Tritium: A Cross-layer Analytics System for Enhancing Microservice Rollouts in the Cloud

Sadie Allen
Boston University

Mert Toslali
Boston University

Srinivasan Parthasarathy
IBM Thomas J Watson Research Center

Fabio Oliveira
IBM Thomas J Watson Research Center

Ayse K. Coskun
Boston University

ABSTRACT

Microservice architectures are widely used in cloud-native applications as their modularity allows for independent development and deployment of components. With the many complex interactions occurring in between components, it is difficult to determine the effects of a particular microservice rollout. Site Reliability Engineers must be able to determine with confidence whether a new rollout is at fault for a concurrent or subsequent performance problem in the system so they can quickly mitigate the issue. We present Tritium, a cross-layer analytics system that synthesizes several types of data to suggest possible causes for Service Level Objective (SLO) violations in microservice applications. It uses event data to identify new version rollouts, tracing data to build a topology graph for the cluster and determine services potentially affected by the rollout, and causal impact analysis applied to metric time-series to determine if the rollout is at fault. Tritium works based on the principle that if a rollout is not responsible for a change in an upstream or neighboring SLO metric, then the rollout's telemetry data will do a poor job predicting the behavior of that SLO metric. In this paper, we experimentally demonstrate that Tritium can accurately attribute SLO violations to downstream rollouts and outline the steps necessary to fully realize Tritium.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; • **Computer systems organization → Cloud computing**.

KEYWORDS

Fault diagnosis, container systems, microservices, version rollouts

ACM Reference Format:
Sadie Allen, Mert Toslali, Srinivasan Parthasarathy, Fabio Oliveira, and Ayse K. Coskun. 2021. Tritium: A Cross-layer Analytics System for Enhancing Microservice Rollouts in the Cloud. In *Proceedings of WoC '21: Workshop on Container Technologies and Container Clouds (WoC '21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3493649.3493656

WoC '21, December 6, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-9171-9/21/12…$15.00
https://doi.org/10.1145/3493649.3493656

1 INTRODUCTION

Microservice applications have complex and dynamic interactions and runtime environments, and this complexity makes it hard to reproduce or diagnose failures in a testing environment. Faults or performance anomalies could be the result of improper cluster configuration, asynchronous service interactions, differences between multiple instances of the same service, actual source code of a service, or countless other issues [11]. One concern for Site Reliability Engineers (SREs) is managing new service rollouts, which constantly happen due to the practice of continuous integration and deployment [4]. These rollouts do not occur in isolation; varying request volume, resource and load fluctuations, and countless other events can happen at or near the same time, making it difficult to determine if the rollout caused a significant change in the system, or if it was due to one of these sources of noise.

1.1 Related Work

Fault diagnosis in microservice systems has already been gaining attention; there are numerous recent works in this space [3, 5, 7, 16, 19, 22–25]. Many past efforts solve a piece of the problem of fault diagnosis, but do not provide a comprehensive picture of the activities in a microservice application. In addition, no prior existing works target rollout-specific fault diagnosis. In this section, we briefly discuss some of the most relevant works and their drawbacks.

In Qiu et al.'s resource management framework FIRM[18], they implement a localization algorithm to identify the microservice at fault for an end-to-end SLO violation. Their algorithm first identifies critical paths (paths of maximal duration starting with client requests) and then uses a binary incremental SVM classifier to decide whether each service in the critical path may be a candidate for being at fault for the SLO violation. This localization algorithm requires training on artificially injected performance anomalies prior to implementation on a system, and its reliance on critical paths means it is only applicable to SLOs related to request latency.

Guo et al. developed a system called Graph-based Microservice Trace Analysis (GMTA). This system abstracts traces into "paths" representing business flows and uses these trace aggregates to aid in visualizing service dependencies and diagnose problems in the system by indicating anomalous traces [7]. While GMTA does provide efficient and flexible storage and access to trace data at several granularities, it is primarily a data storage and visualization tool. It can aid in human understanding of system architecture and problem diagnosis, but lacks any automated detection or pinpointing of issues.

