Proceedings of Machine Learning Research vol 236:957–978, 2024 3rd Conference on Causal Learning and Reasoning

The PetShop Dataset — Finding Causes of Performance Issues across Microservices

Mila Hardt MILAHA@AMAZON.COM
Amazon

William R. Orchard* WILLIAM.ORCHARD@CRUK.CAM.AC.UK
University of Cambridge

Patrick Blöbaum BLOEBP@AMAZON.COM
Amazon

Elke Kirschbaum ELKEKI@AMAZON.COM
Amazon

Shiva Prasad Kasiviswanathan KASIVISW@GMAIL.COM
Amazon

Editors: Francesco Locatello and Vanessa Didelez

Abstract

Identifying root causes for unexpected or undesirable behavior in complex systems is a prevalent challenge. This issue becomes especially crucial in modern cloud applications that employ numerous microservices. Although the machine learning and systems research communities have proposed various techniques to tackle this problem, there is currently a lack of standardized datasets for quantitative benchmarking. Consequently, research groups are compelled to create their own datasets for experimentation. This paper introduces a dataset specifically designed for evaluating root cause analyses in microservice-based applications. The dataset encompasses latency, requests, and availability metrics emitted in 5-minute intervals from a distributed application. In addition to normal operation metrics, the dataset includes 68 injected performance issues, which increase latency and reduce availability throughout the system. We showcase how this dataset can be used to evaluate the accuracy of a variety of methods spanning different causal and non-causal characterisations of the root cause analysis problem. We hope the new dataset, available at https://github.com/amazon-science/petshop-root-cause-analysis, enables further development of techniques in this important area.

1. Introduction

Identifying the origin of unexpected or undesired behavior of a system, also called root cause analysis (RCA), is of extreme relevance in various disciplines. For example, in manufacturing pipelines, if certain parts exceed failure rate thresholds during quality tests, it is crucial to identify and resolve the root cause to avoid financial losses and reputational damage. The timely resolution of such issues is vital as prolonged downtime can result in substantial monetary losses and erode customer trust. Likewise, for online retailers, any decrease in website availability or prolonged page load times directly impacts revenue and customer confidence, underscoring the urgency of RCA in restoring normal operations. Identifying the root cause of such issues, however, can be extremely cumbersome and time-consuming, particularly in complex applications composed of tens or hundreds of microservices. Such microservice architectures have become extremely popular in recent years, since decomposing an application into different microservices that communicate through agreed

* Work done while interning at Amazon Research in Tübingen, Germany

upon application programming interfaces (APIs) enables clear ownership and rapid development. Additionally, the most suitable hardware can be chosen for each component and scaled up or down independently. These are great advantages compared to a monolithic application architecture. However, when problems arise, identifying the cause in these complex systems is challenging and requires not just knowledge of the individual components, but also about the interaction between the components. Oncall engineers may need to look over hundreds of metrics, dig in terabytes of logs, ping people from other teams responsible for various components, before they obtain a clear picture of what went wrong.

To reduce the mean time to resolution, numerous methods have been proposed for automating RCA using available system data. Some of these methods are specifically tailored for RCA in microservice-based applications (e.g. Chen et al., 2014; Lin et al., 2018; Ma et al., 2020; Gan et al., 2021; Liu et al., 2021; Ikram et al., 2022; Li et al., 2022), while others are designed for a broader class of use cases (e.g. Budhathoki et al., 2022). Among these Ikram et al. (2022) has released a dataset based on the Sock-shop application to evaluate such methods. In this paper we introduce a similar dataset also based on microservice architecture with issues injected. However, the datasets differ in a few crucial aspects: our dataset is collected from a microservice based application which contains more than 3x as many components, we inject more issues, cover more types of issues beyond the two considered in the Sock-shop dataset, and include multiple different traffic generation patterns. This allows us to consider a greater diversity of scenarios to study which techniques work well in, which we show in our experimental section.

The dataset encompasses latency, requests, and availability metrics, gathered from a distributed application comprising 41 components, including databases, load balancers, queues, storage systems, and containerized microservices. In addition to normal operation metrics, the dataset includes 68 injected performance issues, such as request overload, memory leaks, CPU hog, and misconfigurations, which increase latency and reduce availability throughout the system. The metrics are annotated with the corresponding issues, serving as ground truth for the analysis. Interestingly, methods that performed well on the Sock-shop data struggle on our dataset. This illustrates the value of having this additional dataset to broaden our understanding of RCA methods, and motivates future research in robust data-efficient RCA methods. This dataset addresses the concern of biased reporting by avoiding selective focus on issues where the developed method demonstrates strong performance.

Our goal is therefore to accelerate research on RCA methods by allowing researchers to focus on the development of new methods instead of generating data for their evaluation. The dataset is publicly available, alongside code for running the experiments described in Section 5 below, at `https://github.com/amazon-science/petshop-root-cause-analysis`. The dataset is designed with an easily extendable data format and accompanying tooling, encouraging broader participation and contribution from others.

The remainder of the paper is organized as follows: next, we present an overview of the causal formalization of root-cause-analysis, laying out the different approaches to answering the question, 'what is a root cause?' In Sec. 3 we describe some of the methods which have been developed for RCA in the context of microservice-based applications and the datasets available to evaluating them. We describe our dataset in Sec. 4 and compare existing methods on it in Sec. 5. After disclosing limitations in Sec. 6 we conclude in Sec. 7.

2. Background

In this section we give a brief survey of different approaches to defining a 'root cause'. Broadly, methods can be categorized according to what layer of the Causal Hierarchy (Pearl and Mackenzie,

2018; Bareinboim et al., 2022) they pose the RCA problem. The causal hierachy delineates a strict hierarchy wherein knowledge at each layer enables reasoning about different classes of causal concepts. Layer 1, $\mathcal{L}_1$, is associational, layer 2, $\mathcal{L}_2$, is interventional, and $\mathcal{L}_3$ is counterfactual. The problem setup is common: the value x_n of target variable X_n has been flagged as anomalous. With jointly observed values $(x_1, ..., x_n)$ of variables $(X_1, ..., X_n)$, we must find the root cause(s), among these variables, of the anomaly x_n.

At $\mathcal{L}_1$, associational approaches (e.g. Shan et al., 2019) do not aim to provide causal explanations for anomalies, but instead to simply prioritise a small number of variables as potential causes of the anomaly so that they can each be manually investigated by an oncall engineer. The principle is simple: the root cause of an anomaly in a target should have experienced anomalous behaviour at a similar time. In some isolated settings this may be sufficient, the oncall engineer introduces causal information through their domain knowledge and so long as a sufficiently small number of metrics are prioritised for inspection the task is manageable in a timely manner. However, in general this method is unlikely to be suitable. Associational approaches cannot distinguish metrics which are anomalous as a result of the anomaly at the target versus being the root cause of it, nor can they distinguish metrics whose association is due to having a common cause. As such, associational approaches are susceptible to false positives which simply applying a stricter cut-off will not solve.

