Causal Latency Modelling for Cloud Microservices

1st Christopher Lohse
School of Computer Science and Statistics
University of Dublin Trinity College
AI for Sustainability
IBM Research Europe
Dublin, Ireland
lohsec@tcd.ie

2nd Diego Tsutsumi†
AI for Sustainability
IBM Research Europe
Dublin, Ireland

3rd Amadou Ba
AI for Sustainability
IBM Research Europe
Dublin, Ireland
amadouba@ie.ibm.com

4th Pavithra Harsha*
AI for IT Automation
IBM T. J. Watson Research Center
Yorktown Heights, USA
pharsha@us.ibm.com

5th Chitra Subramanian*
AI for IT Automation
IBM T. J. Watson Research Center
Yorktown Heights, USA
cksubram@us.ibm.com

6th Martin Straesser*
Department of Computer Science
University of Würzburg
Würzburg, Germany
martin.straesser@uni-wuerzburg.de

7th Marco Ruffini
School of Computer Science and Statistics
University of Dublin Trinity College
Dublin, Ireland
marco.ruffini@tcd.ie

***Abstract*—The use of microservices-based architectures is becoming more prominent due to their advantageous characteristics, such as manageability, scalability, and flexibility. However, their management can be complex, and their performance can be affected by high latencies, which can alter the Service Level Objective (SLO). In order to identify the causes of high latencies, we present a causal modelling framework which is capable of analysing and reconstructing latencies within microservice-based architectures. To this end, we employ causal discovery to identify the causes of latencies. Our model integrates domain knowledge to impose constraints on the causal graph, ensuring the accuracy of the discovered relationships as well as accelerating the causal discovery. To validate our approach, we reconstruct latency metrics using machine learning techniques and demonstrate the effectiveness of our approach by accurately capturing the interrelationships between microservice resources. Our framework provides a better understanding of the causes of latencies that lead to SLO violations, and paves the way for sophisticated mechanisms that enable proactive management of cloud resources.**

***Index Terms*—causality, causal discovery, latency modelling, microservices, SLOs.**

I. Introduction

Over the recent years, microservices-based architectures have emerged as a popular choice for developing scalable, manageable, and flexible software applications [1], [2]. These modular architectures enable software developers to build complex applications through loosely coupled and independently deployable services. However, the complexity of microservices poses significant challenges, particularly in terms of performance management [3]. One of the main issues is high latency, which can lead to the Service Level Objectives (SLOs) violation, and degraded Quality of Service (QoS). Devising mechanisms for predicting and mitigating the causes of latency in microservices-based architectures is therefore essential for maintaining system performance and reliability. To address this challenge, causal mechanisms are being employed in the prediction of end-to-end latency. This is exemplified in [4], where the authors develop a data-driven cluster manager for interactive cloud microservices that is QoS-aware. In a similar way, [5] presents a causal modelling framework for estimating end-to-end latency distributions in microservice-based web applications. [6] uses the Program Evaluation and Review Technique (PERT) to inform the design of their GNN. Their approach determines the causal interaction between microservices through a graph. Furthermore, [7] develops an approach called GRAF, a graph neural network-based proactive resource allocation framework for minimizing total CPU resources while satisfying the SLO. The existing approaches to latency prediction based on causal mechanisms generally have a similar structure. However, they do not consider constrained causal discovery mechanisms with domain knowledge to enhance the causal discovery process.

This paper presents a causal discovery framework that can be used to analyse latency within microservices-based architectures. Causal discovery is the process of identifying and recovering causal relationships between variables in multivariate systems. Two distinct approaches to the recovery of causal relationships can be distinguished: an interventional setting, in which the system can be interacted with and changes forced; and an observational setting, in which the data is based on previous recordings. The modular setup of contemporary microservice architectures presents a significant challenge in identifying relationships between different parameters of the services, such as CPU, memory, and the number of replicas. Prior research has employed causal discovery on a combination of observational and interventional data to find out the causes of failures or high latency outliers in complex microservice settings [8]–[11]. Other works use reinforcement learning with prior causal knowledge in order to enhance the automatic scaling of services to accommodate varying

†Contributed while working at IBM Research Europe.

*Pavithra Harsha, Chitra Subramanian and Martin Straesser contributed to this work by settting up the robotshop depoloyment and providing the data.

demands [12]. The authors in [13] propose a Granger causality framework to detect causal dependencies based on error-log timestamps.

However, these approaches are inadequate for identifying the primary drivers of latency from observational metric data in a real-world deployed cloud microservice architecture without introducing faults or intervening in the system.

Our approach employs causal discovery methods to uncover the underlying causal structure that contributes to high latency. By integrating domain knowledge into our model, we constrain the causal graph, ensuring that the discovered relationships are accurate, as well as accelerating the causal discovery process. In this paper, we make the following contributions:

1) We propose a comprehensive end-to-end framework that infers the topological structure of microservices through the analysis of latency data.
2) We incorporate domain knowledge into the causal discovery process, thereby ensuring the relevance of the inferred relations between the latencies of different microservices.
3) We evaluate the performance of several causal discovery approaches using constraints in practical settings with microservices-based architectures and a real dataset.
4) We validate our framework by reconstructing latencies using machine learning approaches combined with our causal discovery framework. To the best of our knowledge, this is the first work to demonstrate how constrained causal discovery methods can be used to discover factors contributing to latency of microservices from observational data.

