Root Cause Analysis of Failures in Microservices through Causal Discovery

Anonymous Author(s)
Affiliation
Address
email

Abstract

The majority of the production web services use a large number of smaller sub-components (called microservices) that interact with each other, in the form of a complex graph to provide the overall functionality to the user. While modularity of the microservice architecture is beneficial for rapid software development and deployment, maintaining and debugging such a system quickly in case of a failure or a poor performance is challenging. We propose a scalable algorithm for quickly detecting the root cause of failure in complex microservice architectures. The key ideas behind our novel localized hierarchical learning approach are: (1) to treat a failure as an intervention on the root cause to learn the underlying causal structure, (2) only learn the portion of the causal graph related to the root cause, thus avoiding a large number of costly conditional independence tests, and (3) hierarchically learn the relevant causal graph. The proposed technique is highly scalable and can produce useful insights about the root cause, while the use of traditional techniques becomes infeasible due to high computation time. Our solution is application agnostic and only relies on the data collected for diagnosis. For our evaluation, we compare our proposed solution with a modified version of the PC algorithm and the state-of-the-art for root cause analysis. The results show a significant improvement in top-k precision while keeping the execution time reasonable.

1 Introduction

Root Cause Analysis (RCA) is widely used to ensure reliability of production systems in many domains such as IT operations[34], telecommunications[44], and medicine[28]. Modern production systems follow a network topology consisting of multiple components connected to each other through complex dependencies. For example, cloud applications typically follow microservice-based architecture [35]. A cloud application with microservice-based architecture makes development process simple, maintenance easy, and deployment rapid and flexible.[22]. A typical cloud-based application uses several microservices, each performing a small task. Each microservice is instrumented to monitor data which results in thousands of metrics to monitor. The large number of metrics makes it challenging to figure out the root cause of the failures in a timely fashion. It has been reported that it takes on average 3 hours to identify the root cause of a failure without using any automated tools [40]. Since enterprise cloud services provider have prices tied with availability guarantees, a delay in detecting root cause of a fault can result in significant revenue loss.

Causal structure discovery-based [36, 1] Root Cause Analysis has been used recently to find the root cause(s) of a fault [30, 40, 14]. These papers construct a graph with nodes as metrics and a directed edge between two nodes showing the direction and size of the causal effect. A direct application of a causal structure discovery algorithms is infeasible due to the large number of metrics, especially for graphs that are not sparse. Existing approaches reduce the total number of nodes by using a feature

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

selection approach or selecting a specific set of metrics[38, 16, 40]. The obvious problem with this approach is that the root cause metric might be removed by the feature selection approach or the selected set of metrics does not include the true root cause. In addition, the feature selection step might introduce latent variables, which renders the popularly used PC algorithm not applicable.

We make two crucial observations. First, a fault changes the generative mechanism of the failing node. Hence, a fault can be modelled as an intervention on the failing node and the data during the fault period as an interventional dataset. We borrow ideas from recent work on learning causal graph using interventional distribution with unknown interventional targets [12, 20]. In summary, a binary indicator variable for normal and fault period is introduced as an additional node. The target metrics of this nodes are the root causes. Second, given the first observation, we do not need to learn the full graph for RCA. Using the above two observations, we propose a novel localized causal discovery algorithm to learn the causal structure. The algorithm find the root causes of the fault without learning the causal structure of the complete graph. This allows the algorithm to run on a much larger set of metrics. We further optimize the algorithm by selecting the nodes in a hierarchical manner. We call the algorithm ***Root Cause Discovery (RCD)***. We validate RCD on a synthetic dataset and data obtained from Sock-shop application test bed. The results show a significant improvement in runtime and accuracy as compared to the baselines. In addition, we also apply the algorithm on a real-life failure of a microservice from a large cloud service provider. RCD correctly identifies the root cause metrics in a fraction of time than the baseline. Summary of our contributions are as follows:

- We consider failures as interventions on the root cause nodes for the first time in the literature. Doing so allows us to use distributional invariances not only within the observational, but also across observational and interventional data for learning.
- We propose a novel solution to identify the root causes of the failure using a localized hierarchical learning algorithm. Our algorithm is not only more computationally efficient, but it also requires much fewer anomalous samples compared to the existing approaches.
- We evaluate our system on synthetic datasets as, data generated from a real application and also report our finding from a production ready microservice.

2 Background and Motivation

In this section, we provide the background of concepts that underlie our design.

Causality and Causal graphs. A variable is said to cause another variable if intervening on the former induces a change in the distribution of the latter. A causal graph encapsulates the causal relationships between variables using a directed acyclic graph (DAG). Every node of the DAG represents a variable, while a directed edge shows the causal relation between the two variables: $X \rightarrow Y$ indicates that X causes Y.

Structural causal models (SCMs) are typically used to model causality between a set of random variables. Accordingly each variable X is assigned a value based on a functional relation with a subset Pa_X of observed variables and an exogenous noise E_X as $X = f_X(Pa_X, E_X)$. One can then construct the causal graph by assigning the set of observed variables Pa_X as the parents of X for all X. Causal Bayesian networks (CBNs) similarly can be used for defining a causal model without specifying the functional relations, and instead specifying observational and interventional distributions through the so-called truncated factorization formula in accordance with the causal graph. For a formal treatment of the subject, please refer to [26].