It is clear that when the goal is to extract *actionable* insight from the data (i.e. to learn which variable needs to be fixed to address the occurrence of the anomaly) a causal approach is required (i.e. at $\mathcal{L}_2$ or $\mathcal{L}_3$). Although there is not consensus on the way RCA should be characterised as a causal problem, a number of works have proposed formalizations and provided algorithms motivated by them (e.g. Budhathoki et al., 2021, 2022; Ikram et al., 2022; Li et al., 2022). We assume that the data generating process can be modelled by a Structural Causal Model (SCM) whose underlying causal graph is a directed acyclic graph (DAG). In particular, the SCM describes how each variable X_i is generated from its parents PA_i in the causal graph and an unobserved noise term N_i, $X_i := f_i(PA_i, N_i)$, where $N_1, ..., N_n$ are jointly independent (Pearl, 2009).

Common to all causal approaches to RCA is to treat the occurrence of an anomaly as being the result of a change in the causal mechanism generating the root cause variable. Each method then differs in how to define a root cause and what causal information we have available to us at time of analysis. At $\mathcal{L}_2$ we assume we know the causal graph (or can learn it from data) and treat the mechanism change as a distribution change. In particular, following the causal Markov assumption (Pearl, 2009), the joint distribution $P_{\mathbf{X}}$ over $(X_1, ..., X_n)$ factorizes into causal mechanisms

$$P_{\mathbf{X}} = \prod_{i}^{n} P_{X_i|PA_i}.$$

The change in the joint distribution following the anomaly is then the result of mechanism changes in a subset of variables X_T, indexed by a change set T, where

$$P_{\mathbf{X}}^{T} = \prod_{j \in T} \tilde{P}_{X_j|PA_j} \prod_{j \notin T} P_{X_j|PA_j}$$

is the joint distribution resulting from the change of causal mechanisms at each X_j in T from $P_{X_j|PA_j}$ to $\tilde{P}_{X_j|PA_j}$. A variable X_i is then considered a root cause if the change in the joint distribution can be attributed to a change in the causal conditional for X_i. As a causal contribution problem, the task is to quantify the *extent* to which the change in the joint distribution can be attributed to a change in the causal conditional for a potential root cause (Budhathoki et al., 2021), with root causes being

those whose contributions are, in some sense, large. Others reduce the task further and simply call a variable X_i a root cause *if* its causal conditional has changed (Ikram et al., 2022).

At $\mathcal{L}_3$, the finest-grain approach, we assume we know the full SCM (or make suitable assumptions, e.g. additive noise (e.g. Shimizu et al., 2006; Hoyer et al., 2008), such that we can learn it) and the question of whether a variable X_i is a root cause is posed counterfactually. For example, Gan et al. (2021) ask "would the anomaly at X_n have occurred had X_i been at a previously known to be 'normal' value?" Budhathoki et al. (2022) refine this further as a causal contribution problem, quantifying the extent to which the counterfactual (tail) probability of the anomalous event x_n increases owing to the factual mechanism of X_i compared to if it had been as 'normal'.

RCA at both layers $\mathcal{L}_2$ and $\mathcal{L}_3$ is substantially different from determining the behaviour of a target variable X_n with respect to an intervention at X_i. By treating anomalies as the result of a mechanism change, causal RCA asks which variable(s) take on an anomalous value which cannot be explained by their parents. This rules out identifying a variable which simply 'transmits' the value of its parent as a root cause.

3. Related Work

In this section, we examine techniques for RCA, split into those specifically targeted at microservice-based applications and those focusing on other use cases, and discuss the content and availability of the datasets on which the methods were evaluated.

RCA in microservice-based applications *Pinpoint* (Brewer et al., 2002; Kiciman and Fox, 2005) is an early example of a method for anomaly detection and RCA in microservice-based applications and is based on detecting changes in the interactions between application components via requests. Pinpoint is also the first method to be evaluated by injecting faults into the Petstore application, upon which our own dataset is based (see section 4), however no dataset was made publicly available. Another early method for RCA in microservice architectures is *CauseInfer* (Chen et al., 2014) which is based on a traversal of a causal graph between components in a distributed system. CauseInfer was evaluated on a controlled distributed system of 5 machines injecting issues including a CPU hog, memory leak, disk hog, overload, configuration change, and bugs. It has been extended in Lin et al. (2018) which was evaluated on a Sock-shop application (Microservices-Demo, 2022) where CPU hogs, traffic, and pauses were injected. A further extension was presented by Liu et al. (2021). This system ranks the anomalous paths by the correlation of the anomaly index between the potential root cause and a target service. It was evaluated on real issues in the e-commerce system of Alibaba. A similar approach was used in Ma et al. (2020) who present *AutoMAP* which performs a random walk on the causal graph with correlations between potential root causes and the target service serving as edge weights. It was evaluated on simulations and on a real-world enterprise cloud system. Another approach examined using actual operational data from an e-commerce platform (eBay) is the *GRANO* system (Wang et al., 2019). None of these datasets are publicly available.

A counterfactual approach RCA was presented by Gan et al. (2021) who proposed *Sage*. Counterfactuals are evaluated by making use of conditional variational autoencoders (Sohn et al., 2015). Sage is evaluated on a few different microservice architectures developed in DeathStarBench (Gan et al., 2019) reflecting a social network, a hotel reservation system, and a media system. However, again these datasets are not publicly available.

More recently, Li et al. (2022) proposed *CIRCA* which models an issue as an intervention on the root cause node. The method was evaluated in experiments on synthetic data and a banking application. Ikram et al. (2022) extended this line of work and proposed a method for identifying root causes in microservice architectures without relying on a service map. The method was tested

by injecting issues into a Sock-shop application (Microservices-Demo, 2022) with data released Ikram (2023). Our dataset complements this dataset and presents new challenges to identify root causes. In our experiments (Sec. 5) we see that methods that perform well on the Sock-shop data do not necessarily perform well on our dataset.

RCA for other applications Attariyan et al. (2012) identify causes of performance issues in servers such as a configuration setting. They experiment on Apache, lighttpd, Postfix, and PostgreSQL. Budhathoki et al. (2022) extend the counterfactual question of Gan et al. (2021) and account for interactions using Shapely values (Shapley, 1953). They use data on river flows for evaluation. Broadly related is also explainable AI (see e.g. Ribeiro et al., 2016; Sundararajan et al., 2017; Lundberg and Lee, 2017) that tries to explain why a machine learning model returned a certain output and which input features were most relevant for this outcome. While most methods aim to explain *any* output of the model, Idé et al. (2021) focus on explaining anomalous outcomes. They evaluate their approach on a building-energy prediction task. These datasets are very different from data collected in applications based on microservice-architectures.

4. Dataset Description

In this benchmark, we build on an application that has been publicly released in AWS-Samples (2023) and features a pet site for adoptions of different kinds of pets. This application composed of microservices is running on Amazon Web Services (AWS) (AWS.Amazon.com, 2023a). These microservices include storage (distributed file systems, databases), publish-subscribe systems, load balancers, and custom application logic built in a container and deployed using Kubernetes (Kubernetes.io, 2023). The application offers search over pets based on different attributes and permits transactions to adopt and pay for a pet. See the Appendix A for a screenshot of the landing webpage. Furthermore, the release of the application comes with a traffic generator that we use to simulate user interactions with the website. Since our application is build on AWS infrastructure and uses AWS services, in the following, we will refer to them to explain our construction. Note, however, that no prior knowledge or active usage of AWS infrastructure is required to use this dataset. To learn more about the application and its microservices, see AWS-Samples (2023). We injected artificial performance issues into various microservices.

