K-means Application for Anomaly Detection and Log Classification in HPC

Mohamed Cherif Dani[1], Henri Doreau[2(✉)], and Samantha Alt[1]

[1] Intel, 2 Rue de Paris, Meudon, France
[2] CEA, DAM, DIF, 91297 Arpajon, France
henri.doreau@cea.fr

Abstract. Detecting anomalies in the flow of system logs of a high performance computing (HPC) facility is a challenging task. Although previous research has been conducted to identify nominal and abnormal phases; practical ways to provide system administrators with a reduced set of the most useful messages to identify abnormal behaviour remains a challenge. In this paper we describe an extensive study of logs classification and anomaly detection using K-means on real HPC unlabelled data extracted from the Curie supercomputer. This method involves (1) classifying logs by format, which is a valuable information for admin, then (2) build normal and abnormal classes for anomaly detection. Our methodology shows good performances for clustering and detecting abnormal logs.

Keywords: Anomaly detection · HPC · Log processing · K-means

1 Introduction

With the rise of artificial intelligence and intensive simulation in science and engineering, more and more researchers rely on the massive computing power of High Performance Computing (HPC) clusters. Large HPC facilities consist of heterogeneous distributed subsystems, which interact in non-trivial ways. Accidental conditions, hardware or software failures, and sustained heavy load can cause a component in the system to malfunction. Due to the very large number of components within a supercomputer results in significant number of fault occurrences. The fact that HPC systems are designed to maximise performances rather than reliability further increases the risk.

The top-level architectural units, such as storage or compute, are often implemented as interleaved layers of software and hardware that interact together and with numerous services running aside. A single computing cluster operated by CEA at TGCC [1], comprises several thousand compute nodes, which run complex scientific applications and libraries. A job scheduler is in charge of allocating resources and running the tasks as quickly as possible. The nodes communicate together over a high bandwidth/low latency InfiniBand network. External storage servers, connected to the fabric, expose distributed file systems to all clusters of the computing center. The component hierarchy is very deep and the interconnections and dependencies very complex. These subsystems record their activity

S. Benferhat et al. (Eds.): IEA/AIE 2017, Part II, LNAI 10351, pp. 201–210, 2017.
DOI: 10.1007/978-3-319-60045-1_23

using unstructured text lines that constitute a prime source of information for system administrators.

Classical monitoring solutions periodically check whether services are responding[1] [2]. This is indeed essential order to ensure a certain quality of service. The current technology implemented in the HPC system allows for sampling[2] of performance counters at high frequency [3,4], which supplies the user with highly granular insight into the performance of each node in the system. Both approaches lack the ability to provide system administrators with enough information to understand the nature of the issue they are investigating, only that there is an issue occurring. This information is made available through more detail in the console logs.

Robust tools exist to aggregate and centralise the many log streams of a massively distributed computing infrastructure [5]. Nevertheless, the resulting composite signal is difficult to exploit efficiently because of its high throughput, its unstructured format, and its level of noise and information redundancy. In addition, single messages are incomplete in that they do not contain enough information to fully trace a failure back to the root cause. A task as common as identifying faulty components that generate abnormal behaviour can be extremely time-consuming and has to be done by domain experts. Automatic anomaly detection can be directly coupled with resource management systems to mark faulty components and prevent them from being used in production until the problem is solved.

Nowadays, the need for anomaly detection covers almost every domain in engineering. All these domains converge towards a common definition that the anomaly is a deviation from a normal behaviour, which most of the data form. This definition is extensively explored by Agroual [8] and many other authors [9,10]. In this context, we refer to abnormal log lines that contain a description and the source of a failure.

As of today, the common approach of analysing logs first consists in writing and maintaining regular expressions to match known patterns [6]. This does not scale well and requires a significant amount of human work and experience. The main drawback of this approach is that it will systematically lead to a situation where the common messages are properly parsed and the unexpected ones cannot be matched against a regular expression despite being highly valuable.

As noted by Ning et al. [7] console logs have a reduced vocabulary, highly skewed word count distribution, and weak syntax. This makes most natural language processing approaches unsuitable for solving the problem. The choice was therefore made to look for anomalies based on the geometrical structure of the messages rather than try to summarise them.