II. Background on Causal Discovery

Causality is the study of cause and effect relationships. In this context, a variable X is said to cause another variable Y if changes in X lead to changes in Y, denoted as $X \rightarrow Y$ [14]. For multivariate datasets comprising multiple random variables, the causal relationships among variables can be modeled using a Structural Causal Model (SCM). In an SCM, each variable X is assigned a value based on a function of a subset of its respective causal parents Pa_X, such that $X = f_X(Pa_X, \eta_X)$, where η_X is an independent noise term [15]. The SCM can be represented graphically as a Directed Acyclic Graph (DAG) $\mathcal{G}$. Causal discovery aims to infer the structure of $\mathcal{G}$ from observational or interventional data [16].

For non-time series data, with no autocorrelation or lagged dependencies, $\mathcal{G} = (\mathcal{V}, \mathcal{D})$ is a DAG. Where $\mathcal{V}$ represents the set of vertices and $\mathcal{D}$ represents the set of directed edges. The PC-Algorithm [17] is a prominent causal discovery method for such data. It reconstructs $\mathcal{G}$ by performing conditional independence tests to determine the graph's skeleton, starting from a fully connected graph and iteratively removing edges. Subsequently, edges are oriented using a set of rules outlined in [17]. If the direction of an edge cannot be determined, the algorithm outputs a bidirectional edge, which might be directed using additional background knowledge.

For time series data, causal discovery focuses on identifying either the full time series graph $\mathcal{G}$ or a summary graph $\mathcal{G}_{\text{sum}}$. A stationary time series graph is defined as a directed graph $\mathcal{G} = (V \times \mathbb{Z}, \mathcal{D})$, where $V = \{1, \ldots, d\}$, with edges $((i, t-k), (j, t))$ that are invariant under time translation. Usually the existence of a finite maximum time lag $\tau = \max_{i,j \in V}\{k \mid ((i, t-k), (j, t)) \in \mathcal{D}\} < \infty$ is assumed, and that the contemporaneous component of $\mathcal{G}$ is acyclic. The summary graph $\mathcal{G}_{\text{sum}}$ is a directed, potentially cyclic graph over V which contains a directed edge (i, j) if $(i, t-k) \rightarrow (j, t) \in \mathcal{D}$ for some lag k. A notable algorithm for causal discovery in time series data is PCMCI$^+$ [18], which extends the PC algorithm to account for both contemporaneous and time-lagged dependencies while considering autocorrelations. Similar to the PC algorithm, PCMCI$^+$ utilizes conditional independence tests and starts with a nearly fully connected graph as the dependencies can only propagate forward in time. The algorithm applies orientation rules and outputs a bidirectional edge when the direction is not clearly determined [18].

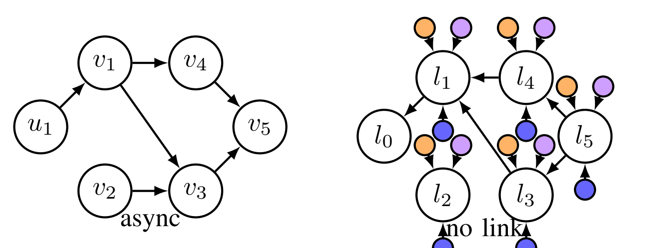
III. Constrained Causal Discovery in Microservices-based Architectures

We first lay out some preliminaries about microservices-based architectures, then present our assumptions. Subsequently, we formalize our proposed causal latency model and present our causal discovery framework.

A. Assumed Background Knowledge

Based on domain-specific knowledge, we assume the following regarding the causal relationships between resources and latency in a microservice application:

A1) Any two endpoints within the same microservice never call each other.
A2) The number of client requests does not directly affect application latency.
A3) High infrastructure usage degrades application performance, not the other way around.
A4) Endpoints of the same microservice are deployed to the same host.
A5) Any metric x_i recorded at any microservice v_i only has a direct influence on the latency l_i of the same microservice, and not on any other latency l_y.



(a) Call Graph

(b) Latency Graph

Fig. 1: Comparison between Call Graph and Latency Graph for five microservices $v_i \in V$ and one user request u_1. Coloured nodes represent the resources attached to a microservice contributing to its latency.