An intervention on a variable is the process of changing the generative mechanism of that variable. Most widely used notion of interventions are in randomized controlled trials (RCTs), and A/B tests, where we would like to understand the causal effect of a certain treatment. This type of intervention is captured by the do-operator of Pearl [26], and destructs the previous generative mechanism. For example, $do(X = x)$ forces X to take the value of x, effectively removing the role of its structural equation. This is also called a *hard intervention*, and its effect in the causal graph is to sever the incoming edges to the intervened node. On the other hand, *soft interventions* are used to model the experiments that do not completely destruct the causal mechanisms, but modify them. For example a soft intervention on X can replace the structure equation $f_X(Pa_X, E_X)$, with some $f'_X(Pa_X, E_X)$, where $f' \neq f$, which then retains the original causal graph but changes the generative mechanism and the conditional distribution $p(X|Pa_X)$.

Microservices and Causality. There are a set of tools that provide observability into a microservice-based system [23, 46, 10]. Most of these tools yield information about the individual microservices or at best the call graph of microservices involved in different types of requests. These tools are used to monitor the health of microservices as well as provide some insights in the case of failures. However, we argue that the observability these tools offer is limited. A detailed description of these can be found in appendix.

Consider a case where two microservices are deployed on a common host. Further, assume that these two microservices are completely unrelated in that they do not invoke each other for any type of request. However, because they are deployed on the same host, they share the physical resources of the host (e.g., network bandwidth). If one of the microservices, say s_1, fails in a way that it starts to consume most of any shared physical resource, the other microservice, say s_2, will also be affected. However, an SRE will have difficulty finding the root cause of the failure that is affecting two unrelated microservices. The problem becomes worse when we start to consider thousands of microservices with multiple instances (an instance of a microservice is its replica). This happens because of the transitive nature of failure propagation. In this instance, all microservices that depend on s_2 will also be affected. This effect, when observed in a cluster of thousands of microservices, makes it challenging to perform fault localization.

Causality can help find the root cause of the failure in such cases. Causal discovery methods are useful when no information about the topology such as call graph is available. Generating a call graph at run-time is costly and might require changes to the application whereas system topology mostly requires an expert's input. With causality, we can find the hidden dependencies between the metrics of different microservices using statistical conditional independence (CI) tests. Doing so enables us to observe the dependencies between microservices that might not be intuitive if we restrict ourselves to only call graphs.

Causal Discovery. Causal discovery is a well-known problem in causal inference and has been studied thoroughly [26, 42, 4, 2]. It has applications in social sciences, bioinformatics, economics, and has recently found its way into machine learning as well [24, 3, 21, 27, 43, 11]. Broadly speaking there are two type of algorithms for causal discovery, **Score-based causal discovery** [7, 31] and **Constraint-based causal discovery** [36, 9]. Score-based algorithms aim to find the causal structure by optimizing a properly defined score function. Among them, Greedy Equivalence Search (GES) [7] is a well-known two-phase procedure that directly searches over the space of equivalence classes.

The constraint-based causal discovery algorithms, on the other hand, start with a complete graph and remove the edges between a pair of variables if they are conditionally independent. To test the independence of two variables, denoted as $X \perp\!\!\!\perp Y$, it uses some well-established statistical conditional independence (CI) tests such as Fisher z-transformation or G-test for hypothesis testing. In contrast to score-based causal discovery algorithms, there have been several studies done for constraint-based algorithms in the presence of interventional data [13, 20, 45].

A renowned constraint-based causal discovery algorithm is the PC algorithm [36]. PC algorithm works in two phases. The first phase is where it tries to learn the skeleton graph (undirected graph) and in the second phase, it tries to orient the edges between the nodes. Starting with a completely connected graph, it picks a connected pair of variables; let us call them X and Y, and runs a CI test to check if the variables are dependent or not. If they are independent, $X \perp\!\!\!\perp Y$, it removes the edge between the two variables. Next, if two variables are still connected in the graph, it tries to find a separating set S, such that conditioning on that set makes the variables independent. To try to find the smallest S, after every iteration, it increases the cardinality of the separating set by 1.

The output of the first phase of the PC algorithm is an undirected skeleton graph, where an edge represents a dependence between the two connecting variables. The next step is to orient the edges of the skeleton graph. The PC algorithm achieves this with a set of rules [18].

From the causal inference literature, there are a plethora of studies that try to learn the underlying causal structure in the presence of observation and interventional data [39, 2, 4, 13, 42]. Furthermore, there are also a set of studies that try to identify the interventional target (the node where intervention happened) [20, 8, 45, 12]. Considering the failures as an intervention on the failing node gives us the ability to leverage these studies and build a customized solution for RCA, which we explain next.

