AutoDebugger: Efficient Root Cause Analysis for Anomaly Jobs (Extended Abstracts)

Fathelrahman Ali[*]
Google
Mountain View, CA
fath.elmisbah@gmail.com

Yiwen Zhu[*]
Microsoft
Mountain View, CA
yiwzh@microsoft.com

Lie Jiang
Microsoft
Beijing, China
liejiang@microsoft.com

Zhen Li
Microsoft
Beijing, China
zhli@microsoft.com

Manting Li
Microsoft
Beijing, China
mantingli@microsoft.com

Kun Huang
Microsoft
Beijing, China
kuhuan@microsoft.com

Lijing Lin
Microsoft
Beijing, China
lilijing@microsoft.com

Xiaolei Liu
Microsoft
Beijing, China
xiaolel@microsoft.com

Long Tian
Microsoft
Beijing, China
ltian@microsoft.com

Subru Krishnan
Microsoft
Barcelona, Spain
subru@microsoft.com

ABSTRACT

In the complex infrastructure of today's cloud environment, analyzing performance issues in Spark jobs is a daunting challenge. This paper introduces AutoDebugger, an automated tool crafted to improve the understanding of runtime anomalies in Spark jobs and facilitate automated triaging. AutoDebugger harnesses machine learning algorithms along with a white-box prediction model and integrates with the Spark Metrics Service to establish a comprehensive analytics pipeline. AutoDebugger efficiently identifies performance outliers, and conducts thorough root cause analyses. Notably, AutoDebugger enhances the existing Root Cause Analysis (RCA) algorithm by delivering a speedup of over 10x. Our experiments validate AutoDebugger's efficacy in pinpointing the root causes of anomalies in real-world Microsoft Fabric Spark jobs, ensuring scalability and near real-time analysis capabilities.

VLDB Workshop Reference Format:
Fathelrahman Ali, Yiwen Zhu, Lie Jiang, Zhen Li, Manting Li, Kun Huang, Lijing Lin, Xiaolei Liu, Long Tian, and Subru Krishnan. AutoDebugger: Efficient Root Cause Analysis for Anomaly Jobs (Extended Abstracts). VLDB 2025 Workshop: Applied AI for Database Systems and Applications (AIDB 2025).

1 INTRODUCTION

The flexibility and ease of use of cloud computing have fueled a significant surge in demand over recent years, as users can seamlessly provision and scale resources as needed. Spark has emerged as a preferred choice for handling large datasets in analytics, supported by major cloud vendors like Amazon AWS [1], Microsoft Azure [11], and Google GCP [5]. In the ever-evolving realm of data processing, the efficiency and reliability of job execution are paramount. These cloud platforms provide extensive dashboards that monitor job execution times and other crucial metrics. Despite these advancements, diagnosing issues efficiently when job anomalies occur remains a formidable challenge. Given the system complexity, slowdowns in job execution are inevitable, caused by various system-induced or user-induced factors [15]. For example, in the Microsoft Fabric environment [12]—an all-in-one platform for database management, analytics, messaging, and business intelligence—over 850 recurring Spark jobs were executed in a single week, with more than 100 exhibiting anomalies, some exceeding the average job duration by over 10,000%. Additionally, over 40% of these jobs had instances where execution time doubled the average. Many of these recurring jobs are critical to company operations, and manual error triaging of these anomalies is both slow and error-prone.

The primary goal of AutoDebugger is to enhance the understanding of runtime anomalies within Spark jobs in the Fabric environment and to provide a tool for automated triaging. In the intricate infrastructure of today's Fabric environment, effectively analyzing performance issues for Spark jobs has become increasingly challenging. In this paper, we present an automated root cause analysis tool deployed in production for Fabric users across all regions. This tool addresses several key challenges:

Fragmented Logs. In Fabric, logs are distributed and fragmented, making manual exploration ineffective for managing the growing volume of job instances. A comprehensive analytic pipeline should aim to consolidate all relevant information from disparate sources.

Efficiency Requirement. The algorithm must exhibit very low latency to ensure scalability and facilitate near real-time analysis immediately following the completion of a customer job. Traditional root cause analysis approaches, particularly those based on treatment testing and interventional methods [2, 10, 16], often suffer from substantial computational overhead, as they typically require extensive counterfactual simulations or interventional data.

Lack of Labeled Data. Developing a specific classification model that pinpoints the root cause of job anomalies necessitates labeled data. Unfortunately, such data is unavailable in our system, making these methods impractical for our needs [9].