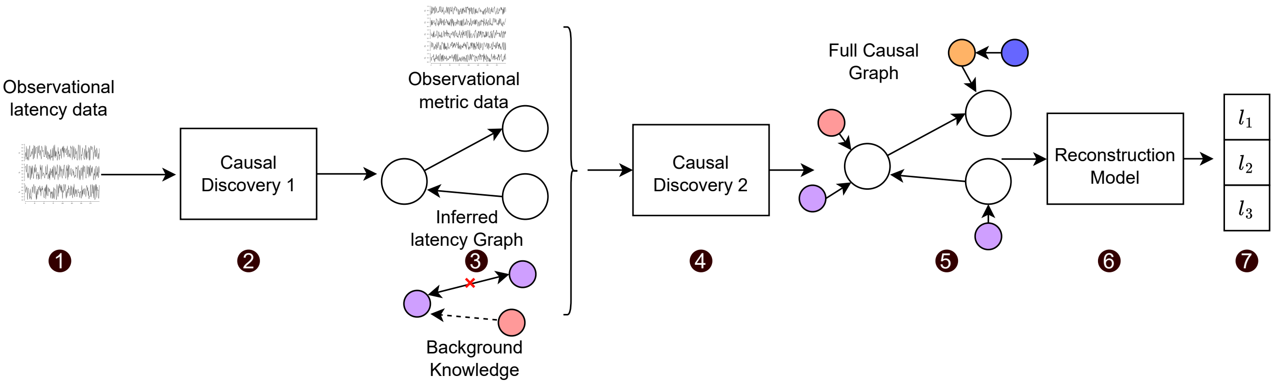


Fig. 2: Overview of the proposed process: ❶ Observational data of the recorded latencies is fed into a causal discovery algorithm ❷. The constructed causal latency graph is then used in combination with expert knowledge and observational data of microservice metrics ❸ and fed into the second causal discovery algorithm ❹, which outputs the full causal graph of the whole system, ❺ is used as a feature selection process for any prediction model ❻ to reconstruct the latency at the microservice level ❼.

A common way to visualise the dependencies in a microservice-based architecture is through a call graph as shown in Fig. 1a, where the microservices are connected via edges $\mathcal{D}$. $(i, j) \in \mathcal{D}$ indicates that the microservice v_i calls the microservice v_j. A more insighful representation of the microservice-based architecture is a latency graph, which can be approximated by the reversed call graph as shown in Fig. 1b. The difference between the reversed call graph and a potential latency graph is that asynchronous calls represented in the call graph are not present in the latency graph.

As we assume that the latency of each microservice does solely depend on its own resources and other latencies we further make the following assumption which is illustrated in Fig. 3:

Assumption 1: Any resource r_{iz} , $z \in R_i$, where $|R_i|$ is the number of resources, can only influence the latency for the endpoint li directly (see Fig. 3). It never directly influences l_j, meaning l_i acts as mediator between any resource $r_{iz} \in R_i$ contributing to l_i. However, we can never intervene on latencies only on resources. This means if we want to intervene on any latency l_i we can only intervene on it by proxy by intervening on its resources R_i.

Since we do not expect cyclical call or latency relationships, we assume that the latency graph takes the form of a DAG. For the latency DAG in Fig. 1b, the structured causal model can be defined as:

$$l_0 := f_{u_1}(l_1, \eta_{l_{u1}}), \quad l_1 := f_{l_1}(l_4, l_3, \eta_{l_1}),$$
$$l_3 := f_{l_3}(l_5, \eta_{l_3}), \quad l_4 := f_{l_4}(l_5, \eta_{l_4}), \quad l_5 := f_{l_5}(\eta_{l_5}),$$

where η_x represents the noise term for each latency. We further know that η_x is not a random noise because the latency of each microservice depends on external factors such as CPU, memory limits and the number of pods. We can further assume that the relationship between any latency l_i and another latency l_j is approximately linear. This means we can rewrite $l_1 := f_{l_1}(l_{p_1}, \eta_{v_1})$, where $l_{p1} = c_3 l_3 + c_4 l_4$ are the latency values of the causal parents of l_1 and c_3, c_4 are constants. η_{l_1} can then be described by the latency limiting resources $r_{1i} \in R_1$ of v_1 plus a remaining random noise term η'_{l_1} thus $\eta_{l_1} = f_{n1}(R_1, \eta'_{l_1})$ where f_{n1} is the noise function for l_1.

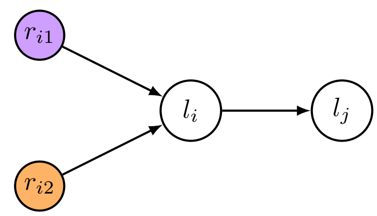


Fig. 3: Graph illustrating the mediator assumption. The resources of l_i only influence l_j with l_i as a mediator.

Thus, we further make the following proposition:

Proposition 1: For the general case the latency of any microservice v_i can be represented as a function of the latency of its causal parents Pa_{li}, its resources R_i and a noise term η'_{vi} (e.g. influenced by different number of calls) by the structural equation f_i with

$$l_i = f_i(Pa_{li}, R_i, \eta'_{vi}). \tag{1}$$

B. Causal Discovery Framework

Considering these assumptions and the approach to modelling latency, we propose a framework that utilizes causal discovery to efficiently build a causal representation of microservice-based architectures. An overview of the methods is given in Fig. 2. We denote the available observational data as $X \in \mathbb{R}^{N \times v}$, where N is the number of observations and v is the number of variables. In step ❷ we first select only the subset of all latency measurements $L \subset X$ (❶) this is used to determine the latency Graph $\mathcal{G}_L$. We use a linear conditional independence test as we assume that the relationship between two latency measurements $l_i, l_j \in L$ given a third latency l_k