4.1. Data Structure

The dataset contains multiple scenarios, each with a set of issues to diagnose. The `graph.csv` describes the *service map* across microservice nodes obtained from tracing. The service map refers to a graphical representation of the flow of function calls within the system, and it visualizes the interactions between different microservices. The metrics at these nodes are recorded in `metrics.csv` for both a normal period (during which no issues were injected) as well as a range of issues in folders named `issues0`, `issues1`, etc. The issues are split evenly at random into `train` and `test` sets according to the microservice they originated from. This separation allows practitioners to conduct any optimizations for their RCA on `train` and report performance separately on `test`. Each split-folder contains directories with issues. Each issue contains a metrics file and a target file with the ground truth root cause described next. The full directory structure is listed in the Appendix B.

4.2. Service Map

We obtain the service map of the application from AWS X-Ray (AWS.Amazon.com, 2023c) by tracing user requests through the application. An edge from A to B indicates that A calls B.

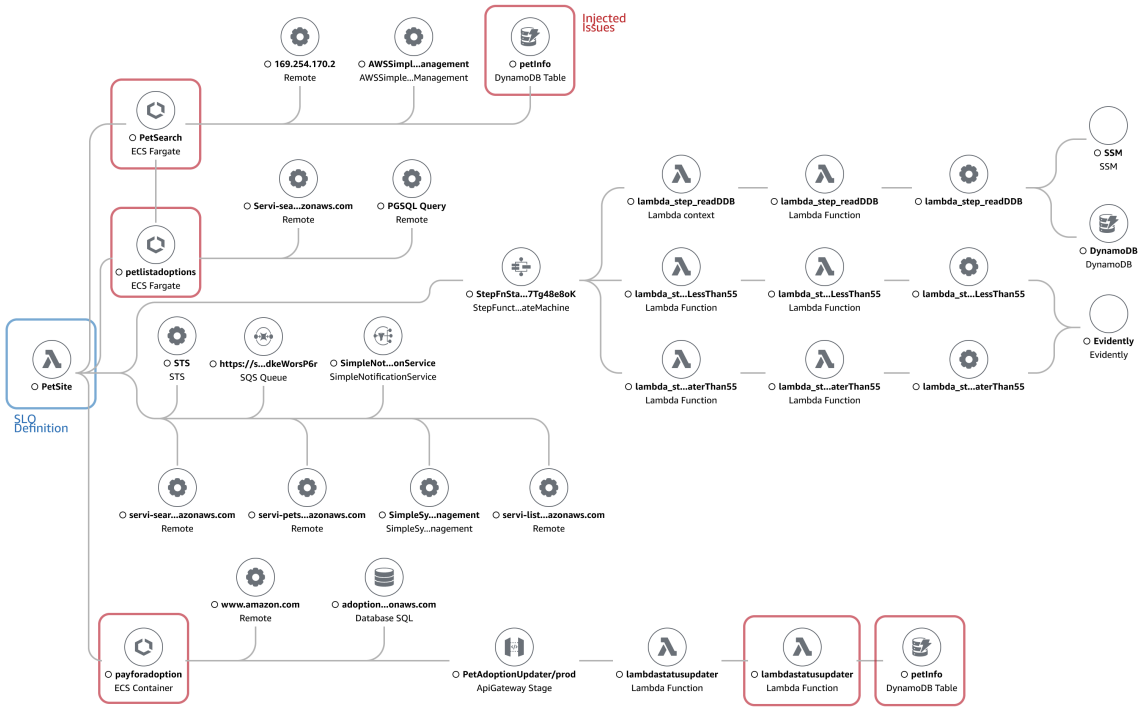

Figure 1: Overview of pet adoption site built on microservices. We injected issues at the highlighted nodes that cause SLO violations of high latencies and low availability at the PetSide node.

Notably, this does not describe the causal graph that connects the metrics originating from those microservices. Nevertheless, it can be useful in constructing a causal graph as noted in Li et al. (2022); Gan et al. (2021); Eulig et al. (2023). The service map can be loaded from `graph.csv`, which contains the adjacency matrix indicating the dependencies of the microservices of the application, as follows: `nx.from_pandas_adjacency(pd.read_csv('graph.csv', index_col=0), create_using=nx.DiGraph)`.

Figure 1 shows the service map of the pet adoption application with directed edges going from left to right. We also highlight nodes in which we injected issues.

4.3. Service-level Objectives (SLOs)

We consider the customer-facing `PetSite` (see Figure 1) as the target node for which we defined service level objectives (SLOs) on what latencies and availabilities are desired to ensure a good end-user experience. For example, a response time SLO at `PetSite` might be: "Ensure 95% of all API requests respond within 200 milliseconds." This SLO sets a performance target for the response time of an API. It specifies that at least 95% of all requests should have a response time of 200 milliseconds or less. This means that the system is expected to meet this performance target for the majority of requests, providing a reliable and responsive user experience. We modified the original code of the application to inject performance issues that lead to violations of such SLOs.

The file `target.json` contains information about the SLO violation at a target node including the name of the node and the metric as well as a time stamp when the issue was injected. Notably, the SLO violation may happen with a certain delay from this injection times. This delay is due to the time for the change to take effect. Some issues are injected through changes in the Simple System Management service through feature flags, that may be cached locally and polled frequently.

Table 1: We list the scenarios that vary in their request traffic. Here, we report quantiles of the number of requests per 5-minutes.

Traffic Scenario	mean	quantile 0.1	quantile 0.5	quantile 0.9
low-traffic	484	464	483	503
high-traffic	690	668	688	714
temporal-traffic	571	376	497	884

Furthermore, the file also lists the actual node from which the issue originated and optionally detailing the metric responsible at that node. It also contains some additional metadata containing details for reproducibility that can be ignored when determining the root cause.

Example 1 *The following describes an example of a SLO violation at the* `PetSite` *microservice with an increase in average latency. The time of the issue injection is recorded as unix timestamp* `1681390208`*, corresponding to 04/13/2023 at 12:50:08pm. The correct root cause for this issue is the* `PetSearch` *node. No additional information about what happened to cause the issue at that node is given. For this case, the file content is the following:*

```
{
   "target": {
      "node": "PetSite",
      "metric": "latency",
      "agg": "Average",
      "timestamp": 1681390208
   },
   "root_cause": {
      "node": "PetSearch",
      "metric": null
   },
   "metadata": {
      "reproducibility": {
         "command": "aws ssm put-parameter --name
            '/petstore/searchdelay' --value '500' --overwrite"
      }
   }
}
```

4.4. Traffic Scenarios

We generated different traffic patterns using a traffic generator: We have a high-traffic, a low-traffic, and a temporal traffic scenario, in which the traffic varies over the course of a day.

Under the low-traffic scenario we have on average 485 requests per 5-min, for the high-traffic scenario we have 690 and for the temporal traffic scenario the number of requests vary and average at 571, see Table 1 for quantiles (see also Tables 5 and 6 in Appendix B for summary statistics of the latency and availability across traffic scenarios, before and after fault injection).