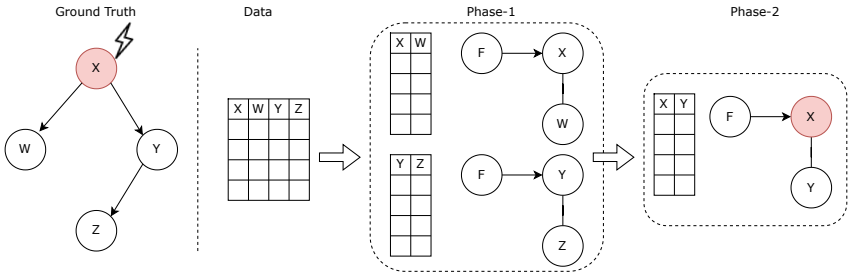

Figure 1: The left side show the ground truth causal graph where X is the interventional node. On the right side, we draw a few samples from the ground truth and try to find the interventional target. The hierarchical learning algorithm works in two phases. In the first phase, we divide the dataset into multiple subsets of size γ, here γ is 2, and run Ψ-PC to find the interventional targets in each subset. For the second phase, we combine the neighbors of F-NODE from all the fragments into one dataset and run Ψ-PC on it to get the final root cause.

3 Hierarchical Learning for RCA

We make the key observation that in a cluster of microservices, a failure can be thought of as an intervention on the failing node by changing its generative mechanism. Consider the latency of a microservice as a random variable L in a causal graph. During the normal state, it follows the probability distribution $P(L|Pa_L)$, where Pa_L is its parents in the graph. However, because of a programming bug or a configuration change, the probability distribution changes to $P' \neq P$. Such a failure can be considered a soft intervention as the failing node remains connected to its parents, but its conditional distribution changes. Accordingly, we define a variable R as a root-cause variable if $P(R|Pa_R)$ varies from normal mode of operation to the anomalous mode of operation. The key benefit of modeling the failure as an intervention is that it enables us to translate the problem of finding the root cause of failure into finding the interventional target. This view allows us to leverage the recent developments in the causal discovery literature that can identify intervention targets.

We picked Ψ-FCI [12] which proposes the state-of-the-art sound and complete algorithm to learn the causal graph as well as the interventional targets from the observational and soft-interventional data. Ψ-FCI considers the case where the latent confounders are present in the data. Accordingly it uses a modified version of the FCI algorithm [36] to learn the causal graph. However, in our work we assume no confounder variables and therefore running FCI is not necessary. Accordingly, we designed a modified version of Ψ-FCI, named Ψ-PC, which internally uses PC, to find the interventional target.

Ψ-FCI, and accordingly Ψ-PC uses an extra node called the F-NODE to represent the effect of an intervention on the system. Such F-NODE's have been used for this purpose in Pearl's work [25] as a tool to prove some of the do-calculus rules. They were also used for causal discovery in several other studies, such as [20, 42]. The utility of such a representation is that one can identify distributional invariances of the form $P_N(X|Pa_X) = P_A(X|Pa_X)$, where P_N and P_A are the distributions under normal mode of operation and anomalous mode of operation, respectively. This can be done through conditional independence tests involving the F-NODE's in a systematic manner by leveraging FCI or PC algorithms on the graph involving the F-NODE. Specifically, let us introduce the probability distribution P^* as $P^*(V|F = 0) = p_N(V)$ and $P^*(V|F = 1) = p_A(V)$, where V is the set of observed variables and F is the F-NODE. Then the invariance $P_N(X|Pa_X) = P_A(X|Pa_X)$ corresponds to the conditional independence statement $X \perp\!\!\!\perp F\,|Pa_X$, that can be tested on P^*. Samples from P^* can be obtained by simply concatenating normal and anomalous datasets with an F node that takes the value of 0 for normal and the value of 1 for the anomalous mode of opearation. Running the PC algorithm [36] on this combined data with the domain knowledge that F-NODE cannot have any incoming edges allows us to discover all and only the root cause nodes, i.e., R such that $P_N(R|Pa_R) \neq P_A(R|Pa_R)$ as the remaining neighbors of F-NODE. The soundness and completeness of Ψ-PC follows from [12]. The detailed algorithm is presented in the appendix.

Hierarchical Learning. We propose a hierarchical learning algorithm called RCD that uses Ψ-PC as a subroutine. Our key insight is that rather than finding the interventional target using all variables, we can divide the data into small subsets of variables, and find the interventional targets in every subset using Ψ-PC. With small sets, we reduce the number of CI tests and hence the run time. We use parameter γ to control the subset size. Ψ-PC outputs candidate interventional targets in each subset.

Note that partitioning the variables into subsets could lead to the descendants of the root cause and the actual root cause falling into different subsets. After running Ψ-PC on a subset, F-NODE will either point to the actual root cause if it exists in that subset or any descendant of the root cause that is not separable by conditioning on the nodes in that subset. Figure 1 depicts this process with 4 nodes.

