Unlocking Outcome-Oriented Predictive Process Monitoring in Hadoop Distributed File Systems

Giuseppina Andresini[1 [0000−0002−5272−644X]], Vincenzo Pasquadibisceglie[1 [0000−0002−7273−3882]], Donato Cancellara[1], Annalisa Appice[1 [0000−0001−9840−844X]], and Donato Malerba[1 [0000−0001−8432−4608]]

Department of Computer Science, University of Bari "Aldo Moro", Italy
{giuseppina.andresini,vincenzo.pasquadibisceglie,annalisa.appice, donato.malerba}@uniba.it, d.cancellara5@studenti.uniba.it

Abstract. A Hadoop Distributed File System (HDFS) is a distributed file system technology widely used to divide big data into data blocks and distribute them in a Hadoop cluster. HDFS Anomaly Detection refers to approaches to identify data blocks whose behaviours deviate from the typical HDFS behaviour. Finally, Outcome-Oriented Predictive Process Monitoring (OOPPM) refers to approaches to predict the outcome of running business process executions based on the sequences of events recorded with their event traces. In this study, we use OOPPM as a springboard to early predict anomalies on data blocks caused by malicious attacks or malfunctions in an HDFS. To this purpose, we handle the sequences of HDFS log messages recorded for the HDFS data blocks (namely HDFS traces), and realize the OOPPM pipeline ICARUS, to monitor running HDFS traces, and early predict anomalies on their HDFS data blocks. In ICARUS, running HDFS traces are transformed in either feature vectors or textual embeddings, to fuel the OOPPM model learning. In addition, the OOPPM model is learned in both the supervised and unsupervised learning settings, by using an Isolation Forest and a Random Forest, respectively. The evaluation of this study assesses the accuracy of the different versions of the OOPPM models developed in ICARUS by using a benchmark LogHub log. This evaluation analysis is conducted orthogonally to the earliness of the anomaly detection.

Keywords: Outcome-Oriented Predictive Process Monitoring · Earliness Analysis · Anomaly Detection · Hadoop Distributed File Systems

1 Introduction

Hadoop is a powerful distributed computing framework that enables data scientists and industries to analyze big data and support smart effective decision-making in real-world problems. It resorts to a distributed architecture to store big data on multiple computation nodes, and perform parallel processing of stored data in a distributed manner. The Hadoop Distributed File System (HDFS) is a key component of Hadoop to store distributed data in the Hadoop cluster [5]. In the HDFS, big datasets are divided into data blocks, identified by

unique block identifiers, and stored in the nodes of a Hadoop cluster. The HDFS is mainly based on two daemons, named NameNode and DataNode, while the HDFS cluster consists of a master node and multiple slave nodes. The master runs the NameNode daemon, which manages the file system and regulates file access for users. When a slave runs a DataNode daemon, the daemon stores data blocks and serves read/write requests from users. Notably, the HDFS logging on the data blocks is a common practice that makes a big volume of HDFS log messages, available at the data block level, to drive the early detection of signs of malicious attacks or malfunctions, which commonly manifest themselves as anomalies. Specifically, each HDFS log message records a specific system event with a set of fields: the referred data block identifier, the timestamp (the occurrence time of the event), and the event description including the activity performed on the data block (e.g., "Received block" or "Packet responder"), its characteristics (e.g., "size" and "source") and the verbosity level (the severity level of the event, e.g., "INFO") [22].

Predictive Process Monitoring (PPM) [4] is a branch of Process Mining, which aims to predict the unfolding of a process execution based on the event data recorded with its running event trace (i.e., running trace). A running trace is a sequence of events, i.e., activities invoked at a specific timestamp by a process execution from the beginning of its execution to the current time. Recent years have seen the wide proliferation of methods designed for PPM in general, and PPM for outcome prediction in particular— herein called Outcome-Oriented Predictive Process Monitoring (OOPPM) [10, 12, 13, 15, 16]. Existing OOPPM methods mainly enable AI models to classify running traces according to a given set of possible categorical execution outcomes in several domains (e.g., healthcare, manufacturing, finance). However, to the best of our knowledge, no previous study explored the use of OOPPM in HDFS anomaly detection problems. Instead, in this study, we start from the observation that the sequences of HDFS message records, grouped at the data block-level, and sorted by timestamp, embody the definition of event traces well, paving the way for leveraging OOPPM in HDFS monitoring.