can be regressed out by a linear function; meaning l_i and l_j are conditional independent given l_k. Then, we determine $\mathcal{G}_L$ using a causal discovery algorithm fulfilling the assumptions about the data. Depending on the frequency of the recorded data it might be necessary to use a time series causal discovery algorithm where the maximum time lag τ has to be determined by using data analysis methods, or if the recorded frequency of the values is too low we might also use non-time series causal discovery algorithms. If we assume latent confounding, meaning that two latency values are caused by the same unknown factor e.g. an external API, then, we also need to consider this information in this step. The graph $\mathcal{G}_L$ is combined with the measurements of the performance metrics and resources $R \subset X = X/L$ (observations in X without the latency) and the background knowledge based on assumptions A1) - A4), which rules out some possible edges (e.g., calls and latency cannot be directly connected based on A2) or helps orient edges (e.g., the CPU cannot be caused by latency but the opposite is possible). In ❸ causal discovery is performed on a microservice level retrieving n sub-graphs G_{R_i} which contain information about latency drivers for each service (❹). In ❹ these sub-graphs can be combined to the overall causal graph $\mathcal{G}$ as they all have the latency nodes $l_i \in \mathcal{G}_L$ as a common element with $\mathcal{G}_L$. Thus, $\mathcal{G} = \mathcal{G}_L \cup (\bigcup_{i=1}^{m} G_{R_i})$. This graph ❺ gives us information of the latency driving factors in the whole system. To test if the determined factors are correct and thus the proposed causal latency model holds we select the causal features X_c to reconstruct the latency of a given endpoint v_i by $X_c = a(m_i, \mathcal{G}, d)$, where a returns the causal ancestors of v_i up to a specified depth d. These features are then fed into a reconstruction model which estimates the structural equation of l_i. If the reconstruction is successful based on an error metric we can assume that the causally determined features are useful in reconstructing the latency. This does not guarantee that the features are indeed causal, but it shows that these features are empirically contributing factors to the latency.

IV. Robot Shop Dataset

Our test use case focuses on Robot Shop[1], an e-commerce deployment that provides a comprehensive environment and functionalities present in real e-commerce websites. Some of the components present are the catalogue, cart, payments, and shipping. Each of these components within the Robot Shop is represented by a distinct microservice. Metrics such as CPU utilisation, calls per seconds and memory utilisation are reported at an endpoint or microservice level. This architectural approach demonstrates the practical implementation of microservices and provides a robust platform for testing various resource provisioning mechanisms specific to a microservice environment. Robot Shop illustrates the advantages and intricacies of a microservices-based application. For our experiments, we used a total of 1,563 values in a sample interval of 1 minute (roughly 26 hours of observational data). We replace the few missing values with previous value imputation, and remove variables with fewer than two unique values across all observations.

[1]https://github.com/instana/robot-shop

V. Experiments and Results

The following section outlines the experimental setup employed for step (❷), the causal discovery of the full causal graph (❹), and the reconstruction of the latency at the endpoint level (❻) for the Robot Shop case study. After this, the obtained results are discussed. Prior to executing the two causal discovery algorithms and the prediction model, a train-test split is conducted, with 10% of the data set aside for evaluation of the reconstruction step of the latency per endpoint. This is done to guarantee that data known in the causal discovery step is not used for assessment of the reconstruction. To validate the suitability of the data for the causal discovery algorithms we test each observed variable for stationarity. Therefore we perform the augmented Dick-Fuller test and reject the null hypothesis of a unit-root with $p_{value} < 0.01$ for each time-series. Thus, the data fulfils the stationarity assumption.

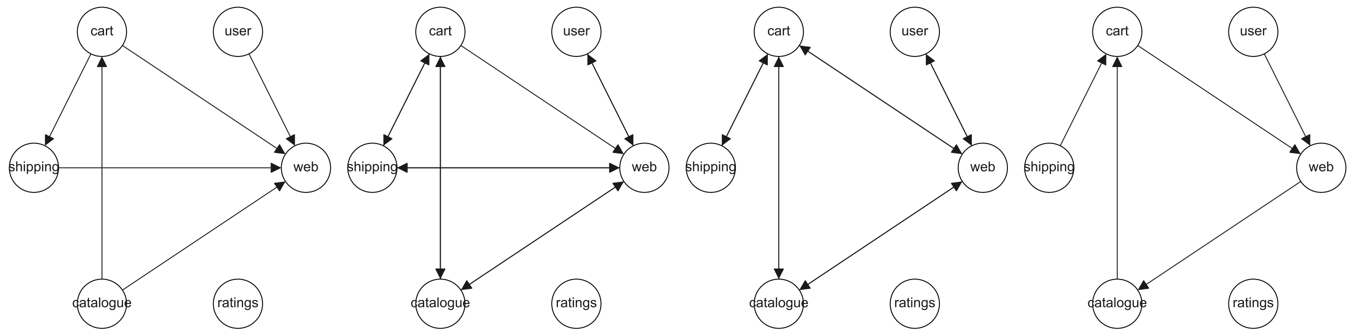
A. Causal Latency Graph Discovery

a) Setup: For the causal latency graph discovery of $\mathcal{G}_L$, we compare the performance of three causal discovery algorithms: PCMCI$^+$, PC, and FCI, using accuracy, recall, F1, and structural Hamming distance (SHD) as validation metrics. As the ground truth, we set the reversed call graph based on the known topology of the Robot Shop deployment as an approximation of the latency graph. This approximation may not represent the true latency graph since some function calls could be asynchronous.

For PCMCI$^+$, we set the maximum time lag $\tau = 3$, determined from autocorrelation plots showing that correlation degrades at this lag. The significance level for the conditional independence test is set to $p_\alpha = 0.01$. We use the linear partial correlation (ParCorr) test due to expected linear dependencies between latency values. The metrics reported are calculated based on the discovered summary graph, as ground-truth data for time-lagged dependencies is unavailable.