In the next phase, we take the union of the interventional targets from all subsets as potential root causes. Specifically, these are the variables that change from normal to anomalous dataset, and whose change cannot be controlled for using the variables in that subset. However, it is possible that had we conditioned on variables in the other subsets, they would be conditionally independent from the F-NODE, and hence be removed from its neighborhood. Accordingly, we can create new subsets using only the nodes in the current neighborhood of F-NODE and repeat this process. In the last stage, we will have a single set of nodes and running Ψ-PC on these reveals the actual root causes. Figure 1 demonstrates this in two stages. Complete pseudocode of the algorithm is given in Algorithm 1.

Algorithm 1 Root Cause Discovery Algorithm (RCD)

Input: Normal dataset D, anomalous dataset D', γ, Ψ-PC (.) [12], k : Max. no. of root causes
Output: A list of root causes $\mathcal{U}$.

procedure RCD(D, D', γ)
  $\mathcal{U} \leftarrow$ Set of variables of D, D'.
  while $|\mathcal{U}| > k$ **do**
    $\mathcal{S} \leftarrow$ A random partitioning of $|\mathcal{U}|$ into subsets of size γ.
    $R \leftarrow \emptyset$
    for all $S \in \mathcal{S}$ **do**
      $G \leftarrow \Psi\text{-PC}(D[S], D'[S])$ # Ψ-PC constructs a graph on $\mathcal{U} \cup \{\text{F-NODE}\}$.
      $R \leftarrow R \cup Ne_G(\text{F-NODE})$ # Extract neighbors of F-NODE.
    $\mathcal{U} \leftarrow R$.
return $\mathcal{U}$

We provide a theorem which shows that RCD is sound and complete for detecting the root causes. The proof can be found in appendix.

Theorem 1 *Given access to a perfect conditional independence oracle, and under the causal sufficiency, and the extended faithfulness*[1] *assumptions Algorithm 1 returns the true root-cause variables.*

Localized Learning. In the previous section, we discussed using RCD to find the root cause of the failure. Note that Ψ-PC focuses on learning not only the interventional targets but also the whole causal graph. This is useful when some downstream task, such as inference, requires the full causal structure [11, 43]. The time complexity of such an algorithm depends on the number of CI tests that need to be executed, which in general is exponential in the number of nodes for non-sparse graphs. By decoupling learning the causal graph from finding the interventional targets, we can improve runtime by significantly reducing the number of CI tests.

In our case, to find the root cause of the failure, we only need to learn the interventional target which translates to only learning the immediate neighborhood of the F-NODE. This key insight allows us to propose a localized learning algorithm that focuses on only learning the neighborhood of the F-NODE. The localized learning algorithm only runs the minimal set of CI tests needed to remove non-root cause nodes from the neighborhood of F-NODE. We can accomplish this by modifying Ψ-PC such that it only runs the CI test where one of the variables under consideration is F-NODE. This localized version of the algorithm is used in our experiments, presented in the next section.

4 Evaluation

We conduct extensive evaluation of our proposed solution to answer the following questions: (1) *How effective is RCD for finding the interventional target?*, (2) *How quickly can RCD find the interventional target?*, (3) *How sample-efficient is RCD?* Our anonymized source code and results are provided[2].

[1] The distributional invariances across datasets cannot be coincidental, please see [12] for a formal definition.
[2] https://www.dropbox.com/s/wf7uopnvtrowvit/rcd.tar.xz?dl=0

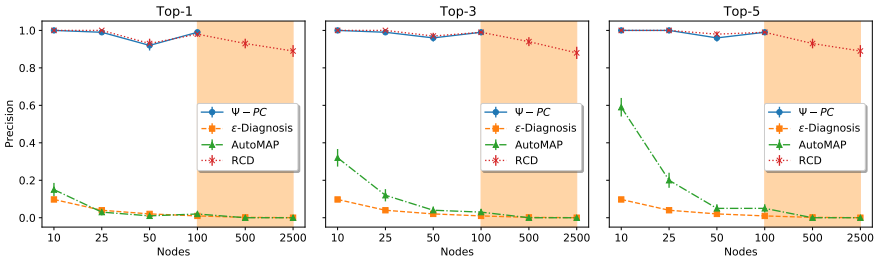

Figure 2: Top-k precision of different algorithms for finding the root cause. RCD achieves similar result as Ψ-PC since number of samples is large at $10,000$. However, it outperforms AutoMAP and ϵ-Diagnosis because they either require human intervention or rely only on lower-order tests such as cross covariance.

Implementation and Testing Setup. We implemented a modified version of RCD (Algorithm 1) for finite number of samples that generates a list of top-k potential root causes based on the p-values of CI tests. The complete algorithm with implementation details is provided in the Appendix. We implemented Ψ-PC in Python using the causal-learn package [5]. We use the Chi-squared test to check the independence between the two variables. To generate synthetic data, we use pyAgrum [29] with a randomly generated DAG to draw samples for the normal and anomalous dataset. For all experiments, we set γ to 5 for RCD. Finally, to get statistically significant results, we ran all the experiments 100 times and plotted the average.

