Fall 12-18-2020

An Unsupervised Anomaly Detection Framework for Detecting Anomalies in Real Time through Network System's Log Files Analysis

Vannel Zeufack

An Unsupervised Anomaly Detection Framework for Detecting Anomalies in Real Time through Network System's Log Files Analysis

A Thesis Proposal presented to

The Faculty of the Computer Science Department

by

Vannel Zeufack

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

December 2020

An Unsupervised Anomaly Detection Framework for Detecting Anomalies in Real Time through Network System's Log Files Analysis

Approved:

DocuSigned by:
Ahyoung Lee
77A8A9BE83C8470...
Dr. Ahyoung Lee - Advisor

DocuSigned by:
Dr. Coskun Cetinkaya
30EA3BD5C7FB4C8...
Dr. Coskun Cetinkaya - Department Chair

DocuSigned by:
Jeffrey Chastine
1C1B4EA2D5D047A...
Dr. Jeffrey Chastine - Dean

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

DocuSigned by:
Vannel Zeufack
F1F34FD9F4584C3...

Vannel Zeufack

Notice To Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Vannel Zeufack

950 Hudson Road SE, Apt 307

Marietta, GA, 30060

The director of this thesis is:

Dr. Ahyoung Lee

680 Arntson Drive, J 306

Marietta, GA, 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

An Unsupervised Anomaly Detection Framework for Detecting Anomalies in Real Time through Network System's Log Files Analysis

An Abstract of

A Thesis Presented to

The Faculty of the Computer Science Department

by

Vannel Zeufack
Previous degree (Bachelor of Science), Kennesaw State University, 2018

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

December 2020

Abstract

Nowadays, in almost every computer system, log files are used to keep records of occurring events. Those log files are then used for analyzing and debugging system failures. Due to this important utility, researchers have worked on finding fast and efficient ways to detect anomalies in a computer system by analyzing its log records. Research in log-based anomaly detection can be divided into two main categories: **batch log-based anomaly detection** and **streaming log-based anomaly detection**. Batch log-based anomaly detection is computationally heavy and does not allow us to instantaneously detect anomalies. On the other hand, streaming anomaly detection allows for immediate alert. However, current streaming approaches are mainly supervised. In this work, we propose a fully unsupervised framework which can detect anomalies in real time. We test our framework on hdfs log files and successfully detect anomalies with an F-1 score of 83%.

An Unsupervised Anomaly Detection Framework for Detecting Anomalies in Real Time through Network System's Log Files Analysis

A Thesis Proposal presented to

The Faculty of the Computer Science Department

by

Vannel Zeufack

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Advisors:

Dr. Ahyoung Lee

Dr. Donghyun Kim

Kennesaw State University

December 2020

Acknowledgement

Many thanks to **Dr. Donghyun Kim** without which my graduate studies would not have been possible

Thanks to **Dr. Ahyoung Lee** for her amazing support

Contents

1. Introduction

In today's computing systems, logging has become a widely embraced and important practice. It is the process of keeping records of events which occurred within a computing system. The records are kept into what is referred to as a log file. According to [3], more than five standards and more than five regulations enforce, in every continent, that a log management system must be included into the security systems. So why are log files so important?

Indeed, computing systems are becoming increasingly large and complex, leading to an increase in vulnerability and failure prominence. By recording events, logs allow system operators to detect and debug failures and easily recover from a working state if necessary.

For many years (and even today), anomaly detection based on log file analysis has been done manually by domain experts who read and interprets log files line by line. However, in this big data era, log files are becoming so large that it is almost impossible to manually and timely analyze them. According to [4], approximately 2.5 quintillion bytes of data are created each day. Every minute, snapchat users share 527,760 photos, YouTube users watch 4,146,600 YouTube videos and 456,000 tweets are sent on Twitter.

To solve this issue, many scholars have researched efficient ways to automate log-based anomaly detection. The state-of-the art approaches leverages the power of Machine Learning algorithms to detect patterns in log files and flag abnormalities. However, performing real time log analysis in an unsupervised setting remains a challenge.

In this work, we propose a fully unsupervised real time framework able to detect anomalies in real time. Our framework exploits the power of clustering to learn most frequent patterns from historical logs and uses fine-grained distance analysis to detect outliers.

The remaining sections of this document are organized as follows. Section 2 describes the most significant researches and their limitations. Section 3 discusses the major challenges of anomaly detection based on log file analysis. Section 4 provides details about the proposed solution. Section 5 describes and discusses our experiments and results. Finally, section 6 concludes and provides perspective for future plans.

2. Significant prior research

For about a decade, machine learning has been the key tool to automate log analysis. Indeed, machine learning algorithms and structures have shown their capability to detect and learn very complex patterns.

Since all significant prior research uses machine learning techniques, they share a common framework [5] shown in the figure below:

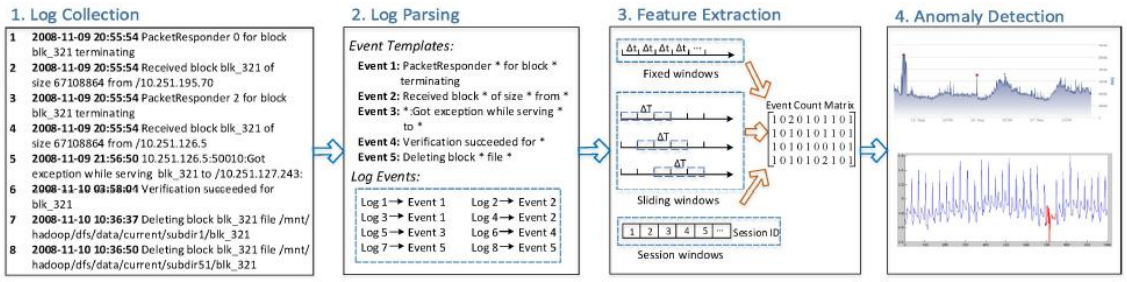


Figure 1 - Log-based Anomaly Detection Framework

Log collection is the first step of the framework and is about gathering data. Logs could be collected in streaming or batch. The next step, log parsing, consists in extracting log templates. The third step, feature extraction, consists in extracting relevant features for the anomaly detection model. The final step uses the extracted features to perform anomaly detection. The anomaly detection system is usually a machine learning model. More details can be found in [5].