Feature Correlation. Another approach to root cause analysis leverages global interpretations of black-box models to assess feature contributions [6]. However, predicting job performance with explainable ML models is challenging due to the large number of

[*]Authors contributed equally.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

collected metrics and features. Pure black-box predictors (e.g., Griffon [15] and more recent models [20]) often struggle with accuracy.

To address these challenges, we developed AutoDebugger, which leverages machine learning algorithms in tandem with a white-box prediction model, fully integrated with the Spark Metrics Service that gathers a comprehensive set of metrics for Fabric Spark jobs. A new algorithm was developed to improve on an existing state-of-the-art Root Cause Analysis algorithm with more than 10x speed up. AutoDebugger makes the following contributions:

- AutoDebugger incorporates domain knowledge in its analysis, revealing concrete relationships between metrics.
- We developed a HybridRCA method that improves latency by more than 10x over state-of-the-art algorithms.
- The system is deployed in production for Microsoft Fabric and has been widely adopted by users.

2 BACKGROUND

In Spark job execution within Fabric, we observe that jobs originating from the same **Spark Job Definition** (SJD) or **notebook** (i.e., recurrent jobs) often exhibit varying execution times. For instance, consider a Spark job that performs daily aggregation of customer interaction data across multiple regions and writes the summarized results to a report table. Although such jobs run regularly, their performance can vary significantly due to infrastructure-related delays (e.g., I/O or shuffle bottlenecks) as well as evolving business logic (e.g., increased input volume). We define an **anomalous job** as a job instance whose execution time significantly exceeds expectations, as identified using state-of-the-art anomaly detection algorithms [13]. Each job instance is associated with a large number of metrics that correlate with execution time. However, when an anomalous job appears alongside normal jobs from the same SJD or notebook, isolating the root causes becomes particularly challenging. Understanding these causes is crucial for mitigating performance issues and enhancing job reliability. Inspired by systems such as Griffon [15], our goal in this work is to provide a quantitative breakdown of each factor's contribution to performance degradation in anomalous job instances.

2.1 Spark Metrics Service

Diagnosing job anomalies is challenging due to the need for detailed metrics. The Spark Metrics Service (SMS) addresses this by implementing a Spark listener to capture metrics and a post-processing pipeline that aggregates and stores data in Cosmos DB, providing crucial input for analysis (Figure 1). SMS offers over 40 metrics for Fabric Spark jobs, including read time, cores allocated, and shuffle time, all accessible via an intuitive user interface.

Based on the collected data, several analytics pipelines, including Anomaly Detection and Root Cause Analysis (RCA), are triggered to diagnose job slowness. However, the RCA algorithm's complexity leads to high latency (over 100 seconds per job). Thus, a key objective of our work is to enhance its efficiency, enabling faster and more reliable performance analysis.

3 ALGORITHM

In this section, we present the proposed algorithm for efficient root cause analysis, named HybridRCA.

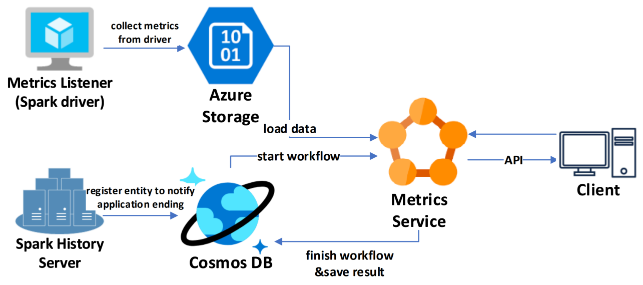
collect metrics from driver
Metrics Listener (Spark driver)
10 01
Azure Storage
load data
start workflow
API
Client
Metrics Service
register entity to notify application ending
Spark History Server
Cosmos DB
finish workflow &save result

Figure 1: End-to-end architecture of Spark Metrics Service.

3.1 Problem Formulation

Root Cause Analysis (RCA) aims to quantify the contribution of each metric or feature to the observed anomaly in a Spark job. Given a causal graph G constructed from domain knowledge and execution metrics, and an anomaly instance characterized by deviations in a set of features, RCA identifies which features have the most significant causal effect on the observed performance degradation.

Formally, let X_0 denote the observable outcome of interest (e.g., job execution time), and let $u \in \mathcal{V}$ denote a feature from the set of input metrics. The objective of RCA is to compute the *causal contribution* of feature u to the anomaly in X_0, denoted as:

$$S_G(u \longrightarrow X_0), \tag{1}$$

where $S_G(u \longrightarrow X_0)$ represents the attribution score of node u to node X_0 under the causal graph G.

