE	pam	syslog	authentication_success
7 File system full.	1007	5	FALSE	syslog	errors	low_diskspace
7 File system full.	1007	6	FALSE	syslog	errors	low_diskspace
3 Successful sudo to ROOT executed	5402	18	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	18	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	19	FALSE	syslog	sudo	gpg13_7.6
3 PAM: Login session opened.	5501	19	FALSE	pam	syslog	authentication_success
3 Successful sudo to ROOT executed	5402	20	FALSE	syslog	sudo	gpg13_7.6

4.3 Data cleaning

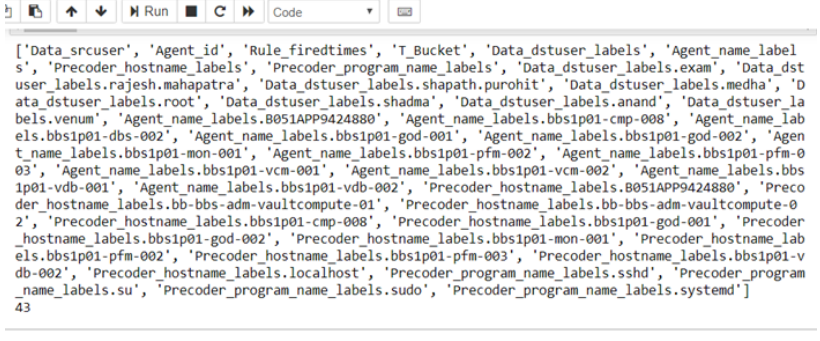
After we extracted features from the file, it came out to be more than 100 features and most of the features were categorical and derived data. We applied one-hot encoding mechanism to convert this categorical data to individual columns where the column values were 0 or 1 corresponding to which column it has been placed. Figure 8 shows the screenshot of columns generated after one-hot encoding was applied and Figure 9 shows the total feature those got generated after on-hot encoding was applied to all the categorical columns we had in our dataset.

Figure 8 Columns generated before and after one-hot encoding

	location	TimeStamp	Data_srcip	Data_dstuser	Agent_name	Agent_id	Rule_level	Rule_description	Rule_id	I
0	/var/log/auth.log	2018-12-02 00:02:29	172.16.33.226	varsham	blr1p01-cmp-008	1993	3	sshd: authentication success.	5715	
1	/var/log/auth.log	2018-12-02 00:29:14	172.16.33.226	varsham	blr1p01-cmp-008	1993	3	sshd: authentication success.	5715	
2	/var/log/auth.log	2018-12-03 06:32:13	10.253.201.10	infra	blr1p01-god-001	2165	3	sshd: authentication success.	5715	
3	/var/log/auth.log	2018-12-03 07:09:15	10.253.201.10	infra	blr1p01-god-001	2165	3	sshd: authentication success.	5715	
4	/var/log/auth.log	2018-12-04 07:39:56	172.16.32.52	praveen	B051APP8274175	2359	3	sshd: authentication success.	5715	

	@timestamp_labels.2018-12-02T18:30:06.806Z	@timestamp_labels.2018-12-02T18:30:09.811Z	@timestamp_labels.2018-12-02T18:30:12.815Z	@timestamp_labels.2018-12-02T18:30:18.822Z	@timestamp_labels.2018-12-02T18:30:23.837Z
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

Figure 9 Total features generated after one-hot encoding



4.4 Feature extraction

Now as we got clean data, we applied PCA based on the rule ID for feature extraction purpose. PCA assumes that the directions with the largest variances are the most 'important' (i.e., the most principal), we first found the eigenvalue because the amount of variance retained by each principal component is measured by the eigenvalue. Eigenvalues are large for the first PCs and small for the subsequent PCs. That is, the first PC corresponds to the directions with the maximum amount of variation in the dataset. Figure 10 shows the screenshot of all features along with explained_variance_ratio after PCA and the Scree plot drawn for all features with their variance plotted in descending order is shown in Figure 11.

From the Figures 10 and 11 plot, we extracted 41 features those were of maximum relevance.

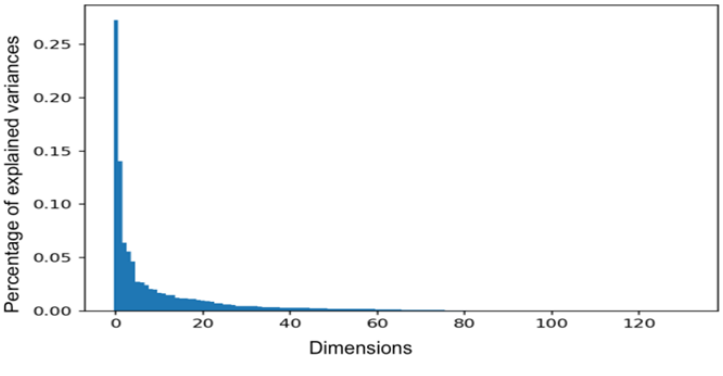
After applying PCA, we combined all PCA values across all rule IDs by making the union of them, to create a unique list of principal components. In case of the unavailable information for any attribute, we replaced it by '*U*'. Hence now a format could be