4.5. Metrics

In our pet adoption application we collect the number of requests, their latency (average and different quantiles) and average availability at each microservice always over 5 minute windows.

microservice	PetSite							petInfo_AWS::DynamoDB::Table						
metric	latency					requests	availability	latency					requests	availability
statistic	Average	p90	p50	p95	p99	Sum	Average	Average	p90	p50	p95	p99	Sum	Average
time														
2023-04-13 12:45:00	0.083632	0.227089	0.044066	0.270083	0.357580	695.0	99.712230	0.013723	0.025674	0.003875	0.047272	0.337898	1284.0	100.0
2023-04-13 12:50:00	0.101622	0.247955	0.045317	0.333652	0.552535	671.0	100.000000	0.013397	0.028259	0.003841	0.045765	0.323774	1216.0	100.0
2023-04-13 12:55:00	0.109615	0.266084	0.046886	0.467992	0.555613	698.0	99.426934	0.016027	0.028762	0.003795	0.048882	0.345327	1287.0	100.0
2023-04-13 13:00:00	0.089611	0.236024	0.045869	0.281809	0.450128	675.0	99.555556	0.014104	0.025089	0.003845	0.046468	0.341427	1220.0	100.0
2023-04-13 13:05:00	0.087218	0.234109	0.044786	0.275017	0.345848	656.0	100.000000	0.014511	0.027894	0.003780	0.049658	0.351707	1171.0	100.0

Figure 2: Exemplary printout of the metrics for two microservices with 5 measurements each. The multi-column format contains the node, the metric, and different statistics for the aggregation of the metric over 5 minute time windows.

Here, availability is defined as $1 - \frac{\#errors}{\#requests}$. We obtained these metrics from Amazon CloudWatch (AWS.Amazon.com, 2023b), a tool to collect and analyze resource and application data. Figure 2 illustrates the metrics for two exemplary nodes and five time stamps. The `metrics.csv` file can be loaded using pandas with `pd.read_csv('metrics.csv', header=[0, 1, 2], index_col=0)`.

4.6. Injected Issues

We list out the different issues we injected below, including two of the most common issues in cloud systems: memory leaks and CPU hogs (see Mariani et al., 2018; Ikram et al., 2022), and provide a summary in Appendix B (Table 4). We parameterized many of these issues by the time delay or the random fraction of traffic for which they would occur. Further, we added AWS System Manager parameters to many of the nodes that we can modify on the fly to turn on and off issues of varying severity without requiring a deploy. Overall, we made sure the error percent or delay in milliseconds were large enough to lead to noticeable spikes on the `PetSite` for at least some traffic scenarios. We repeated each issue twice so that we can study variability. Overall, we also cover a wide range of anomaly severities, from mild deviations from normality (∼1-2 standard deviation events) to rare extreme events (>3 standard deviations) (see Table 7 and Figures 5 and 6 in Appendix B for summary statistics and the distributions IT anomaly scores for latency and availability metrics at `PetSite`). This will enable practitioners at companies to see how closely the severities of issues explored in our dataset match those they observe during issues, and determine whether or not our work is relevant for their setting.

Requests Overload: We overload the database with external requests hitting the throttling limit. This causes issues in the `petInfo DynamoDB Table` node marked in Figure 1. This issue is mimics a real issue we encountered.

Memory Leak: The application comes with a memory leak that may trigger for requests of a specific type (concerning the adoption of bunnies only). This causes errors in the `payforadoption ECS Fargate` service that propagates to the `PetSite`. Memory leaks are common issues in cloud systems (see Mariani et al., 2018; Ikram et al., 2022).

CPU hog: We introduce a CPU hog in the `lambdastatusupdater Lambda Function` that keeps the CPU busy for a certain duration of time through multiplications of random numbers, which leads to an increased latency that propagates to the `PetSite`. CPU hogs represent common issues in cloud systems (see Mariani et al., 2018; Ikram et al., 2022).

Misconfiguration: The application comes with a misconfigured storage bucket in the `PetSearch ECS Fargate` microservice. This reflects a common problem of insufficient permissions. We randomly select 1% - 2% of requests subject to this error. We additionally trigger a miscon-

Figure 3: Example showing how delays in the `payforadoption ECS Fargate` microservice reflect in the response times at the PetSite.

figured database table name in the `lambdastatusupdater Lambda Function` and the `petlistadoptions ECS Fargate` microservice. These misconfigurations lead to a decrease in availability of `PetSite`. For the `lambdastatusupdater Lambda Function` this error affects requests for kittens. For the `petlistadoptions ECS Fargate` microservice the percentage of requests that used an invalid database table ranges from 2% to 10%.

Other Delays: We also introduced delays in `PetSearch ECS Fargate` and `payforadoption ECS Fargate` through sleep statements. An example for such a delay and its propagation to `PetSite` is shown in Figure 3. These naive sleep statements are meant to model performance regressions of microservices that can be introduced by code changes.

To instrument delays in `PetSearch ECS Fargate` we introduced a Systems Manager parameter that governs the delay in milliseconds (between 500 and 2000) that is injected for search requests of bunnies. In order to limit the impact from retrieving this parameter we implemented a caching layer of the parameters refreshing them once a minute. That way we limit requests to the Systems manager while guaranteeing some freshness. As a consequence though there can be some time that passes between the trigger time (when we turn on the parameter) and the service refreshes the parameter. The fact that the delay only impacts the bunny requests means that its not all latency percentiles are effected the same way. To instrument delays in `payforadoption ECS Fargate` we introduced a Systems Manager parameter that governs the delay in milliseconds (between 250 and 1000) that is injected for all requests.

In total we have 68 issues triggered at 5 different nodes across the three traffic scenarios causing SLO violations due to high latency or low availability at the PetSite node. For each of the 68 issues,

Table 2: We list the number of issues affecting the latency and availability at the PetSite node for each of the distinct traffic scenarios.

Traffic Scenario	number of latency issues	number of availability issues
low-traffic	14	12
high-traffic	14	12
temporal-traffic	8	8

there is exactly one ground-truth root cause, based on where we inject the issue. Table 2 lists how they are split across the three traffic scenarios and latency and availability performance measures.

4.7. Evaluation

A method for RCA can take as input the service map, the metrics, and information about the SLO violation (including the target node and metric) and needs to output a list of potential root causes composed of a node and potentially a metric with a confidence score. This could look as follows:

```
def analyze_root_causes(graph: nx.DiGraph, target_node: str,
    target_metric: str, target_statistic: str, normal_metrics:
    pd.DataFrame, abnormal_metrics: pd.DataFrame) ->
    List[PotentialRootCause]:
    """Finds root causes of a performance issue in target_node."""
```

Using these outputs, we provide a method to evaluate the given RCA approach on the dataset. We compute the top-1 and top-3 recall. Given that there is a unique root cause that represents the ground truth the top-1 recall captures the accuracy of the method to determine the correct root cause. In practice, a method can be used to present an oncall engineer with a ranked list of root-causes. The engineer can then investigate further. To capture the quality of this ranked list, we additionally compute the top-3 recall. An approach can be evaluated calling the following method:

```
def evaluate(analyze_root_causes, dir: str, split: str=None) ->
    pd.DataFrame:
    """Computes the top-k recall of the method to analyze root causes."""
```

5. Experiments

We provide an initial comparison of proposed RCA methods on the 68 injected issues.