We compare PCMCI$^+$ with the PC algorithm to assess whether lagged times series information improves edge discovery and orientation. Additionally, we use the FCI algorithm to examine if accounting for latent confounders, like an external API in some cloud deployments, enhances causal discovery. For both PC and FCI, we use the linear Fisher-Z conditional independence test with $\alpha = 0.01$. All algorithms use A1) as background knowledge, removing any connections between latency values of endpoints of the same host.

b) Results: Table I shows the accuracy, precision, recall, and F1 scores for the three causal discovery algorithms at both the endpoint and microservice levels. At the microservice level, connections are aggregated: if an endpoint in microservice v_i connects to an endpoint in v_j, an edge (m_i, m_j) is added to the graph. For PCMCI$^+$, the discovered summary graph G_{sum} is derived from the results. At the endpoint level, results are compared without post-processing, except for removing edges where endpoints from the same microservice



(a) True Graph (b) PCMCI$^+$ (c) Stable PC (d) FCI

Fig. 4: Results provided by the different Causal Discovery algorithms compared to the ground truth graph.

are connected. Recall is a key metric here, as it measures the proportion of accurately inferred edges. PCMCI$^+$ recovers all edges at the microservice level with reasonable precision. As shown in Fig. 4, incorrect edges are mainly due to uncertain orientation. This is consistent across endpoint graphs, where no spurious influences are found; only the direction of the causes may be misidentified.

The time series data in PCMCI$^+$ appears to help in more accurately orienting edges. At the endpoint level, PC and PCMCI$^+$ have similar recall, while the FCI algorithm has a lower structural Hamming distance (SHD), suggesting fewer edge modifications are needed to achieve the ground truth graph. For step ❸, the graph from PCMCI$^+$ is selected as it includes time-dependent information useful for understanding system dynamics, making it the preferred input for ❸.

Algorithm	Accuracy		Precision		Recall		F1		SHD	
	m	e	m	e	m	e	m	e	m	e
PCMCI$^+$	**0.83**	0.91	0.55	0.42	**1.00**	**0.83**	**0.71**	**0.56**	**5**	8
PC	0.80	0.91	0.50	0.42	**0.83**	**0.83**	0.63	**0.56**	6	8
FCI	**0.83**	**0.93**	**0.60**	**0.50**	0.50	0.50	0.54	0.50	**5**	**6**

TABLE I: Results provided by the different causal discovery algorithms at a microservice level (m) and an endpoint level (e) for ❷ the first step of the causal discovery.

B. *Full Causal Graph Discovery*

a) Setup: In order to construct the complete causal graph $\mathcal{G}$, it is necessary to perform causal discovery for each endpoint to obtain the corresponding G_{Ri}. The resulting subgraph dependencies are then added to the complete causal graph.

We again use the PCMCI$^+$ algorithm for this. The lagged mutual information and autocorrelation plots reveal a time-dependency up to three time steps behind, thus we again set $\tau = 3$. The p-value is set to 0.01, and the Regression conditional independence test [19] is employed, as this test is capable of handling categorical variables, as is the case for the number of pods used in our experiment. Subsequently, background knowledge A1) - A5) is included to constrain causal discovery and reduce the amount of conditional independence testing of the algorithm, as not every variable combination needs to be checked.

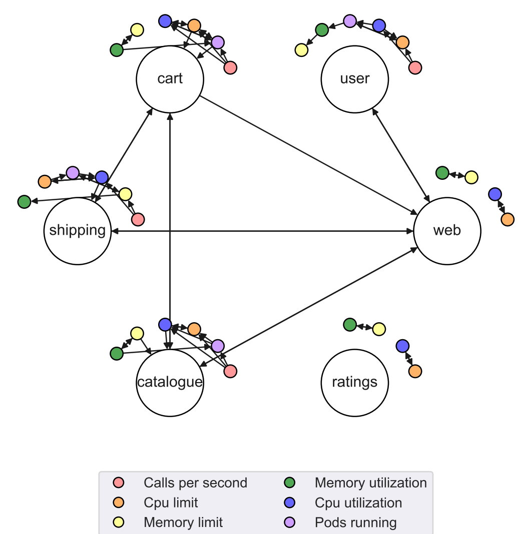


Fig. 5: Discovered causal graph from observational data summarised at the microservice level. White circles represent the microservices and coloured circles represent the relevant resources for each service.

b) Results: Fig. 5 presents the retrieved full causal graph $\mathcal{G}$ of the entire topology at the microservice level, thus providing better visibility. The graph provides a visual representation of the service dependencies. The graph indicates which resources are the primary drivers of latency in a particular service.

The main drivers of latency are identified as the cart, shipping and catalogue microservices. The relevant resources here are the running pods and CPU limit for cart, CPU usage for shipping, and memory limit and CPU usage for catalogue. The latency of the user service is not affected by any of its resources, nor is the web service, which is the entry point for user requests.

