Provenance-enhanced Root Cause Analysis for Jupyter Notebooks

Ruyue Xin[1a], Simon Stallinga[1a], Hongyun Liu[1], Peng Chen[2b], Zhiming Zhao[1b]
[1]Multiscale Networked Systems (MNS), University of Amsterdam, Amsterdam, Netherlands
[2]School of Computer and Software Engineering, Xihua University, Chengdu, China
[a]These authors contributed to the work equally and should be regarded as co-first authors
[b]Corresponding author. Email: chenpeng@mail.xhu.edu.cn, z.zhao@uva.nl

***Abstract*—With Jupyter notebooks becoming more commonly used within scientific research, more Jupyter notebook-based use cases have evolved to be distributed. This trend makes it more challenging to analyze anomalies and debug notebooks. Provenance data is an ideal option that can create more context around anomalies and make it easier to find the root cause of the anomaly. However, provenance rarely gets investigated in the context of distributed Jupyter notebooks. In this paper, we propose a framework that integrates two data types, provenance and detected performance anomalies based on performance data. We use the combined information to visually show the end-user the provenance at the time of the anomaly and the root cause of the anomaly. We build and evaluate the framework with a notebook extended with anomaly-generating functions. The generated anomalies were automatically detected, and the combined information of provenance and anomaly creates a valuable subset of the provenance data around the time an anomaly occurred. Our experiments create a clear and confined context for the anomaly and enable the framework to find the root cause of performance anomalies in Jupyter notebooks.**

***Index Terms*—Jupyter notebooks, provenance data, root cause analysis, anomaly detection**

I. Introduction

Jupyter notebooks have become more commonly used within scientific research, especially in data science [1]. Their friendly interface and ease of interaction are gaining more daily popularity. Scientific users tend to perform data analysis in a distributed manner because of the increase in the size of datasets [2]. However, Jupyter notebooks do not offer a built-in debugger themselves [3], which makes it harder to detect the anomalies and debug accordingly. In addition, cells in a Jupyter notebook are usually executed in non-chronological order, which makes it difficult to follow the execution flow [4]. Distributed systems allow for a speedup in processing those amounts of data in Jupyter notebooks. While a good way of identifying the root cause of problems within these systems is missing. Therefore, it is essential to have a method that can automatically identify the root cause and give context.

For debugging programming files, different strategies have been provided. One of the first actions taken in debugging is to read the cell with abnormal behavior and understand what it is doing. Strategies to check the output of the cell have been developed [5]. These Strategies come down to a good understanding of the workflow of the cell by the programmer. For larger executions, workflow management systems such as Taverna exist. Taverna stores the execution information data in a provenance format, such as PROV, that can be queried by the end-user [6]. Numerous other frameworks can be used for performance metrics like CPU usage or memory occupancy [7]. The combination of the provenance data and the performance metrics gives a complete picture of the workflow execution context. But, there does not exist a method that uses this complete picture to localize the root causes of anomalies in the execution.

Thus the research question of this paper is formulated as: **how can the combined information of provenance with performance data aid in finding the root cause of performance anomalies in Jupyter notebook executions?** This paper presents a framework that can automatically detect anomalies in performance data and combine the detected anomalies with provenance data to create a clearer insight into the possible root causes of the execution. Our contributions are as follows:

- We propose a framework to localize the root causes of detected anomalies in Jupyter notebooks based on provenance data.
- We compare different anomaly detection models and discuss their results for collected performance data.
- We visualize the provenance data and create a clear picture of the communication between functions at the time of the anomaly.

The rest of the paper is organized as follows. In section II, we review existing research about anomaly detection, provenance, and root cause analysis. In section III, a provenance-based root cause analysis framework and its components are introduced in detail. In section IV, we conduct experiments to evaluate the performance of the framework, including anomaly detection and root cause analysis results. Finally, we provide discussion and conclusion in section V and section VI.

II. Related work

Since workflows in Jupyter notebooks have become more complex and grown in size, lots of research is related to this topic to detect and trace abnormal behavior that occurs in the workflows, e.g., CPU or I/O anomalies [8] [9]. From the existing work, three main research topics can be distilled: anomaly detection, provenance, and root cause analysis.

A. Anomaly detection

Performance anomalies can be detected with a wide range of well-researched methods [10]–[12]. With the vast amounts of monitoring data, machine learning-based techniques are increasingly used for anomaly detection. Because rare labels exist in monitoring data, unsupervised learning methods are mostly used [13]. We can classify them into the nearest neighbor and cluster-based methods.

