AutoMAP: Diagnose Your Microservice-based Web Applications Automatically

ABSTRACT

The high complexity and dynamics of the microservice architecture make its application diagnosis extremely challenging. In this study, we design a novel tool, named AutoMAP, which enables dynamic generation of service correlations and automated diagnosis leveraging multiple types of metrics. In AutoMAP, we propose the concept of anomaly behavior graph to describe the correlations between services associated with different types of metrics. Two binary operations, as well as a similarity function on behavior graph are defined to help AutoMAP choose appropriate diagnosis metric in any particular scenario. Following the behavior graph, we design a heuristic investigation algorithm by using forward, self, and backward random walk, with an objective to identify the root cause services. To demonstrate the strengths of AutoMAP, we develop a prototype and evaluate it in both simulated environment and real-work enterprise cloud system. Experimental results clearly indicate that AutoMAP achieves over 90% precision, which significantly outperforms other selected baseline methods. AutoMAP can be quickly deployed in a variety of microservice-based systems without any system knowledge. It also supports introduction of various expert knowledge to improve accuracy.

KEYWORDS

Microservice architecture; web application; anomaly diagnosis; root cause; cloud computing

1 INTRODUCTION

The emergence of microservice architecture promotes easier abstraction and modularity for implementation, reuse, as well as independent scaling of web application development [1]. However, as services and their dependencies are evolving through continuous refactoring, localizing the sources of anomalies in large-scale microservice-based web application is more challenging than ever before [2]. The challenge roots in three aspects:

Dynamic application structure. Due to the various nature of services, static troubleshooting approaches such as thresholding schemes [3] may fail to obtain reliable model applies for frequently changing situations [4]. As a result, recent works typically start with a system structure, and then diagnose anomalies via analyzing patterns following the structure [5][6]. Such structures, e.g. network topology and service-calling dependencies, are generally extracted from historical data collected by monitoring different components, such as the log files, audit events and network data packets. It is time and effort-consuming, even unrealistic in some legacy systems to develop a central running component to collect such data and generate these structures.

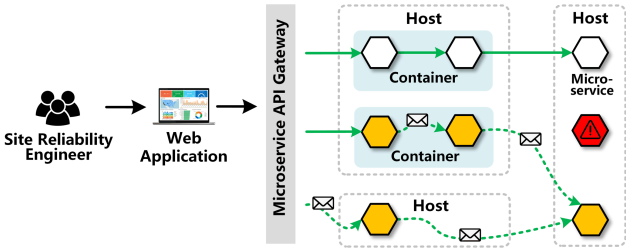


Figure 1: A simple example of anomaly propagation in web application built on microservices. The hexagon represents a service. Red color stands for the root cause, yellow for affected service, arrow for synchronous (solid) or asynchronous request (dashed).

Indirect anomaly propagation. As the component granularity becomes smaller in microservice architecture, services reside on distributed hosts and containers, their calling process could be synchronous as direct calling or asynchronous via message proxy or publish/subscribe component [7]. Therefore, the propagation of anomaly is not bounded by the calling dependency any more. Figure 1 presents an example where a web application invokes services running on different hosts and containers through the API gateway. Even anomaly occurs in a microservice that is not been called, it may affect other services in the same host or container, and cause the anomaly propagation. Therefore, even if we know the calling dependency of services, we lack a more dynamic diagnosis mechanism due to the existence of indirect fault propagation.

Multiple types of metric. Algorithm based on single metric may fail to identify the root cause, as single type of metric is not enough to characterize the anomalies occur in diverse services [8]. In addition, the asynchronous calling procedures makes metrics often unable to directly reflect the propagation dependency. When the application is experiencing anomalies, although modern architectures provide us a wide variety of healthy metrics, we still lack an automated mechanism that selects them appropriately according to the characteristics of involved services.

Consequently, to tackle the anomalies in microservice-based web applications, operation team has to maintain deep domain knowledge of the system. It is extremely challenging to keep updating such knowledge, particularly when the architecture evolves quickly through rapid refactoring of new features.