Some prior approaches aim to make use of more than one type of data from the application. Luo et al. proposed a fault diagnosis approach that leverages event and time-series data [15]. Their goal

was to evaluate the correlation between events and time-series to identify when specific events consistently lead to issues in the system. They modeled their correlative analysis as a two-sample hypothesis problem. The strength of their work is the utilization of heterogeneous data types for fault diagnosis. However, the system must be trained on a relatively large data set to obtain statistically significant results, and does not utilize information about the system architecture to focus on the most relevant data for a given problem.

Another approach by Shan et al. focuses on diagnosing the root cause of small window long-tail (SWLT) latency (i.e., tail latency within a one minute or smaller window [21]). They proposed an unsupervised root cause analysis methodology that uses threshold-based detection to detect anomalous latency levels. A two-sample test algorithm is used to identify significantly changed time-series metrics, and these are returned as potential root causes. This approach has several limitations. First, it is only designed to work with the specific detection and diagnosis of SWLT latency. Their approach does not take into account the system architecture and topology, so every metric is tested as a potential root cause (even metrics for containers that are upstream[1] from the detected SWLT latency and on a different node). Finally, this analysis is correlative; their approach does not causally link the changed time-series to the long-tail latency, so the two could happen simultaneously by coincidence.

A few others have strived to demonstrate causal relationships between SLO violations and their sources. Qiu et al. aimed to find the root cause *metrics* using a causality graph and rank the top k possible causes [19]. Their approach's drawback is that the only potential root causes are key performance indicators. SLOs violations cannot be attributed to events, whereas in many cases, SREs would need to know about events to remedy the problem. Only attributing SLO violations to metrics adds another step for SREs as they must determine what aspect of the metric changed and how the change affects the SLO.

1.2 The Vision for Tritium

We propose Tritium to provide a way for SREs to visualize their cluster as well as determine causal relationships between rollouts and SLO violations. At a high level, the steps of our diagnosis system are as follows: (1) identify SLO violations using *change point detection* methods, (2) monitor cluster events (using a container-orchestration platform such as Kubernetes [12]) to identify new version rollouts, (3) use topology graph gleaned from traces (e.g., Jaeger) to determine if a given SLO violation is in a rollout's "area of effect", and (4) perform statistical analysis on metric data to determine if the rollout is at fault. See Figure 1 for a high level diagram of our system.

Tritium works based on the founding insight that if a rollout is responsible for a change in an upstream or physically neighboring SLO metric, that change will also reflect in the telemetry data (latency, error rate, memory/CPU usage, etc.) of the rollout itself. Thus, the rollout's telemetry data should serve as a reasonably accurate predictor of the SLO. However, if other changes in the system are responsible, then the rollout's telemetry data will do a poor job predicting the behavior of the SLO metric during the anomalous period. Our specific contributions with the proposal of Tritium are as follows:

1. We identify key challenges in designing a system to attribute SLO violations to specific system changes (i.e., microservice rollouts).
2. We present Tritium, a prototype system integrating many data types to confidently attribute SLO violations to specific system changes. Tritium introduces causal impact analysis [2] to the application of fault diagnosis in microservice applications. We also have prototyped a GUI with metric, event, and topology data views.
3. We demonstrate the efficacy of our approach for Tritium when identifying if a service rollout is problematic in Train Ticket [13], a benchmark with 41 microservices.

2 DESIGN CHALLENGES

Tritium's main goal is to determine if SLO violations are caused by rollouts. However, it also intends to provide SREs with as much additional information about SLO violations as possible and enable them to visualize and narrow down on issues in their cluster. We address the following fault diagnosis design challenges with our proposed solution.

Cross-layer data utilization: Tritium uses metric data, event data, and topology data gleaned from traces to establish a comprehensive picture of a cluster. In particular, we can vastly reduce our search space by determining the "area of effect" of a given SLO, which consists of metric/event data in upstream services or services co-located on the same node.