	SVR Causal				SVR (ground truth) Causal				Lasso all				XGBoost all			
	τ	d	r^2	MSE	τ	d	r^2	MSE	τ	d	r^2	MSE	τ	d	r^2	MSE
web_user_unique_id	3	1	0.88	31.66	1	1	0.88	**31.3**	3	0	0.82	7579.87	1	-	**0.96**	10.28
web_ship_confirm	2	1	0.61	580.27	0	1	**0.99**	**1.01**	3	1	-82	4522.09	1	-	0.98	32.41
web_catalogue_products	3	1	0.82	21.31	3	1	**0.83**	20.74	3	2	0.57	150.15	1	-	0.84	**18.64**
web_cart_add	3	1	0.92	280.48	2	1	**0.93**	**257.03**	1	0	-3.52	8.11	0	-	0.95	190.94
user_unique_id	3	1	0.92	2.18	0	-	-	-	3	0	0.6	346.57	1	-	**0.98**	**0.67**
shipping_confirm	0	1	0.79	314.82	0	1	**0.98**	**33.01**	3	1	0.77	11.06	1	-	0.47	788.62
catalogue_products	3	2	**0.82**	**1.85**	3	2	-0.19	12.13	3	2	0.2	16222.78	1	-	0.86	1.46
catalogue_product	3	1	**0.96**	27.92	0	-	-	-	1	2	0.8	50.53	1	-	**0.96**	**26.71**
cart_shipping	0	1	**0.94**	**85.84**	1	1	-0.1	1638.4	0	0	-2.01	123382.31	1	-	0.17	1240.63
cart_add	3	1	0.92	122.25	0	1	0.92	**121.37**	3	1	-3.7	46.61	3	-	**0.97**	51

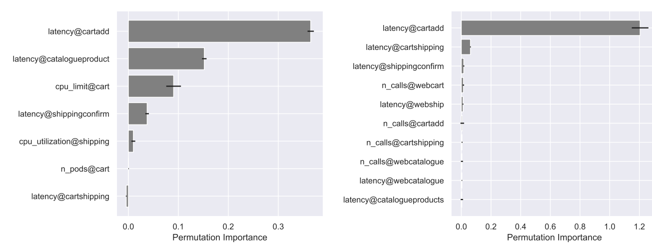
TABLE II: Latency prediction at each endpoint level. We determine the best τ and d value and then report the respective best model. The SVR ground truth model uses the ground truth latency graph based on the reversed call graph with discovered features for $\mathcal{G}_{\mathcal{R}}$. The XGBoost and the Lasso model uses all features (47) the other models use the causal features as described above.

C. Latency Reconstruction Model

a) Setup: The latency reconstruction model serves two purposes: 1. It acts as a proxy for evaluating the second causal discovery step in the absence of ground truth data. 2. It helps identify which variables impact latency, offering insights into which service or resource may be increasing latency at a specific endpoint. To select causal features, we consider depths $d \in 1, 2, 3$, where depth 1 includes second-degree predecessors as features, depth 2 includes third-degree predecessors of an endpoint v_i in the full causal graph $\mathcal{G}$. The optimal depth and maximum time lag (τ) are determined for each endpoint and model, with τ ranging from 0 to 3.

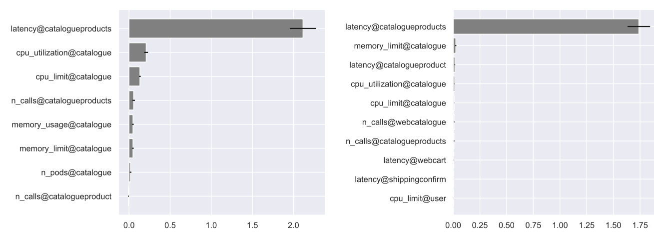
We use a Lasso model [20], a linear model with feature selection, and Support Vector Regression (SVR) model [21] with a radial basis function kernel, thus making it a non-linear model. An XGBoost model [22], known for strong feature selection and capturing non-linear relationships, is used as a benchmark with access to all features except the endpoint under reconstruction. The optimal time lag for XGBoost is also selected. The models are evaluated using the r^2 score, with 1 being the best, and mean squared error (MSE), where a lower value indicates better performance. For the SVR the best hyperparameter are determined over a grid of 1100 different model over a 10-fold cross-validation, the Lasso parameters are also selected using 10-fold cross-validation.

b) Results: Table II shows the metrics for latency reconstruction using either features from the causal graph, and in case of the XGBoost baseline model using all features except the endpoint's own latency. The SVR model, using only causal features, is the only one with an r^2 value over 0.5 for all endpoints. Except for cart_shipping and shipping_confirm, the XGBoost benchmark generally outperforms other models. The Lasso model struggles with complex routes, indicating linear models are insufficient for these cases. Reconstructions are not feasible for the catalogue_product and user_unique_id services with causal ground truth latency graphs due to the absence of causal parents. For cart_shipping, input features like CPU limit and pods running are inadequate for latency reconstruction. Similarly, catalogue_product has only one dependency, and its resources are insufficient for accurate predictions. Lagged data of the latency value at the endpoint level might be needed for this due to observed autocorrelation of latency.