The cutting-edge results in log-based anomaly detection research can be classified into two broad categories: **offline anomaly detection systems and online anomaly detection systems**.

2.1. Offline anomaly detection systems

Offline anomaly detection systems are mainly used to debug or detect anomalies after they occurred. Log analysis is performed after a significant number of logs is collected (may be at end of the day or the week or the month). Some of these approaches use unsupervised learning while others use supervised learning.

Approaches using unsupervised learning do not require labeled logs (logs for which we know a priori what is normal and what is abnormal). There is no learning phase. They extract features and perform anomaly detection on the full log file. Here, we can mention PCA [6] which extracts state ratio vectors and message count vectors and uses Principal Component Analysis to detect anomalies. We also have LogCluster [7] which assigns weights to events and uses Agglomerative Hierarchical Clustering to detect patterns. Invariants Mining [8] extracts messages counts and uses singular value decomposition to learn invariants from the logs.

Supervised learning approaches require labeled logs. They learn patterns from labeled logs and can flag, from a new batch, any pattern which deviates from the normal patterns. This is the case of LogRobust [9] who uses attention-based Bi-LSTM neural network to learn normal patterns and then detect anormal behavior given new batches.

2.2. Online anomaly detection systems

Online or streaming anomaly detection systems perform anomaly detection in real time, as the events are recorded. Currently, these approaches use supervised learning. They learn patterns offline from normal logs and are placed in production to detect abnormal events in real time.

One of the most prominent online anomaly detection system is DeepLog [10] which trains, offline, a log key anomaly detection model and a parameter value anomaly detection model. The trained models are used to detect anomalies in real time. Another interesting work is LogAnomaly [11] which learns normal patterns using LSTM neural network and can detect sequential and quantitative anomalies simultaneously.

3. Research Challenges

Three major challenges emerge in log-based anomaly detection research: *unsupervised streaming anomaly detection, feature engineering and new log entries management*.

3.1. Unsupervised streaming anomaly detection

Streaming anomaly detection can be defined as the process of finding issues and alerting in real-time. Indeed, in this big data era with faster computers, data is generated at a huge pace. Moreover, because nowadays computer systems are becoming increasingly complex, they are increasingly prone to failure. Hence, there is a necessity to build automated anomaly detection systems which can perform analysis at the same pace as data are generated.

Even though some researchers were able to find ways to perform streaming anomaly detection, it remains a gap as the current state of the art approaches require normal (labeled) logs from which to learn normal patterns. However, obtaining labeled logs is sometimes expensive (require domain experts) and time consuming. This leads to the necessity of a streaming anomaly detection system which does not require normal logs.

3.2. Feature engineering

In the previous works, the most exploited feature is the count of log templates by window. However, many approaches have shown satisfying results by exploring various types features including timestamp statistics, log parameter vectors and state ratio vectors. The diversity in feature extraction can be attributed to the richness and diversity of information included into log files. Hence, it is hard to determine exactly what features are best for anomaly detection.

3.3. Handling new log entries

In a production environment, it could happen that a previously unseen event occurs. While some of the previous works flagged new log entries as abnormal, other approaches proposed that new log entries should be analyzed manually by domain experts [10] and others proposed that new log entries should to be approximated to the known log entry with most similar template [11]. Handling new log entries has shown to be a challenge as it requires the system to be retrained offline to account for the change.

4. Methodology

In this work, we will tackle the first major challenge which is to be able to build an unsupervised streaming anomaly detection system. To achieve that, we propose an unsupervised anomaly detection framework for detecting anomalies in real-time. Compared to other approaches, our framework does not require prior clean logs and can perform real time anomaly detection. An overview of our framework is shown in figure 2:

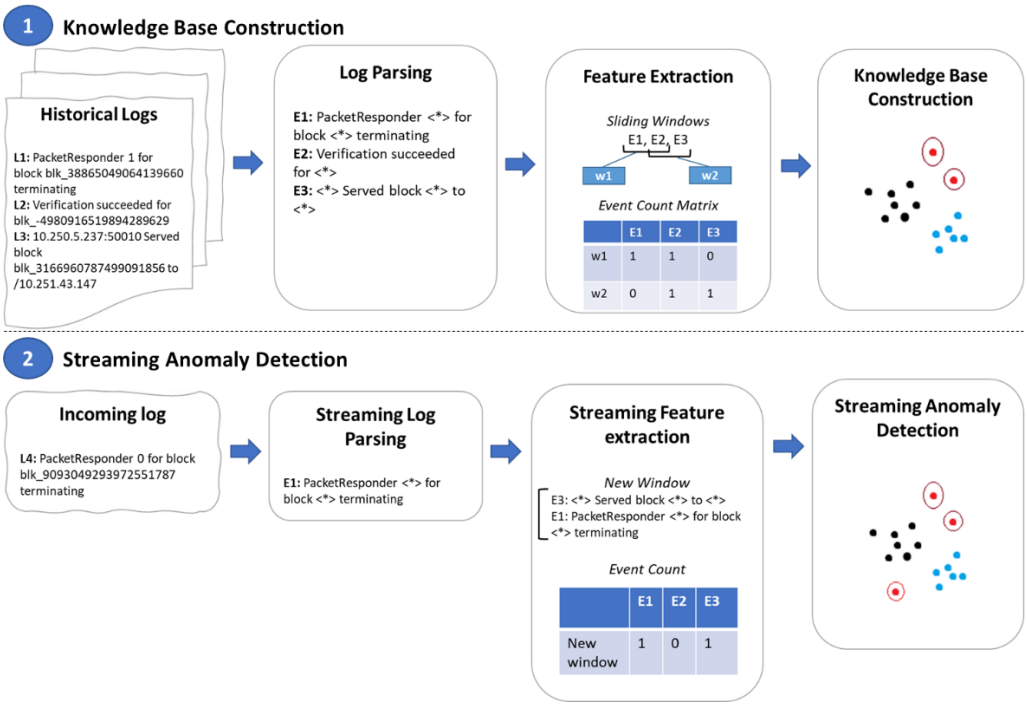


Figure 2 – Unsupervised Anomaly Detection Framework

4.1. Framework overview

Our framework has two main steps**: knowledge base construction and streaming anomaly detection.** The knowledge base construction phase learns normal patterns from the historical logs. During the streaming anomaly detection phase, an event is flagged as abnormal if it exhibits a pattern which differs from normal behavior.

