Root Cause Analysis of Failures in Microservices through Causal Discovery

Azam Ikram [1] Sarthak Chakraborty [2] Subrata Mitra [2] Shiv Kumar Saini [2] Saurabh Bagchi [1] Murat Kocaoglu [1]

[1]Purdue University [2]Adobe Research

Microservice-based Architecture

Modern cloud applications follow a network topology that consists of multiple components (microservices) connected through complex dependencies. It provides the following attractive features:

- Easy development: Developers can write independent microservices in parallel.
- Smaller and faster deployments: Smaller codebase leads to quick deployments and therefore enables Continuous Deployment.
- Uncomplicated management: Individual microservices can be scale-down and scale-up.

Root Cause Analysis (RCA)

RCA is the process of finding the root cause of a failure.

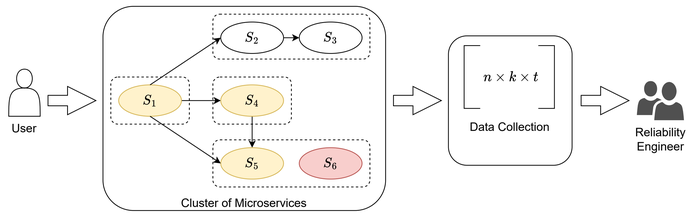


Figure 1. The workflow of a microservice-based system from the usage point of a client and a Site Reliability Engineer (SRE). Every oval is a microservice and the edges represent the call graph. Red-colored microservices signify a failure but the effect is observed on the yellow-colored microservices.

Challenges

Debugging a microservice-based application is difficult because of the following reasons;

1. Extremely high workload and complex dependency structure between the microservices leads to easy easy fault propagation.
2. Enormous amount of tracing and logging data makes it tedious to narrow down the failure to a few services.
3. The delay in detecting the root cause of a fault can result in significant revenue loss. Therefore, the root cause needs to be detected as soon as possible which leads to limited number of anomalous samples.

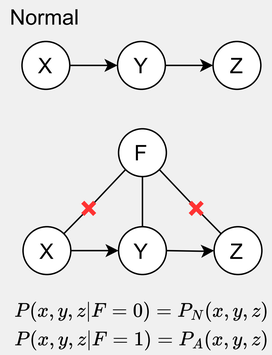
Existing Solutions

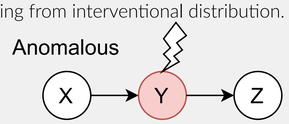
The existing solutions fall short because they;

1. make parametric assumptions between the metrics of different microservices (such as linear relationship between the CPU utilization of A and latency of B.
2. require expert domain knowledge about the architecture of the application or rely on historical data from the past failures.

Key Observation

Consider anomalous data as the data coming from interventional distribution.





$$P(x, y, z|F = 0) = P_N(x, y, z)$$
$$P(x, y, z|F = 1) = P_A(x, y, z)$$

$$P(x|F = 0) = P(x|F = 1) \Leftrightarrow X \perp\!\!\!\perp F$$
$$P_N(x) = P_A(x)$$

$$P(z|y, F = 0) = P(z|y, F = 1) \Leftrightarrow Z \perp\!\!\!\perp F|Y$$
$$P_N(z|y) = P_A(z|y)$$

$$P(y|x, F = 0) = P(y|x, F = 1) \Leftrightarrow Y \not\perp\!\!\!\perp F|X$$
$$P_N(y|x) \neq P_A(y|x)$$

Figure 2. The mapping from anomalous data to interventional data allows us to use causal discovery algorithms with unknown targets to find the interventional target.

In our context, the interventional target is the root cause of the failure.

Root Cause Discovery (RCD) Algorithm

1. Hierarchical: Split the data into small subsets and find candidate targets.
2. Localized: Only find the interventional targets.