Based on the premises introduced above, the present study aims at unlocking OOPPM to early detect HDFS failures manifesting themselves as data block-level sequences of HDFS messages (herein named HDFS traces) labeled with anomalous outcome. To this purpose, we describe an OOPPM pipeline, named ICARUS (predIctive proCess monitoring in hAdoop distRibUted file Systems), to forecast the anomalous outcome for running HDFS traces. In ICARUS, running HDFS traces are obtained in different running stages of the data blocks' life cycle (i.e., after 60 seconds from the the data block creation, after 30 minutes, and so on), and processed for the OOPPM model development. This model development is done along two alternative representations of running HDFS traces: a feature vector representation – that is obtained by computing the counts of distinct activities recorded in running HDFS traces and the time passed from the traces' beginning, and an embedding vector representation – that is obtained by encoding the story telling of running HDFS traces through a well known pre-

trained text encoder (e.g., MediumBert). Finally, in ICARUS, the OOPPM model for anomaly detection is trained in both a supervised setting (i.e., with the label supervision and using a Random Forest algorithm) and an unsupervised setting (i.e., without the label supervision and using an Isolation Forest algorithm). The accuracy performance of the different configurations of ICARUS is evaluated in the benchmark Loghub HDFS log [21]. Specifically, the experimental study allows us to explore how an OOPPM pipeline, designed for HDFS anomaly detection, can be influenced by the adopted trace representation and the learning setting. In addition, this evaluation is done orthogonally to the earliness of the anomaly detection. We note that, to the best of our knowledge, exploring the performance of the "early" detection of HDFS anomalies, in addition to using OOPPM for HDFS anomaly detection, is an innovative contribution in this study.

The paper is organized as follows. Section 2 describes the related work, while Section 3 introduces the basics of this study. Section 4 describes the proposed OOPPM pipeline. Section 5 presents the experimental study. Finally, Section 6 sketches the future work.

2 Related work

The PPM research in anomaly detection has been traditionally dominated by data mining and machine learning approaches designed to learn a model of the normal behaviour of a business process, and identify any deviation during the process executions as an anomaly.

The pioneering research for anomaly detection in event logs includes event-level anomaly detection approaches mainly based on PCA [18] and invariant mining [8], which leverage statistical correlations and invariant relationships between logged events, respectively, to detect anomalous events. More recently, the increasing availability of large-scale event logs recorded during the executions of several processes has also encouraged the adoption of Deep Learning techniques for event-level anomaly detection. One of the first contributions in this direction is DeepLog [6], which uses a LSTM network, trained on logged event sequences of historical process executions, to predict the next-event of any new event sequence and raise an alert as real event that deviates from its prediction. LogAnomaly [9] continues the research in this direction by combining sequential pattern learning with quantitative information extracted from logs.

Another relevant PPM research line, that is also aligned with the specific requirements of anomaly detection in HDFS logs, shifts the focus from detecting anomalies at the level of individual events to detecting anomalies at the level of entire event traces, such those recorded in a HDFS long during HDFS data blocks' sessions. For examples, approaches like LogRobust [20] and PLELog [19] learn global abstractions of event traces, to classify event traces as normal or anomalous. However, these approaches primarily focus on identifying anomalous patterns in complete process executions. Hence, they cannot actually support HDFS security managers, to monitor the progress of data block sessions over time, and provide "early" alerts of any potential anomalies before the com-

pletion of the sessions. Furthermore, trace-level anomaly detection approaches, commonly formulated in PPM, are not designed to handle HDFS-specific structural information such as IP addresses, packet sizes, or software component-level information. Hence, in this study, we address the trace-level anomaly detection task referred to the sessions of HDFS data blocks. To this purpose, we extend the scope of the traditional OOPPM approaches to HDFS logs. In particular, we focus on the early detection of HDFS failures, which manifest as data block-level sequences of HDFS messages (hereafter referred to as HDFS traces) associated with anomalous outcomes.