To address these challenges, we aim to develop an automated diagnostic tool with several capabilities: first, it should be able to generate anomaly topology automatically without prior knowledge; second, it should be able to characterize services and anomalies automatically based on multiple types of metric; third, it should be able to select appropriate metric to conduct root cause

detection automatically. We present our solution - AutoMAP. In summary, our contributions include:

1. We propose the concept of *Anomaly Behavior Graph*. This graph model depicts the correlation between services when the anomaly propagates. In order to reveal the nature of service and anomaly pattern, we define the addition (+) and subtraction (-) operations on behavior graph, and leverage them to generate the profile of service and anomaly.

2. We design a similarity function on anomaly profile, and use it to search the most relevant metric in historical records. An automated metric-weight learning approach and an investigation algorithm are proposed. The algorithm leverages forward, self and backward random walk to identify root cause heuristically.

3. We validate AutoMAP in simulated and real production environment and compare it with the selected baseline approaches. Experimental results show that AutoMAP provides over 90% precision, significantly outperforming other methods. The results also verify that AutoMAP can optimize itself effectively and efficiently.

2 RELATED WORK

In this section, we review the related work on root cause detection in distributed systems. Various research efforts have been devoted to similar issues such as network traffic analysis [9], web application anomaly detection [10] and debugging [11], service failure detection [28] and prediction [29]. For example, Gertler et al. proposed to detect anomalies by examining monitoring data of individual component with a thresholding scheme [6]. Wang et al. proposed a bottleneck detection method that correlates throughput and load with a tiered network system model at fine granularity [7]. However, we find it is difficult to obtain a reliable threshold for various situations in practice, especially in dynamic microservice architecture.

Machine learning techniques, such as decision tree [12], clustering [13, 14] and Markov prediction model [15], have also been leveraged to identify abnormal nodes in networks. There are also efforts focusing on knowledge discovering based on performance metrics [13] and network topology [14]. To this end, a central master node is commonly required to collect records from distributed monitoring facilities [16]. These records include log files, audit events, network traffic statistics, and even sensory measurements in physical systems. Most of these solutions need predefined system topology [17] or service calling relationship [16]. As a result, further efforts explore this issue by automatically discovering system topology [18], and then identify anomalies in a heuristic way [19]. For instance, a structure called "invariant graph" [20] is proposed to depict the anomaly propagation topology. Although, links in static network structure represent a part of the causal relations, the actual relations in microservice architecture are more dynamic.

Kim et al. [23] implements a real-time metric collection system and anomaly detection framework, named MonitorRank. It provides an unsupervised and heuristic way to diagnose root cause services based on random walking strategy. However, MonitorRank also requires preliminary domain knowledge and service calling topology of the target system. For microservice-based systems, obtaining the ground truth calling topology is of high cost, as it is constantly changing. In view of this, CloudRanger [30] and Microscope [31] are proposed to reconstruct the topology based on the statistical characteristics on metrics, which can diagnose system anomaly without obtaining a ground truth calling topology.

Another key issue is how to choose appropriate type of metric in diagnosis. Some works discussed how to deal with multiple types of metrics [21]. For example, NetMedic is proposed to generate dependency relationship graph for small enterprise networks using fault propagation templates [8]. Similarly, Sherlock discovers fault-related inference graphs using multi-level metrics from network monitoring data and system logs [22]. Large-scale microservice architecture generates high-dimension dependency relationships that poses greater challenges to find the actual root causes heuristically and design an adaptive mechanism to find the best metric combination to reflect the features of different microservice. To solve this issue, MS-Rank [32] framework preliminarily proposes a self-adaptive mechanism for dynamically implied-metrics generation and metric-scheme selection based on historical diagnostic records.

However, we notice that existing algorithms do not analyze and utilize the historical features of services and anomalies. In enterprise-level operation and maintenance, two aspects of knowledge play an important role in anomaly diagnosis: experience of historical diagnosis and the characteristics of different services. Therefore, targeting these challenges, the main differences and advantages of this study include: (i) the concept of behavior graph and its calculation method to reflect the statistical characteristics of services and anomalies; (ii) quantification of service/anomaly and similarity function; (iii) automatic metric selection mechanism for specific anomaly scenario; (iv) real-world verification on enterprise cloud system.

3 PROBLEM STATEMENT

3.1 Problem Definition

To generalize the problem, we treat microservice-based web application as a "grey box", which means we only have several types of monitoring metrics, without knowing any system knowledge, such as calling topology and service functionalities.