Let $\Phi \subseteq \mathcal{V}$ be the subset of features that directly contribute to X_0 (i.e., its immediate descendants in the graph). We assume the total deviation in X_0 between the abnormal and normal execution is attributable to the combined contributions of all its direct causes. Let $\mathbb{E}^*[X_0]$ and $\mathbb{E}[X_0]$ denote the expected values of X_0 in the abnormal and normal cases, respectively. Then the sum of attribution scores over Φ should satisfy:

$$\sum_{u \in \Phi} S_G(u \longrightarrow X_0) = \mathbb{E}^*[X_0] - \mathbb{E}[X_0]. \tag{2}$$

For example, given a Spark job delayed by 10 minutes, we aim to attribute this delay to its immediate causes in the causal graph (e.g., *QueuingTime*, *ApplicationRuntime*), ensuring their attribution scores sum to the total delay. We further decompose delays in *ApplicationRuntime* into components like *StartingTime*, *IdleTime*, *CompilationTime*, and *ExecutionTime*, based on their causal impact. We assume the causal graph G is constructed from domain knowledge and system specifications. Future work may incorporate structure learning techniques (e.g., NOTEARS [19] or FCI [18]) to automate or augment graph construction.

3.2 The Birth of HybridRCA

The traditional treatment-based algorithm [2, 16] applies the do-calculus [17], constructed based on domain knowledge, to evaluate the causal-effect graph [14] that tackles the issues of intercorrelation between input features. The algorithm has been proven to be complete, though suffering from computational complexity, particularly when handling high-dimensional data and complex causal models. Specifically, to evaluate the impact of one feature anomaly on the final observable variable, one needs to enumerate all the combinations of the features given they are both abnormal

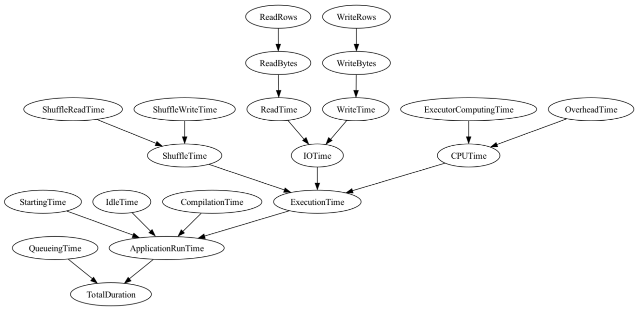

Figure 2: Causal relationships for a subset of metrics.

or not and compute the impact of the factor based on trained ML predictors. For instance, to compute the impact of factor X_1 on the observation X_0, which is also impacted by factors X_2 and X_3, we need to train an ML predictor for X_0 and compute the conditional estimation of X_0 given as the powerset of factors X_2 and X_3, i.e., ($\Omega = \{\}, \{X_2\}, \{X_3\}, \{X_2, X_3\}$), to be abnormal and the difference of observational variable's value given X_1 is normal or not. i.e.,

$$E(X_0|X_1 \cup \Phi \text{ abnormal}) - E(X_0|X_1 \text{ normal}, \Phi \text{ abnormal}), \forall \Phi \in \Omega. \quad (3)$$

The above computation, to iterate over all sets in Ω, takes $O(2^N)$ where N is the number of variables. We have 40+ variables, hence the algorithm takes approximately 100 seconds for a single anomaly job instance. Therefore, there is a pressing need for improvement.

Using metrics from Spark Metrics Service (SMS) and domain knowledge, we constructed the causal graph (Figure 2). We observed that deterministic relationships, such as *ReadRows* and *ReadBytes*, eliminate the need for the $O(2^N)$ algorithm, allowing certain metric scores to be computed in $O(1)$. Additionally, linear relationships, such as *CPUTime*, *ExecutorComputingTime*, and *OverheadTime*, support a divide-and-conquer approach, enabling analysis of smaller subgraphs instead of a single large graph.

Thus, we propose the HybridRCA algorithm, which improves efficiency by applying "divide-and-conquer" to do-calculus. It (1) decomposes the graph to compute local contributions from subgraphs and (2) integrates these results into the full graph.