3 Basics

In this section we introduce the basics regarding information recorded in HDFS logs. An HDFS log records a stream of HDFS messages logged on data blocks run in an HDFS. Each HDFS log message describes the activity executed on a data block at a specific time. For each activity, the source IP, destination IP, packet size and core internal class may be also logged in the message as activity's characteristics. A data block-level HDFS trace, shortly referred herein as HDFS trace, is a sequence of HDFS log messages concerning a specific data block.

Formally, let $\mathcal{I}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{S}$, $\mathcal{D}$, $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{O}$ denote the universes of data block identifiers, HDFS activity names, timestamps, source IPs, destination IPs, packet data size values, core internal class names, and outcome labels (missing, normal, anomaly), respectively. In addition, let $\mathcal{H} = \mathcal{I} \times \mathcal{A} \times \mathcal{T} \times \mathcal{S} \times \mathcal{D} \times \mathcal{P} \times \mathcal{C} \times \mathcal{O}$ denote the universe of HDFS log messages. An **HDFS log message** is a tuple $hdfs = (id, a, t, ipS, ipD, p, c, o) \in \mathcal{H}$, where $id \in I$ is the data block identifier, $a \in A$ is the performed activity, $t \in T$ is the timestamp, $ipS \in S$ is the source IP of the activity, $ipD \in D$ is the destination IP of the activity, $p \in P$ is the size of the allocation block in the activity, $c \in \mathcal{C}$ is the core internal class of the activity, and $o \in \mathcal{O}$ is the outcome label. Notice that ipS, ipD, p and c record activity characteristics. c is mandatory with each activity name, while ipS, ipD and p refer to optional information. The outcome information is available in historical HDFS logs for the model development in the supervised setting. If known, the same outcome label is recorded in all HDFS log messages referred to the same data block. An example of an HDFS log message $hdfs$ is reported in the following:

$hdfs.id$: "blk_3050920587428079149",
$hdfs.a$: "BLOCK* NameSystem.addStoredBlock:blockMap",
$hdfs.t$: "Nov.9.2008::20.41.32"
$hdfs.ipS$: "?"
$hdfs.ipD$: "10.251.43.115"
$hdfs.p$: "67108864"
$hdfs.c$: "dfs.FSNamesystem"
$hdfs.o$: "?"

where "?" denotes a missing information. An **HDFS log** $\mathcal{L}$ is a stream of HDFS log messages: $hdfs_1, hdfs_2, \ldots, hdfs_i, \ldots$ so that, for all $j \geq 1$, $hdfs_j \in \mathcal{E}$ and

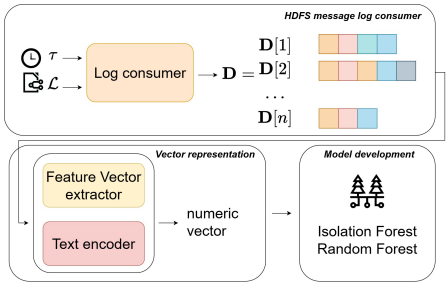

Fig. 1: ICARUS pipeline

$hdfs_j.t \leq hdfs_{j+1}.t$. Given a data block identifier $id \in \mathcal{I}$, and a timestamp $t \in \mathcal{T}$, the **running HDFS trace** $\sigma(id, time)$ of id at the temporal stage $time$, with $time \in \mathcal{T}$, of the data block lifecycle is the sequence $\sigma(id, time) = \langle hdfs_1, \ldots, hdfs_n \rangle$ such that: (1) $hadfs_j \in \mathcal{L}$ for all $j = 1, \ldots, n$, (2) $hdfs_j.id = id$ for all $j = 1, \ldots, n$, and (3) $hdfs_j.t \leq hdfs_{j+1}.t \leq time$ for all $j = 1, \ldots, n-1$.