Table 1: Notations

Notation	Definitions
$G(V, E, W)$	Metric-weighted correlation graph with weight matrix W
$v_{fe}, \boldsymbol{v}_{rc}$	Front-end service; set of root cause services
I_i, O_i	Set of in and out-neighbor nodes of node v_i
n, m, ℓ	Number of services, types of metrics; incident period length
$\mathbb{M}, \mathbf{M}$	$\mathbb{M}_m = [\mathbf{M}_m]_{n\times\ell}$, No.$m$ metric measurement
$\|x\|_0$	Number of non-zero coordinates of x
$\mathbb{C}$, $\mathbf{C}$, $c_{i,j,k}$	$\mathbb{C}_k = [\mathbf{C}_k]_{n\times n}, c_{i,j,k} = [\mathbf{C}_k]_{i,j}$, correlation of v_i to v_j given $\mathbb{M}_k$
$\mathbf{P}$, $p_{i,j}$	$[\mathbf{P}]_{i,j} = p_{i,j}$, transition probability from service v_i to v_j
α	Significance in conditional independence tests
ρ	The strength parameter of backward transition

We formalized the problem. Suppose that an anomaly is observed in a front-end service $v_{fe} \in V$ during the incident period ℓ. Here, $n = |V|$ and m are the number of services and types of metrics, respectively. We denote the raw metrics as $\mathbb{M}$, where $|\mathbb{M}| = m$, $\mathbb{M}_m = [\mathbf{M}_m]_{n\times\ell}$ and $\mathbf{M}_m$ records the measurements of No.m metric for n services during ℓ. Our target is to identify a set of services $\boldsymbol{v_{rc}} \subset V$ that causes the observed anomaly in v_{fe}. Table 1 summarizes the major notations used in this study.

3.2 AutoMAP

To solve this issue, we develop a novel tool, named AutoMAP (Automated Microservice-based Web Applications Patrol), which enables dynamic generation of service correlations and automated diagnosis leveraging multiple types of metrics. Fig. 2 depicts its framework.

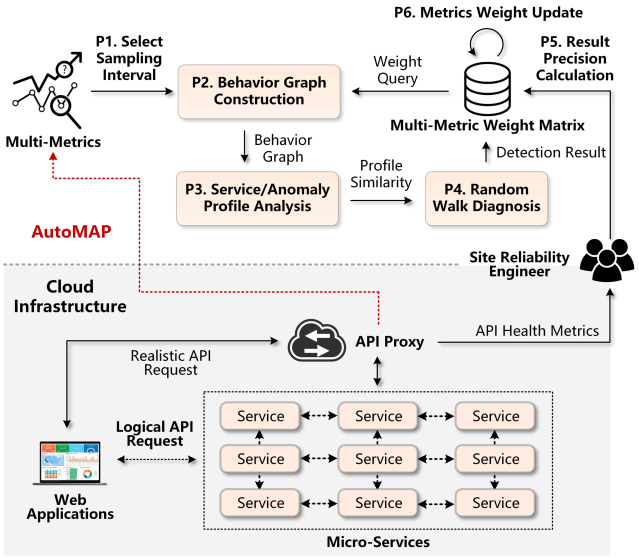


Figure 2: The framework of AutoMAP

AutoMAP collects the details of each microservice request and status of containers/hosts based on an API proxy. The diagnosis starts if an anomaly is detected in front-end service. It decomposes the root cause detection task into several iterative phases:

- **P1.** Select a sampling interval parameter on raw metrics;
- **P2.** Construct the anomaly behavior graph using multiple types of metrics;
- **P3.** Extract the profile of anomaly using "+" and "−" operations on behavior graph;
- **P4.** Conduct heuristic root cause detection algorithm following the behavior graph;
- **P5.** Verify the result and calculate the precision;
- **P6.** Update the metric-weight matrix. Repeat **P1** to **P6** if a new anomaly occurs.

It is worth mentioning that AutoMAP is an automated tool which does not require any system knowledge. Even non-expert users can use it to locate the root cause. For SRE, AutoMAP can provide them a reference diagnosis result to significantly improve the efficiency of system maintenance. We discuss the major tasks of AutoMAP in details in the following sections.