Nearest neighbor-based techniques are commonly used method for detecting anomalies [14]–[16]. Nearest neighbors do not require labels for the data. These methods do not need a training phase; only distances between data points are calculated. However, this makes the testing phase of the models computationally more expensive than other techniques. These techniques also tend to suffer in high-dimensional data, and if distance computation between points is hard [17]. Finally, these models do not perform well if anomalies occur in the data often.

Like the nearest neighbor, cluster-based techniques [18]–[21] also lack performance in high-dimensional data and if the distance computation is hard. Also, cluster methods detect anomalies as a byproduct of clustering; they are not optimized for anomaly detection. However, this model needs to be trained on the data. This training phase is computationally expensive. However, this results in a computationally cheap test phase. Like the nearest neighbor, cluster-based techniques also do not need labels to function for anomaly detection.

B. Provenance

Provenance can be used in scientific computing to track the evolution of different research assets, e.g., metadata, systems, workflow, or data [22]. Provenance metadata describes an arbitrary process modeled by an arbitrary model with metadata [23]. This meta-provenance is used to automatically detect repairs in software-defined networks [24]. In addition, Gehani et al. [25] created an architecture that records provenance metadata for scientific reproducibility purposes.

In Jupyter notebooks, workflow provenance can be collected. A workflow can be seen as a directed graph where each node is a function or module with input, output, and parameters, and each of the edges is a transfer of information [26] [27]. Elias et al. [28] created a framework, cross-context workflow execution analyzer (CWEA), to connect provenance data with system logs and visualize the combination to aid scientists in detecting anomalies. Souza et al. [29] used workflow provenance data to build a holistic view of the life cycle of scientific machine learning models. Costa et al. [30] proposed architecture for managing workflow provenance data in distributed environments. They offer to use interrelated provenance database servers, forming an integrated environment for managing distributed provenance.

C. Root cause analysis

Root cause analysis is used in many areas to trace a certain outcome to its origin, such as medical [31] [32]. In computer science, it is used to detect the root cause of bugs or errors in workflows [33]. Soldani et al. [34] distilled two main ways for root cause analysis: log-based and distributed tracing.

Logs are often presented in workflows and give contextual information. Zawawy et al. [35] used this method to filter logs according to input from the user. Then this framework used annotated goal trees to model constraints and conditions. Lu et al. [36] extracted a structure and features from the raw log files first. They then detect anomalies and identify the root cause with a General Regression Neural Network (GRNN). For distributed tracing, one of the most basic forms is to create a methodology to compare the traces visually [34]. Gelle et al. [37] included kernel level events in the high-level logs to the tracing to improve the detection of anomalies and relate them to their root cause. Bento et al. [38] went from visualizing tracing data for human inspection to automating the analysis of tracing data.

In conclusion, different methods have been researched and compared for anomaly detection. Various forms of data give different best performers for anomaly detection methods [7]. Therefore, we consider to evaluate detection results with different anomaly detection methods based on performance data collected from Jupyter notebooks. Many structures are extracted from logs or other formats, but the ready provenance is rarely mentioned as a source. Therefore, a root cause analysis method for Jupyter notebook based on provenance data can be explored more.

III. Performance Diagnosis Framework

In this section, we present a provenance-based root cause analysis framework, which can be seen in Figure 1. The framework starts with collecting provenance and performance data from a running Jupyter notebook. Then performance data is used to detect anomalies, and detected anomalies are harmonized and combined with the provenance data. Finally, the combined information gives more context to increase the chance of finding the root cause of the anomalies.

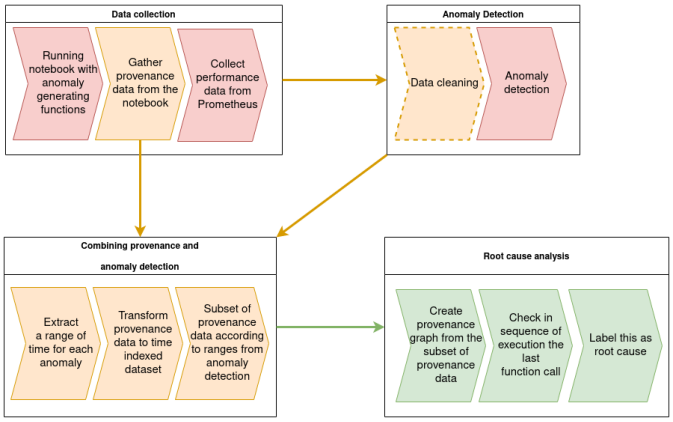

Fig. 1. The design of the provenance-based root cause analysis framework. The red arrows are files imported from others, the orange arrows are the parts we developed, and the green arrows represent the novel part of the framework as a contribution.