Narrowing down useful metric data: Prometheus [17], the monitoring software we use to obtain metric data from our cluster, collects hundreds of different metrics for each service, container, and node every few seconds. Not all of these metrics are useful in predicting a given SLO. Thus, we correlate all of these metrics with the specific SLO to determine those that best reflect its behavior in a normal, healthy state. This narrows our search space to focus on the most likely metrics to help in diagnosis.

Rollout vs. not rollout: The primary goal of our system is to establish whether or not a new rollout is at fault for some upstream or neighboring SLO violation. This is important because it can tell SREs if the problem is based in the source code of the new service version or it is a resource management/architectural issue.

Solution robustness: Another benefit to our approach is that it can be applied generally to any SLO. Although we perform experiments primarily with latency as the SLO, a user could pick any metric reported by monitoring database (e.g., Prometheus) and determine if a rollout affects it.

Establishing causation: It is essential not to just establish correlation between a rollout event and a change in an SLO. To make the correct decisions around rollouts and resource management, causal relationships are necessary. Thus, we attempt to causally link issues in SLOs with potential causes (rollout events) in this work.

Additional insights when a rollout is not at fault: If one finds that a new rollout is not at fault for an SLO violation, it is beneficial to provide as much information to SREs as possible to aid them in fault diagnosis by identifying the best predictor metrics of the anomalous SLO. These metrics give SREs a starting point to continue their own diagnosis of the issue.

Automated prediction testing: Once an SLO change is attributed to a rollout, this hypothesis can be tested by an automated rollback. If

[1] Service A is "upstream" of service B if service A depends on service B to complete its task (and service B is "downstream" of service A).

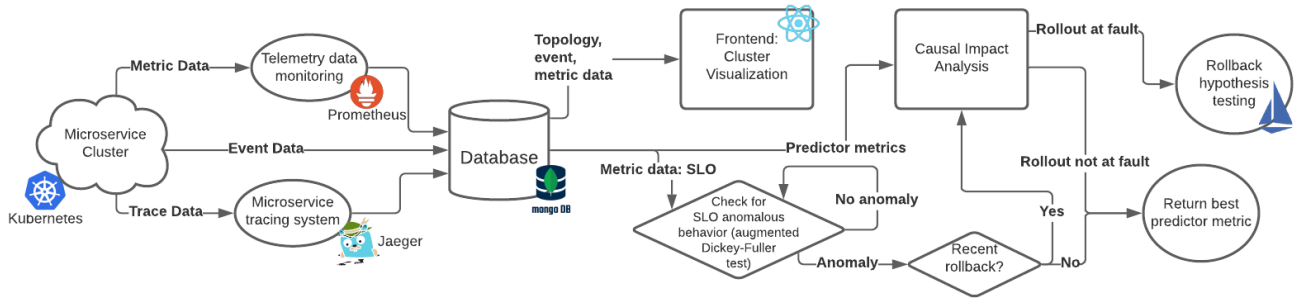

Figure 1: An overview of Tritium. Various types of data are collected and stored in a database. The data is sent to our interactive front-end which renders topology, event, and metric data views. SLO metric data is also sent to an anomaly detector. If anomalous behavior is detected AND there was recently a new rollout, causal impact analysis is triggered to determine if the rollout is at fault for the SLO violation. Otherwise, metrics most highly correlated with the SLO during the anomalous behavior are returned.