4.1.1. Knowledge base construction

This phase aims at building prior knowledge based on which our system will detect abnormalities. This knowledge base is built from historical logs which may or may not contain anomalies (in contrary to previous works who assumes having clean logs to start with). **The key idea is to determine the most frequent patterns exhibited within a predefined window size**. For example, considering a network log file, we can learn that, within a window of size of 100 events, users usually make 10 queries. Analyzing the log

file within small chunks (windows) is suitable for faster anomaly detection as we will no more need to have the full log file to perform an analysis.

To learn the frequent patterns, the historical logs (training logs) are first parsed to extract log templates [12] (also named log keys in some other works). Next, the log file is divided into equal size windows. For each window, a message count vector is built. Each message count vector is combined to form an event count matrix. We use OPTICS [13] to cluster together similar windows. Finally, the centroid for each of the clusters is computed and represents our frequent patterns.

4.1.2. Streaming Anomaly Detection

The streaming anomaly detection phase allows to determine, in real-time, whether an event is normal or abnormal. To determine whether an incoming event is normal, we analyze it in the context of the most recent window. **We argue that the normality of an event can be inferred from the most recent events which occurred before it**.

Every time a new event occurs, it is first parsed and matched to its corresponding log template. Next, a window, including the new event and the $k - 1$ most recent events before it, is created. Then a message count vector is created for the newly created window.

Next, each message count vector is compared to the closest vector from the knowledge base. If the distance between the vectors is above a threshold δ then, the vector is considered **abnormal**. Otherwise, the vector is considered **normal**. The threshold δ is set as the maximum distance from the centroid to the farthest vector within the corresponding cluster.

Finally, if the vector was considered normal, then we consider the current event as normal. Otherwise, we investigate further by determining whether the current event is responsible for the abnormality of the window.

4.2. Log parsing

Log parsing is the process of extracting for each log line, its template and parameters. As illustrated in Figure 3, every log line is made a **constant part** and a **variable part**. Log parsing extracts the constant part. Since log parsing is not the focus of our study, we reuse the state-of-the art parser proposed in **Drain** [12]. *Drain* is an online log parsing approach with fixed depth tree. It allows to parse logs efficiently in real-time.

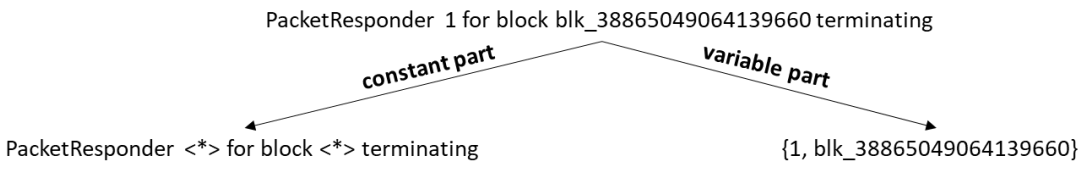


Figure 3 - Log Parsing Illustration

4.3. Feature Extraction

As [10], *we believe that the occurrence of an event strongly depends on the occurrence of the most recent events that occurred before that event*. Hence, we divide the log file into sliding windows. For each window, we capture its pattern by building an event count vector for the window.

An event count vector is built by summing up the number of occurrences of each event within the window. For example, given a system containing 3 events, the vector $[10, 5, 80]$ would mean that, within the window, event 1 has occurred 10 times, event 2 has occurred 5 times and event 3 has occurred 80 times.

After combining all the event count vectors, we obtain an $n * m$ event count matrix where n is the number of windows and m is the total number of events within the system.

4.4. Clustering using OPTICS [13]

To group together similar windows and therefore determine frequent patterns, we use a clustering algorithm, namely **OPTICS (Ordering points to identify the clustering structure)** [14]. OPTICS is a better version of DBSCAN (Density based spatial clustering of applications with noise) [15] algorithm which solves DBSCAN's issue of not being able to cluster points in varying density datasets. Compared to K-Means clustering [16] and Agglomerative Hierarchical clustering [17], OPTICS algorithm is not sensitive to outliers and therefore perfect for anomaly detection. Moreover, it can determine arbitrary shaped clusters and requires few parameter tunings.

OPTICS is a density-based algorithm. It allows to make groups without having to specify the number of groups in advance. This is a great asset for our problem as we do not know in advance how many clusters to expect. Density-based clustering algorithms consider a cluster to be an area with a huge density of points. The clusters are separated by areas of low density. The points in low density areas are usually considered noise or outliers.

OPTICS is based on the observation that given a $MinPts$ (minimum number of points), clusters with a higher density are embedded within clusters with a lower density. The key idea is that higher density points should be processed first. OPTICS retains the clustering order using: the **core distance and the reachability distance**.

The Core Distance is the minimum value of radius required to consider a point as a core point. If a point is not a core point, then its Core Distance is undefined.

The Reachability Distance: the Reachability Distance between two points p and q is the maximum of the Core Distance of p and the Euclidean Distance (or some other distance metric) between p and q.

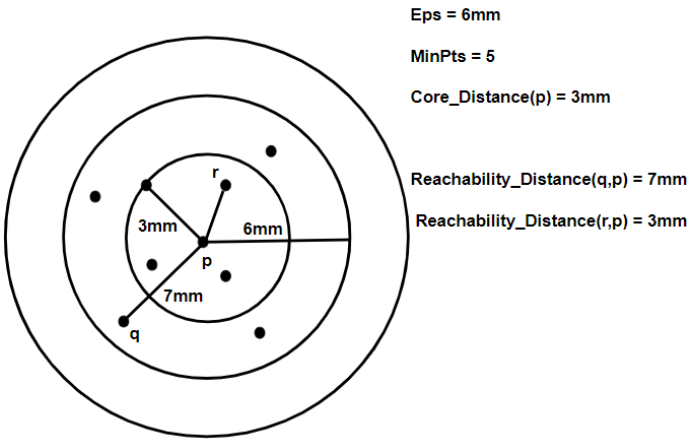


Figure 4 - Core Distance vs Reachability distance [13]

Using OPTICS algorithm, we group together similar windows. Next for each cluster, we compute its centroid. The centroid of each cluster numerically represents a frequent pattern.