A. Data generation and collection

For a running Jupyter notebook, we collect provenance and performance data as shown in Figure 1. We start with a notebook with different anomalies generated. Then, we mainly collect provenance data and computer performance data for subsequent analysis.

1) Collecting provenance: for a Jupyter notebook with anomalies, we collect provenance data via a wrapper function that wraps around every function in the Jupyter notebook. The wrapper logs provenance data is in a W3C PROV format. The functions write the informing information to a log file. The provenance data has the following structure. When a communication took place, we can represent it as function a is informed by function b at time t.

2) Collecting performance data: The open-source monitoring and alerting toolkit Prometheus[1] is used for collecting computer performance data. Performance data includes system resource data, like CPU usage, memory/SWAP occupation, I/O transfer. This data is extracted from Prometheus as multivariate time-series data. At the start of the execution of the notebook, a function registers the start time. At the end of the notebook, a function is added that registers the completion time of the notebook and makes a request to Prometheus for all the collected monitoring data between the start and finish time.

B. Anomaly detection

With collected performance data, we apply four classic anomaly detection models because they usually focus on different features in data and have different performance [7]. The four anomaly detection models are: k-nearest neighbor (KNN), local outlier factor (LOF), support vector machines (SVM), and isolation forest (IForest).

KNN is an unsupervised anomaly detection algorithm [39]. It works by looking around a data point with a certain radius, counting all the neighbors within the radius, and assigning the data point to the cluster with the most neighbors within the radius. To use this technique for anomaly detection, a data point that is not within reach of a minimum number of other data points is labeled as an anomaly.

LOF is an algorithm that detects local anomalies [40]. A data point is considered a local anomaly based on the local neighborhood. LOF uses two functions to determine the local anomaly. It uses KNN to determine the K-distance, or the distance from a point to its k^{th} nearest neighbor to define the neighborhood. LOF also uses the reachability density (RD), which takes the distance to the edge of the neighborhood if the point is within the neighborhood and the distance to the point outside the neighborhood.

SVM is a supervised machine learning algorithm that is frequently used in classification problems [41]. SVM finds hyperplanes in a space that separates one class from the rest. A hyperplane is expressed as seen in the equation.

IForest. [42] created an anomaly detection method they called Isolation Forest (IForest). This method only needs a tiny proportion of data to create an effective model. Resulting in linear time complexity and low memory requirements. Under the hood, IForest is similar to random forest methods since it relies on assembling binary tree structures for the dataset. IForest is different from the random forest because anomalies are nodes in the tree with a short average path length.

With anomaly detection methods, we can determine when anomalies happen, which is anomaly periods. Based on anomaly periods, we can combine provenance data, check function relations, and localize root causes.

C. Combining provenance graph with anomaly detection

This component starts with extracted anomaly periods from the anomaly detection data. After that, the transformation of the provenance data is explained. Finally, it is explained how the previous two steps allow for the creation of provenance subsets that correlate to the period of the anomalies.

1) Different data types: Anomaly detection data is a binary classification over time; it is in the form of: at time t no anomaly but at $t + 1$ there is an anomaly. Provenance data, however, is in the form of agent a and is informed by agent b at time t. Here an agent is a monitored function in a Jupyter notebook. The only way to correlate the two data types is based on time. However, the provenance data is very accurate, at millisecond precision, whereas the anomaly detection data is sparser, with a Δt of 3 seconds.

2) Detecting the time of the anomaly: To combine anomaly detection with the provenance data, the first step is to find the time when the anomaly occurred. This is done by searching the location of the first item of several identifications of anomalies in the binary anomaly detection array. This location is then used to find the time at which the anomaly occurred in the performance data that is gathered from Prometheus. Now the start time and finish time of the detected anomaly are known. With the time known, we need to figure out what function communication occurred around this time.

3) Transformation of provenance data: To make it easier to query the provenance data, we then transform the provenance into a format indexed with time. Since anomaly detection is not that accurate over time, the provenance data doesn't need the accuracy that is present in the data. A data frame is created with all unique communications between the functions in the notebook for the provenance data. So if function a talks to function b and function b talks to function c than there is a column with $a \rightarrow b$ and a column $b \rightarrow c$. Since function a doesn't communicate with function c directly, the link is not present in the columns. This prohibits the average amount of memory needed for the column space to always be in $\Theta(n^2)$ and be more in line with $\Theta(n \log n)$. During every second, all the function calls are counted, and the count is added to the column with the right communication. This gives a row as seen in table I.