Theorem: *Given access to a perfect conditional independence oracle, and under the causal sufficiency, and the extended faithfulness assumptions RCD returns the true root cause variables.*

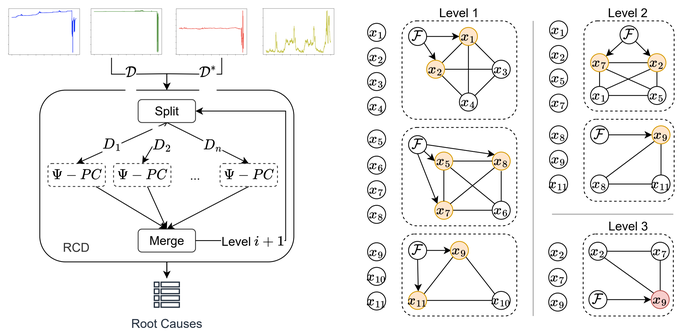


Figure 3. RCD (left) follows the divide-and-conquer approach. It first splits the data and finds the interventional targets from each subset. In the second phase, it combines the candidate root causes of all the subsets and performs the same steps recursively. The example (right) shows an execution of RCD with 11 nodes. The orange nodes are potential root causes that are carried to the next level for further processing and the red node (x_9) is the eventual root cause.

Evaluation

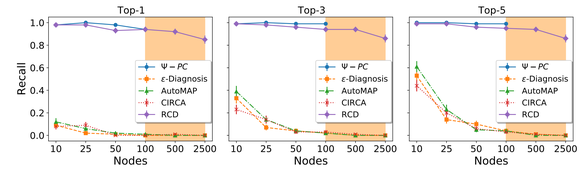


Figure 4. RCD achieves similar top-k recall as Ψ-PC. It outperforms the other baselines because they either require domain knowledge, human intervention, or rely only on lower-order tests such as cross covariance.

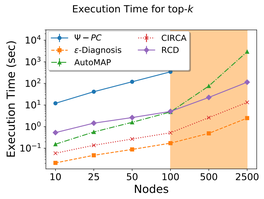


Figure 5. The execution time required for RCD increases slower with the number of nodes than Ψ-PC (note the logarithmic scale on the Y-axis), which does not terminate in a reasonable time for 500 and more nodes. ϵ-Diagnosis achieves the best execution time but its usefulness is limited as its recall drops significantly when the number of node increase

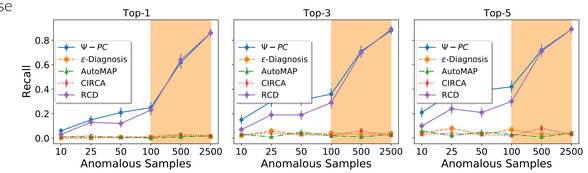


Figure 6. As the number of samples grow, the top-k recall of RCD and Ψ-PC starts to increase as well. That is to say, RCD can provide the same benefits as Ψ-PC while reducing the execution time significantly.

Table 1. The top-7 recall of ϵ-Diagnosis and RCD on data collected for three outages from a production-based cloud application. The length of the vector represents the number of services that were flagged as root cause and the individual number shows the number of metrics belonging to a particular service. The green color illustrates that algorithm was able to correctly detect the root cause whereas the red color shows the algorithm could not find the root cause.

			Rank of Services from top-7		Time (sec)	
Outage	Metrics	Duration (min)	ϵ-Diagnosis	RCD	ϵ-Diagnosis	RCD
$\mathcal{A}$	137	65	[**1**,1,1,1,1,1,1]	[3,1,1,1,1]	0.145	112
$\mathcal{B}$	147	72	[**2**,1,1,1,1,1]	[3,1,1,1,1]	0.186	239.8
$\mathcal{C}$	150	210	[**3**,1,1,1,1]	[4,1,1,1]	0.146	22.57

Conclusion

- Root cause analysis of failures is a challenging task
- Considering the failure as an intervention on the failing node enables us to use approaches from causal discovery literature to find the root cause.
- RCD is a sound hierarchical and localized algorithm to estimate the root cause without the exponential runtime.