Adding anomaly detection to the system monitoring process leads to healthier and better functioning systems. By applying text mining techniques and unsupervised learning to actual system logs from *petascale* HPC clusters, we have developed an approach to filter the messages and provide the system

[1] http://shinken-monitoring.org.
[2] https://graphiteapp.org.

administrators with a selection of the most useful entries to identify and understand anomalies and failures.

2 Problem Statement Within HPC

An HPC environment has domain-specific constraints and characteristics with high component count and complexity. These subsystems change very quickly in order to stay cutting-edge, which leads to frequent firmware and OS upgrades. Therefore, administration and monitoring techniques must be scalable with regard to the number of nodes as well as not be a large impact on performance. Current research is oriented toward the reach of *exascale*. It is expected that future Exascale machines will contain many more base components rather than larger and faster components. This will lead to more failures to investigate within a larger log stream, thus the critical need for efficient automation.

The techniques used to reliably propagate console messages from the nodes that generate them to a log processing cluster is out of the scope of this article. This work explores means to process the logs and present them in a useful way to the system administrators. In particular, identify outliers among the aggregated stream of messages.

We focused on unstructured text messages, so any attempt to apply machine learning techniques on the problem had to come with a matrix representation of the log corpus. The chosen technique had to be efficient, in terms of CPU and memory footprint, so as to be applicable for the very large volume of logs that a HPC cluster can generate.

As scalability of mining algorithms are a large factor in this systems automation, the genomic sequence mining algorithms described in [11] are not applicable due to the scalability limitations. Linking resource usage to console log [12,13] returned interesting results, but failed at identifying problems that are not described by both datasets.

Definitions

A ***log message***, or *record*, consists of one or multiple lines of text attached to a timestamp. A record describes an event that occurred on a component of the system at that time. Messages of a component are emitted and recorded sequentially. The sequence of log messages is referred to as system log or log data.

A record can be divided into fields, which in turn fall into two categories. The well-known ones (such as date, time, hostname, process name, etc.) usually prefix the messages and format-free information follows. The latter contains the most relevant information but is by far the hardest to process. Only a small minority of applications publish their log format and specifications.

We call ***anomalies*** the log records that describe abnormal situations. They are distant from the records that are emitted during nominal operations. The definition of abnormality depends on some terms/tags in the log line *(Fatal, Error, Err, etc.)*, and many other patterns validated by experts.

Log format varies significantly, an example of these logs is described in the following figure:

```
2015-11-xx 15:21:09,568 ERROR tuned.utils.commands:
Executing hdparm error: [Errno 2] No such file or directory

Dec 6 03:56:37 [158XXX] node1789.c-curie.hpc.cea.fr
pengine: error: unpack_resources:         Resource start-up
disabled since no STONITH resources have been defined

141912041 2016 Dec 7 03:35:01 x kern warning kernel
Lustre: DEBUG MARKER: Sun Dec 7 03:35:01
```

Fig. 1. Heterogeneous log format sample

3 Unsupervised Log Clustering and Anomaly Detection

A lot of methods and tools exist in machine learning to analyse and to detect anomalies in data centers, some of them mentioned previously. The simplest rely on regular expressions, while others empower more advanced methods like Natural language processing (NLP) or machine learning [8].

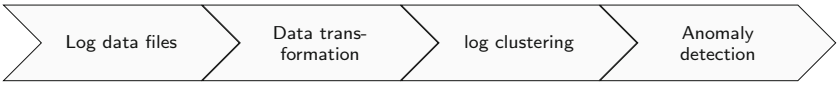

Fig. 2. Different phases before achieving detection

Within an unsupervised environment, where no prior information is available, the detection process of anomalies is complex and challenging. As in every domain, many elements within the data center can lead to an anomaly. Furthermore, the amount of data generated daily is huge. The detection process is defined in Fig. 2.

Our first dataset contains more than 300 log lines of heterogeneous logs, extracted from different templates. Three of these logs are presented in Fig. 1. The entire dataset contains about 15 different log formats. One of these formats constitutes our second dataset, that will be used to demonstrate the concept of the anomaly detection and log clustering. This dataset contains more than 32 million entries used to test the clustering efficiency and scalability, and 1 million log lines used for anomaly detection.