Now, it is easy to find the provenance in a given period. To search even quicker, missing seconds, which can be caused by a function that took a while to execute, are filled in with all 0 values. This results in every row describing the situation in

[1]https://prometheus.io/

Time	$a \rightarrow b$	$b \rightarrow c$
1655387530	100	100
1655387531	1	0

TABLE I
EXAMPLE OF HOW THE PROVENANCE DATA FRAME LOOKS LIKE AFTER THE TRANSFORMATION

the second after the previous row. This means that the location of a row with a certain time can be calculated by subtracting the first time from the desired time, which will be helpful for root cause analysis.

4) Creation of subsets of the provenance data: The final part of the combination process is to use the periods extracted from the anomaly detection to select a subset of the provenance data. The time is used to figure out the index. This is possible because of the transformation of the provenance data. The known indexes result in fast access to the row in the provenance data frame where the start and finish of the anomaly have been detected. The rows in between these times are extracted from the provenance data frame.

D. Root cause analysis

This component consists of three parts as shown in figure 1. It starts with the creation of a provenance graph from the subset. After the graph is constructed, the execution flow is checked to discover the last function call. The final part is the root cause analysis.

The root cause analysis starts with creating the graph from the subset of the provenance data. An example of this subset graph is shown in Figure 2. The node in red is the most likely root cause, which is figured out by checking the sequence of the last execution call in the provenance data before the transformation. The final step is to label the informed node as red as the most likely root cause and the informant node as orange to enhance the insight into the last function calls.

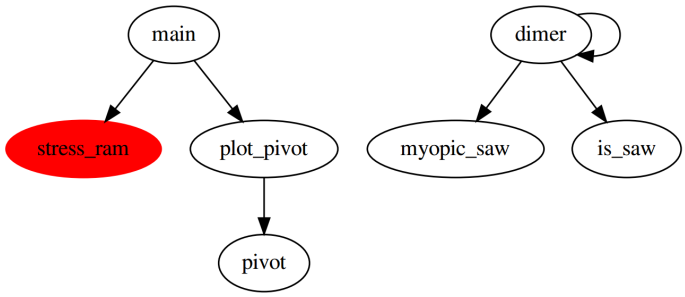

Fig. 2. The flow graph of the function communication when an anomaly has been detected. The node in red is the most likely root cause.

IV. EXPERIMENTS AND RESULTS

We provide experiments to evaluate the root cause analysis framework. First, we will introduce data generation. Then, we will provide anomaly detection results for performance data. Finally, we show the root cause analysis results.

A. Data generation

1) Running Jupyter notebook: We chose a notebook that used Monte-Carlo simulations to simulate self-avoiding random walks[2]. The Markov chains are computationally heavy if they are long enough. The heavy computation makes the notebook run long enough for accurate performance data. We generate two different anomalies: CPU anomaly and memory anomaly.

CPU anomaly generating function. One of the methods to generate anomalies is to overload the CPU. When a function in a Jupyter notebook executes, the CPU will allocate the execution to one core by default unless it is specified to use more cores. Since the random walk notebook does not use multiprocessing, the anomaly would be if suddenly the CPU would activate multiple cores. If a notebook would use multiprocessing that the opposite would be true. A function is created to generate this type of anomaly that would activate multiple cores for a specified time. This causes a spike in CPU usage and would be seen as an anomaly. The function was built to last 30 to 60 seconds to ensure that the spike was present in Prometheus since the data was only collected at an interval of 3 seconds, and it took some time for Prometheus to detect the increase in CPU load.

Memory anomaly generating function. Another method of generating an anomaly is to create a memory overflow. While executing a Jupyter notebook, the notebook stores all variables and intermediate answers in memory. In distributed computing, the container often gets a constrained amount of memory allocated. When the notebook exceeds this amount, the Jupyter notebook kernel crashes. For example, this overflow of memory can be caused by a too large dataset loaded into memory or arrays that get too big. For simulating the overflow of memory, a function is created that allocates a part of memory for an array and steadily increases the size of the array, thus increasing the memory needed to store the array until the kernel crashes.