Methods

We test both methods relying on the causal graph and those that do not. The causal formalization of RCA makes clear that we would expect methods which are given causal information, in the form of the causal graph, to perform better than those which are either not causal (owing to false positives) or which must learn the causal graph or SCM from the data (owing to needing to make strong assumptions and the statistical challenge of learning with limited data). However, as in practice we often do not know the causal graph, we also assess methods that do not require it.

Methods that require a causal graph Inspired by Chen et al. (2014); Lin et al. (2018); Liu et al. (2021), the *traversal* method identifies a node as a root cause under two conditions: 1) none of its parent nodes exhibits anomalous behavior, and 2) it is linked to the SLO violating node exclusively through nodes showing anomalous behavior. This thereby encodes the requirement that the anomalous behaviour at a root cause should not be explained by its parents, as follows from the formalism that

an anomaly occurs due to a mechanism change at the root cause, and that it should contribute to the anomalous behaviour at the target. Although simple, traversal is therefore an $\mathcal{L}_2$ method. For labeling a node X as anomalous based on a given observation x, we use the MAD-score with a threshold of 5. This score is the normalized distance to the median defined as $\frac{|x-\text{median}[X]|}{\text{MAD}[X]}$, where $\text{MAD}[X]$ denotes the median absolute deviation. We have experimented with other anomaly scores implemented within the Python library *DoWhy* (Blöbaum et al., 2022) that we describe in the Appendix C.1 from which MAD-score performed best.

CIRCA (Li et al., 2022) likewise requires a causal graph, and identifies root causes by testing which causal conditional distributions have changed. Practically this is achieved by fitting a linear Gaussian SCM to the data from the normal period, and comparing the predicted value of each variable given its parents to the true value in the abnormal period. CIRCA is therefore does not treat RCA counterfactually and is an $\mathcal{L}_2$ method.

Counterfactual Attribution (Budhathoki et al., 2022), like CIRCA, requires an input causal graph which is then used to fit an SCM, but instead treats RCA as a counterfactual contribution problem (as described in Sec. 2), returning Shapley value-based contribution scores for each potential root cause. This is therefore an $\mathcal{L}_3$ method.

Methods that do not require a causal graph Without a causal graph we must either learn one from the data or else rely on associational, $\mathcal{L}_1$, methods. A simple heuristic used in practice is to rank potential root causes according to their *correlation* with the target in the abnormal period. In this study we filter for nodes which have been detected as anomalous using the MAD-score, use the Pearson correlation coefficient and rank according to its *p*-value as a simple baseline.

ϵ-Diagnosis (Shan et al., 2019) constructs a test statistic based on the energy distance correlation coefficient (Székely and Rizzo, 2014) and performs a two-sample test to identify which services changed significantly from the normal to abnormal periods. As this does not assess whether the change in a service can be explained by its parents, this is a $\mathcal{L}_1$ method.

RCD (Ikram et al., 2022) exploits the fact that mechanism changes can be modelled as soft interventions, and thereby applies work on causal discovery with unknown interventions (Jaber et al., 2020) to identify root causes. More precisely, by introducing a binary 'intervention indicator' variable, testing which variables' causal conditional distributions have changed between the normal and abnormal periods amounts to checking conditional independence statements involving the indicator: for the variables which cannot be made conditionally independent of the intervention indicator it must also be the case that their 'anomalousness' cannot be explained by their parents. These variables are therefore the children of the intervention indicator in an extended causal graph, which can be identified under faithfulness and causal sufficiency. RCD is therefore an $\mathcal{L}_2$ method.

For evaluation we use the implementation by Salesforce (2023) for the methods CIRCA, ϵ-Diagnosis and RCD and the implementation by (Blöbaum et al., 2022) for the counterfactual attribution method. We opensource (link to be added upon acceptance) implementations of the traversal and the correlation methods. In the Appendix C.2 we describe the configuration options we tested for each method and which one performed best and is used in the results below.

Results. As expected the results presented in Table 3 illustrate that methods with access to a causal graph typically perform better than those without. However, counterfactual attribution struggles at identifying root causes for drops in availability. This is likely due to the low variability in availability across the normal period, making learning an SCM challenging. By comparison, as traversal needs only to perform anomaly detection, it is robust to low variability in the normal period and to small numbers of data points in the abnormal period.

When it comes to methods that do not have access to the graph, however, the performance drops dramatically. In fact, none of the proposed approaches outperform the basic ranked correlations. This

Table 3: Top-3 recall of the RCA methods measuring the accuracy of including the correct root-cause node in the top-3 results.

		graph given			graph not given		
traffic scenario	metric	traversal	circa	counter-factual	ϵ-diagnosis	rcd	correlation
low	latency	0.57	**0.86**	0.71	0.00	0.21	0.57
low	availability	**1.00**	**1.00**	0.42	0.00	0.75	0.92
high	latency	0.79	**1.00**	0.86	0.00	0.07	0.79
high	availability	**1.00**	0.00	0.00	0.33	0.00	0.92
temporal	latency	**1.00**	**1.00**	0.50	0.12	0.75	0.75
temporal	availability	**1.00**	**1.00**	0.25	0.12	0.75	0.75

calls for more research into the development of robust causal attribution methods which to not depend on knowing the causal graph. While RCD and ϵ-diagnosis achieved good performances on hundreds of data-points (e.g. 1 second aggregated metrics) as illustrated by Ikram et al. (2022) they are not designed to handle limited data as present in this dataset. It is noteworthy, that in the top-1 recall (see Table 8 in Appendix C.3) ranked correlation outperforms all other methods including those with access to the graph. We hypothesize that both the data size and the graph structure play a role here. Pairwise-correlations work better with little data compared to estimating SCMs or discovering graph structures. Additionally, in these experiments we focus on SLO violations at the PetSite node, which, in the causal graph, is a leaf and does not share a common cause with another node. As a result false positives arising from being unable to distinguish its causes and effects, and association due to common causes, are not an issue. We hope future work sheds more light on error sources (including errors arising from finite samples, and causal graph misspecification). To investigate whether method performance would vary according to the severity of the injected issue, we additionally evaluated performance stratified by IT anomaly score (see Appendix C.1), and find that most methods do not consistently perform better across both latency and availability issues when the strength of the observed anomaly is larger (see Tables 9 and 10).

Additionally, we tested the specificity on normal data but found that all methods fabricate root-causes during normal operations. We leave the adaption of the methods for improved specificity for future work. Setting a threshold on the scores can be a first starting point.

6. Limitations

This dataset is generated from an Web application running on cloud infrastructure created for demonstration. It is not an application running in production with real user traffic. The traffic is generated artificially. This is a crucial limitation in our dataset. In real applications it is possible that issues affect user behavior. In case of errors or increased latencies users may refresh a page, restart an application or abandon their session. These reactions lead to a change in request which in turn can affect metrics across all microservices of an application. As such, causal approaches which assume the underlying causal graph is acyclic may perform well on our benchmark but not in the presence of such feedback loops. Work on extending causal formalizations of RCA to non-recursive models is an important future direction.