The transformation of the data is an important phase that has a large impact on the quality of clustering. Our goal is to cluster the logs by format and message type using respectively the first and the second datasets. If we consider the log-files analysis process as a classic text mining problem, where we can for example cluster the data by the subject of interest, or extract useful information

such as correlated words. Hence, since we assume no prior knowledge of the type of logs to keep consistent with the heterogeneity and rapid changes of the system. The log data are transformed into a frequency matrix that can be easily consumed by machine learning methods. To do so, we keep only the type of logs (error, info, debugging, notices, etc.), and the log description *(cleaned up)*. The cleanup of the message consisted in forcing lowercase, removing punctuation signs, and stemming process. For log messages that contained a file path, we concatenated the path into one word to optimise our representation and to avoid sparse representation by producing too many terms.

The documents term (log-term) matrix, which contains the frequency of terms regarding all the logs, is our first input from the pre-processing phase. The term frequency and inverse document frequency (tfidf) [14] is a widely used matrix in text mining that evaluates how important a term is to a document within a collection of documents. This is defined as:

$tdidf(d,t) = tf(d,t) * log(\frac{|D|}{df(t)})$, where $tf(t)$ is the number of occurrences of term t in a document divided by the total number of terms in the documents.

An example of this transformation is illustrated in Fig. 3:

```
Original log:
  148228XXX 2016 Dec 21 03:35:01 node3435 cron info CROND (root) CMD
  (/usr/bin/test -e /dev/lnet && /usr/sbin/lctl mark >/dev/null 2>&1) 2
Transformed  log:
  info crond root cmd usr-bin-test dev-lnet usr-sbin-lctl mark dev-null
```

Fig. 3. Log transformation

The K-means family methods are used extensively in text mining [9] and outlier and anomaly detection [8]. A performance discussion of K-means clustering in comparison to other approaches such as hierarchical (hclust) and density (dbscan) clustering is done in the next section.

3.1 Log-Files Clustering Using K-means

Our process is illustrated by Fig. 4 where the main input can contain logs with millions of formats. Then for each format group, we apply another clustering to build normal and abnormal classes.

The K-means [15] is an unsupervised algorithm which aims to group the data into K clusters, where K needs to be user defined. The K-means setup can be resumed into major setup: assignment setup and update setup. The knowledge of K is not an obstacle in this study, unlike in many unsupervised problems since we are dealing with known template that generates the logs. So, the number of K can be estimated easily.

Otherwise, an estimation of K is conceivable in the case where new templates are setup or new logs appear. For such estimation, several statistic criteria exist like Akaike Information Criterion [16] or Bayesian Information criterion [17].

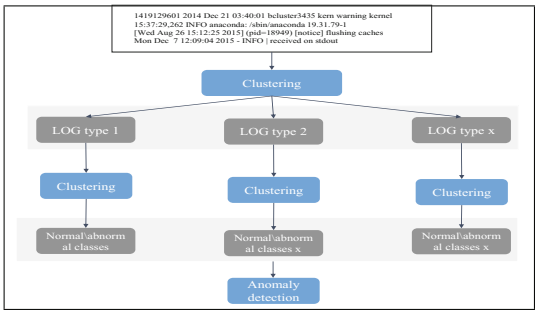

Fig. 4. Log clustering stages and anomaly detection

The classic K-means similarity measure is, in general, the Euclidean distance which is defined as:

$$D_e = (\sum_{t=1}^{n} |w_{t,a} - w_{t,b}|^2)^{1/2}$$

Where $T = t_1, ...t_m$ is the term set and $w_{t,a}$ is the term weights computed using $(tdidf)$.

Generally, the document term matrices are known to be highly dimensional. A situation where standard k-means is less powerful than other techniques.

Consequently, we tried during the prepossessing phase to simplify our document term matrix, to get less terms by deleting the least significant ones and merge paths in one word, etc. Also the standard distance definition of k-means based on centroids was replaced by the cosine similarity measure which is more efficient for document term clustering than Euclidian distance [19].

It can be considered as the correlation between two term vectors, with certain independence of document length. So, with two documents t_1, t_2 forming the term-set $T = t_1, ..., t_n$, the cosine is denoted by:

$$D_c = \frac{\overrightarrow{t_1}.\overrightarrow{t_2}}{|t_1||t_2|}$$

In simple manner, when two documents are identical the distance is equal to 1, and if they are very different the distance equals 0. After computing the similarity measures, each data point is assigned to the cluster with the highest cosine similarity measure (the classic k-means steps).