4 ICARUS Pipeline

In this section, we present ICARUS, the OOPPM pipeline to consume an HDFS log, and predict anomaly-related data block-level outcomes in the "early" stages of the data blocks' lifecycles. This consists of the following components: an HDFS message log consumer, a vector encoding system, a model development system, and an anomaly monitoring system. HDFS logs are used as input of the model development system and the anomaly monitoring system, respectively. Both systems are equipped with an HDFS message log consumer. In addition, they use the vector encoding system to obtain a vector representation of the running HDFS traces that achieve the stage of their data block lifecycle for which they are ready to be processed in the HDFS message log consumer. Figure 1 shows the described pipeline.

4.1 HDFS message log consumer

The HDFS message log consumer takes an HDFS log $\mathcal{L} \subseteq \mathcal{E}^*$ and a data block lifecycle stage quantifier $\tau \in \mathbb{N}$ as input. Specifically, τ denotes the time that should pass from the activation of the considered data block in the HDFS to consider that the running HDFS trace of the data block has finally achieved a lifecycle stage for which it is ready-to be processed in the pipeline. Let $time \in \mathcal{T}$ denote the current timestamp. The HDFS message log consumer consumes each new HDFS message $hdfs \in \mathcal{L}$ as it is logged in $\mathcal{L}$ and records it in a data synopsis $\mathbf{D}$. This data synopsis is an Hash table to record HDFS log messages consumed from $\mathcal{L}$ until the current timestamp $time$. Each cell of $\mathbf{D}$ represents a running data block identifier id, while the entry $\mathbf{D}[id]$ contains the running HDFS trace $\sigma(id, time) \in \mathcal{L}$. Once the τ-based stage of the lifecycle of a data block id is

achieved in $\mathbf{D}[id]$ (i.e., the difference between the timestamps of the last HDFS log message and the first HDFS message recorded in $\mathbf{D}[id]$ is greater than or equal to τ), the running HDFS trace $\mathbf{D}[id]$ is considered ready to be processed in either the model development or the model monitoring systems.

4.2 Vector representation

Given a ready-to-process running HDFS trace $\sigma = hdfs_1, hdfs_2, \ldots, hdfs_n$, this is transformed into a real-valued vector by using two alternative vector extractors:

- **Feature Vector extractor**: This transforms σ into a numeric vector that includes the following features:
  - Activity counts: for each activity $a \in \mathcal{A}$, a count feature is created by counting the number of occurrences of a in σ.
  - Passed time: a time feature is created by measuring the time passed from the beginning of σ until the recording of the last event in σ.
  - IP counts: two IP count features are created by counting the number of distinct IPs recorded in σ as source IP and destination IP, respectively.
  - Core class count: a count feature is created by counting the number of distinct core internal classes recorded in σ.
  - Average packet size: a real feature is created by computing the average of the sizes of packets eventually logged in σ. If this information is missing in all HDFS messages recorded in σ, then the average packet size feature is set equal to zero.
- **Text encoder**: This transforms σ into a numeric vector that is the embedding of the story telling of σ, as it is obtained with a pretrained LLM. In this study, we consider MediumBert [17] as embedding pretrained model for the text encoding of a running HDFS trace, but the used of any other LLM encoder can be experimented. The trace story telling is obtained adapting the story telling trace template introduces in [12] to the specific domain of this study. Specifically, σ is converted into a sequence of sentences, with one sentence for each HDFS log message recorded in σ. In details, the story telling of an HDFS log message $hdfs = (id, a, t, ipS, ipD, p, c, o)$ is created according to the template: **core class**: ⟨ c ⟩ **activity**: ⟨ a ⟩ [**src**: ⟨ ipS ⟩] [**dest**: ⟨ ipD ⟩] [**of size**: ⟨ p ⟩] ⟨ t ⟩ **seconds ago**. As an example, the story telling of the HDFS log message reported in Section 3 is: "**core class: dfs.FSNamesystem activity: BLOCK* NameSystem.addStored Block:blockMap dest: ip1 of size: 67108864 25 seconds ago**". Final notes regard the transformation of IPs in "ip1", "ip2", and so on. This is done before producing the trace story telling, and by renaming IPs according to the order they appear in the HDFS log . This renaming operation follows the guidelines reported in [2], to transfer any decision model trained on IPs' information across multiple network infrastructures.