3.3 Graph Decomposition

The graph decomposition leverages *prominent nodes* to partition a large causal graph G into smaller, more manageable subgraphs. A prominent node is a special type of node in the causal graph whose contribution to its parent node can be explicitly isolated from that of its siblings. This structural property enables the graph to be decomposed around such nodes, facilitating more efficient root cause analysis.

For each prominent node, the graph can be divided into two parts: (1) the *subgraph rooted at the prominent node*, and (2) the *remaining portion of the graph*, where the original subgraph rooted at the prominent node is replaced by the node itself as a leaf.

For example, in Figure 3a, we identify Y_0 as a prominent node because its contribution to X_0 is independent of other inputs, such as X_1, X_2, etc. We can thus partition the graph into: (1) the subgraph rooted at Y_0 (corresponding to G_2 and its descendants), and (2) the modified graph G_1, which contains the rest of the structure with Y_0 treated as a leaf node.

In the following section, we provide a formal definition of prominent nodes and describe the decomposition process in detail.

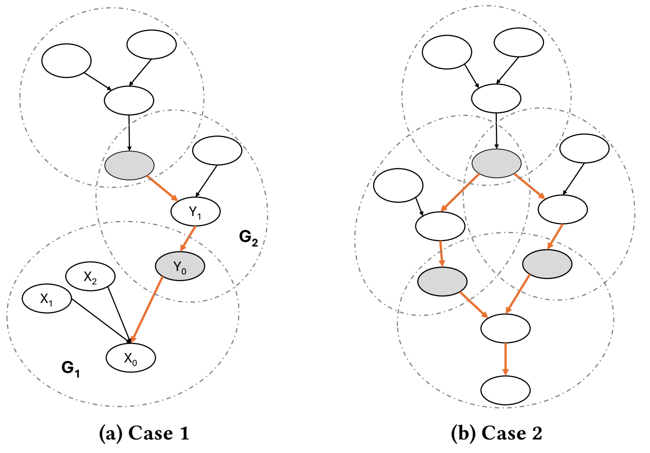

Figure 3: Graph decomposition and combination.

3.3.1 *Assumption: Prominent Node Selection.*

Consider a node X_0 in the graph, as illustrated in Figure 3a. The relationships of X_0 with other nodes in the graph are defined as follows:

$X_1, ..., X_c, Y_0$:	Direct children of X_0;
$X_1, ..., X_c, Y_0, Y_1, ..., Y_e$:	Descendants of X_0;
$Y_1, ..., Y_e$:	Descendants of Y_0;

The relationship between X_0 and its children is expressed as:

$$X_0 = f(Y_0) + g(X_1, ..., X_c) + N_0, \quad (4)$$

where $f(Y_0)$ describes the contribution of Y_0, $g(X_1, ..., X_c)$ captures the contribution of $X_1, ..., X_c$, and N_0 represents the noise associated with X_0. In this context, Y_0 is identified as a *prominent node* because its contribution to X_0 can be explicitly separated through the function $f(Y_0)$. This separability makes Y_0 a natural conjunction point for graph decomposition. Its prominent role simplifies breaking the graph into smaller subgraphs, allowing the subgraph rooted at Y_0 to be analyzed independently while preserving the original structural relationships.

In Figure 2, *ShuffleTime*, and *IOTime* are identified as prominent nodes because of the independent contribution of *ShuffleTime* and *IOTime*.

3.3.2 *Decomposition Approach.*

To perform the decomposition, we divide the graph G into two separate graphs, namely G_1 and G_2, using the following steps:

- G_1 is obtained by removing the subgraph rooted at Y_0 (any prominent nodes) from G and replacing it with Y_0 as a leaf node. If Y_0 is already a leaf node, G_1 remains the same as G.
- G_2 represents the subgraph rooted at Y_0, where the node Y_0 is replaced by $f(Y_0)$.

Instead of running the root cause analysis (RCA) algorithm on the original graph G, we run separate instances of the RCA algorithm on G_1 and G_2. We denote the attribution score of the node u to node v obtained from the RCA algorithm on the graph H as $S_H(u \to v)$.