created based on which we can check which alert is fired. So we grouped all alerts from a particular rule IDs under frequency attribute. Then we chose the source_host from which the alert is generated and timestamp at which it is generated. Now the remaining attributes basically were representing the footprint of a user from the moment he has entered the system till the moment he leaves the same. So next we grouped all the features related to user foot print and maintained them separately under a name 'profile'. Here each profile has a unique signature. Each signature we uniquely identified with a profile_id – thus further reducing features in the dataset. All these form a proper dataset to be processed. Now algorithm can be applied to pull data, pass it to the model, identify the outliers and reports them as anomalies. Now in order to deal with the high volume of data we must deal with smaller chunk at-a-time. We freeze the slot to be monitored on each 15 minutes. To implement this, we divided the 24 hours logs to 15 minutes time buckets, where each bucket is uniquely identified by bucket-ID. Thus finally we arrive at to only three without making any compromise with data quality.

Figure 10 Screenshot of features along with variance

```
{('Agent_id_PCA',): 0.8222010751922796,
 ('Rule_firedtimes_PCA',): 0.17768004652147104,
 ('T_Bucket_PCA',): 8.747173884965891e-05,
 ('Data_dstuser_labels_PCA',): 1.7678513548335797e-05,
 ('Agent_name_labels_PCA',): 2.2839672853472474e-06,
 ('Precoder_program_name_labels_PCA',): 2.2552388839477263e-06,
 ('Precoder_hostname_labels_PCA',): 2.04961196816613e-06,
 ('Data_dstuser_labels.infra_PCA',): 1.7892262078972601e-06,
 ('Data_dstuser_labels.jegan_PCA',): 1.5666831488198506e-06,
 ('Data_dstuser_labels.manishkumar.singh_PCA',): 1.1234847052819769e-06,
 ('Data_dstuser_labels.root_PCA',): 1.0907483346354132e-06,
 ('Data_dstuser_labels.shiny_PCA',): 9.394410050536195e-07,
 ('Data_dstuser_labels.venum_PCA',): 3.773211956263694e-07,
 ('Agent_name_labels.B051APP6006088_PCA',): 1.037713301558657e-07,
 ('Agent_name_labels.B051APP9424880_PCA',): 7.741313633731444e-08,
 ('Agent_name_labels.blr1p01-cmp-008_PCA',): 5.5561181543311396e-08,
 ('Agent_name_labels.blr1p01-dbs-002_PCA',): 1.5565468823469425e-08,
 ('Agent_name_labels.blr1p01-god-001_PCA',): 7.977786839162058e-33,
 ('Agent_name_labels.blr1p01-god-002_PCA',): 7.977786839162058e-33,
```

Figure 11 Scree plot for all features with variance (see online version for colours)



As a final result, we reduced the high dimensional data produced to only three features, i.e., profile, source host and time of the day which we expressed as bucket-ID.

4.5 Storage of data for reference

Each profile that got created with a unique feature set, has the data 'profile' as constant which enabled us to create signatures per event type. Hence every unique event has a unique signature. We used reverse indexed in-memory database server – Redis to store signature reference as we need to find the unique signature ID from the log. For the purpose of storing signature reference we used an in-memory database Redis. Redis is open source software with BSD license. Redis, is a fast, open-source, in-memory key-value data store which is used as a database, cache, message broker, and queue. All Redis data resides in-memory, very unlike the databases that store data on disk or SSDs. As the need to access disks is no more required, seek time delays are avoided in in-memory data stores such as Redis and the data can be accessed in microseconds. Redis features versatile data structures, high availability, geospatial, Lua scripting, transactions, on-disk persistence, and cluster support making it simpler to build real-time internet scale apps. Next in order to store signature data we used Cassandra – a NoSQL database. Apache Cassandra is a highly scalable, high-performance distributed database. It is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It performs efficiently on 'read' queries, thus is highly recommendable for 'read heavy' database. So we stored the profiles here and used profile_id as a representative for each instance in our dataset. To summarise, we put the dataset created hence and get a unique identifier which is an alpha-numeric identifier. We store it to a reverse-indexed database Redis. Now, the stream will look for all fields along with the unique value, if available it stamps it and if it is not there it will create and stamp it. Next it is written to CASSANDRA which can be put to model for further processing.