4.5. Streaming Anomaly Detection Algorithm

To assess the normality of an event, we first assess the normality of the window containing the event and the $k-1$ most recent events before it. If the window is normal, so is the event. If the window is considered abnormal, we assess whether the current event is a major cause for that abnormality. To achieve that, we find the top j events accounting for the abnormality. If the current event is within the top j candidates, we flag it as abnormal. Otherwise, we flag it as normal.

5. Experiments

In this section, we describe the dataset used for our experiments, our evaluation metrics and our evaluation results.

5.1. HDFS logs Dataset [6]

For our experiments we use HDFS (Hadoop Distributed File System) log data set. It a dataset generated by running Hadoop-based jobs on more than 200 Amazon's EC2 nodes. The dataset has been labeled by Hadoop domain experts.

Hadoop is an open source framework for efficiently storing and manipulating big data [18]. Among 11,197,954 log entries in the dataset, about 2.9% are abnormal. More details about the dataset can be found in [6]. Of the overall 11 million logs, we used 3 million for our experiments.

5.2. Experimental setup

In our experiments, we used 70% of the dataset as training set and 30% as test set.

The code was written using Python 3.7.9 on a windows 10 Enterprise 64-bit desktop with processor Intel(R) Core (TM) i7-9700 CPU @ 3.00GHz (8 CPUs), ~3.0GHz (16 GB RAM). We used the OPTICS implementation from sklearn 0.23.2.

5.3. Evaluation metrics

Our framework is evaluated according to **precision, recall and F1-Score**.

In the following formulas, TP $(True\ Positive)$ represents the number of reported anomalies which were real anomalies. FP $(False\ Positive)$ represents the number of reported anomalies which were not real anomalies. FN $(False\ Negative)$ represents the number of reported normal windows which were anomalies.

Precision determines the percentage of reported anomalies which were real anomalies. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

Recall determines the percentage of anomalies which were caught among all the anomalies. It is computed as follows:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

F1-Score is the weighted average of precision and recall and is computed as follows:

$$F1_{Score} = 2 * \frac{Recall * Precision}{Recall + Precision} \quad (3)$$

5.4. Evaluation results

In this section, we describe the results from implementing our framework on HDFS dataset. We provide the log parsing results, the feature extraction results, and the framework efficiency with respect to our evaluation metrics.

5.4.1. Log parsing Results

Using Drain [10] we were able to determine that our log file contains about 48 different events. Their repartition is shown on the figure below:

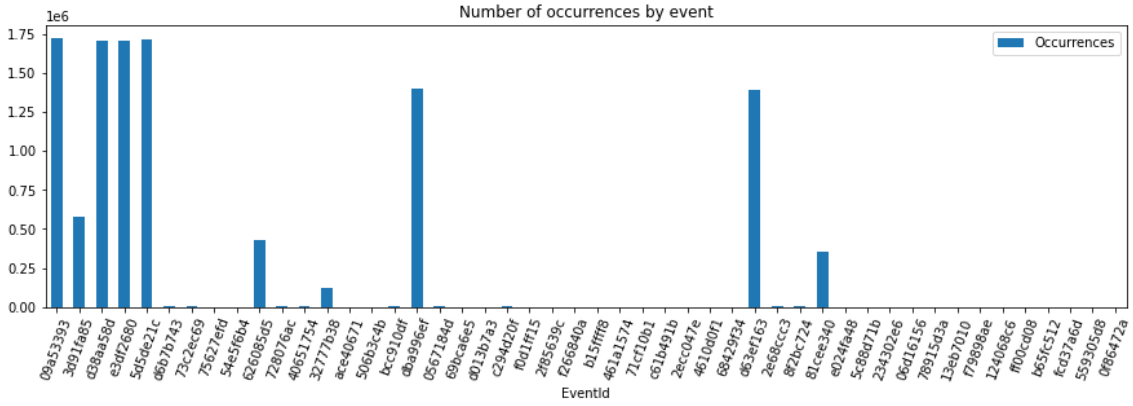


Figure 5 - Number of occurrences of each type of events in the log file

From figure 5, we can notice that only 10 of those events have a real significance in the log file. The table below shows 5 of the most occurring events as well as their number of occurrences in the log file. The following figure shows the repartition of the 10 most occurring events.

Table 1 - Five of the most recurring events

EventId	EventTemplate	Occurrences
09a53393	Receiving block <*> src: <*> dest: <*>	1723232
3d91fa85	BLOCK* NameSystem.allocateBlock: <*> <*>	575061
d38aa58d	PacketResponder <*> for block <*> <*>	1706728
e3df2680	Received block <*> of size <*> from <*>	1706514
5d5de21c	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>	1719741