Additionally, we do not consider cases where there is more than one root cause as discussed in Oesterle et al. (2023). While two services independently failing at once is rather unlikely, multiple root causes can still occur in practice e.g. due to failures at unmeasured confounders.

Further, the issues we have injected into microservices are likely to be different from issues encountered in real-world applications. While they include some common issues in cloud systems (i.e. memory leaks and CPU hogs) (see Mariani et al., 2018; Ikram et al., 2022) in addition to request overloads and misconfigurations, they lack coverage and are not representative. Furthermore, we performed some tuning of the parameters for issue injection (the fraction of requests to be affected or the delay in ms) to make sure that they have a pronounced effect in the system, and thereby introduce a bias for stronger anomalous events. We might expect therefore that the dataset not give a complete picture of method performance for weak anomalous events, however, when the evaluation is stratified by anomaly severity, we do not see that methods perform consistently better for stronger anomalies (see Tables 9 and 10 in Appendix C.3).

Lastly, the methods we experimented with have not been tuned or massaged. That way they give us a sense for the performance they can give out-of-the-box for this new dataset. Moreover, we have not included all RCA methods in the comparison.

7. Conclusion

In this paper, we introduced a dataset that encompasses metric data from a microservices-based application during both normal and anomalous operational periods. We believe that this dataset could serve as a valuable resource for evaluating and benchmarking RCA techniques. For illustration, we report the top-3 recall for a selection of published RCA methods. These methods approach the problem of defining a root cause following different formalizations spanning the causal hierarchy (Bareinboim et al., 2022). We find that causal methods perform well when we provide the causal graph, but that methods relying on learning the causal graph or the full SCM do not yield satisfactory performance when data is limited. In this case the simple baseline of ranking potential root causes according to the p-value for correlation with the target, and filtering for variables detected as anomalous, provides a strong baseline against which new causal methods should be evaluated. We thus hope that with this dataset we can enable future research into robust methods for root cause analysis that work well with limited data. In order to work in practice, such methods should not require access to the causal graph and should be able to work with a handful of abnormal measurements to make a timely diagnosis.

We anticipate that this dataset will inspire further exploration and advancement in the field of RCA methods. Additionally, we encourage the community to contribute by expanding this dataset to cover other applications and supplement it with performance issues observed in real-world production systems.

References

Mona Attariyan, Michael Chow, and Jason Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, page 307–320, USA, 2012. USENIX Association. ISBN 9781931971966.

AWS-Samples. One observability demo, 2023. URL https://github.com/aws-samples/one-observability-demo.

AWS.Amazon.com. Amazon web services (aws), 2023a. URL https://aws.amazon.com/.

AWS.Amazon.com. Amazon cloudwatch, 2023b. URL https://aws.amazon.com/cloudwatch/.

AWS.Amazon.com. Aws x-ray, 2023c. URL https://aws.amazon.com/xray/.

Elias Bareinboim, Juan Correa, Duligur Ibeling, and Thomas Icard. On pearl's hierarchy and the foundations of causal inference (1st edition). In Hector Geffner, Rina Dechter, and Joseph Y. Halpern, editors, *Probabilistic and Causal Inference: the Works of Judea Pearl*, pages 507–556. ACM Books, 2022.

Patrick Blöbaum, Peter Götz, Kailash Budhathoki, Atalanti A. Mastakouri, and Dominik Janzing. Dowhy-gcm: An extension of dowhy for causal inference in graphical causal models, 2022.

E. Brewer, M. Y. Chen, E. Fratkin, A. Fox, and E. Kiciman. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings International Conference on Dependable Systems and Networks*, page 595, Los Alamitos, CA, USA, June 2002. IEEE Computer Society. doi: 10.1109/DSN.2002.1029005. URL https://doi.ieeecomputersociety.org/10.1109/DSN.2002.1029005.

Kailash Budhathoki, Dominik Janzing, Patrick Bloebaum, and Hoiyi Ng. Why did the distribution change? In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1666–1674. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/budhathoki21a.html.

Kailash Budhathoki, Lenon Minorics, Patrick Bloebaum, and Dominik Janzing. Causal structure-based root cause analysis of outliers. In *ICML*, 2022.

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, 2014. doi: 10.1109/INFOCOM.2014.6848128.

Elias Eulig, Atalanti A Mastakouri, Patrick Blöbaum, Michaela Hardt, and Dominik Janzing. Toward falsifying causal graphs using a permutation-based test. *arXiv preprint arXiv:2305.09565*, 2023.

Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An open-source

benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 3–18, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi: 10.1145/3297858.3304013. URL https://doi.org/10.1145/3297858.3304013.

Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: Practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 135–151, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446700. URL https://doi.org/10.1145/3445814.3446700.

Patrik Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper_files/paper/2008/file/f7664060cc52bc6f3d620bcedc94a4b6-Paper.pdf.

Tsuyoshi Idé, Amit Dhurandhar, Jiří Navrátil, Moninder Singh, and Naoki Abe. Anomaly attribution with likelihood compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4131–4138, 2021.

Azam Ikram. Sock-shop data, 2023. URL https://github.com/azamikram/rcd/tree/master/sock-shop-data.

Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. Root cause analysis of failures in microservices through causal discovery. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 31158–31170. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf.

Amin Jaber, Murat Kocaoglu, Karthikeyan Shanmugam, and Elias Bareinboim. Causal discovery from soft interventions with unknown targets: Characterization and learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9551–9561. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6cd9313ed34ef58bad3fdd504355e72c-Paper.pdf.

E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, 2005. doi: 10.1109/TNN.2005.853411.

Kubernetes.io. Kubernetes, 2023. URL https://kubernetes.io/.

Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 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*, KDD '22, page 3230–3240, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539041. URL https://doi.org/10.1145/3534678.3539041.

Jinjin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, editors, *Service-Oriented Computing*, pages 3–20, Cham, 2018. Springer International Publishing. ISBN 978-3-030-03596-9.

Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '21, page 338–347. IEEE Press, 2021. ISBN 9780738146690. doi: 10.1109/ICSE-SEIP52600.2021.00043. URL https://doi.org/10.1109/ICSE-SEIP52600.2021.00043.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

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*, WWW '20, page 246–258, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380111. URL https://doi.org/10.1145/3366423.3380111.

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, 2018. doi: 10.1109/ICST.2018.00034.

Microservices-Demo. Sock-shop - a microservice demo application, 2022. URL github.com/microservices-demo/microservices-demo.

Michael Oesterle, Patrick Blöbaum, Atalanti A Mastakouri, and Elke Kirschbaum. Beyond single-feature importance with icecream. *arXiv preprint arXiv:2307.09779*, 2023.

Judea Pearl. *Causality*. Cambridge University Press, Cambridge, second edition, 2009.

Judea Pearl and Dana Mackenzie. *The Book of Why*. Basic Books, New York, 2018. ISBN 978-0-465-09760-9.

Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. pages 97–101, 02 2016. doi: 10.18653/v1/N16-3020.