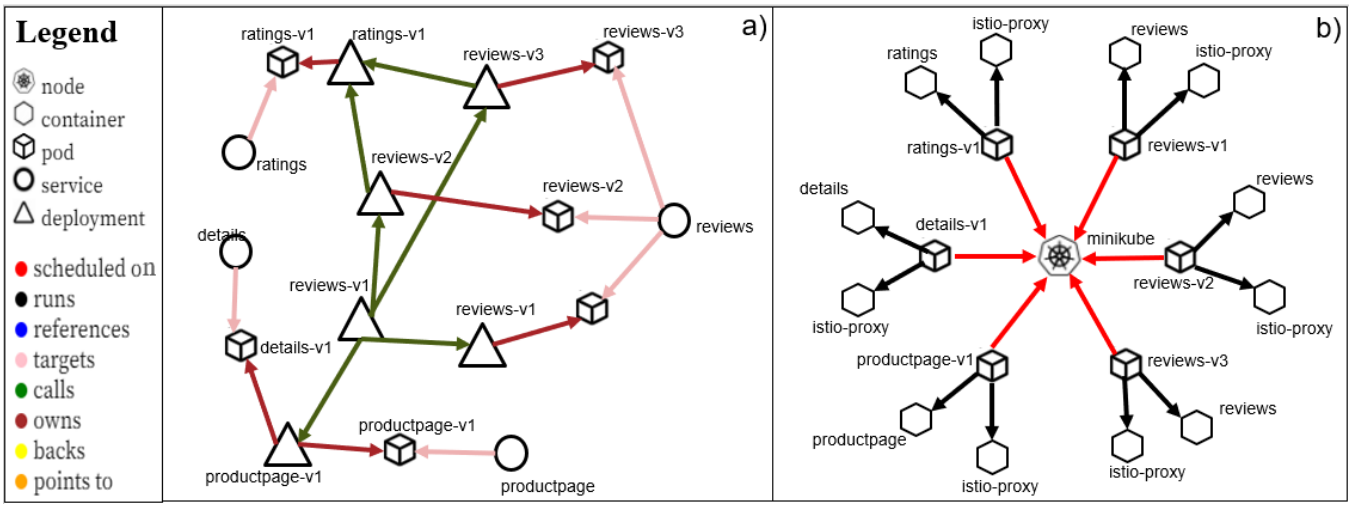

Figure 2: Two time-point topology views of the Bookinfo [1] application. These views are components of the cluster visualization tool. We show two different views of the same cluster at the same point in time to highlight the way in which users could filter the view to focus on different relationships: a) Call-graph view showing services, deployments, and pods; b) A topology view showing nodes, pods, and containers.

the SLO returns to normal, this confirms the prediction. Otherwise, our prediction may be wrong, so that the service can be rolled out again.

3 SYSTEM VISION

We implement a prototype of Tritium's fault diagnosis and visualization system using Kubernetes, Istio [8], Prometheus, and Jaeger [9]. In practice, Tritium will operate alongside a microservice application, collecting data and performing analyses. Users will have control over the data they see and the SLOs they care about.

Fault Diagnosis: The inputs to our fault diagnosis system are event and architecture data obtained from the Kubernetes API, trace data from Jaeger, metric data from Prometheus, and a list of SLOs, which is provided by the user. Two things must happen in order to trigger Tritium's analysis. First, a new microservice rollout must occur, and second, there must be a significant change in an SLO. The system then works as follows:

1. Identify best predictor variables for the anomalous SLO through correlative analysis (i.e., Pearson). This analysis is performed prior to the anomalous behavior when the metric in question is behaving healthily.
2. Predict behavior of that SLO during the anomalous period using the identified predictor variables and the causal impact algorithm [2].
3. Evaluate causal impact results: (a) If the behavior prediction is accurate without using downstream rollout metrics as predictors, the rollout is not to blame. Determine the best predictor metric, providing more guidance to SREs about where their problem might be (e.g., via R^2 or systematic checking via CI). (b) Otherwise, add/use

telemetry data from the new rollout service as predictors for causal impact. If predictions are now accurate/significantly improve with data from new rollout, the rollout likely caused the issue.

Cluster Visualization: The cluster visualization component of Tritium complements the fault diagnosis and provides additional diagnostic information for SREs. It consists of the following data views:

1. Timestamped Topology View: This view will show a graphical representation similar to that of Kiali [10] of the cluster's topology, with vertices representing nodes, pods[2], containers, services, etc. and different edges representing the relationships between them (scheduled on, runs, targets, owns, etc.). In the snapshot view, it will display the state of the cluster at a given time. In the delta view, it will indicate the topological differences in the cluster between two given time points (e.g., the addition or removal of any pods or containers). These views will allow SREs to see how the cluster changes when something goes wrong. See Figure 2 for examples of this view.
2. Time-series View: This view is similar to that provided by Prometheus, allowing SREs to observe all telemetry data. It is useful for seeing changes in the system state not reflected in the topological data.
3. Event View: This view provides a timeline of Kubernetes events. It is primarily useful when used in conjunction with the topology view or the time-series view, so SREs can see when topology changes or metric spikes co-occur with events.