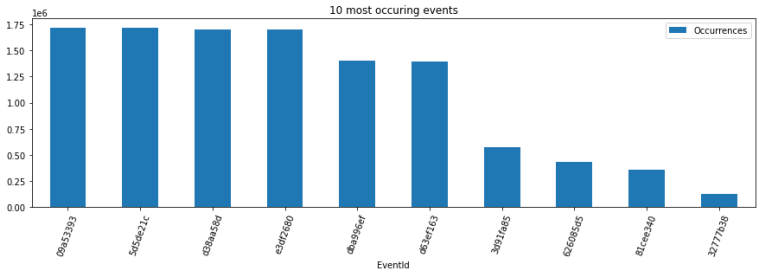


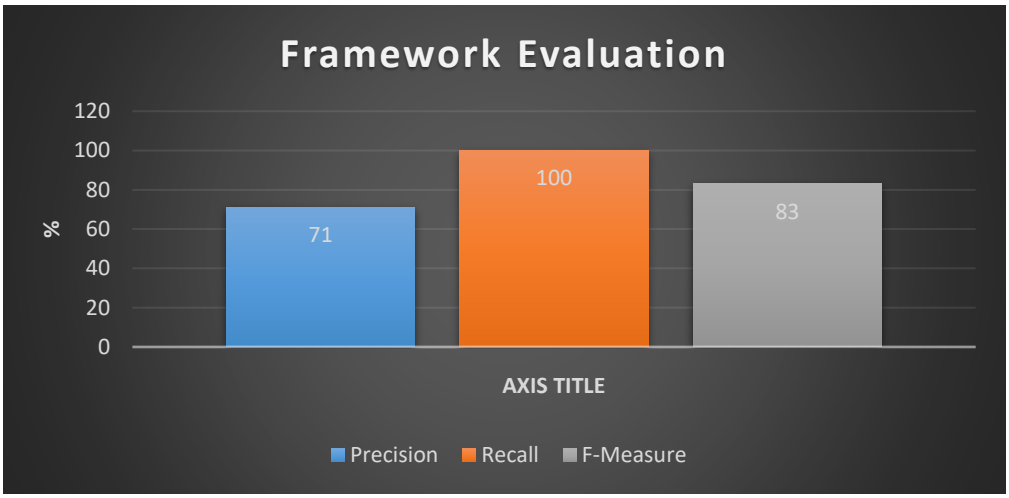
Figure 6 - Count of the 10 most occuring events in the logs

5.4.2. Feature Extraction Results

In our experiments, we used **50** as window size. Approximately 2 million (size of training dataset) windows were generated for training.

5.4.3. Framework efficiency

The chart below shows are framework efficiency.



71% of reported anomalies by our framework were indeed real anomalies. Our framework was able to detect all anomalies with a recall of 100%. The final combined F-measure yields 83% which is a quite high result for an unsupervised framework.

5.4.4. Discussion

It should be noted that we assume that the training logs are in majority normal.

From our results, we can realize that our framework perform pretty well given that it has only being trained on ~2 million log lines. Moreover, we find that the precision is relatively average which is mainly due to the unsupervised nature of the framework. The precision could be made higher by increasing the training size.

Also, we can argue that our results are better than DeepLog [8] which only works in a supervised setting, given normal logs for training.

6. Conclusion and Future Work

With the increasing complexity of today's computer systems, timely and efficient anomaly detection has become necessary to prevent failures. As log files record precious information, they show up as a valuable resource for debugging and preventing failures. However, log file sizes have grown too large for humans to perform timely and efficient analysis. To solve this issue, many researchers have proposed automated anomaly detection frameworks. However, the current state-of-the art fails at providing an anomaly detection framework which can detect anomalies in real time without requiring normal or labeled logs.

In this work, we proposed an unsupervised framework for real time anomaly detection. Our framework does not require neither normal nor labeled logs. Our framework has two main stages: a knowledge base construction stage which uses clustering for determining frequent patterns and a streaming anomaly detection phase for detecting anomalous events in real time. Our framework shows a novel perspective to anomaly detection in which, rather than alerting whenever an event trace is abnormal, we alert whenever an event seems abnormal in the context of the most recent events which occurred before it. We experimented our framework on an HDFS log dataset and obtained a great F-1 score of ~83%.

In the future, we intend to explore the feasibility of an online unsupervised anomaly detection framework which can update itself in real-time without requiring periodic retraining. Moreover, we intend our feature extraction by exploiting the impact of the other rich features embedded in log files.

References

[1] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys,* vol. 41, no. 3, 2009.

[2] R. Ahamed, A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed and M. Imrand, "Real-time big data processing for anomaly detection: A Survey," *International Journal of Information Management,* vol. 45, pp. 289-307, 2019.

[3] D. Forte, "The importance of log files in security incident prevention," in *Network Security*, 2009.

[4] B. Marr, "Forbes," 21 May 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#5b9fd28e60ba.

[5] S. He, J. Zhu, P. He and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in *IEEE 27th International Symposium on Software Reliability Engineering*, 2016.

[6] W. Xu, L. Huang, A. Fox, D. Patterson and M. I. Jordan, "Detecting Large-Scale System Problems by Mining Console Logs," in *Proc. of the ACM Symposium on Operating Systems Principles*, 2009.

[7] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang and X. Chen, "Log Clustering based Problem Identification for Online Service Systems," in *IEEE/ACM 38th IEEE International Conference on Software Engineering Companion*, 2016.

[8] J.-G. LOU, Q. FU, S. YANG, Y. XU and J. LI, "Mining Invariants from Console Logs for System Problem Detection," in *Proc. of the USENIX Annual Technical Conference*, 2010.

[9] X. Zhang, Y. Xu, H. Zhang, Y. Dang, C. Xie, X. Yang, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen and D. Zhang, "Robust Log-Based Anomaly Detection on Unstable Log Data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[10] M. Du, F. Li, G. Zheng and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in *ACM Conference on Computer and Communications Security (CCS)*, Dallas, 2017.

[11] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun and R. Zhou, "Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

[12] P. He, J. Zhu, Z. Zheng and a. M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *IEEE 24th International Conference on Web Services*, 2017.

[13] M. Ankerst, M. M. Breunig, H.-P. Kriegel and J. Sander, "OPTICS: Ordering Points To Identify the Clustering Structure," in *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2011.

[14] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, "OPTICS-OF: Identifying Local Outliers," in *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge*, Prague, 1999.

[15] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[16] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-Means Clustering Algorithm," *Journal of the Royal Statistical Society, Series C,* pp. 100 - 108, 1979.

[17] D. Defays, "An efficient algorithm for a complete-link method," *The Computer Journal - British Computer Society,* vol. 20, no. 4, p. 364–366, 1977.

[18] AWS, "What is Hadoop?," [Online]. Available: https://aws.amazon.com/emr/details/hadoop/what-is-hadoop/.

[19] T. Akidau, "O'Reilly," 5 August 2015. [Online]. Available: https://www.oreilly.com/radar/the-world-beyond-batch-streaming-101/.

[20] M. Shukla, Y. P. Kosta and M. Jayswal, "A Modified Approach of OPTICS Algorithm for Data Streams," in *Engineering, Technology & Applied Science Researh*, 2017.

[21] J. Sander, "Density Based Clustering," *Encyclopedia of Machine Learning,* 2011.

[22] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia and G. Lu, "LogMaster: Mining Event Correlations in Logs ofLarge-scale Cluster Systems," in *31st International Symposium on Reliable Distributed Systems*, 2012.