Salesforce. Pyrca, 2023. URL https://github.com/salesforce/PyRCA.

Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. ϵ-diagnosis: Unsupervised and real-time diagnosis of small- window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*, WWW '19, page 3215–3222, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313653. URL https://doi.org/10.1145/3308558.3313653.

L. S. Shapley. *17. A Value for n-Person Games*, pages 307–318. Princeton University Press, Princeton, 1953. ISBN 9781400881970. doi: doi:10.1515/9781400881970-018. URL https://doi.org/10.1515/9781400881970-018.

Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, and Antti Kerminen. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(72):2003–2030, 2006. URL http://jmlr.org/papers/v7/shimizu06a.html.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 06–11 Aug 2017.

Székely and Rizzo. Partial distance correlation with methods for dissimilarities. 42, 2014. doi: 10.1214/14-aos1255.

Hanzhang Wang, Phuong Nguyen, Jun Li, Selcuk Kopru, Gene Zhang, Sanjeev Katariya, and Sami Ben-Romdhane. Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform. *Proc. VLDB Endow.*, 12(12):1942–1945, aug 2019. ISSN 2150-8097. doi: 10.14778/3352063.3352105. URL https://doi.org/10.14778/3352063.3352105.

Appendix A. Application

Figure 4 shows a screenshot of the pet adoption webpage. It supports search and purchasing features powered by different microservices.

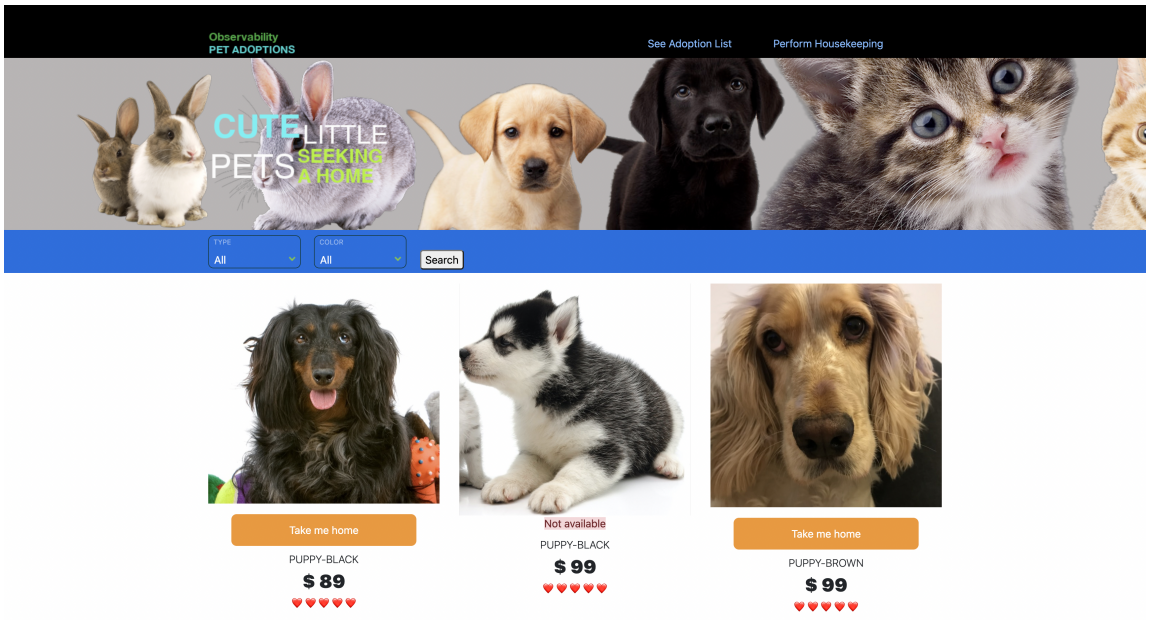

Figure 4: Screenshot of the landing webpage of the pet adoption application.

Appendix B. Dataset

The data is stored according to the following structure:

```
scenario/
   graph.csv
   noissue/
      metrics.csv
   train/
      issue0/
         metrics.csv
         target.json
      issue1/
         metrics.csv
         target.json
      ...
   test/
      issue0/
         metrics.csv
         target.json
```

Table 4 gives a detailed breakdown of each of the injected issues.

Node	Type of Node	Target Metric	Type of Issue	Relevance	Traffic Impacted	Parameter Range	Total Count	Description
petInfo	database	availability	traffic spike / throttling	as practitioners we have run into this type of issue	a fraction of traffic will be throttled	-	6	We generate many read requests directly hitting the database, triggering throttling.
petSearch	containerized micro-service on ECS	availability	human configuration error	configurations are often done manually and cannot be tested as easily as code	random fraction of all traffic	1%, 2%, 3%	10	A random fraction of requests attempts to access a directory in the file system that does not exist.
petSearch	containerized micro-service on ECS	latency	performance degradation	performance degradations can happen due to deploying less performant code, change in hardware, overall resource exhaustion	bunny requests	0.5sec, 1sec, 2sec	10	Bunny requests are delayed by some number of seconds.
payforadoption	containerized micro-service on ECS	availability	memory leak	among the most common issues in cloud systems, see Mariani et al. (2018); Ikram et al. (2022)	bunny requests	n/a	6	A memory leak is triggered for all requests.
payforadoption	containerized micro-service on ECS	latency	performance degradation	performance degradations can happen due to deploying less performant code, change in hardware, overall resource exhaustion	all	0.25sec, 0.5sec, 1sec	10	All requests are delayed by some number of seconds.
statusupdater	serverless data-processing system on lambda	availability	human configuration error	configurations are often done manually and cannot be tested as easily as code	kitten requests	n/a	6	Kitten requests attempt to access a database table that does not exist.
statusupdater	serverless data-processing system on lambda	latency	performance degradation	performance degradations can happen due to deploying less performant code, change in hardware, overall resource exhaustion	all requests	0.25sec, 0.5sec, 1sec	10	All requests are delayed by some number of seconds.

Table 4: Breakdown of injected issues. In all cases, issues were repeated twice.

traffic scenario	condition	mean (ms)	std
low	normal	9.49	0.57
low	abnormal	13.46	8.53
high	normal	9.82	0.42
high	abnormal	14.42	11.38
temporal	normal	10.04	0.90
temporal	abnormal	24.10	21.34

Table 5: Mean and standard deviations of latency across traffic scenarios and normal and abnormal conditions.

traffic scenario	condition	mean (%)	std
low	normal	99.74	2.48
low	abnormal	98.60	1.85
high	normal	99.91	0.23
high	abnormal	98.66	1.80
temporal	normal	99.85	0.95
temporal	abnormal	98.19	2.37

Table 6: Mean and standard deviations of availability across traffic scenarios and normal and abnormal conditions.

traffic scenario	metric	min	max
low	latency	1.05	7.07
low	availability	2.70	5.97
high	latency	3.71	7.07
high	availability	1.52	7.07
temporal	latency	5.27	8.10
temporal	availability	4.25	5.53

Table 7: Minimum and maximum IT anomaly scores across each traffic scenario and metric.