4.3 Model development

The model development system takes the collection of the numeric feature vectors as input to fuel the model development phase. Input feature vectors represent the running HDFS traces consumed through the HDFS message log consumer of an historical HDFS log. They are obtained through the feature vector extractor or the text encoder, according to the security analyst decision. In the supervised setting, input feature vectors are labelled with binary anomaly labels (also recorded in the HDFS log to fuel the model development supervision). Anomaly labels assume binary values: normal (0) or anomalous (1). Independently of the adopted learning setting, the model development phase trains an anomaly detection function $f\colon \mathbb{R} \mapsto \{0,1\}$, where m is the size of the input feature vectors. In the evaluation of this work, f is trained with a Random Forest [3] in the supervised setting, while an Isolation Forest [7] in the unsupervised setting. Both are competitive machine learning methods proved accurate in several process monitoring studies [11, 1]. In any case, any other supervised method for classification or unsupervised method for anomaly detection may be used in the presented pipeline. Final notes concerns the parameter optimization done within model development phase. This is done by resorting to a 5-fold stratified cross validation and selecting the best parameter set-up according to F1 score. For Random Forest, the parameter optimization step is done for: number of trees $NTrees \in \{100, 200, 300\}$, maximum depth $max\_depth \in \{\text{None}, 10, 20\}$, minimum samples split $min\_samples\_split \in \{2, 5, 10\}$, maximum features $max\_features \in \{\sqrt{\cdot}, \log_2\}$, and class weight $class\_weight \in \{\text{balanced}, \text{None}\}$. For the Isolation Forest, the parameter optimization step is done for: the number of trees $NTrees \in \{100, 200, 300\}$, maximum samples $max\_samples \in \{0.5, 0.8, 1.0\}$, maximum features $max\_features \in \{0.5, 0.8, 1.0\}$, and contamination $contamination \in \{0.001, 0.01, 0.2\}$. The remaining parameters are set as suggested in the implementation documentation. We used the Random Forest [1] and Isolation Forest [2] implemented in sklearn.

4.4 Anomaly monitoring

The anomaly monitoring system takes an anomaly detection function $f\colon \mathbb{R} \mapsto \{0,1\}$ as input. This is used as the decision oracle to query, in order to alert possible anomalies on the data blocks logged in a testing HDFS log. The testing HDFS log is consumed using an HDFS message log consumer configured in the same way as the consumer used in the model development system, i.e, the same lifecycle stage quantifier τ is used to extract the historical running HDFS traces that fuel model development system, as well as the new running HDFS traces that fuel the anomaly monitoring system. As the running HDFS trace of a test data block achieves the status of ready-to-be-monitored in the HDFS message

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

[2] `https://sklearn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest`

Table 1: LogHub description (training log and testing log): number of HDFS messages, number of distinct data blocks, average data block duration in hours, percentage of anomalous data blocks

HDFS Log	N. messages	N. Data blocks	Avg. Data block duration	%Anomaly
Training log	81294	3036	13.9 h	2.83%
Testing log	18706	760	13.7 h	3.28%

log consumer of the anomaly monitoring system, it is transformed into a numeric feature vector as done within the model development system. The anomaly label is finally predicted for the test data block by querying f with the numeric feature vector representation obtained for the running HDFS trace logged for the data block until now.