4 METRICS

4.1 Testbed System and Raw Metrics

The testbed system of our tool is a top-ranked cloud platform (due to the double-blind review requirement, we do not mention its name in this version). This platform provides hundreds of categories of microservices. Massive web applications are built on them, running on millions of machines in multiple cloud centers around the world, serving over a million users and produce billion-level API requests every single day.

Table 2: Metrics

Metric	Notation	Definition
Latency	$\mathbf{M}_{lat}$	Average latency of service calling
Throughput	$\mathbf{M}_{thr}$	Average request times per unit time of service
Power	$\mathbf{M}_{con}$	Congestion function, $power = \frac{throughput}{latency}$ [24]
CPU	$\mathbf{M}_{cpu}$	CPU-usage of the located host
I/O	$\mathbf{M}_{io}$	I/O count of the located host
Memory	$\mathbf{M}_{mem}$	Memory consumption % of the located host
Availability	$\mathbf{M}_{avl}$	Service available %

In this study, we consider 7 types of performance metrics: *latency*, *throughput*, *power*, *CPU*, *I/O*, *memory* and *availability*. We use $\mathbb{M} = \mathbf{M}_{\{lat,thr,con,cpu,io,mem,avl\}}$ to denote the records of these metrics (See Table II). Note that these metrics are chosen because they are typical and easy-to-obtain from service API request records. Users can also define and choose any other metric to characterize particular types of anomalies.

4.2 Sampling Interval

AutoMAP starts from choosing an appropriate sampling interval on raw metrics. This parameter directly affects the accuracy of the following diagnosis. If we sample the metric too frequently, every second for example, it may produce redundant fluctuations that do not reflect the actual calling dependency. On the contrary, if we aggregate original sampling using a larger interval, many effective metric changes will be lost, making it fails to catch the propagation topology. Given this fact, we propose a sampling interval selection algorithm for microservice architecture. This parameter depends on the characteristics of system. A reasonable selection is the statistical average calling intervals for all services. In other words, a service is called once on average during this interval. Thus, we compute it using a frequency-weighted average of request interval, that is:

$$\sum_{i=1}^{n} \frac{The\ number\ of\ calling\ v_i}{The\ number\ of\ total\ service\ calling} * (interval\ of\ calling\ v_i),$$

where v_i represents No.i service of V. In real production environment, the propagation path is much more complicated as the anomaly usually affects numerous services directly or indirectly. In most situations, non-expert lacks a priori knowledge about the functionalities of services and the calling relationships between them. Therefore, our work aims to build an automated approach to model the correlations between services using metrics, to help web application developers and site engineers analyze the anomaly.

5 BEHAVIOR GRAPH CONSTRUCTION AND COMPUTATION

5.1 Behavior Graph Construction

As the calling dependencies in microservice architecture are complex and dynamic, instead of spending a lot of time to analyze the service-calling topology of a web application, experienced SRE usually chooses to troubleshoot based on intuition about the type of anomaly. More specific, they select the most suspected service based on observation on its metrics and compare to historical experiences. For example, when many I/O services lose their responses, a problem may occur in database-related services. If the availability of a computational service decreases, its container maybe unstable and we may further trace to the hardware deficiency of host machine. It can be seen that, the intuitive knowledge of SRE mainly includes two aspects: experiences from historical diagnosis and the characteristics of various services. This section proposes a model named *Anomaly Behavior Graph* to extract correlations from metrics and help us discover similar anomalies in records. It is defined formally as follows:

Anomaly Behavior Graph. $G(V, E, W)$ is an anomaly behavior graph describing the impact correlations between vertices (i.e. services) in V, where E is the edge set and W is the weight matrix for edges. Given any service pair v_i and v_j, $W_{i,j,k} \in [0,1]$, $\sum_{k=1}^{m} W_{i,j,k} = 1$. We set edge $e_{ij} \in E$ (from v_i to v_j) to 1 when $\left\|[W]_{i,j}\right\|_0 > 0$ (here $\|x\|_0$ calculates the number of non-zero coordinates of x). An edge $v_i \to v_j$ with a weight $W_{i,j,k} > 0$ indicates that v_i is impacted by v_j given $\mathbf{M}_k$ with a confidence $W_{i,j,k}$.