3.3.3 *Combination Strategy.*

Next, we derive the attribution scores obtained from the original graph G using the attribution scores obtained from the combined graphs G_1 and G_2. For the variables $(X_1, ..., X_d)$ and Y_0, we use the attribution scores obtained from G_1, as they are identical to the scores obtained from G.

$$S_G(u \to X_0) = S_{G_1}(u \to X_0), \forall u \in X_1, ..., X_c, Y_0. \tag{5}$$

For variables $Y_1, Y_2, ..., Y_e$, we use a scaled version of the attribution scores obtained from G_2.

$$S_G(u \to X_0) = S_{G_2}(u \to Y_0)\frac{S_G(Y_0 \to X_0)}{E^*(Y_0) - E(Y_0)}, \forall u \in Y_1, ..., Y_e, \tag{6}$$

where the value of $\frac{S_G(Y_0 \to X_0)}{E^*(Y_0)-E(Y_0)}$ "scales" the computed contribution of nodes according the contribution of Y_0 to X_0 (the root node of the whole graph, i.e., the target) where $E^*(Y_0)$ denotes the anomaly value of Y_0 and $E(Y_0)$ the "normal" value of this feature. One can prove that as long as the contribution of Y_0 and the other variables to X_0 can be separated, this chain rule computes the exact value of the contribution. In cases such as Figure 3b, where multiple paths exist from a node to the root (e.g., via both direct and indirect dependencies), the total contribution of that node is computed as the sum of its propagated contributions along all valid paths. This chaining rule is critical for ensuring accurate attribution in non-trivial graphs.

3.4 Time Complexity Analysis

The original method computes results for every subset of all variables, requiring $O(2^N)$ repetitions. In contrast, the proposed method applies the same process to each subgraph separately, reducing complexity to $O(\sum_i 2^{n_i})$, where n_i is the number of nodes in the i^{th} subgraph. This optimization is particularly beneficial for large variable sets with linear relationships that naturally partition the graph into smaller subgraphs.

4 EXPERIMENT

Production Workloads. To validate the algorithm, we evaluated records from over 30 real Fabric recurrent job groups, comprising more than 2,000 job instances (see Figure 4). Each job had over 50 instances and at least one anomaly. The primary cause identified was longer idle time, partially due to user errors and provisioning delays. The average root cause analysis time per anomaly job was 147 seconds using the original causal structural methods, compared to just 12 seconds with HybridRCA. For root causes contributing at least 5% to the issues, the ranking remained consistent with the original do-calculus algorithm, with an average error of 0.4% and a maximum absolute error of 5%.

Synthetic Scenarios. Additionally, we developed a custom **anomaly data generator** designed to emulate realistic customer usage patterns within Microsoft Fabric Spark. Using this framework, we created five representative scenarios that are known to frequently cause anomalies in real-world settings: (1) Executor Downscale Scenario: A notebook initially runs on a large Spark cluster with 8 executors, but the number of executors is later reduced to 3, resulting in increased job runtime. (2) Executor Upscale Scenario: A notebook begins execution with only 3 executors on a small cluster and is subsequently scaled up to 8 executors, potentially altering job behavior and resource contention. (3) Query Variation Scenario: The notebook executes the same query for multiple hours daily, but the query content varies slightly from day to day, introducing fluctuations in performance. (4) Data Scale-Up Scenario: The notebook first processes a 100GB dataset, but the dataset size is later increased to 1000GB, stressing the system's memory and I/O capabilities. (5) Data Scale-Down Scenario: The notebook initially processes a 1000GB dataset, but the data volume is reduced to 100GB in subsequent runs, potentially altering the job's execution path and optimization strategy. These synthetic scenarios were constructed to isolate different sources of performance variation. The system consistently identified the correct root causes across all those test cases, demonstrating its robustness and suitability for deployment in real operational environments.

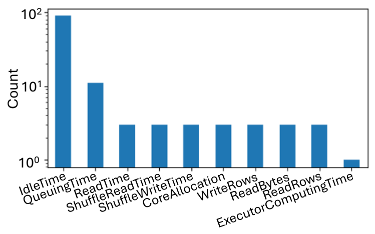

Figure 4: Root cause analysis results on recurrent job groups.

5 RELATED WORK

Various algorithms tackle root cause analysis (RCA) in complex systems. The Bayesian Inference-based RCA (BI) [7] models conditional probabilities but operates at the dataset level, lacking instance-specific insights. The Hypothesis Testing-based RCA (HT) [8] identifies root causes via statistical tests but assumes linear relationships and independent effects, overlooking complex dependencies. Sawmill [10] computes Average Treatment Effect to infer root cause for query latency. Adaptive Interventional Debugging (AID) [4] employs a feedback-driven approach with dynamic instrumentation to iteratively refine fault localization in software execution environments. Explanation Tables [3] generate human-readable rules that link specific parameter settings to failures, providing an interpretable method for understanding binary outcomes in computational workflows. The do-calculus [17], widely used for estimating causal effects from observational data, eliminates the need for controlled experiments by leveraging domain knowledge. However, applying do-calculus to high-dimensional data, such as Spark job performance metrics, presents significant computational challenges due to the necessity of evaluating all possible feature combinations.