(a) SVR Causal Features (b) XGBoost all Features

(a), (b) Comparison for web_cart_add endpoint

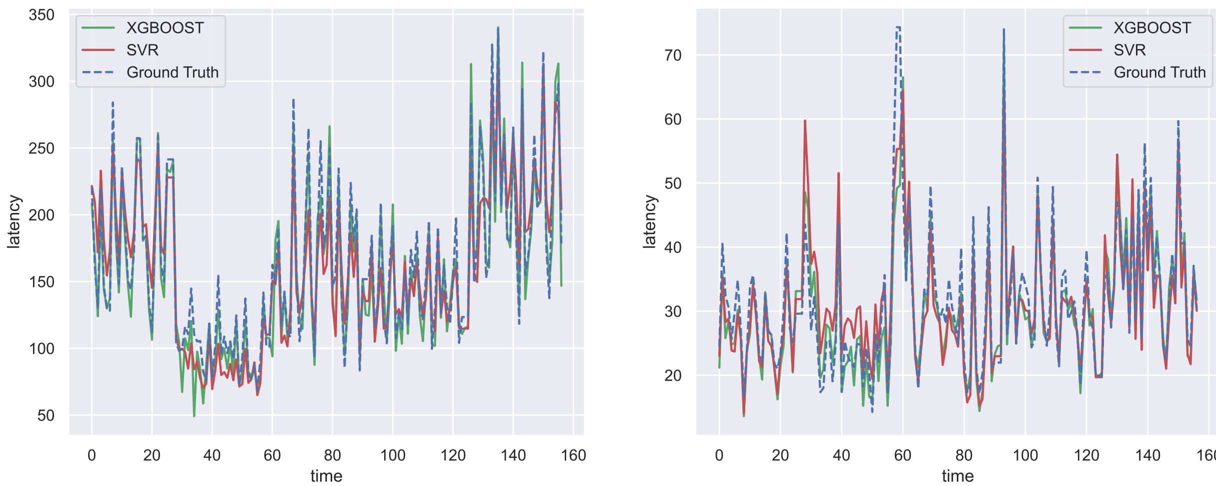


(c) SVR Causal Features (d) XGBoost all Features

(c), (d) Comparison for web_catalogue_product endpoint

Fig. 6: Comparison of permutation importance based on r^2 of XGBoost model fitted on all features and SVR model fitted on causal dependencies for a $d = 1$.

Overall, predictions using the inferred latency graph are more consistent than those using the ground truth latency graph, suggesting that the reversed call graph may not fully capture the underlying dynamics. Fig. 6 illustrates that achieving the highest performance is less critical than obtaining actionable estimators. In the SVR model, features like



(a) Web_cart_add endpoint

(b) Web_catalogue_product endpoint

Fig. 7: Comparison between the predictions of SVR with causal features and XGBoost model with all features. Latency is reported in milliseconds.

cpulimit@cart and cpu-utilisation@shipping are actionable, whereas XGBoost uses features like latency and calls per second, which cannot be directly intervened upon. Thus, a model using all features might achieve better predictions but does not always provide actionable insights for reducing latency.

As illustrated in Fig. 7, both the SVR with causal features and the XGBoost model are able to reconstruct the latency at the selected endpoint, which also generalises to other endpoints. While the SVR model generally underperforms compared to XGBoost, the focus is not on performance alone but on demonstrating that latency reconstruction is feasible using causal features.

The good result for the SVR with causal features shows that we can estimate latency values l_i based on causal predecessors, suggesting that the proposed latency model in (1) holds empirically. We further investigate the contributing features to the prediction with permutation importance, while for example the web_cart_add endpoint has meaningful features contributing to the SVR predictions, such as cpulimit@cart and cpu-utilisation@shipping as well as latency@cartadd. For the model without causal features solely features which are not intervenable such as latency and calls per second have high permutation score. Additionally, some latency values for endpoints which according to the ground truth should not influence the latency at the web_cart_add are determined as important features using permutation importance. We observe similar behaviour for all other endpoints.

The results were obtained on an M3-Max MacBook with 64GB RAM, with 12 performance cores at up to 4.06 GHz and 4 efficiency cores at 2.8 GHz. The first causal discovery takes 2 seconds, the second takes 37 seconds, and the reconstruction model takes 3 seconds due to hyperparameter tuning. The overall evaluation process, including hyperparameter choice and multiple model training, takes 60 minutes.

VI. Deployment Considerations and Discussion

A. Deployment Considerations

The proposed algorithm offers several potential advantages in scenarios of practical deployment. Firstly, the algorithm operates effectively due to the integration of domain knowledge constraints and the assumption of linear relationships among variables. This efficiency allows for rapid analysis, even in complex systems. Secondly, interesting insights can be derived from observational data alone, thus avoiding the necessity for costly and potentially inefficient interventions on a deployed system. Thirdly, the approach provides a graphical representation of the primary drivers of latency, thereby offering an intuitive visualisation and understanding of the causal relationships. Furthermore, the model can be executed in near real time, allowing for the adaptation of causal dependencies as the system evolves. Moreover, by leveraging the mediator assumption, the model identifies actionable predictors that can be targeted for intervention. These predictors are based on factors that can be directly influenced, in contrast to metrics such as calls per second or latencies, which are impractical to set to zero or any fixed value in a real-world application.

As previously stated, the total execution time for the provided example with 48 variables is quick, less than a minute on a laptop, and the number of computational complexities increases by no more than the square in the worst case, which is improbable given the extensive background knowledge. Consequently, the approach should be applicable to significantly larger use cases.