The usage of kmeans for anomaly detection was inspired by many studies [8,9,18] that use the same method for anomaly detection. Before discussing the clustering results, we will explain in the next section the anomaly detection process using K-means.

4 Anomaly Detection and Log Classification Using K-Means

In many anomaly detection problems, the unsupervised part is used for labelling data, then semi-supervised or supervised method like OSVM [9] are used for detection and classification. However, we propose in this part of the work to use K-means centroids and Euclidean distance to classify the new logs as abnormal or not.

In [18] the authors propose to build normal and abnormal classes using K-means. A data point with a distance closer to the normal centroid and under a threshold is labelled as normal. Otherwise the data point is abnormal. This process of detection is adapted to our problematic and used on 1 million of log lines to detect anomalies.

4.1 Results Discussion

First in this section, we will compare the clustering results of different data set. The number of k is known for the three datasets (k = 15 for the 300 logs dataset, and k = 5 or 6 for the other datasets).

Heterogeneous Classification

Clustering the heterogeneous logs is the first relevant information that we can provide to the admin. Technically, because every log is provided from a specific template or system, grouping these logs allows us to link specific anomaly detection to a specific log source, i.e. hardware fault or software application. In addition to K-means, we have also tested dbscan, which is used for clustering logs [7] as well as noise and outlier detection, and hclust with an implementation package of hierarchical clustering in R.

To evaluate the detection and quality of the clustering results, we have chosen to use the common F-measure evaluation criterion. It expresses the performance of an anomaly detection method using the precision P and sensitivity R measures where: $P = \frac{TP}{TP+FP}$ and $R = \frac{TP}{TP+FN}$ and TP, FP, TN, FN are respectively: True Positive, False Positive, True Negative and False Negative.

To measure the performance of our approach, The F-measure is denoted by $F = 2 \times \frac{P \times R}{P+R}$ where $F \simeq 1$ indicates a good performance and $F \simeq 0$ means that the method does not detect any anomaly in the data.

Table 1. Anomaly detection performance (F-measures) in different data sets with different number of clusters.

	Dbscan	Hclust	Kmeans
300 Log	0.39	0.63	0.87
32M Log	0.20	0.64	0.80
01M Log	0.5	0.7	0.94

The original log files are composed of 15 clusters *(k = 15)*. Using dbscan only 9 clusters were formed, with 39% performance. These results are not only due to misclassification of logs, but can also be explained by the particularity of dbscan to not cluster logs that do not respect the density parameters. With an f-score of 63% Hclust misclassified data and returned similar logs with only minor differences. We observe the same thing for K-means, but with a much higher performance (87%) (Fig. 5).

```
Cluster:
info broker master archive old log file
info broker master  move old log file var-log-shinken shinken log to var-log-shinken-archives shinken log
info shinken  poller master  init connection scheduler master https tipasa  tipasa ocre cea
info shinken  poller master  connection ok schedul  schedul  master
info receive master  receive master  stop workers
Cluster:
info crond  root  cmd  usr-bin-test dev-lnet usr-sbin-lctl mark dev-null
cron info crond  root  cmd  usr-bin-test  dev-lnet usr-sbin-lctl mark dev-null
cron info crond  root  cmd  usr-lib64-sa-sa
kern warning kernel
kern warning kernel lustre  debug marker
Cluster:
api ni lnetstartuplndnis    add lni
api ni lnetstartuplndnis    add lni
lprocosc oscwractive    activate   ignore repeat request
lprocosc oscwractive    activate   ignore repeat request
ostosc ffffeaf  communicate operation ostconnect failed
```

Fig. 5. Sample of clustering

Using a classic server to process the original dataset (32 million log lines) which contains only six format families *(k = 6)*. Clustering accuracy for K-means is 80%, 20% for dbscan with heavy processing, and an intermediate score of 64% for hclust. However, these results are accompanied with a great challenge of scalability where execution time took a few minutes for K-means and many hours for dbscan.

Anomaly Detection

To build normal and abnormal classes we followed the definition of anomalies given above. The clustering performances on the training subset of 1 million logs are presented in Table 1. Two global clusters were built manually, the normal cluster contains 5 sub-clusters, and the abnormal one which contains mostly error tags.