We envision that a master time selector will control the time intervals for these views. All three will have a multitude of filtering options (time filtering, event type filtering, vertex type filtering, time-series filtering, etc.) to hone in on the exact layer or area of the cluster SREs want to know more about. We plan to add visual indicators of fault diagnosis hypothesis into these views (highlighting vertices, specific metrics, etc.) and surface the hypotheses themselves as part of the UI. For example, a hypothesis might be: the rollout in service X at time t caused the increased latency of upstream service Y. Together, Tritium's fault diagnosis and visualization tools will form an extremely powerful system for both fault diagnosis and general cluster understanding.

4 TRITIUM USE CASES

This section outlines a few specific use cases to demonstrate the utility of Tritium. Tritium can be configured to provide information about any SLO. In our experiments, we focus on request latency, which is an important operational metric. When the request latency of a front-end service increases, many downstream services could be responsible for this increase. Tritium could be used in this scenario to identify the downstream service at fault. The topology data collected from Jaeger traces allows Tritium to hone in on the downstream services that could potentially be causing the issue. Then, Tritium can use the metric and event data to run causal impact analysis and determine if changes in downstream latency explain the latency spike in the upstream service.

Another situation in which Tritium could be useful is if a service experiences latency increase due to memory or CPU pressure on its node. In this situation, it is not necessarily a downstream service that would be at fault. If a service whose pods are co-located on the service of interest's node experiences increased memory/CPU consumption, this could starve the service of resources, causing increased latency. Again, this is where the topology information collected by Tritium becomes advantageous. In addition to knowing which services are downstream of the service of interest, the data collected from the Kubernetes API includes physical relationships between nodes, pods, and services. This information tells Tritium which services are co-located on the same machine. When an SLO violation is detected, the metric data from these services are added to the search space for fault diagnosis. We can then see if a rollout of a co-located service causes an increase in memory or CPU usage that is related to the latency increase of the service of interest.

[2]A pod is group of one or more containers; it is the smallest deployable unit of computing that one can create and manage in Kubernetes.

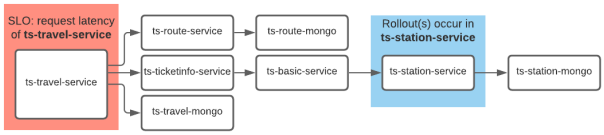

Figure 3: A subset of the call graph for the Train Ticket application relevant to our experiments. Our SLO is the latency of the upstream service *ts-travel-service*, and we roll out new versions of *ts-station-service*.

5 DESIGN OF TRITIUM

In this section, we provide more details on several of the technical design components of Tritium.

Detecting Change Points in Time-series Metrics: Tritium's fault diagnosis system does not trigger unless anomalous behavior is detected in an SLO. We have looked into several methods to detect major shifts in time-series behavior. A threshold-based method is undesirable because SREs may not have a specific threshold in mind that would be unacceptable for a metric to pass. Also, in situations where a metric spikes, but is not quite above the threshold, SREs may still want to determine the cause of the spike. Instead of determining individual thresholds for each SLO, we use the augmented Dickey-Fuller test. Alternatives non-threshold methods such as Pelt search or the binary segmentation method could also be used.