Baseline and Evaluation Metrics. We compare our solution to the following baseline methods;

- ϵ-Diagnosis [32]: A distance-based algorithm for RCA of microservice failures. The authors proposed an algorithm to find the root cause specifically for microservices with the problem of small-window long-tail latency. They used the coefficient of variation (COV) on the normal and anomalous data to find microservices that changed significantly after the failure. The comparison with ϵ-Diagnosis will show the importance of our use of the CI test.
- AutoMAP [16]: The state-of-the-art solution to find the root cause of the failure in a cluster of microservices using causal inference. It constructs a weighted causal graph using a modified version of PC algorithm where the edge weight between the two microservices signifies the (in)dependence of a performance metric. It concludes by traversing nodes based on a random walk algorithm and finding the root cause from the weighted causal graph using the correlation metric.
- Ψ-PC: A modified version of Ψ-FCI that focuses on learning the interventional target from the observational and interventional data in the presence of latent variables. The comparison will highlight that learning the complete causal graph is not necessary for finding the root cause. Furthermore, it will also establish the time gain from RCD by not running the CI tests for the irrelevant variables.
- RCD: The proposed hierarchical solution with localized learning, and the optimization for avoiding the edge orientation phase of Ψ-PC.

Two quantitative metrics are generally used to evaluate the performance of different algorithms related to root cause detection: execution time and precision at top-k [16, 41, 19, 40]. In our case, precision at top-k is the probability of getting the correct root cause from the top k outputs of the algorithm.

Finding Interventional Target. Our first experiment compares the detectability of different RCA algorithms. We run this experiment by controlling the number of nodes in the random DAG we generated to collect the synthetic data. The max in-degree of all the DAGs is set to 3 and the total number of states for every node is 6. Furthermore, for every experiment, we draw 10,000 samples for the normal and anomalous states of the system.

Figure 2 shows the top-k precision with $k = 1/3/5$. In general, we see that RCD can identify the interventional target with very high precision. More specifically, it achieves 98% precision in top-1. In terms of other algorithms, Ψ-PC performs similarly to RCD (but cannot run beyond 100 nodes). This is because RCD internally uses a modified version of Ψ-PC to learn the interventional target, but solves the scalability problem of Ψ-PC. On the flip side, precision of AutoMAP for $k = 5$ is

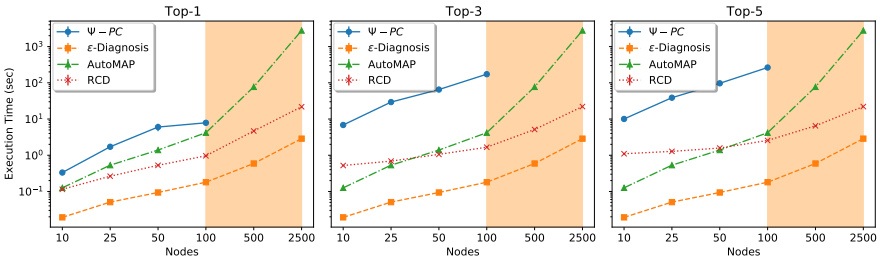

Figure 3: The execution time of different algorithms to find the top-k root cause candidates. The time required for RCD increases slower in the number of nodes than Ψ-PC, which does not terminate in a reasonable time for 500 and more nodes. AutoMAP performs better than Ψ-PC but still incur a high execution time because it internally uses PC. ϵ-Diagnosis achieves the best execution time but it usefulness is limited as its precision drops significantly as the number of node increases.

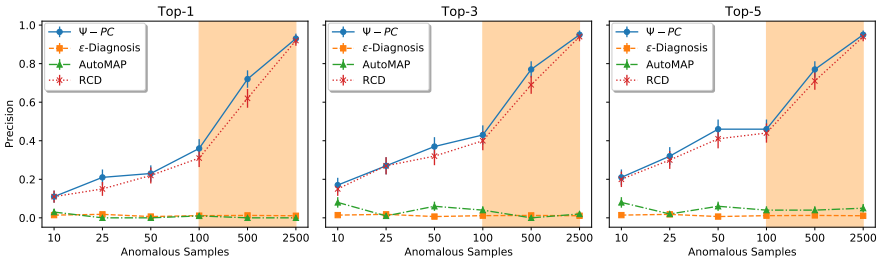

Figure 4: Sample Efficiency of different root cause localization algorithms. As the number of samples grow, the top-k precision of RCD and Ψ-PC can starts to increase. That is to say, RCD can provide the same benefits as Ψ-PC while reducing the execution time significantly.

just above 50% when there are only 10 nodes and it drops to 0 when the number of nodes increases. One of the key reason for this result is that the accuracy of AutoMAP depends on the historical data about the failures. Hence, for the initial rounds of the execution (i.e., without expert human input), AutoMAP performs worse as it only relies on the correlation between the variables. Finally, we can observe that ϵ-Diagnosis performs much worse than RCD because of its use of COV that does not take into account other variables when deciding the independence between two variables. ϵ-Diagnosis performs worse with larger number of nodes as there is more room to miss the root cause and classify a wrong microservice to be the root cause simply because that metric shows a bigger change.