5 Experimental setting

In this section we illustrate the evaluation study conducted to assess the accuracy performance of the various configurations of ICARUS. To this purpose, we used LogHub [21] that is a HDFS log generated in a private cloud environment using benchmark workloads, and manually labelled through handcrafted rules to identify the anomalies. The evaluation explored the accuracy of ICARUS along the trace representation strategy and the learning setting used with respect to the earliness of the predictive performance done.

5.1 Data and experimental set-up

We considered the HDFS messages recorded in LogHub between 2008-11-09 20:35:18 and 2008-11-10 10:57:38. These messages refer to 3796 data blocks: 97.07% normal data blocks and 2.93% anomalous data blocks. For the evaluation of this study, we adopted the experimental setup commonly used in OOPPM (e.g., [12, 14]). Specifically, we sorted the HDFS traces of LogHub by the timestamp of first event recorded in each trace and selected the first 80% of the HDFS traces for the training stage (for a total of 3036 data blocks), and the remaining 20% of the HDFS traces (for a total of 760 data blocks) for the testing stage. Table 1 reports a summary of the characteristics of the LogHub training data and testing data used in the experimental study.

We considered the OOPPM models developed with the following versions of ICARUS: **S-F**, **S-E**, **U-F**, and **U-E**, where **S** denotes the use of the supervised learning setting, **U** denotes the use of the unsupervised learning setting, **F** denotes the use of the feature vector extractor, and **E** denotes the use of the text encoder. In all versions, we trained the OOPPM anomaly detection models consuming the training LogHub log, while we measured the accuracy performance of the trained models on the testing LogHub log. The accuracy performance was measured with the F1 of Precision and Recall, computed considering the class

"anomaly" as the positive class of the evaluation. To examine the earliness of the predictive ability of the OOPPM models developed and tested in the experimentation, we used the HDFS message log consumers of ICARUS by considering the data blocks ready for the model development and monitoring systems after 60 seconds (1 minute), 1800 seconds (30 minutes), 3600 seconds (1 hour), 10800 seconds (3 hours) and 21600 seconds (6 hours), respectively, from their activation in the HDFS. To this purpose, we ran all versions of ICARUS in the early detection configurations defined with $\tau \in \{60, 1800, 3600, 10800, 21600\}$. As a baseline experimentation of all tested variants, we also considered the late detection configuration LATE, where we used the full HDFS traces recorded for each data block in both the training LogHub log and testing LogHub log.

5.2 Results

Figure 2 shows the F1 of **S-F**, **S-E**, **U-F** and **U-E**, measured in both the considered early detection configurations of ICARUS and the baseline late detection configuration.

The results show that the two supervised versions of ICARUS – **S-F** and **S-E** – commonly outperform the two unsupervised versions –**U-F**, **U-E**– in almost all the tested configurations. The only exceptions are observed in the early detection configuration run with $\tau = 1800$, where **U-F** outperforms the remaining variants, and in the early detection configuration run with $\tau = 10800$, where instead **U-E** outperforms the remaining variants. In any case, we also note that the difference between the performance of the supervised and unsupervised versions of ICARUS becomes more remarkable in the late detection configuration only. With regard to the vector representation, the peaks of F1 in both the early detection configurations and the late detection configuration are achieved with the feature vector extractor.

Finally, focusing the attention on the performance in the early detection configurations, all versions of ICARUS achieve F1 greater than 60% (with a peak of 68.4% with the supervised models of **S-F** and **S-E**) already after 60 seconds from the beginning of the data block sessions (i.e., with $\tau = 60$). Furthermore, the highest F1 is achieved in the early detection configurations when the anomaly detection decisions are delivered after 180 seconds from the beginning of data block sessions (i.e., with $\tau = 180$), while the early detection performance decreases in the remaining configurations, to increase again when anomaly detection decisions are done at the completion of the sessions (i.e., with LATE). This suggests that the discriminative signs of anomalies appear in the first and last steps of a data block session, while the profile of normal and anomalous sessions resemble each other for a long central phase of the sessions. In any case, this deserves further investigations in the future.