In order to obtain a behavior graph, we need to start from a complete, undirected and fully-weighted graph, gradually remove the weight of edge and orient the directions of edges using conditional independence test. This process consists of four steps:

- **Step 1.** Generate a complete, undirected and fully weighted graph $G(V, E, W)$, where $W_{i,j,k} = 1$ for $\forall i, j \in [1, n]$ and $\forall \mathbf{M}_k, k \in [1, m]$;
- **Step 2.** For each type of metric $\mathbf{M}_k$, test conditional independence of any pairs v_i, v_j. Set $W_{i,j,k} = 0$ if conditional independence between v_i and v_j is accepted;
- **Step 3.** Remove edge $e_{i,j}$ if $W_{i,j,k} = 0$ for $\forall k \in [1, m]$. Set $W_{i,j,k} \leftarrow W_{i,j,k}/||W_{i,j}||_0$;
- **Step 4.** Orient v-structures and the remaining edges in G.

For the sake of clarity, we summarize this process in Algorithm 1. The result of this algorithm is a weighted-CPDAG (completed partially directed acyclic graph) describing the correlations between services V characterized by $\mathbb{M}$. In Step 2 and 3, we test conditional independence given a significance α, for any service-pair recursively. Particularly, we say v_i and v_j are conditionally independent given v_k if $P(v_i \cap v_j | v_k) = P(v_i|v_k)P(v_j|v_k)$ when $P(v_k) > 0$. It indicates that the occurrence of v_i and the occurrence of v_j are independent events in their conditional probability distribution given v_k. If and only if v_i, v_j are conditionally independent given any subset of S, we call v_i, v_j are separated by S. Let $S(v_i, v_j, k)$ denote the conditional set that separates v_i and v_j given $\mathbf{M}_k$, $adj(G, v_i, k)$ be the set of nodes connected to v_i in G (in other words, any v_j satisfying $W_{i,j,k} = 1$), we test all pairs (v_i, v_j) for conditional independent using $\mathbf{M}_k$, given any single node in $adj(G, v_i, k)\backslash\{v_j\}$ or $adj(G, v_i, k)\backslash\{v_i\}$. If there exists any service v_q let (v_i, v_j) conditionally independent, we set $W_{i,j,k} = 0$ (i.e. remove the edge) and insert v_q into $S(v_i, v_j, k)$ and $S(v_j, v_i, k)$. Once all one-step adjacent pairs and all types of metrics have been tested, a new graph G is generated and we continue in this way by increasing the size of the conditioning set. It stops when all neighborhoods in G are smaller than the size of the conditional set. In this process, $\alpha \in (0,1)$ is the threshold for conditional independence test. When α is approaching to 0, the conditional independence hypothesis is easier to be accepted, thus, more edges will be removed from G. On the contrary, if we set a larger α, more edges will be remained.

Algorithm 1. Anomaly Behavior Graph Construction

Input. Metrics $\mathbb{M}$, Vertex V, separation function S, significance α

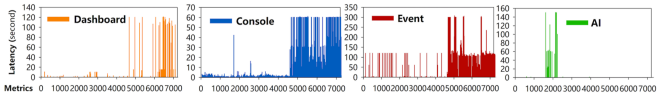
```
new G(V, E, W), level = 0
for ∀(v_i, v_j) ∈ V if |adj(G, v_i, k)\{v_j}| ≥ level
    for ∀v_m ⊂ adj(G, v_i) with |m| = level
        for ∀M_k ∈ M, k ∈ [1, m]
            if v_i, v_j conditionally independent given v_m, α, M_k
                set W_{i,j,k} = 0
                insert v_m into S(v_i, v_j, k) and S(v_j, v_i, k)
            end if
        end for
    end for
    level ← level + 1
end for
for any i, j, k let W_{i,j,k} ← W_{i,j,k}/||W_{i,j}||_0
for ∀v_i − v_j − v_l ∈ G
    if ∃k ∈ [1, m] s.t. v_j ∉ S(v_i, v_l, k) ∧ v_j ∉ S(v_l, v_i, k)
        orient v_i − v_j − v_l into v_i → v_j ← v_l
    end if
end for
repeatedly applying Rule 1-3
```