Execution Time. Most microservice-based cloud applications are latency sensitive thus any tool that claims to find the root cause of the failure has to minimize the execution time. To measure the execution time of different algorithms, we conducted the experiments on synthetic data with a varying number of variables and drew 10K samples for both datasets (normal and anomalous). We measured the time it takes to find the top-k potential root causes where $k = 1/3/5$. Figure 3 illustrates the results (note the logarithmic scale on the Y-axis).

The first thing to note here is that RCD outperforms all the baselines. On average, for 1,000 nodes, it can detect the actual root cause of the failure in the top 5 candidates in just above 34 seconds. Secondly, the execution time of Ψ-PC grows exponentially as the number of nodes increases. For just 500 nodes, one execution of Ψ-PC with $k = 5$ was taking more than 150 minutes. Here we can observe the benefits of using a hierarchical and localized approach to learn the root cause. Because of the time constraint, we only ran Ψ-PC for a maximum of 100 nodes.

Considering other baselines, we see that AutoMAP still incurs a high delay because of its dependence on PC algorithm which tries to learn the whole causal graph. Similarly, the execution time of ϵ-Diagnosis almost stays constant. The reason for this is that it only considers variables in a pairwise fashion, thus, the number of tests for COV grows only quadratically with the number of nodes. The consequence of only considering the variables in a pairwise fashion is that it leads to a higher number of false positives which significantly affects the top-k precision.

Sample Efficiency. Timely fix of failures in a large scale cloud-based application is of utmost importance. In other words, the root cause of the failure needs to be detected as early as possible. Consequently, one would expect a large amount of data for the normal state of the system however, the number of samples for the anomalous dataset will be quite limited. To check the applicability of any algorithm for RCA, we need to see how sample efficient that approach is. To that end, we performed an experiment by varying the number of samples and checking the top-k precision of different algorithms. For this experiment, we kept the number of nodes and number of samples for normal dataset constant (100, and 10,000 respectively) and varied the number of anomalous samples. Figure 4 illustrates the results.

The first thing to note here is that, in most cases, RCD is as effective in detecting the root cause as Ψ-PC even with limited number of samples. More specifically, with only 500 anomalous samples, RCD can find the root cause in top 5 with the precision of 71% (as compared to 77% for Ψ-PC). Whereas AutoMAP and ϵ-Diagnosis can only detect about 4% and 1.1% of the root causes respectively.

They key takeaway from our experiments is that RCD provides almost identical top-k precision as compared to Ψ-PC while reducing the execution time significantly. RCD gains this benefit by 1. splitting the data into small fragments and then running the causal discovery algorithm 2. by only learning the neighborhood of F-NODE rather than learning the complete underlying causal graph.

5 Case Study

Sock-shop. To test the applicability and quantify the performance gain of different root cause discovery algorithms, we setup a test-bed using a real microservice-based application. We use Sock-shop [33], a replica of an online application that sells socks. It consists of 13 microservices written in different technologies, where each microservice is deployed on its separate VM/container. The communication between the microservices happens through API requests over HTTP. Moreover, all these microservices provide a large amount of statistical information in terms of different metrics (for instance, CPU and memory utilization, latency, and the number of errors).

We developed an independent workload generator using Locust [15] to send the traffic to the Sock-shop application. To mimic a real application, we followed sinusoidal distribution with the mean of 50 to draw the number of users every second. Every user first signs up on the system, browses a few items, and finally orders one. This whole process sends multiple requests to all the microservices.

There are two very common generic failures in cloud computing applications: CPU hog and memory leak [6, 17, 16]. We injected these two types of failures in microservices by running `stress-ng` [37]. We accomplish this by modifying the Docker images of all the microservices to install `stress-ng` and then running it on the container of the targeted microservice. With this setup, we injected failures in all of the major microservices of Sock-shop (carts, catalogue, orders, payment, and user). Out of the 13 microservices, we selected these because they were the critical user-facing services. Any change to their performance cause the other microservices to get affected as well, thus making it difficult to detect the root cause with a simple threshold-based scheme.

We ran the Sock-shop application for 5 minutes to collect the data for the normal state of the system. After that, we injected the failure and ran the application for 5 more minutes to collect the anomalous dataset. We repeated the experiment 5 times for two different types of failures (CPU hog and memory leak) and in total we gathered 50 datasets. To measure the performance of root cause detection algorithm, we ran an algorithm 50 times on every dataset and quantified their top-k precision. In this context, the top-k precision can be interpreted as the probability that the algorithm can find the actual root cause in a list of top-k potential root causes.

Table 1 shows the top-k precision of different algorithms for detecting the root cause on the data collected from Sock-shop. RCD performs better than the baselines in general. More specifically, it achieves an average precision of 63% and 64% for CPU hog and memory leak failures. Considering baselines Ψ-PC performs poorly for most of the microservices, the average top-k precision for CPU hog and memory leak is 0.16 and 0.2 respectively. The main reason for this is the limited number of samples for normal dataset, which makes Ψ-PC unable to learn the required causal structure. In one of the cases (Memory leak failure in Orders), ϵ-Diagnosis performs better than RCD.