Determining Predictor Metrics: Prometheus collects hundreds of different metrics. We need to devise a method to determine the best predictor metrics for a given SLO in a healthy system state. We use Pearson correlation to this end. Given a particular metric, we start with all metrics collected from the service/pod/container related to said metric. From this list, we calculate the Pearson correlation of the SLO with each. We then use all the metrics that have a Pearson correlation > 0.60, a cutoff that indicates a significant relationship between the time series. This process tends to yield 10-20 metrics (see section 6 for examples of correlated metrics). The benefits of this methodology are: (1) it works for any SLO, (2) one could then selectively collect highly correlated metrics to save on running time, and (3) one can tune it to obtain a certain number of metrics by raising or lowering the correlation threshold or taking the top k correlated metrics.

Causal Impact Analysis: To infer causal relationships, we utilize Google's causal impact algorithm [2]. While this algorithm was originally designed for applications in econometrics (inferring the causal impact of market interventions on outcome metrics such as sales),

Table 1: Varying load schedule for experiment 2

Load	Start time	Duration	# of workers
1	0	20	10
2	3	2	1
3	6	2	3
4	13	2	2
5	16	2	3

we realized it had ample relevance to the fault diagnosis problem. The causal impact algorithm fits a Bayesian structural model on past observed data to make predictions about what future data would look like. This "historical" data is the time-series behavior before some intervention (in our case, a microservice version rollout). The algorithm then compares the counterfactual (predicted) data against what is actually observed to draw statistical conclusions. This algorithm is better suited to the rollout fault diagnosis problem compared to other causal inference methods such as Granger causality[6] because it allows you to specify an event time/change point and analyze the effect of that event as opposed to just modeling one time-series based on another.

Rollback Hypothesis Testing: If the causal impact analysis suggests that a downstream rollout is indeed at fault for an SLO violation, it is desirable to be able to verify this suggestion. We suggest automated rollback as a way to do this. We can use the Istio service mesh to roll back the culprit microservice and then observe the effect of this action on the state of the SLO. If rolling back the new version fixes the issue, it confirms that our rollout candidate caused the problem.

6 EXPERIMENTAL RESULTS

To verify our approach, we performed two experiments on a cluster running the Train Ticket benchmark of microservices. Each experiment acts as proof-of-concept for a different aspect of Tritium's design. In both experiments, we focused on latency of the *ts-travel-service* as our SLO. We correlated hundreds of metrics with the SLO and chose 14 with correlation coefficients >0.6. Examples of highly correlated metrics included *container memory usage* and *request volume*.

Experiment 1. Constant Load, Delayed Service: In this experiment, we establish that our approach can identify a rollout as a cause of increased latency in an upstream service provided the application is under constant load.

We induce a constant load of 10 concurrent workers on the Train Ticket application (get request to the ts-travel-service) for 20 minutes. 10 minutes in, we roll out a new version of ts-station-service in which a 100ms delay has been injected. We then aim to detect the effect of this injected latency on the upstream ts-travel-service latency. The call graph for the subset of the Train Ticket application relevant to our experiments is shown in Figure 3.

Experiment 2. Varying Load, Delayed Service: We establish that our approach can identify rollout as a cause of increased latency in an upstream service provided the application is under varying load.

This experiment is a slightly more challenging version of experiment 1. Like the first experiment, a new rollout is deployed 10 minutes into the experiment. However, throughout the experimental period, the number of workers inducing a load on the application varies according to Table 1 (start time and duration given in minutes).