Output. Reverse any edge direction in G

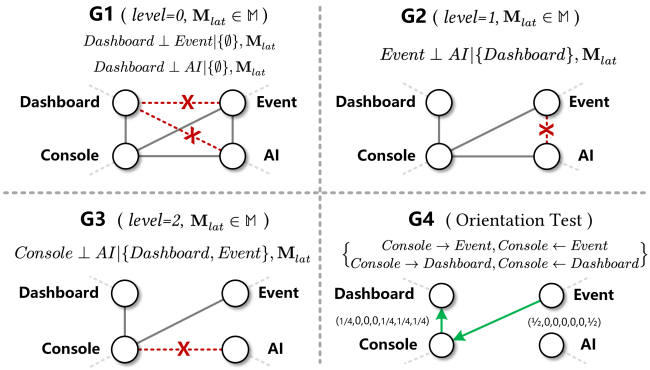
Step 4 orients undirected skeleton into behavior graph. To do this, we search the graph for triple (v_i, v_j, v_l) that satisfies $e_{i,j} = 1, e_{j,l} = 1$ and $e_{i,l} = 0$. All such triples will be oriented as $v_i \to v_j \leftarrow v_l$ (called v-structure) if v_j is neither in $S(v_i, v_l, k)$ nor in $S(v_l, v_i, k)$ for $\forall k \in [1, m]$. After this, for any remaining undirected edge, we repeatedly apply **Rule 1-3** to check whether any of its two possible directions introduces a new v-structure or directed cycle:

- **Rule 1.** Orient edge $v_j - v_l$ as $v_j \to v_l$, whenever there is a directed edge $v_i \to v_j$ such that v_i and v_l are not adjacent (otherwise, a new v-structure will be created);
- **Rule 2.** Orient edge $v_i - v_j$ as $v_i \to v_j$, whenever there is a chain $v_i \to v_l \to v_j$ (otherwise, a directed cycle will be created);
- **Rule 3.** Orient edge $v_i - v_j$ as $v_i \to v_j$, whenever there exist two chains $v_i - v_p \to v_j$ and $v_i - v_l \to v_j$ such that v_p and v_l are not adjacent (otherwise, a new v-structure or a directed cycle will be created).

Algorithm Complexity. The complexity of Algorithm 1 is bounded by the degree of G. Let d be the maximal degree of any vertex in G. In the worst case, the number of conditional independence tests is bounded by $m * 2\binom{n}{d}\sum_{i=0}^{d}\binom{n-1}{i}$. The computational complexity increases exponentially with m and d. The precision and efficiency of this algorithm can be significantly improved by taking into account domain knowledge. More specifically, we can remove edge or specify its direction based on the known service-calling dependencies.



(a) Raw data



(b) Behavior graph Construction

	$\mathbf{M}_{lat}$	$\mathbf{M}_{thr}$	$\mathbf{M}_{con}$	$\mathbf{M}_{cpu}$	$\mathbf{M}_{io}$	$\mathbf{M}_{mem}$	$\mathbf{M}_{avl}$	**Overall Weight**
Event → *Console*	1	0	0	0	1	1	1	$(1/4, 0, 0, 0, 1/4, 1/4, 1/4)$
Console → *Dashboard*	1	0	0	0	0	0	1	$(1/2, 0, 0, 0, 0, 0, 1/2)$

(c) Weight calculation

Figure 3: Example of behavior graph construction

Example. Fig.3 elaborates the details of constructing a demo behavior graph consisting of four microservices using $\mathbf{M}_{lat}$. This incident is classified as "performance downgrade" by SRE team. Cloud application users report that Web UI and command-line interface (CLI) are slow in response. To start the algorithm, we set $level = 0$ in **G1** and test all the service pairs for their conditional independence. As we find $Dashboard \perp Event|\{\emptyset\}$, $Dashboard \perp AI|\{\emptyset\}$. Therefore, we remove edge *Dashboard-Event* and *Dashboard-AI* from **G1**. In **G2**, we test the independence when $level = 1$ and find $Event \perp AI|\{Dashboard\}$. That is, *Event* is conditionally independent with *AI* given *Dashboard*. Therefore, we insert *Dashboard* into $S(Event, AI)$ and remove the edge *Event-AI*. In **G3**, we find *Console* is conditionally independent with *AI* given $\{Event,\ Dashboard\}$ and remove *Console-AI*. The iteration stops when $level = 3$ because $|adj(G, v_i)\backslash\{v_j\}| < level$, $\forall v_i \in G$. After this, we obtain a skeleton "*Event-Console-Dashboard*". In the last step, with the help of connected v-structures, we orient the skeleton by repeatedly rule-checking for any of the edge directions. Finally, we obtain the part of behavior graph using $\mathbf{M}_{lat}$: $Event \rightarrow Console \rightarrow Dashboard$. Likewise, we find $Event \rightarrow Console$ using $\mathbf{M}_{io}$, $\mathbf{M}_{mem}$ and $\mathbf{M}_{avl}$. Edge $Event \rightarrow Console \rightarrow Dashboard$ is detected using $\mathbf{M}_{avl}$. Therefore, according to the weight calculation table shown in Fig. 3, the weight of $Event \rightarrow Console$ is $(\frac{1}{4}, 0,\ 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$, and the weight of $Console \rightarrow Dashboard$ is $(\frac{1}{2}, 0, 0, 0, 0, 0, \frac{1}{2})$.