The CPU data was collected in a 6-hour session, and the memory data in a session of five hours. This resulted in 7586 performance data points for CPU and 5958 for memory. For the provenance data, we collect 74618134 CPU provenance data points and 58468995 memory provenance data points. After data collection, we input these data into our framework, and perform anomaly detection and root cause analysis.

B. Anomaly detection

1) CPU anomaly detection: We perform CPU anomaly detection experiments first. This test was conducted by loading the performance data as received from Prometheus. We remove the columns with a standard deviation of 0 and run the four anomaly detection methods. This gave the results as seen in Table II. The models do perform worse based on accuracy alone. However, all the models detect anomalies with the updated data. Only Isolation forest has decreased in overall

[2]https://github.com/gabsens/SelfAvoidingWalk

performance. SVM, with columns with a standard deviation of 0 removed, is the best performer in detecting anomalies.

	Accuracy	F1	Precision	Recall
KNN	0.81677	0.13570	0.12306	0.12907
LOF	0.84129	0.25823	0.23417	0.24561
SVM	0.86699	0.37048	0.29391	0.32778
IForest	0.83272	0.21504	0.19474	0.20439

TABLE II
THE ACCURACY, PRECISION, RECALL AND F1 SCORE FOR THE DIFFERENT APPROACHES ON THE CPU DATASET WITH THE COLUMN WITH STD OF 0 REMOVED

2) Memory anomaly detection: The second experiment we conducted was the detection of memory anomalies. For this test, memory anomalies were generated, as discussed in section IV-A. We started with feeding the models the whole dataset as it was retrieved from Prometheus. In Table III, the performance of the anomaly detection models is presented after the columns with a standard deviation of 0 were removed. Notice that Isolation Forest, in contrary to the results in Table II, performs better with the 0 standard deviation columns removed. As with the CPU anomaly dataset, the models detect anomalies in this case, except for SVM, which still has a precision, recall, and f1 score of 0. With the memory anomalies, Isolation Forest is the best performer for detecting the anomalies.

	Accuracy	F1	Precision	Recall
KNN	0.91323	0.43025	0.58986	0.49757
LOF	0.86606	0.19463	0.26728	0.22524
SVM	0.92716	0.00000	0.00000	0.00000
IForest	0.92833	0.50588	0.69355	0.58503

TABLE III
THE ACCURACY, PRECISION, RECALL AND F1 SCORE FOR THE DIFFERENT APPROACHES ON THE MEMORY DATASET WITH THE COLUMN WITH STD OF 0 REMOVED

In conclusion, anomaly detection results show that Isolation forest is the best performer in detecting memory-related anomalies, and SVM is the best method for CPU-related anomalies. However, anomaly detection is not performing as well as it should. It is striking to see that the performance of anomaly detection differs a lot between CPU-related and memory-related anomalies. This is likely to the data that Prometheus gives back. For memory, Prometheus records the amount of reserved memory a lot better than the load of the CPU. This gives the anomaly detection model a better representation of the spikes in memory than in CPU usage spikes. Another explanation is that the memory anomalies have more impact on other metrics, which cause more data change during the anomaly period. While CPU change will not affect other metrics more.

C. Root cause analysis

We start with an inquiry about the root cause analysis performed on the CPU dataset. The results consist of graphs that indicate the provenance data when an anomaly is detected. After the inquiry of the memory dataset, the same is done with the dataset of the memory anomalies.

1) CPU root cause analysis: With CPU anomalies detected by SVM, the time anomalies occurred, and the provenance data around the time, the root cause analysis model creates a provenance graph to aid with the localization of the root cause of the anomaly. The root cause localization accuracy of the root cause analysis is at 38%.

From the anomaly detection, the time ranges for the anomaly are extracted. With the CPU data, this came down to 414 anomalies. This was way too much, and after investigating, it turned out that the anomaly time ranges that are calculated for collecting the provenance data often overlapped with each other. This means that the anomaly detection algorithm, SVM in this case, skipped some points in between, and now it seems as if there are two anomalies. To tackle this, we checked the time ranges for overlap, and if they did, we merged them. This resulted in a decrease of anomalies to 29, which is more likely. In Figure 3 the resulting root cause analysis graphs are displayed. As described previously, the red node is the most likely root cause, and the orange node is the function that informed the root cause.