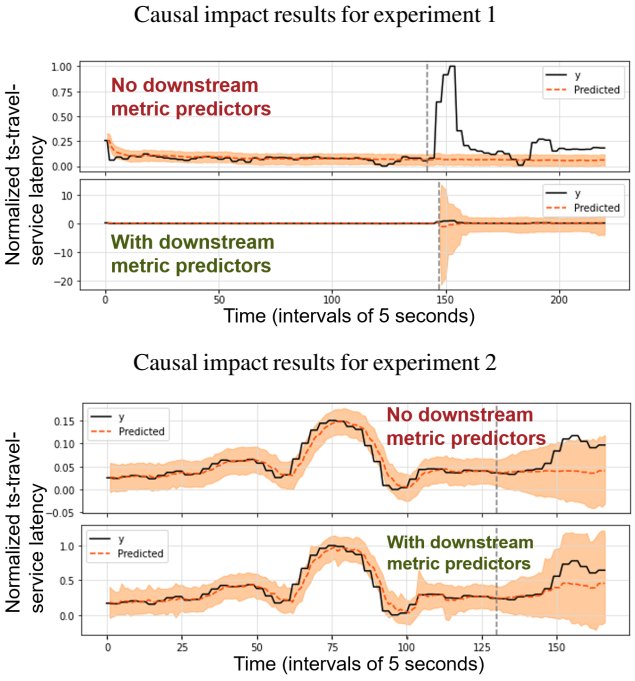

Figure 4: Results from causal impact analysis. The top graphs in each subfigure show predictions when no downstream metric data are included as predictors, and the bottom graphs show predictions with the downstream data.

We performed experiments on a 12-node Kubernetes cluster (each node with 8 cores and 64GB memory) running the Train Ticket benchmark microservice application. We recorded metric and event data during 20 minute periods. 10 minutes in, we rolled out a new version. We then ran causal impact on the period pre- and post-rollout time intervals to determine if the rollout was at fault for increased latency in an upstream service. Figure 4 shows results for both experiments.

In experiment 1, we see Tritium does a poor job predicting the behavior of ts-travel-service latency before we add predictors from the new rollout service. The causal impact package is 100% sure that an external event has influenced the behavior of our SLO. Then, when we provide telemetry data from ts-station-service, we see an improvement in the prediction (the true behavior of ts-travel-service latency is within the margin of error of our prediction). The likelihood that an external event has influenced the upstream latency decreases to just 50.65%. A confidence of 95% is typically desired to say an external event has influenced SLO behavior, so providing the downstream telemetry data puts the confidence far below this level.

A similar result is seen in experiment 2. Initially, Tritium estimates the probability that an external event is responsible for changes in ts-station-service latency as 93.41%. After including data from the downstream rollout, this drops to 74.53%. The noise of the varying request volume makes the difference smaller, but it is still clear that the rollout helps improve the predictions of the upstream service latency.

In these experiments, we see promising results regarding the efficacy of our approach. Even in noisy environments, we are able to

determine that downstream rollouts affect our SLO. We believe these early results are promising indicators for Tritium's feasibility.

7 DISCUSSION AND OPEN QUESTIONS

One open problem in Tritium is determining the exact methodology to construct the topological graph, specifically the call-graph component. The physical relationships can be updated based on event monitoring (e.g., Pod Deleted/Pod Created events), but as the service relationships are collected from traces, which are generated frequently, it is not practical to look at every trace and update the call graph accordingly. Also, in the event where anomalous, potentially fault-indicative paths are observed, should the call graph be updated? One option is to query traces once at system start and then periodically check for changes. Alternatively, Tritium could leverage event monitoring and check for changes in the call graph whenever there is a change in the physical topology.

One may also experiment with the window lengths for causal impact analysis to see if there is an optimal length that yields clearer results. If the windows are too long, we run the risk of additional system noise making the diagnosis problem more difficult, but shorter window lengths may not provide enough data for statistical confidence.

There is also the question of finding the optimal "alternative hypothesis testing" methodology in the event that a rollout is ruled out as the cause of an SLO violation. As previously discussed, the main result from Tritium's fault diagnosis is the answer to the yes/no question: is a given downstream rollout responsible for the change in a SLO? If the answer to this question is no, we want to provide as much information to SREs as possible about what else could be causing the problem. One potential idea is to determine the *best* predictor among the initial predictor metrics. We could use correlative methods to see which of the predictors is most correlated with the SLO during the anomalous behavior, or run several causal impact experiments with one predictor at a time and see which one performs the best. Another option is to use an explainable AI engine such as SHAP [14] or LIME [20] to assist.

There are also open questions surrounding the hypothesis testing via rollback. Is it better to do a partial or complete rollback? Could the policy depend on the confidence of the causal impact results? Another avenue to consider that has been explored in previous work [5] would be auto-scaling of resources (CPU or memory) in response to SLO violations attributed to bottlenecks in those resources. Such a strategy would be a counterpart to automatic rollbacks when a metric is determined to be at fault.