6 CONCLUSION

In this work, we presented an efficient root cause analysis framework for performance anomalies in Spark jobs, leveraging a causal graph and efficient attribution techniques. Our method accurately identifies key contributors to performance degradation and has been validated through synthetic anomaly scenarios and real production workloads. In the future, we plan to extend the current definition of prominent nodes to allow for more granular decomposition and generalize the algorithm to other graph-based RCA methods. Furthermore, we envision AutoDebugger as a critical component of an autonomous database system, where root cause signals can be fed into adaptive control loops for workload reconfiguration, resource tuning, or query rewriting.

REFERENCES

[1] Amazon.com, Inc. 2025. *Amazon Web Service*. Retrieved May 28, 2025 from https://aws.amazon.com/

[2] Patrick Blöbaum, Peter Götz, Kailash Budhathoki, Atalanti A. Mastakouri, and Dominik Janzing. 2022. DoWhy-GCM: An extension of DoWhy for causal inference in graphical causal models. *arXiv preprint arXiv:2206.06821* (2022).

[3] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. *Proc. VLDB Endow.* 8, 1 (Sept. 2014), 61–72. https://doi.org/10.14778/2735461.2735467

[4] Anna Fariha, Suman Nath, and Alexandra Meliou. 2020. Causality-Guided Adaptive Interventional Debugging. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 431–446. https://doi.org/10.1145/3318464.3389694

[5] Google. [n.d.]. Google Cloud Platform. https://cloud.google.com. Accessed: 2024-05-25.

[6] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[7] David Heckerman. 2008. A tutorial on learning with Bayesian networks. *Innovations in Bayesian networks: Theory and applications* (2008), 33–82.

[8] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 3230–3240. https://doi.org/10.1145/3534678.3539041

[9] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. 2017. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 389–396.

[10] Markos Markakis, An Bo Chen, Brit Youngmann, Trinity Gao, Ziyu Zhang, Rana Shahout, Peter Baile Chen, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. 2024. Sawmill: From Logs to Causal Diagnosis of Large Systems. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 444–447. https://doi.org/10.1145/3626246.3654731

[11] Microsoft. 2024. *Microsoft Azure*. Retrieved May 28, 2024 from https://azure.microsoft.com

[12] Microsoft. 2024. *Microsoft Fabric*. Retrieved May 28, 2024 from https://www.microsoft.com/en-us/microsoft-fabric

[13] Microsoft. 2025. *What is Anomaly Detector?* Retrieved May 28, 2025 from https://learn.microsoft.com/en-us/azure/ai-services/anomaly-detector/overview

[14] Judea Pearl. 2009. *Causality*. Cambridge university press.

[15] Liqun Shao, Yiwen Zhu, Siqi Liu, Abhiram Eswaran, Kristin Lieber, Janhavi Mahajan, Minsoo Thigpen, Sudhir Darbha, Subru Krishnan, Soundar Srinivasan, et al. 2019. Griffon: Reasoning about Job Anomalies with Unlabeled Data in Cloud-based Platforms. In *Proceedings of the ACM Symposium on Cloud Computing*. 441–452.

[16] Amit Sharma and Emre Kiciman. 2020. DoWhy: An end-to-end library for causal inference. *arXiv preprint arXiv:2011.04216* (2020).

[17] Ilya Shpitser and Judea Pearl. 2006. Identification of conditional interventional distributions. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence* (Cambridge, MA, USA) *(UAI'06)*. AUAI Press, Arlington, Virginia, USA, 437–444.

[18] Peter Spirtes. 2001. An Anytime Algorithm for Causal Inference. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Thomas S. Richardson and Tommi S. Jaakkola (Eds.), Vol. R3. PMLR, 278–285. https://proceedings.mlr.press/r3/spirtes01a.html Reissued by PMLR on 31 March 2021.

[19] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. 2018. DAGs with NO TEARS: Continuous Optimization for Structure Learning. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/e347c51419ffb23ca3fd5050202f9c3d-Paper.pdf

[20] Yiwen Zhu, Rathijit Sen, Robert Horton, and John Mark Agosta. 2023. Runtime Variation in Big Data Analytics. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–20.