Real Data. We validate our approach on a real dataset collected from a production ready microservice that is hosted on AWS cloud-native system (name hidden for the double-blind review process). The

Table 1: Top-k precision of different algorithms for detecting the root cause on data from the Sock-shop application. Ψ-PC suffers mainly because of the limited number of samples for normal state of the system whereas RCD can achieve reasonable precision due to localized search. AutoMAP performs worse because of its reliance on the historical data.

		CPU hog				Memory leak			
		Ψ-PC	AMAP	ϵ-Diag.	RCD	Ψ-PC	AMAP	ϵ-Diag.	RCD
Carts	top-1	0.6	0.13	0.6	**0.74**	0.0	0.18	0.2	**0.6**
	top-3	0.6	0.30	0.6	**0.71**	0.0	0.26	0.2	**1.0**
	top-5	0.6	0.32	0.6	**0.71**	0.0	0.32	0.2	**0.94**
Catalogue	top-1	0.0	0.2	0.4	**0.6**	0.0	0.01	0.2	**0.2**
	top-3	0.0	0.26	0.4	**0.69**	0.0	0.06	0.2	**0.6**
	top-5	0.0	0.31	0.4	**0.66**	0.0	0.1	0.2	**0.8**
Orders	top-1	0.0	0.01	0.2	**0.60**	0.0	0.32	**0.4**	0.34
	top-3	0.0	0.33	0.2	**0.80**	0.0	0.49	**0.8**	0.57
	top-5	0.0	0.41	0.2	**0.80**	0.0	0.63	**0.8**	0.74
Payment	top-1	0.0	0.1	0.2	**0.31**	0.0	0.01	**0.2**	0.01
	top-3	0.0	0.1	0.2	**0.4**	0.0	0.12	0.2	**0.46**
	top-5	0.0	0.1	0.2	**0.46**	0.0	0.26	0.2	**0.69**
User	top-1	0.2	0.31	0.2	**0.8**	0.2	0.18	0.2	**0.6**
	top-3	0.2	0.45	0.2	**0.49**	0.2	0.28	0.4	**0.89**
	top-5	0.2	0.49	0.2	**0.63**	0.2	0.34	0.4	**0.67**
Average		0.16	0.26	0.32	**0.63**	0.04	0.28	0.32	**0.64**

data was collected from Grafana, a monitoring tool that tracks and logs the metric values of several components of a running system at certain time intervals. We collected data spanning 24 hours for a day when the outage had occurred which we will be using as the anomalous data, while the normal dataset was collected for the day prior to when the outage had occurred.

According to the incident reports generated and curated by the SRE team, the AWS service instances located at specific regions were unavailable for a brief amount of time, which impacted several customers and the outage was labeled to be of the type of Severity 1 red status. The system was in a failed state after a wrong deployment had crashed three databases instances. With all the deployments paused for a brief period, the SRE team was able to restore the database nodes.

With close supervision from the SRE team, we were able to gather all the metrics that were relevant to the outage. In total, we collected 14 database-related metrics from the AWS service, which broadly covers all the monitoring interfaces for the database. The type of metrics ranged from database workload to throughput and network latency as well as bytes read or written during an I/O operation. Using the data collected by the above process, we ran Ψ-PC and RCD to detect the root cause for the outage. AutoMAP detect a root cause only at the service level and the real data involved multiple metrics but for only the single database service. Therefore, running AutoMAP would have yield a vacuous result, and we omitted it. Ψ-PC as well as RCD pointed to the Buffer Pool[3] of the database engine as the top-1 root cause, while throughput, amount of bytes read and written by the database engine and workload were among the top-5 root causes. After an interview with the SRE team for ground-truth, they concurred with our determination of the root cause for the outage.

The average time taken by Ψ-PC for detecting the root cause was **114.75 seconds**, averaged over 10 executions, while RCD took an average of **0.25 seconds**. This corroborates our claim stated in Section 1 that RCD detects root cause much faster than even the modified version of PC (Ψ-PC).

6 Conclusion

We proposed a novel root cause analysis algorithm that systematically leverages the distributional invariances across normal and anomalous datasets, taking inspiration from the recent causal discovery algorithms. Our tailored algorithm sidesteps learning the full causal graph and rather focuses on only the root causes. This provides not only significant benefits in runtime, but also in the number of required anomalous samples. Future work includes a localized hierarchical root cause discovery algorithm that incorporates the existence of latent variables. Furthermore, methods specialized for time-series data could be incorporated in RCD, which is a limitation of the current work.

[3] A buffer pool is an area of main memory that has been allocated by the database manager for the purpose of caching table and index data as it is read from disk.

References

[1] Andrew Arnold, Yan Liu, and Naoki Abe. Temporal causal modeling with graphical granger methods. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 66–75, 2007.

[2] Elias Bareinboim and Judea Pearl. Causal inference and the data-fusion problem. *Proceedings of the National Academy of Sciences*, 113(27):7345–7352, 2016.

[3] Kenneth A Bollen. *Structural equations with latent variables*, volume 210. John Wiley & Sons, 1989.

[4] Philippe Brouillard, Sébastien Lachapelle, Alexandre Lacoste, Simon Lacoste-Julien, and Alexandre Drouin. Differentiable causal discovery from interventional data. *Advances in Neural Information Processing Systems*, 33:21865–21877, 2020.