8 CONCLUSION

Making sense of the vast amounts of data generated by microservices to identify causes of performance issues is difficult. We introduce steps toward designing a fault diagnosis and visualization system that utilizes cross-layer data to link SLO violations to rollouts. We demonstrate promising results that Tritium's fault diagnosis system is capable of establishing causal relationships between SLO violations and the downstream rollouts responsible.

Acknowledgments: We would like to thank our reviewers for their feedback on this work. This research is partially supported by IBM Thomas J. Watson Research Center.

REFERENCES

[1] bookinfo [n. d.]. Bookinfo. https://istio.io/latest/docs/examples/ bookinfo/.
[2] Kay H. Brodersen, Fabian Gallusser, Jim Koehler, Nicolas Remy, and Steven L. Scott. 2015. Inferring causal impact using Bayesian structural time-series models. *Annals of Applied Statistics* 9 (2015), 247–274.
[3] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE Conference on Computer Communications (INFOCOM)*. 1887–1895. https://doi.org/10.1109/INFOCOM.2014.6848128
[4] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology* 57 (2015), 21–31.
[5] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. 2019. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 19–33. https://doi.org/10.1145/3297858.3304004
[6] Clive Granger. 1969. Investigating Causal Relations by Econometric Models and Cross-Spectral Methods. *Econometrica* 37, 3 (1969), 424–38. https://EconPapers.repec.org/RePEc:ecm:emetrp:v:37:y:1969:i:3:p:424-38
[7] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis. In *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 1387–1397. https://doi.org/10.1145/3368089.3417066
[8] istio [n. d.]. Istio service mesh. https://istio.io/.
[9] Jaeger [n. d.]. Jaeger: open-source, end-to-end distributed tracing. https://www.jaegertracing.io/. https://www.jaegertracing.io/
[10] kiali [n. d.]. Kiali, Service mesh management for Istio. https://kiali.io/. https://kiali.io/
[11] KubernetesFailures [n. d.]. Kubernetes Failure Stories. https://github.com/hjacobs/kubernetes-failure-stories. https://github.com/hjacobs/kubernetes-failure-stories
[12] KubernetesProduction [n. d.]. Kubernetes: Production-Grade Container Orchestration. https://kubernetes.io/. https://kubernetes.io/
[13] John Langford, Lihong Li, and Alex Strehl. 2007. Train Ticket : A Benchmark Microservice System. http://hunch.net/~vw/
[14] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *31st International Conference on Neural Information Processing Systems (NIPS)*. 4768–4777. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf
[15] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating Events with Time Series for Incident Diagnosis. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[16] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. 2018. Localizing Faults in Cloud Systems. In *IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 262–273.
[17] Prometheus [n. d.]. Prometheus. https://prometheus.io/. https://prometheus.io/
[18] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 805–825. https://www.usenix.org/conference/osdi20/presentation/qiu
[19] Juan Qiu, Qingfeng Du, Kanglin Yin, Shuang-Li Zhang, and Chongshu Qian. 2020. A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications. *Applied Sciences* 10, 6 (2020). https://doi.org/10.3390/app10062166
[20] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *CoRR* abs/1602.04938 (2016). arXiv:1602.04938 http://arxiv.org/abs/1602.04938
[21] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. ?-Diagnosis: Unsupervised and Real-Time Diagnosis of Small- Window Long-Tail Latency in Large-Scale Microservice Platforms. In *the World Wide Web Conference (WWW)*. 3215–3222. https://doi.org/10.1145/3308558.3313653
[22] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: Actionable Insights from Monitored Metrics in Microservices. *CoRR* abs/1709.06686 (2017). arXiv:1709.06686 http://arxiv.org/abs/1709.06686
[23] A. Traeger, I. Deras, and E. Zadok. 2008. DARC: Dynamic Analysis of Root Causes of Latency Distributions. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. 277–288.
[24] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*.
[25] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 683–694.