6 Conclusions

In this study, we proposed ICARUS, an OOPPM pipeline for early anomaly detection in Hadoop Distributed File System (HDFS) environments. It processes

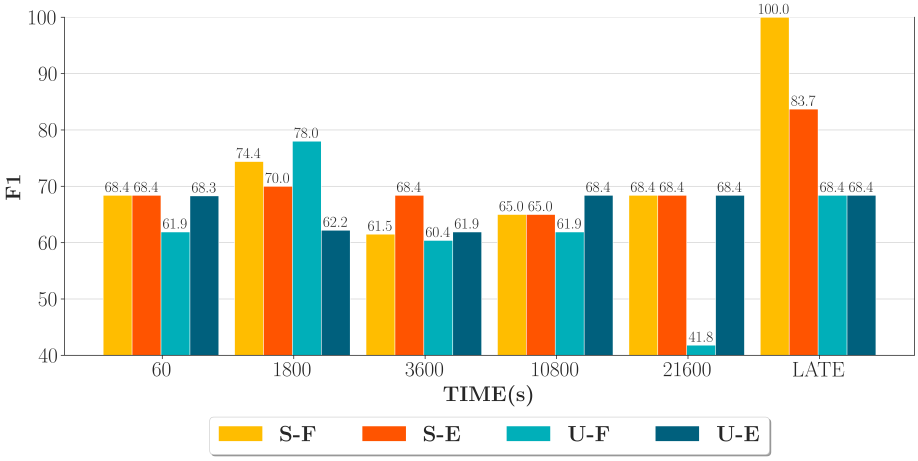

Fig. 2: F1 of **S-F**, **S-E**, **U-F** and **U-E**, run in the early detection configurations monitoring the anomalous behaviour of running HADFS traces with $\tau \in \{60, 1800, 3600, 10800, 21600\}$ seconds) and the late detection configuration LATE monitoring the anomaly behaviour of complete HDFS traces

sequences of HDFS log messages (HDFS traces), transforming them into feature vectors and textual embeddings suitable for the developments of machine learning models. ICARUS supports both supervised and unsupervised learning using Random Forest and Isolation Forest, respectively, by providing complementary approaches to anomaly detection. The evaluation done in the benchmark LogHub log shows that the different versions of the propose pipeline achieve remarkable accuracy performance in detecting anomalous sessions early in the session lifecycle of the corresponding HDFS data blocks. As future work, we plan to extend ICARUS to a streaming setting, to be able to process HDFS events in real time, handling potential concept drifts commonly expected in streaming scenarios. This would enable a transition from offline batch learning to online predictive monitoring. In addition, we plan to investigate the integration of Deep Learning-based approaches in the presented pipeline, and equip decisions with explanation insights.

Acknowledgment

We acknowledge financial support under the National Recovery and Resilience Plan (NRRP), M4C2I1.1, funded by the European Union – NextGenerationEU– Project Title aRtificial intElligence for Process Analytics (REPA) - Grant Assignment Decree No. 2022CJWPNA (CUP Università degli Studi di Bari Aldo Moro H53C24000960006) by the Italian Ministry of University and Research (MUR).