[23] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu and D. Pei, "Syslog Processing for Switch FailureDiagnosis and Prediction in Datacenter Networks," in *IEEE International Workshop on Quality of Service*, 2017.

[24] Z. LIU, T. QIN, X. GUAN, H. JIANG and C. WANG, "An Integrated Method for Anomaly DetectionFrom Massive System Logs," in *IEEE SPECIAL SECTION ON SECURITY AND TRUSTED COMPUTING FORINDUSTRIAL INTERNET OF THINGS*, 2018.

[25] R. Vaarandi, B. Blumbergs and M. Kont, "An Unsupervised Framework for Detecting Anomalous Messages from Syslog Log Files," in *IEEE/IFIP Network Operations and Management Symposium*, 2018.

[26] Y. Cui, S. Ahmad and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," 2016.

[27] C. Monni and M. Pezz, "Energy-Based Anomaly DetectionA New Perspective for Predicting Software Failures," in *IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019.

[28] M. Astekin, H. Zengin and H. S. ozer, "Evaluation of Distributed Machine LearningAlgorithms for Anomaly Detection fromLarge-Scale System Logs: A Case Study," in *IEEE International Conference on Big Data (Big Data)*, 2018.

[29] C. Bertero, M. Roy, C. Sauvanaud and G. Trédan, "Experience Report: Log Mining using Natural LanguageProcessing and Application to Anomaly Detection," 2017.

[30] W. i. X. u, L. Huang, A. Fox, D. Patterson and M. Jordan, "Online System Problem Detection by Mining Patterns of Console Logs," in *Ninth IEEE International Conference on Data Mining*, 2009.

[31] G. J. Ferrer, "Real-time Unsupervised Clustering," 2016.

[32] Z. Zhao, S. Cerf, R. Birke, B. Robu, S. Bouchenak, S. B. Mokhtar and L. Y. Chen, "Robust Anomaly Detection on Unreliable Data," 2020.

[33] S. Ahmada, AlexanderLavina, S. Purdya and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Elsevier,* vol. Neurocomputing, pp. 134 - 147, 2017.

[34] R. Dominguesa, M. Filipponea, P. Michiardia and J. Zouaouib, "A comparative evaluation of outlier detectionalgorithms: experiments and analyses," *Elsevier,* 2018.

[35] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan and X. Zhang, "Fast Memory Efficient Local Outlier Detectionin Data Streams," in *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 2016.

[36] P. Bodık, M. Goldszmidt, A. Fox, D. B. Woodard and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," *ACM digital libary,* 2010.

[37] M. Chen, A. Zheng, J. Lloyd, M. Jordan and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing*, 2004.

[38] Y. Liang, Y. Zhang, H. Xiong and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs," in *Seventh IEEE International Conference on Data Mining*, 2007.

[39] P. He, J. Zhu, S. He, J. Li and M. R. Lyu, "An Evaluation Study on Log Parsing and Its Use in Log Mining," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016.

[40] V. CHANDOLA, A. BANERJEE and V. KUMAR, "Anomaly detection: A survey," *ACM DL,* 2009.

[41] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso and O. Kao, "Self-Attentive Classification-Based Anomaly," in *ICDM*, 2020.

[42] K. J. S. C. P. Anton A. Chuvakin, Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management.

[43] Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan and S. Li, "Which Variables Should I Log?," in *TSE*, 2019.

[44] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in *ICDM*, 2016.

[45] V. CHANDOLA, A. BANERJEE and V. KUMAR, "Outlier Detection : A Survey," *ACM Computing Surveys,* 2007.

[46] L. Kalinichenko, Shanin and Taraban, "Methods for Anomaly Detection: a Survey," *CEUR Workshop,* 2014.

[47] F.Sabahi and A.Movaghar, "Intrusion Detection: A Survey," in *The Third International Conference on Systems and Networks Communications*, 2008.

[48] J. Zhu, J. LIU, M. R. Lyu, P. He, S. HE and Z. Chen, "Log Analytics Powered by AI," [Online]. Available: https://github.com/logpai.

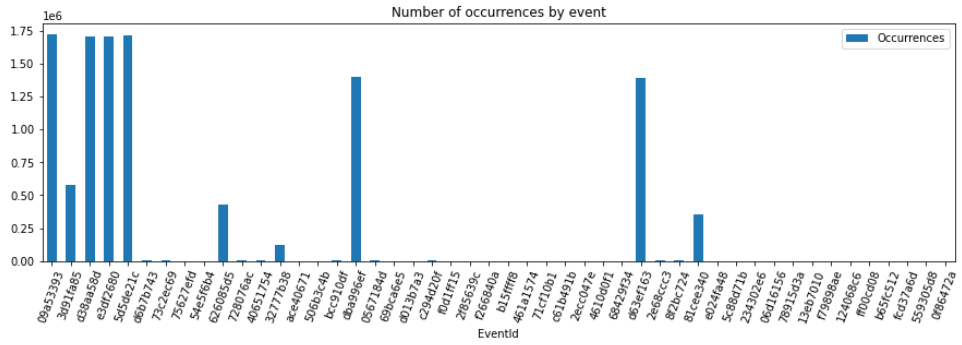
[49] G. Gan and M. K.-P. Ng, "k -means clustering with outlier removal," *Pattern Recognition Letters,* 2017.

[50] Y. Gong, R. O. Sinnott and P. Rimba, "RT-DBSCAN: Real-time Parallel Clustering of Spatio-Temporal Data using Spark-Streaming," in *International Conference on Computational Science*, 2018.