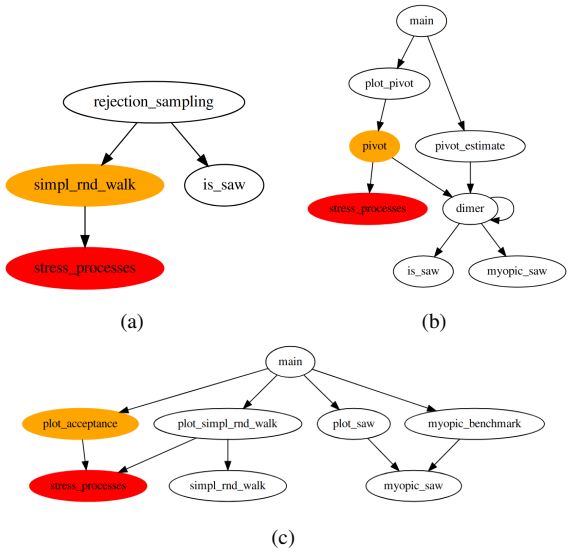

Fig. 3. Provenance graphs with root cause analysis on CPU anomaly dataset. (a) the red node is the root cause. (b) function calls that happened around the anomaly. (c) root causes that are function calls within function calls.

2) RAM root cause analysis: With the detected anomalies in the memory anomaly data using IForest, the time the anomalies occurred, and the provenance data around that time, the model creates a provenance graph to aid with the localization of the root cause of the anomaly. This results in graphs like in Figure 4. The root cause localization accuracy of the model on the memory dataset is at 55%. This shows the correct root cause of the anomaly and the function call relations around it to aid with further analysis. The red node is the last function that was executed in the period that was investigated. This is the most likely point of failure. For further analysis, the orange marked node is the function that informed the red node. Figure 4(a) shows an execution where the anomaly function is called directly in a cell, so no other function was involved. Figure

4(b) and 4(c) shows an example where the function call to the anomaly generating function, was within several functions. It still correctly labeled the 'stress_ram' function as the root cause.

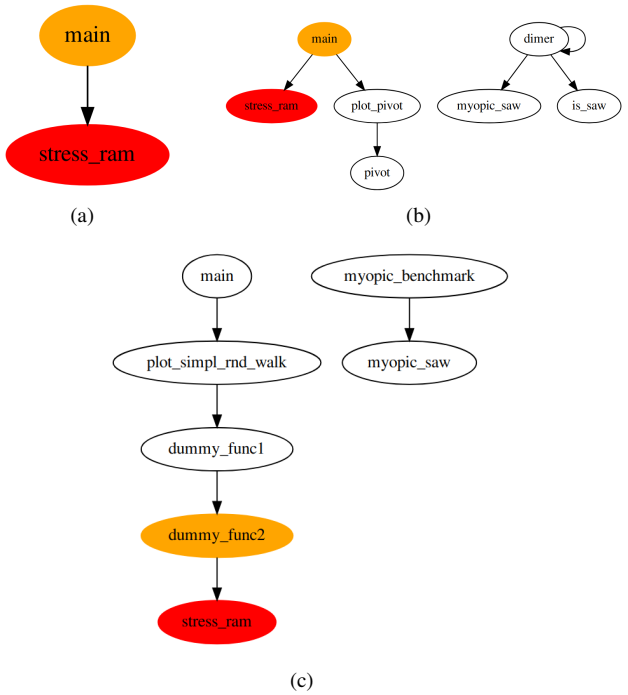

Fig. 4. Provenance graphs with root cause analysis on memory anomaly dataset. (a) the red node is the root cause. (b) function calls that happened around the anomaly. (c) root causes that are function calls within function calls.

In conclusion, the results show that the graphical representation is a clear overview of the provenance data at the time of a detected anomaly. The straightforward method of using the last function call as the root cause is an excellent baseline. For the graphs with the anomaly generating function, in almost all cases, the model selected those functions as the most likely root cause.

On the contrary, the high rate of false-positive results is most likely due to the lower performance of anomaly detection. The lack of precision is reflected later in the model while combining the data. Furthermore, the false positives in anomaly detection create graphs with anomalies that are not there. This has the undesired effect of scientists looking for flaws in their code that are not there.

V. Discussion

The provenance-based anomaly detection model can automatically detect performance anomalies in the CPU and memory. With these anomalies, it can find function communication in the vicinity of the anomaly. This allows the model to build a provenance graph and select a likely root cause in the graph. This gives researchers a good starting point for analyzing the root cause of the occurred anomaly; this means that one of the design requirements, the automation of the system, has been passed.

There are still aspects of the model that have not been explored yet and can potentially improve its performance. One of these things is the configurations of the anomaly detection methods. Due to their unsupervised nature, the parameters of the models are a key factor in the performance of the model since they determine how the model operates. Unfortunately, it was impossible to explore the different settings of the models to improve their performance.