To detect anomalies, we extracted (test set) from normal and abnormal clusters about 200 log lines (100 abnormal, 100 normal log lines). And then tried to use the Euclidean distance from centroid to classify the test set. We did not make any assumption or threshold about the normal or abnormal clusters, and the detection process showed very satisfying results. A sample of the results is presented in Fig. 6.

The dataset we used contains only few types of abnormal logs, which explains the good clustering and detection performances. To evaluate the method more efficiently, we would need more types of abnormal logs.

```
Abnormal cluster :
err corosync   totem  marking ringid  interface jobalt faulty
err corosync   totem  marking ringid  interface jobalt faulty
err corosync   totem  marking ringid  interface jobalt faulty
err corosync   totem  marking ringid  interface jobalt faulty
err corosync   totem  marking ringid  interface jobalt faulty
err corosync   totem  marking ringid  interface jobalt faulty

Normal Cluster:
info stonith ng   info stonith command processed stexecute  lrmd
info stonith ng   info stonith command processed stexecute lrmd
info stonith ng   info log operation restofence curiel getting status ipmijo
info stonith ng   info log operation restofence curiel getting status ipmijo
notice corosync   totem automatically recovered ring
notice corosync   totem automatically recovered ring
notice corosync   totem retransmit list
notice corosync   totem retransmit list
notice corosync   totem retransmit list
```

Fig. 6. Sample of abnormal and normal cluster

5 Conclusion and Future Works

This work illustrates the ability to provide system administrators with a useful vision of system logs. It significantly speeds up troubleshooting and failure analysis by clustering heterogeneous logs. The efficiency of the technique, its scalability, and the fact that it works on unlabelled data makes it particularly appropriate for very large and constantly changing data centers such as HPC facilities. The results are limited by the wealth of data, since we have hardly any prior information about the data. Pre-processing is indeed an important phase in such methods and we need to improve the new data representation in order to use more sophisticated algorithms and techniques such as co-clustering or density methods. Abnormal logs detected can serve as a strong input to troubleshoot and identify the root-cause of an anomaly.

References

1. Morey, J.-M.: Numerical simulation at CEA. In: Proceedings of SNA + MC (2013)
2. David, J.: Building a Monitoring Infrastructure with Nagios. Prentice Hall PTR, Upper Saddle River (2007)
3. Bautista, E., Whitney, C., Davis, T.: Big data behind big data. In: Arora, R. (ed.) Conquering Big Data with High Performance Computing, pp. 163–189. Springer, Cham (2016)
4. Sigoure, B.: OpenTSDB scalable time series database (TSDB) (2012)
5. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: a distributed messaging system for log processing. In: Proceedings of The NetDB, pp. 1–7 (2011)
6. Reelsen, A.: Using elasticsearch, logstash and kibana to create realtime dashboards (2014)
7. Ning, X., Jiang, G., Chen, H., Yoshihira, K.: HLAer: a system for heterogeneous log analysis
8. Aggarwal, C.C., Yu, P.: Outlier detection with uncertain data. In: SDM (2008)
9. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. **41**, 15 (2009)
10. Gupta, M., Han, J., Aggarwal, C., Gao, J.: Outlier detection for temporal data: a survey. IEEE Trans. Knowl. Data Eng. **26**, 2250–2267 (2014)

11. Stearley, J.: Towards informatic analysis of syslogs. In: Cluster Computing. IEEE (2004)
12. Chuah, E., Jhumka, A., Narasimhamurthy, S., et al.: Linking resource usage anomalies with system failures from cluster log data. IEEE (2013)
13. Gurumdimma, N., Jhumka, A., et al.: CRUDE: combining resource usage data and error logs for accurate error detection in large-scale distributed systems. IEEE (2016)
14. Rajaraman, A., Ullman, J.D.: Data mining. In: Mining of Massive Datasets (PDF) (2011)
15. MacQueen, J.B.: Some Methods for classification and Analysis of Multivariate Observations. University of California Press, Berkeley (1967)
16. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Petrov, B.N., Csáki, F. (eds.) 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2–8 (1971)
17. Schwarz, G.E.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
18. Münz, G., Li, S., Carle, G.: Traffic anomaly detection using k-means clustering. In: GI/ITG-Workshop MMBnet, September 2007
19. Larsen, B., Aone, C.: Fast and effective text mining using linear-time document clustering. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999)