Appendix : Source Code

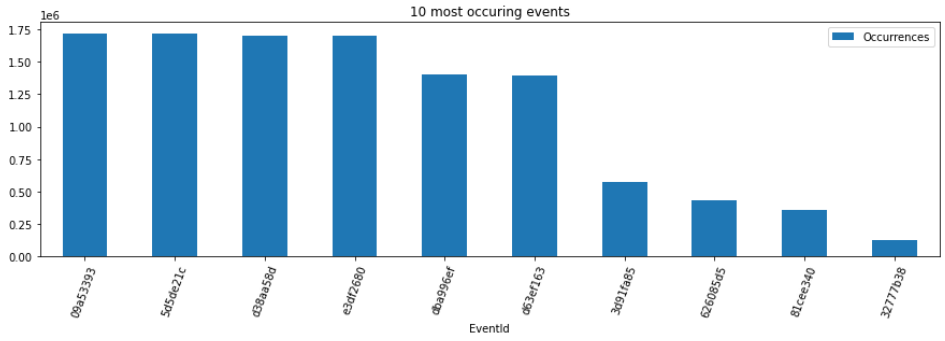
In [1]:
```
#Read log file and log templates

import pandas

full_log_file = pandas.read_csv('Dataset/HDFS.log_structured.csv')
full_log_keys = pandas.read_csv('Dataset/HDFS.log_templates.csv')
experiment_size = 3000000
log_file = full_log_file[:experiment_size]
```

In [2]:
```
#Display the various log keys and their number of occurrences in a graph

import matplotlib.pyplot as plot

full_log_keys.plot.bar(x="EventId", y="Occurrences", rot=70, title="Number of occurrences by event", figsize=(15,4));
plot.show(block=True);
```



In [3]:
```
# Get 10 most relevant log_templates

sorted_log_keys = full_log_keys.sort_values(by=['Occurrences'], ascending=False)
log_keys = sorted_log_keys[:10].reset_index(drop = True)
log_keys.plot.bar(x="EventId", y="Occurrences", rot=70, title="10 most occuring events", figsize=(15,4));
print(log_keys)
plot.show(block = True)
```

```
    EventId                                      EventTemplate  Occurrences
0  09a53393            Receiving block <*> src: <*> dest: <*>      1723232
1  5d5de21c  BLOCK* NameSystem.addStoredBlock: blockMap upd...      1719741
2  d38aa58d              PacketResponder <*> for block <*> <*>      1706728
3  e3df2680           Received block <*> of size <*> from <*>      1706514
4  dba996ef                        Deleting block <*> file <*>      1402047
5  d63ef163  BLOCK* NameSystem.delete: <*> is added to inva...      1396174
6  3d91fa85          BLOCK* NameSystem.allocateBlock: <*> <*>       575061
7  626085d5                          <*> Served block <*> to <*>       428726
8  81cee340         <*>Got exception while serving <*> to <*>       356207
9  32777b38                    Verification succeeded for <*>       120036
```



In [4]:
```
# Get log key ids

log_key_ids = list(log_keys['EventId'])

event_dict = {}
for i in range (len(log_key_ids)):
    event_dict[log_key_ids[i]] = i
print(event_dict)
```

```
{'09a53393': 0, '5d5de21c': 1, 'd38aa58d': 2, 'e3df2680': 3, 'dba996ef': 4, 'd63ef163': 5, '3d91fa85': 6, '626085d5': 7, '81cee340': 8, '32777b38': 9}
```

Split data into training and test set

In [5]:
```
total_logs = len(log_file)
training_size = int(total_logs * 70 / 100)

training_logs = log_file.iloc[:training_size]
test_logs = log_file.iloc[training_size:]

print('Total logs: ', total_logs)
print('Total training logs: ', len(training_logs))
print('Total test logs: ', len(test_logs))
```

```
Total logs:  3000000
Total training logs:  2100000
Total test logs:  900000
```

In [6]:
```
window_size = 50
num_of_events = 10
cols = ['lineId', 'eventId', 'blockId', 'e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8', 'e9']
```

In [7]:
```
import re

'''
  extracts the block id from the parameter list
'''
def get_block(param_list):
    blk_regex = 'blk_-*\d+'
    match = re.search(blk_regex, param_list)

    if (match):
        return match.group()
    else:
        return ''
```

In [11]:
```
import numpy as np

'''
  builds an event count given a sequence of events
'''
def build_event_count(event_sequence):
    lastIdx = len(event_sequence) - 1
    lineId = event_sequence.iloc[lastIdx]['LineId']
    eventId = event_sequence.iloc[lastIdx]['EventId']
    blockId = get_block(event_sequence.iloc[lastIdx]['ParameterList'])

    cnt = np.zeros(num_of_events)

    for i in range (len(event_sequence)):
        if (event_sequence.iloc[i]['EventId'] in log_key_ids):
            current_id = event_dict[event_sequence.iloc[i]['EventId']]
            cnt[current_id] = cnt[current_id] + 1

    return [lineId, eventId, blockId] + list(cnt)
```

In [12]:
```
# Build event count matrix

import csv

'''
  Builds an event count matrix given a log file

  @param logs: the input log file
  @param output_file: the file path of the file containing the result
  @param cols: the column names of the resulting matrix
  @param widow_size: the window size
'''
def build_event_count_matrix(logs, output_file, cols, window_size):
    with open(output_file, 'w', newline='') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(cols)

        first_window = logs[:window_size]
        first_event_count = build_event_count (first_window)
        csvwriter.writerow(first_event_count)

        previous_event_count = first_event_count

        for i in range(window_size, len(logs)):
            current_event_count = previous_event_count

            eventId_removed = logs.iloc[i - window_size]['EventId']
            if (eventId_removed in log_key_ids):
                removed_eventId_index = event_dict[eventId_removed] + 3;
                current_event_count[removed_eventId_index] = current_event_count[removed_eventId_index] - 1

            current_eventId = logs.iloc[i]['EventId']
            if (current_eventId in log_key_ids):
                current_eventId_index = event_dict[current_eventId] + 3;
                current_event_count[current_eventId_index] = current_event_count[current_eventId_index] + 1

            #set lineId, eventId and blockId of current_event_count to current_event's lineId, eventId and blockId
            current_event_count[0] = logs.iloc[i]['LineId']
            current_event_count[1] = logs.iloc[i]['EventId']
            current_event_count[2] = get_block(logs.iloc[i]['ParameterList'])

            csvwriter.writerow(current_event_count)
            previous_event_count = current_event_count
            i = i + 1
```

In [13]:
```
#build event_count for training_logs

training_event_count_matrix_file = 'Dataset/train_event_count_matrix.csv'
build_event_count_matrix(training_logs, training_event_count_matrix_file, cols, window_size)
```