During the creation of the performance anomaly function, it came forward that the way the logging of the provenance data prohibited it from monitoring the function that was executed in parallel during multi-core processing. The explanation is that multiple cores would simultaneously write to the same file, creating clashes, so python prevented this from happening and raised an error. Unfortunately, this meant that the function doing the calculations to create the CPU overload was not included in the provenance data. Therefore, gathering provenance data needs to be adapted to monitor the multiprocessing functions.

Finally, the range of the simulated anomalies is not diverse enough. Due to time limitations, only a memory overflow or sudden fill-up and increased CPU load have been simulated. This leaves room for a lot of other anomalies that can occur. The models may detect some anomalies, but since they are unsupervised, they do not need to train for the new kind of anomalies.

VI. Conclusion

To conclude, our framework monitored a notebook to extract provenance and performance data for memory and CPU anomalies. We tested several anomaly detection methods on the performance data and concluded that IForest and SVM were the best performers for their respective datasets. Then the detected anomalies and the provenance data were harmonized, combined, and analyzed to detect the root cause. To show the results, a new way of visualizing the provenance graphs is used to improve contextual awareness around the occurrence of an anomaly.

There are several areas where the model can be improved or expanded to improve the analysis of the root cause of performance anomalies that occur in scientific notebooks. For example, adjusting the parameters of the unsupervised anomaly detection methods, storing more data, or using other sources for root cause analysis. Furthermore, we consider improving the model to perform run-time analysis in the future.

Acknowledgment

This research is funded by the European Union's Horizon 2020 research and innovation program under grant agreements 825134 (ARTICONF project), 862409(BlueCloud project) and 824068 (ENVRI-FAIR project). The research is also supported by EU LifeWatch ERIC.

REFERENCES

[1] Z. Zhao, S. Koulouzis, R. Bianchi, S. Farshidi, Z. Shi, R. Xin, Y. Wang, N. Li, Y. Shi, J. Timmermans, *et al.*, "Notebook-as-a-vre (naavre): from private notebooks to a collaborative cloud virtual research environment," *Softw Pract Exp. spe.3098 https://doi.org/10.1002/spe.3098.*, 2022.

[2] P. Martin, L. Remy, M. Theodoridou, K. Jeffery, and Z. Zhao, "Mapping heterogeneous research infrastructure metadata into a unified catalogue for use in a generic virtual research environment," *Future Generation Computer Systems*, vol. 101, pp. 1–13, 2019.

[3] J. W. Johnson, "Benefits and pitfalls of jupyter notebooks in the classroom," in *Proceedings of the 21st Annual Conference on Information Technology Education*, pp. 32–37, 2020.

[4] J. Wang, T.-y. Kuo, L. Li, and A. Zeller, "Restoring reproducibility of jupyter notebooks," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering: Companion proceedings*, pp. 288–289, 2020.

[5] I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst, "Debugging distributed systems: Challenges and options for validation and debugging," *Queue*, vol. 14, no. 2, pp. 91–110, 2016.

[6] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, *et al.*, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic acids research*, vol. 41, no. W1, pp. W557–W561, 2013.

[7] R. Xin, H. Liu, P. Chen, P. Grosso, and Z. Zhao, "Fired: a fine-grained robust performance diagnosis framework for cloud applications," *arXiv preprint arXiv:2209.01970*, 2022.

[8] A. Nouri, P. E. Davis, P. Subedi, and M. Parashar, "Exploring the role of machine learning in scientific workflows: Opportunities and challenges," *arXiv preprint arXiv:2110.13999*, 2021.

[9] R. Mork, P. Martin, and Z. Zhao, "Contemporary challenges for data-intensive scientific workflow management systems," in *Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*, pp. 1–11, 2015.

[10] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[11] X. Xu, H. Liu, and M. Yao, "Recent progress of anomaly detection," *Complexity*, vol. 2019, 2019.

[12] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–35, 2015.

[13] P. Chen, H. Liu, R. Xin, T. Carval, J. Zhao, Y. Xia, and Z. Zhao, "Effectively detecting operational anomalies in large-scale iot data infrastructures by using a gan-based predictive model," *The Computer Journal*, 2022.

[14] T. Huang, Y. Zhu, Q. Zhang, Y. Zhu, D. Wang, M. Qiu, and L. Liu, "An lof-based adaptive anomaly detection scheme for cloud computing," in *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, pp. 206–211, IEEE, 2013.