References

1. Aldrich, C., Liu, X.: Monitoring of mineral processing operations with isolation forests. Minerals **14**(1) (2024). https://doi.org/10.3390/min14010076
2. Baahmed, A.R.E., Andresini, G., Robardet, C., Appice, A.: Using graph neural networks for the detection and explanation of network intrusions. In: Machine Learning and Principles and Practice of Knowledge Discovery in Databases - International Workshops of ECML PKDD 2023, Revised Selected Papers, Part III. pp. 201–216. Communications in Computer and Information Science, Springer (2023). https://doi.org/10.1007/978-3-031-74633-8_13
3. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
4. Ceravolo, P., Comuzzi, M., De Weerdt, J., Di Francescomarino, C., Maggi, F.M.: Predictive process monitoring: concepts, challenges, and future research directions. Process Science **1**(1), 2 (2024). https://doi.org/10.1007/s44311-024-00002-4
5. Chnib, M., Gabsi, W.: Detection of anomalies in the hdfs dataset. In: 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA). pp. 243–250 (2023). https://doi.org/10.1109/SERA57763.2023.10197797
6. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1285–1298. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134015
7. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, ICDM 2008. pp. 413–422 (2008). https://doi.org/10.1109/ICDM.2008.17
8. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: 2010 USENIX annual technical conference (USENIX ATC 10) (2010)
9. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., Zhou, R.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. pp. 4739–4745. International Joint Conferences on Artificial Intelligence Organization (7 2019). https://doi.org/10.24963/ijcai.2019/658
10. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D., Modugno, G.: ORANGE: outcome-oriented predictive process monitoring based on image encoding and cnns. IEEE Access **8**, 184073–184086 (2020). https://doi.org/10.1109/ACCESS.2020.3029323
11. Pasquadibisceglie, V., Appice, A., Malerba, D.: LUPIN: A LLM approach for activity suffix prediction in business process event logs. In: 6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14-18, 2024. pp. 1–8. IEEE (2024). https://doi.org/10.1109/ICPM63005.2024.10680620
12. Pasquadibisceglie, V., Appice, A., Malerba, D., Fiameni, G.: Leveraging a large language model (LLM) to predict hospital admissions of emergency department patients. Expert Syst. Appl. **287**, 128224 (2025). https://doi.org/10.1016/J.ESWA.2025.128224
13. Pasquadibisceglie, V., Castellano, G., Appice, A., Malerba, D.: FOX: a neuro-fuzzy model for process outcome prediction and explanation. In:

3rd International Conference on Process Mining, ICPM 2021, Eindhoven, The Netherlands, October 31 - Nov. 4, 2021. pp. 112–119. IEEE (2021). https://doi.org/10.1109/ICPM53251.2021.9576678
14. Pasquadibisceglie, V., Donadello, I., Appice, A., Lanz, O., Maggi, F.M., Fiameni, G., Malerba, D.: Multimodal predictive process monitoring and its application to explainable clinical pathways. Inf. Syst. **139**, 102698 (2026). https://doi.org/10.1016/J.IS.2026.102698
15. Peeperkorn, J., Ortega Vázquez, C., Stevens, A., De Smedt, J., vanden Broucke, S., De Weerdt, J.: Outcome-oriented predictive process monitoring on positive and unlabelled event logs. In: Process Mining Workshops. pp. 255–268. Springer Nature Switzerland (2023). https://doi.org/https://doi.org/10.1007/978-3-031-27815-0_19
16. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Transactions on Knowledge Discovery from Data **13**(2), 1–57 (2019). https://doi.org/10.1145/3301300
17. Turc, I., Chang, M., Lee, K., Toutanova, K.: Well-read students learn better: The impact of student initialization on knowledge distillation. CoRR **abs/1908.08962** (2019)
18. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Largescale system problem detection by mining console logs. In: Proceedings of SOSP. vol. 9, pp. 1–17 (2009)
19. Yang, L., Chen, J., Wang, Z., Wang, W., Jiang, J., Dong, X., Zhang, W.: Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). pp. 230–231 (2021). https://doi.org/10.1109/ICSE-Companion52605.2021.00106
20. Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.G., Chintalapati, M., Shen, F., Zhang, D.: 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. p. 807–817. ESEC/FSE 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338906.3338931
21. Zhu, J., He, S., He, P., Liu, J., Lyu, M.R.: Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics . In: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). pp. 355–366. IEEE Computer Society, Los Alamitos, CA, USA (Oct 2023). https://doi.org/10.1109/ISSRE59848.2023.00071
22. Zhu, J., He, S., He, P., Liu, J., Lyu, M.R.: Loghub: A large collection of system log datasets for ai-driven log analytics. In: 34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023. pp. 355–366. IEEE (2023). https://doi.org/10.1109/ISSRE59848.2023.00071