Clustering

In [14]:
```
# Reading the train event count matrix

train_matrix = pandas.read_csv(training_event_count_matrix_file, usecols = cols)
train_matrix = train_matrix[['e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8', 'e9']]
```

In [15]:
```
import hdbscan

clusterer = hdbscan.HDBSCAN(min_cluster_size=11)
cluster_labels = clusterer.fit_predict(train_matrix)
```

Getting representative count vectors

In [23]:
```
#Group count vectors by cluster_id

from collections import OrderedDict

clusters = OrderedDict()

for i in range (0, len(cluster_labels)):
    label = cluster_labels[i]
    if label != -1:
        current_vector = train_matrix.loc[i].tolist()
        if label in clusters:
            clusters[label] = clusters.get(label) + [current_vector]
        else:
            clusters[label] = [current_vector]

cluster_counts = {}
for cluster in clusters:
    cluster_counts[cluster] = len(clusters.get(cluster))
```

In [25]:
```
#computing representative count vectors

representatives = OrderedDict()

for cluster in clusters:
    current_rep = np.zeros(num_of_events)
    current_cluster_vectors = clusters.get(cluster)
    for vector in current_cluster_vectors:
        for i in range (num_of_events):
            current_rep[i] = current_rep[i] + vector[i]
    current_rep = current_rep/num_of_events
    representatives[cluster] = current_rep
```

Computing Anomaly Thresholds

In [20]:
```
import math

def get_distance (v1, v2):
    dist = 0
    for i in range (len(v1)):
        dist = dist + math.pow(v2[i] - v1[i], 2)
    return math.sqrt(dist)
```

In [26]:
```
thresholds = OrderedDict()

for i in representatives:
    #find farthest event count in its cluster
    max_dist = get_distance(representatives.get(i), clusters.get(i)[0])
    for j in range (1, len(clusters.get(i))):
        dist = get_distance(representatives.get(i), clusters.get(i)[j])
        if (dist > max_dist):
            max_dist = dist
    #set max_dist as threshold
    thresholds[i] = max_dist
```

Anomaly Detection

In [22]:
```
# Build event count for test data

test_data = pandas.concat([training_logs.tail(window_size - 1) , test_logs], ignore_index = True)
test_event_count_matrix_file = 'Dataset/test_event_count_matrix.csv'
build_event_count_matrix(test_data, test_event_count_matrix_file, cols, window_size)
```

In [27]:
```
# computes the distance dictionary

def get_dist_dict(event_count, representative):
    result = OrderedDict()
    for i in range (num_of_events):
        result[i] = math.pow(event_count[i+3] - representative[i], 2)
    return result
```

In [28]:
```
# Computes distance to closest representative

def get_distance_to_closest_rep (event_count):
    rep_keys = list(representatives)
    current_dist_dict = get_dist_dict(event_count, representatives[rep_keys[0]])
    current_dist = math.sqrt(sum(current_dist_dict.values()))
    closest_rep_dict = current_dist_dict
    closest_dist = current_dist
    closest_threshold = thresholds[rep_keys[0]]

    for i in range (1, len(representatives)):
        current_dist_dict = get_dist_dict(event_count, representatives[rep_keys[i]])
        current_dist = math.sqrt(sum(current_dist_dict.values()))
        if current_dist < closest_dist:
            closest_rep_dict = current_dist_dict
            closest_dist = current_dist
            closest_threshold = thresholds[rep_keys[i]]

    sorted_closest_dict = dict(sorted(closest_rep_dict.items(), key=lambda item: item[1]))
    current_event_id = event_dict.get(event_count[1])

    if (current_event_id != None):
        anomaly_level = list(sorted_closest_dict.keys()).index(current_event_id)
    else:
        anomaly_level = 0

    return closest_dist, closest_threshold, anomaly_level
```

In [*]:
```
# events classification

test_matrix = pandas.read_csv(test_event_count_matrix_file, usecols = cols)
event_classification_file = 'Dataset/event_classification.csv'
event_classification_cols = ['lineId', 'eventId', 'blockId', 'window_label', 'event_label']

with open(event_classification_file, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(event_classification_cols)

    for i in range (len(test_matrix)):
        current_event = test_matrix.iloc[i]
        dist, threshold, anomaly_level = get_distance_to_closest_rep(current_event)
        if (dist > threshold):
            window_label = 'Anomaly'
            if (anomaly_level > 6):
                event_label = 'Anomaly'
            else:
                event_label = 'Normal'
        else:
            window_label = 'Normal'
            event_label = 'Normal'

        csvwriter.writerow([current_event[0], current_event[1], current_event[2], window_label, event_label])
```

In [*]:
```
# block classifications in relation of each of their respective events

block_classification = OrderedDict()
for i in range (len(event_classification)):
    current_row = event_classification.iloc[i]
    if (current_row[2] in block_classification):
        block_classification[current_row[2]] = block_classification[current_row[2]] + [current_row[4]]
    else:
        block_classification[current_row[2]] = [current_row[4]]
```

In [*]:
```
#get true labels and convert to dictionary

with open('Dataset/anomaly_label.csv', mode='r') as infile:
    reader = csv.reader(infile)
    labels = {rows[0]:rows[1] for rows in reader}
```

In [*]:
```
# Getting final classification result for each block

final_classification_file = 'Dataset/final_classification.csv'

with open(final_classification_file, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(['block', 'experimental_label', 'expected_label'])
    for key, value in block_classification.items():
        total_normal = value.count('Normal')
        if ((total_normal / len(value)) >= 0.5):
            csvwriter.writerow([key, 'Normal', labels[key]])
        else:
            csvwriter.writerow([key, 'Anomaly', labels[key]])
```

Evaluation

In [*]:
```
#precision, recall and f1-score

final_classification = pandas.read_csv(final_classification_file)

tp = 0 #true positives
fp = 0 #false positives
fn = 0 #false negatives

for i in range (len(final_classification)):
    current_block = final_classification.iloc[i];
    if (current_block[1] == 'Anomaly'):
        if (current_block[2] == 'Anomaly'):
            tp = tp + 1
        else:
            fp = fp + 1
    else:
        if (current_block[2] == 'Anomaly'):
            fn = fn + 1

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (recall * precision) / (recall + precision)

print(precision)
print(recall)
print(f1_score)
```