[15] T. Pham and S. Lee, "Anomaly detection in the bitcoin system-a network perspective," *arXiv preprint arXiv:1611.03942*, 2016.

[16] M. Ahmed, N. Choudhury, and S. Uddin, "Anomaly detection on big data in financial markets," in *2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 998–1001, IEEE, 2017.

[17] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[18] A. Marcos Alvarez, M. Yamada, A. Kimura, and T. Iwata, "Clustering-based anomaly detection in multi-view data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 1545–1548, 2013.

[19] G. Pu, L. Wang, J. Shen, and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 146–153, 2020.

[20] J. Li, H. Izakian, W. Pedrycz, and I. Jamal, "Clustering-based anomaly detection in multivariate time series data," *Applied Soft Computing*, vol. 100, p. 106919, 2021.

[21] R. C. Ripan, I. H. Sarker, S. M. Hossain, M. Anwar, R. Nowrozy, M. M. Hoque, M. Furhad, *et al.*, "A data-driven heart disease prediction model through k-means clustering-based anomaly detection," *SN Computer Science*, vol. 2, no. 2, pp. 1–12, 2021.

[22] B. Magagna, D. Goldfarb, P. Martin, M. Atkinson, S. Koulouzis, and Z. Zhao, "Data provenance," in *Towards Interoperable Research Infrastructures for Environmental and Earth Sciences: A Reference Model Guided Approach for Common Challenges* (Z. Zhao and M. Hellström, eds.), pp. 208–225, Springer International Publishing.

[23] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?," *The VLDB Journal*, vol. 26, no. 6, pp. 881–906, 2017.

[24] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "Automated network repair with meta provenance," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pp. 1–7, 2015.

[25] A. Gehani, D. Tariq, B. Baig, and T. Malik, "Policy-based integration of provenance metadata," in *2011 IEEE International Symposium on Policies for Distributed Systems and Networks*, pp. 149–152, IEEE, 2011.

[26] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1345–1350, 2008.

[27] F. Z. Khan, S. Soiland-Reyes, R. O. Sinnott, A. Lonie, C. Goble, and M. R. Crusoe, "Sharing interoperable workflow provenance: A review of best practices and their practical application in cwlprov," *GigaScience*, vol. 8, no. 11, p. giz095, 2019.

[28] E. el Khaldi Ahanach, S. Koulouzis, and Z. Zhao, "Contextual linking between workflow provenance and system performance logs," in *2019 15th International Conference on eScience (eScience)*, pp. 634–635, IEEE, 2019.

[29] R. Souza, L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Vital Brazil, M. Moreno, P. Valduriez, *et al.*, "Workflow provenance in the lifecycle of scientific machine learning," *Concurrency and Computation: Practice and Experience*, p. e6544, 2021.

[30] F. Costa, D. De Oliveira, and M. Mattoso, "Towards an adaptive and distributed architecture for managing workflow provenance data," in *2014 IEEE 10th International Conference on e-Science*, vol. 2, pp. 79–82, IEEE, 2014.

[31] T. C. A. Teixeira and S. H. D. B. Cassiani, "Root cause analysis: evaluation of medication errors at a university hospital," *Revista da Escola de Enfermagem da USP*, vol. 44, pp. 139–146, 2010.

[32] L. A. Lynn and J. P. Curry, "Patterns of unexpected in-hospital deaths: a root cause analysis," *Patient safety in surgery*, vol. 5, no. 1, pp. 1–25, 2011.

[33] R. Xin, P. Chen, and Z. Zhao, "Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications," *arXiv preprint arXiv:2209.02500*, 2022.

[34] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–39, 2022.

[35] H. Zawawy, K. Kontogiannis, and J. Mylopoulos, "Log filtering and interpretation for root cause analysis," in *2010 IEEE International Conference on Software Maintenance*, pp. 1–5, IEEE, 2010.

[36] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, "Ladra: Log-based abnormal task detection and root-cause analysis in big data processing with spark," *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.

[37] L. Gelle, N. Ezzati-Jivan, and M. R. Dagenais, "Combining distributed and kernel tracing for performance analysis of cloud applications," *Electronics*, vol. 10, no. 21, p. 2610, 2021.

[38] A. Bento, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, "Automated analysis of distributed tracing: Challenges and research directions," *Journal of Grid Computing*, vol. 19, no. 1, pp. 1–15, 2021.

[39] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[40] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104, 2000.

[41] W. S. Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[42] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008.