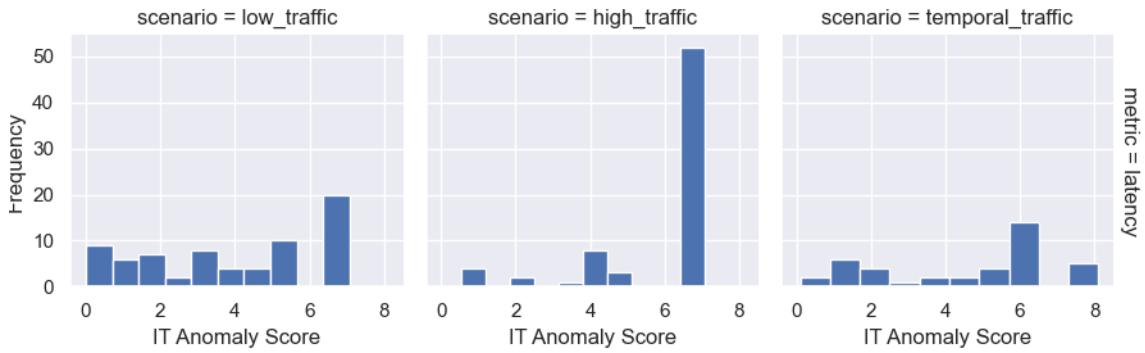

Figure 5: Distribution of IT anomaly scores at PetSite for latency issues.

Appendix C. Experiments

C.1. Anomaly Scores

For the anomaly traversal we experimented with the following anomaly scores:

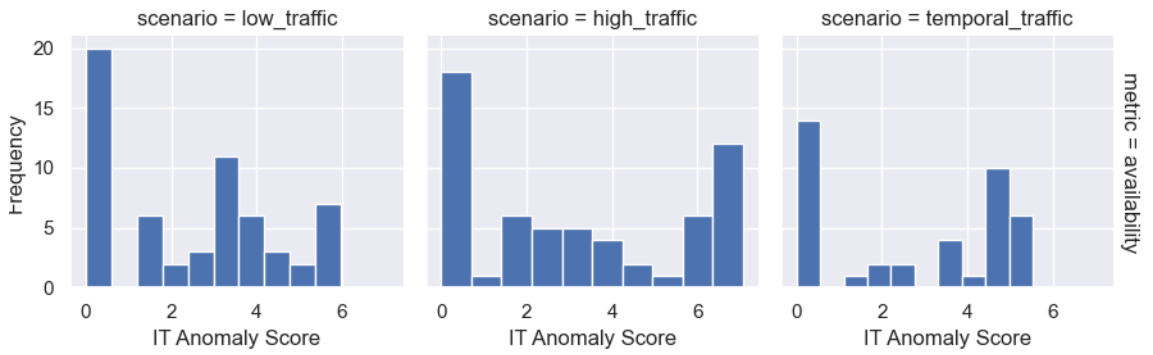

Figure 6: Distribution of IT anomaly scores at PetSite for availability issues.

- **z-score:** This is a measure of the normalized distance from the mean, calculated as $\frac{|x-\mathbb{E}[X]|}{\sqrt{\text{Var}[X]}}$.
- **MAD-score:** This score is the normalized distance to the median defined as $\frac{|x-\text{median}[X]|}{\text{MAD}[X]}$, where $\text{MAD}[X]$ denotes the median absolute deviation.
- **IT-score:** This approach translates the scores defined above into an information theoretic quantity, as per Budhathoki et al. (2022), $-\log(g(X) \geq g(x))$. Here, g is a 'feature' map that functions as an anomaly scorer, producing a score such as the z-score or MAD-score.
- **Median quantile-score:** This score resembles the log-probability of the 2-sided quantile calculated by $-\log(2 \cdot \min\{\Pr[X >= x], \Pr[X <= x]\})$.

C.2. Methods

We experimented with a few options for the various methods and report the best results. In particular, for RCD we ran it with the global option, on all metrics, and with a mean imputation scheme. Best worked the interpolation as imputation method that we report in the results. For ϵ-diagnosis we also tried setting the significance level α to 0.01, using interpolation, and selecting only metrics of the same type as the target metric (e.g. latency or availability) vs using both. Using both performed better and is shown in the results. For CIRCA, we also tried both mean and interpolation for the imputation, both max and sum aggregation, and with and without the descendant adjustment. We include the best performing setting with the mean imputation and the max aggregation in the results.

For correlation we experimented with using the coefficient for ranking and/or the p-value.

The counterfactual method had the longest run-time owing to the method's reliance on Shapley values.

C.3. Results

Table 8 shows top-1 recall for identifying the root cause. Tables 9 and 10 show the top-3 recall for each method stratified according the strength of the anomaly detected at the target (PetSite). In particular, small anomalies are those with an IT anomaly score (see Appendix C.1 above) lower than the median score across issues of the same type (either latency or availability), whereas large anomalies are those with a score greater than the median.

		graph given			graph not given		
traffic scenario	metric	traversal	circa	counter-factual	ϵ-diagnosis	rcd	correlation
low	latency	**0.57**	0.36	0.36	0.00	0.07	0.43
low	availability	0.50	0.42	0.00	0.00	0.58	**0.75**
high	latency	0.57	0.50	0.57	0.00	0.00	**0.64**
high	availability	0.33	0.00	0.00	0.00	0.00	**0.83**
temporal	latency	**1.00**	0.75	0.38	0.12	0.25	0.62
temporal	availability	0.38	0.38	0.00	0.00	0.50	**0.62**

Table 8: Top-1 recall of the RCA methods measuring the accuracy of identifying the correct root-cause node.

		graph given			graph not given		
traffic scenario	anomaly size	traversal	circa	counter-factual	ϵ-diagnosis	rcd	correlation
low	small	**1.00**	**1.00**	0.25	0.00	**1.00**	0.88
low	large	**1.00**	**1.00**	0.75	0.00	0.25	**1.00**
high	small	0.75	0.00	0.00	0.25	0.00	**1.00**
high	large	0.75	0.00	0.00	0.38	0.00	**0.88**
temporal	small	**1.00**	**1.00**	0.33	0.17	0.67	0.67
temporal	large	**1.00**	**1.00**	0.00	0.00	**1.00**	**1.00**

Table 9: Top-3 recall for methods for availability issues stratified by the size of the anomaly at the target. Small: IT anomaly score < median score for availability issues, large: IT anomaly score > median for availability issues.

		graph given			graph not given		
traffic scenario	anomaly size	traversal	circa	counter-factual	ϵ-diagnosis	rcd	correlation
low	small	0.40	**1.00**	0.80	0.00	0.20	0.20
low	large	0.67	**0.78**	0.67	0.00	0.22	**0.78**
high	small	0.67	**1.00**	0.83	0.00	0.00	0.50
high	large	0.88	**1.00**	0.88	0.00	0.12	**1.00**
temporal	small	**1.00**	**1.00**	0.50	0.00	0.67	0.67
temporal	large	**1.00**	**1.00**	0.50	0.50	**1.00**	**1.00**

Table 10: Top-3 recall for methods for latency issues stratified by the size of the anomaly at the target. Small: IT anomaly score < median score for latency issues, large: IT anomaly score > median for latency issues.