[5] causal-learn - causal discovery for python, 2022. [Accessed: 2022-05-19].

[6] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1887–1895. IEEE, 2014.

[7] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

[8] Daniel Eaton and Kevin Murphy. Exact bayesian structure learning from uncertain interventions. In *Artificial intelligence and statistics*, pages 107–114. PMLR, 2007.

[9] Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019.

[10] Grafana - the open observability platform, 2022. [Accessed: 2022-05-19].

[11] Paul Hünermund and Elias Bareinboim. Causal inference and data fusion in econometrics. *arXiv preprint arXiv:1912.09104*, 2019.

[12] Amin Jaber, Murat Kocaoglu, Karthikeyan Shanmugam, and Elias Bareinboim. Causal discovery from soft interventions with unknown targets: Characterization and learning. *Advances in neural information processing systems*, 33:9551–9561, 2020.

[13] Murat Kocaoglu, Amin Jaber, Karthikeyan Shanmugam, and Elias Bareinboim. Characterization and learning of causal graphs with latent variables from soft interventions. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] JinJin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *International Conference on Service-Oriented Computing*, pages 3–20. Springer, 2018.

[15] Locust - an open source load testing tool, 2022. [Accessed: 2022-05-19].

[16] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. Automap: Diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020*, pages 246–258, 2020.

[17] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. Localizing faults in cloud systems. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 262–273. IEEE, 2018.

[18] Chris Meek. *Complete orientation rules for patterns*. Carnegie Mellon [Department of Philosophy], 1995.

[19] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

[20] Joris M Mooij, Sara Magliacane, and Tom Claassen. Joint causal inference from multiple contexts. 2020.

[21] Raha Moraffah, Mansooreh Karami, Ruocheng Guo, Adrienne Raglin, and Huan Liu. Causal interpretability for machine learning-problems, methods and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(1):18–33, 2020.

[22] Sam Newman. *Building microservices*. " O'Reilly Media, Inc.", 2021.

[23] The opentracing project, 2022. [Accessed: 2022-05-19].

[24] Rainer Opgen-Rhein and Korbinian Strimmer. From correlation to causation networks: a simple approximate learning algorithm and its application to high-dimensional plant gene expression data. *BMC systems biology*, 1(1):1–10, 2007.

[25] Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

[26] Judea Pearl. *Causality*. Cambridge university press, 2009.

[27] Judea Pearl. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, 62(3):54–60, 2019.

[28] Mohammad Farhad Peerally, Susan Carr, Justin Waring, and Mary Dixon-Woods. The problem with root cause analysis. *BMJ quality & safety*, 26(5):417–422, 2017.

[29] pyagrum - introduction to pyagrum, 2022. [Accessed: 2022-05-19].

[30] Juan Qiu, Qingfeng Du, Kanglin Yin, Shuang-Li Zhang, and Chongshu Qian. A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications. *Applied Sciences*, 10(6):2166, 2020.

[31] Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3(2):121–129, 2017.

[32] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*, pages 3215–3222, 2019.

[33] sock-shop - a microservice demo application, 2022. [Accessed: 2022-05-19].

[34] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Comput. Surv.*, 55(3), feb 2022.

[35] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232, 2018.

[36] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

[37] stress-ng, 2022. [Accessed: 2022-05-19].

[38] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. Sieve: Actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 14–27, 2017.

[39] Sofia Triantafillou and Ioannis Tsamardinos. Constraint-based causal discovery from multiple interventions over overlapping variable sets. *The Journal of Machine Learning Research*, 16(1):2147–2205, 2015.

[40] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 492–502. IEEE, 2018.

[41] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. Microdiag: Fine-grained performance diagnosis for microservice systems. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, pages 31–36. IEEE, 2021.

[42] Karren Yang, Abigail Katcoff, and Caroline Uhler. Characterizing and learning equivalence classes of causal dags under interventions. In *International Conference on Machine Learning*, pages 5541–5550. PMLR, 2018.

[43] Changwon Yoo, Vesteinn Thorsson, and Gregory F Cooper. Discovery of causal relationships in a gene-regulation pathway from a mixture of experimental and observational dna microarray data. In *Biocomputing 2002*, pages 498–509. World Scientific, 2001.

[44] Keli Zhang, Marcus Kalander, Min Zhou, Xi Zhang, and Junjian Ye. An influence-based approach for root cause alarm discovery in telecom networks. In *International Conference on Service-Oriented Computing*, pages 124–136. Springer, 2020.

[45] Kun Zhang, Biwei Huang, Jiji Zhang, Clark Glymour, and Bernhard Schölkopf. Causal discovery from nonstationary/heterogeneous data: Skeleton estimation and orientation determination. In *IJCAI: Proceedings of the Conference*, volume 2017, page 1347. NIH Public Access, 2017.

[46] Openzipkin - a distributed tracing system, 2022. [Accessed: 2022-05-19].

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Conclusion contains limitations of our work.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Footnote of page 5 contains a link to our code and data.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Footnote of page 5 contains a link to our code and data.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]