5 Conclusions and future work

In this paper, we have proposed profiling method on the JSON output of OSSEC/HIDS logs those are generated in complex large-scale micro-service-based installations for analysis purpose. This method works for further reduction of N-features to only three dimensions, so we can perform any analytics and visualise the data, once a standard feature selection technique is applied. Finally use of in-memory DB and NoSQL database servers enable us to handle large volume of instances easily for further processing of finding out anomalies in case of complex micro-service architecture. But the limitation in the proposed setup becomes heavy for small infrastructures. We need to constantly modify the unique feature list as and when we get more rules from OSSEC. Once we build data lakes, i.e., we push all logs to a distributed streaming platform; OSSEC that we have used being a monolithic system, may not be able to handle such high-volume data. OSSEC is also modified to handle more than 1,500 clients. Hence we need to create clusters of OSSEC to handle large installations. In this case we can have an analytics engine for big data processing, with built-in modules for streaming, mapping features and creating profile ID. Thus to make the conversion engine scalable how to use big data analytics engine will be the focus of our study in future. The spark jobs enable us to scale horizontally and handle huge incoming data which can be processed in large numbers. A REDIS cluster can help us also handle huge amount of

unique events. The key to maintain such systems is to do a regular purge job so we can keep tab on the volume of data in data stores.

References

Aniceto, R., Xavier, R., Guimarães, V., Hondo, F., Holanda, M., Walter, M.E. and Lifschitz, S. (2015) 'Evaluating the Cassandra NoSQL database approach for genomic data persistency', *International Journal of Genomics* [online] https://doi.org/10.1155/2015/502795.

Breier, J. and Branišová, J. (2015) 'Anomaly detection from log files using data mining techniques', in *Information Science and Applications*, pp.449–457, Springer, Berlin, Heidelberg.

Brownlee, J. (2017) *Why One-Hot Encode Data in Machine Learning?*, 28 July [online] https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/ (accessed 26 February 2020).

Fowler, M. (2012) *NoSQL Definition*, 9 January [online] https://martinfowler.com/bliki/NosqlDefinition.html (accessed 24 February 2020).

Jain, R.K. and Trivedi, P. (2016) 'OSSEC based authentication process with minimum encryption and decryption time for virtual private network', in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, IEEE, pp.442–445.

Kabakus, A.T. and Kara, R. (2017) 'A performance evaluation of in-memory databases', *Journal of King Saud University – Computer and Information Sciences*, Vol. 29, No. 4, pp.520–525, DOI: 10.1016/j.jksuci.2016.06.007.

Lou, J-G., Fu, Q., Yang, S., Xu, Y. and Li, J. (2010) 'Mining invariants from console logs for system problem detection', in *USENIX Annual Technical Conference*, pp.23–25.

Martinez-Mosquera, D. et al. (2020) 'Modeling and management big data in databases – a systematic literature review', *Sustainability*, Vol. 12, No. 2, p.634.

Potdar, K., Pardawala, T. and Pai, C. (2017) 'A comparative study of categorical variable encoding techniques for neural network classifiers', *International Journal of Computer Applications*, Vol. 175, pp.7–9, DOI: 10.5120/ijca2017915495.

Raj, A. (2019) *Feature Reduction Using – PCA & LDA*, 23 January [online] https://medium.com/@adityaraj_64455/feature-reduction-using-pca-lda-338b9fe64f59 (accessed 21 February 2020).

Rouse, M. (2017) *NoSQL (Not Only SQL Database)* [online] https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL (accessed 24 February 2020).

Tandon, P., Sleiman, F.M., Cafarella, M.J. and Wenisch, T.F. (2016) 'HAWK: hardware support for unstructured log processing', *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp.469–480, DOI: 10.1109/ICDE.2016.7498263.

Tovarňák, D. and Pitner, T. (2019) 'Normalization of unstructured log data into streams of structured event objects', *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp.671–676.

Vidal, R., Ma, Y. and Sastry, S.S. (2016) 'Principal component analysis', in *Generalized Principal Component Analysis*, pp.25–62, Springer, New York, NY.

Wang, Z. and Zhu, Y. (2017) 'A centralized HIDS framework for private cloud', *18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Kanazawa, pp.115–120.

Xu, W., Huang, L., Fox, A., Patterson, D. and Jordan, M.I. (2009) 'Detecting large-scale system problems by mining console logs', *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles – SOSP'09*, DOI: 10.1145/1629575.1629587.

Websites

https://education.howthemarketworks.com/covariance-analysis/ (accessed 21 February 2020).

https://math.stackexchange.com/questions/885410/covariance-matrix-of-various-x-y-z-cartesian-coordinates (accessed 21 February 2020).

https://redis.io (accessed 10 April 2020).

https://aws.amazon.com/redis/ (accessed 10 April 2020).

http://cassandra.apache.org/ (accessed 6 April 2020).