B. Discussion

This paper shows how to causally model cloud microservice applications. We do so by showing that we can reconstruct the topology of the deployment with known causal discovery algorithms. Furthermore, we empirically find meaningful contributors to high latency, which we evaluate with the reconstruction model due to the absence of ground truth data. This is all done purely on observational data without the need of potential costly interventions. However, access to interventional data could potentially strengthen the evaluation process if used in addition to observational data, leading to more accurate causal graphs. Furthermore, the full strength of using causal features to reconstruct latency lies in being robust in the presence of distribution shifts such as in case of an anomaly in the system. Thus, comparing the results of the reconstruction model with and without causal features on anomalous data could be a sensible set of experiments to include. Nevertheless we believe that our approach can help to gather interesting insights that can be derived from observational data alone, thus avoiding the necessity for costly and potentially inefficient interventions on a deployed system. Therefore, making it applicable in practical industry applications.

VII. Conclusions

We develop a latency modelling approach inspired by causality, with the objective of gaining insight into the underlying cause-and-effect relationships that influence latency within a microservice-based architecture. By utilizing causal discovery techniques, our methodology aims to find the elements that contribute to high latency. Furthermore, we propose a novel causal discovery framework that efficiently reconstructs latency graphs with high accuracy at the microservice level based on background domain knowledge. The results of our experiments demonstrate that the causally derived features achieve r^2-scores above 0.5 for all reconstructed latency on endpoints. This suggests that the features obtained through our causal discovery framework are effective in reconstructing latency, thereby demonstrating their efficiency in identifying contributing factors to performance degradation in cloud-native applications. Finally, due to the extensive assumption and background knowledge used for the causal discovery, our approach recovers the causal graph in a relatively short time suggesting that the approach could also work for large scale applications. Potential future work could build upon the proposed causal latency model to estimate the full causal graph of a microservice application while finding the best ways to automatically scale cloud resources by using causal reinforcement learning.

Acknowledgment

This work was partially supported by the European Union's Horizon Research and Innovation program under AC3 project and grant agreement No 101093129.

References

[1] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE, 2016, pp. 44–51.

[2] Y. Abgaz, A. McCarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley, and P. Clarke, "Decomposition of monolith applications into microservices architectures: A systematic review," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4213–4242, 2023.

[3] M. Straesser, J. Grohmann, J. von Kistowski, S. Eismann, A. Bauer, and S. Kounev, "Why is it not solved yet? challenges for production-ready autoscaling," in *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*, 2022, pp. 105–115.

[4] Y. Zhang, W. Hua, Z. Zhou, E. Suh, C. Delimitrou, and WeizheHua, "Sinan: Ml-based and qos-aware resource management for cloud microservices," *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232118940

[5] Y. Zhang, R. Isaacs, Y. Yue, J. Yang, L. Zhang, and Y. Vigfusson, "Latenseer: Causal modeling of end-to-end latency distributions by harnessing distributed tracing," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, 2023, pp. 502–519.

[6] D. S. H. Tam, Y. Liu, H. Xu, S. Xie, and W. C. Lau, "Pert-gnn: Latency prediction for microservice-based cloud-native applications via graph neural networks," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2155–2165. [Online]. Available: https://doi.org/10.1145/3580305.3599465

[7] J. Park, B. Choi, C. Lee, and D. Han, "Graf: a graph neural network based proactive resource allocation framework for slo-oriented microservices," *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:244841439

[8] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, "Root Cause Analysis of Failures in Microservices through Causal Discovery," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 158–31 170, Dec. 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/c9fcd02e6445c7dfbad6986abee53d0d-Abstract-Conference.html

[9] K. Budhathoki, L. Minorics, P. Bloebaum, and D. Janzing, "Causal structure-based root cause analysis of outliers," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 2357–2369. [Online]. Available: https://proceedings.mlr.press/v162/budhathoki22a.html

[10] S. Jha, J. Rios, N. Abe, F. Bagehorn, and L. Shwartz, "Fault localization using interventional causal learning for cloud-native applications," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2024, pp. 141–147.

[11] B. Żurkowski and K. Zieliński, "Root cause analysis for cloud-native applications," *IEEE Transactions on Cloud Computing*, 2024.

[12] T. Tournaire, Y. Jin, A. Aghasaryan, H. Castel-Taleb, and E. Hyon, "Factored Reinforcement Learning for Auto-scaling in Tandem Queues," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2022, pp. 1–7, iSSN: 2374-9709. [Online]. Available: https://ieeexplore.ieee.org/document/9789809

[13] Q. Wang, L. Shwartz, G. Y. Grabarnik, V. Arya, and K. Shanmugam, "Detecting causal structure on cloud application microservices using granger causality models," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 558–565.

[14] J. Pearl, *Causality*. Cambridge university press, 2009.

[15] ——, "Causal diagrams for empirical research," *Biometrika*, vol. 82, no. 4, pp. 669–688, 1995.

[16] P. Spirtes and K. Zhang, "Causal discovery and inference: concepts and recent methodological advances," in *Applied informatics*, vol. 3. Springer, 2016, pp. 1–28.

[17] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.

[18] J. Runge, “Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets,” in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 1388–1397.

[19] M. Tsagris, G. Borboudakis, V. Lagani, and I. Tsamardinos, “Constraint-based causal discovery with mixed data,” *International journal of data science and analytics*, vol. 6, pp. 19–30, 2018.

[20] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

[21] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, “Support vector regression machines,” *Advances in neural information processing systems*, vol. 9, 1996.

[22] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.