5.2 Addition Operation and Service Profile

In this section, we show how AutoMAP analyzes the type of services using behavior graph automatically. Recall that our cloud platform provides over 200 categories of microservices, without any prior knowledge, it is a challenging task to classify various services according to their characteristics. A straight-forward way is using multiple historical observations of metrics in normal status. To this end, we generate dozens of behavior graphs using normal metrics, and aggregate them to show the characteristics of service. Thus, we define the addition operation on behavior graph:

Addition operation. Addition is a binary operation on behavior graphs, denoted as "+". Let $G(V, E, W) = G_a(V_a, E_a, W_a)$ $+G_b(V_b, E_b, W_b)$, where $V = V_a \cup V_b$, $E = E_a \cup E_b$ and

$$[W]_{i,j,k} = \frac{\|[W_a]_{i,j}\|_0 * [W_a]_{i,j,k} + \|[W_b]_{i,j}\|_0 * [W_b]_{i,j,k}}{\|[W_a]_{i,j}\|_0 + \|[W_b]_{i,j}\|_0}, i, j \in [1, n], k \in [1, m].$$

Given a set of time period $\boldsymbol{w} = \{w_1, w_2, \cdots, w_k\}$, and a set of behavior graph $\{G_1, G_2, \cdots, G_k\}$ generated using metrics collected during $\boldsymbol{w}$, let $\mathbb{G} = \sum_{i=1}^{k} G_i$ for $\forall v_i \in V$. We call the average of all the weight vectors of edges from v_i to its out-neighbors O_i as the profile of v_i. Thus, $Profile(v_i) = \frac{\mathbf{1}}{|O_i|}\sum_{l \in O_i} W_{i,l}$. This vector indicates the feature of v_i's impact made on others.

In Fig. 4, we present an example to show the details of service profile generation. In this example, we construct several normal behavior graphs including four services, and aggregate them by addition operation. Specifically, in Fig. 4a, we examine all the directed edge from *Dashboard* to its neighbors. By averaging the weight of edges $Dashboard \rightarrow \{Console, Event,\ AI\}$, we obtain the service profile vector of *Dashboard*, i.e. (**0.54**, 0, 0.11, 0, 0.08, 0.08, 0.19). It indicates that the impact of *Dashboard* on other services is dominated by latency ($\mathbf{M}_{lat}$). In other words, latency is the most significant metric that should be used for investigation when it is experiencing anomalies. We call the type of metric with highest weight in profile vector as the dominant metric. Similarly, we find $\mathbf{M}_{con}$ and $\mathbf{M}_{lat}$ are the dominant metrics for service *Event* and *Console*, respectively (See Fig. 4b and Fig. 4c for more details).

Furthermore, we classify services in our testbed system into five typical categories according to their dominant metrics: **R**epresentational, **C**omputing, **N**etworking, **S**torage, **E**nvironmental (denoted as **R**/**C**/**N**/**S**/**E** in the figures for short). For example, for representational services, we are generally concerned about their delays. In our cloud platform, application Dashboard and Console are the most common representational services. Similarly, the performance of storage services is mainly manifested by their I/O. Table III summarizes five categories of typical service profile and lists their representative services in our cloud platform. Note that the purpose of this classification is to facilitate the automated diagnosis of anomalies. Minor inconsistencies between the classification result and actual service functionalities will not affect the effectiveness of AutoMAP.

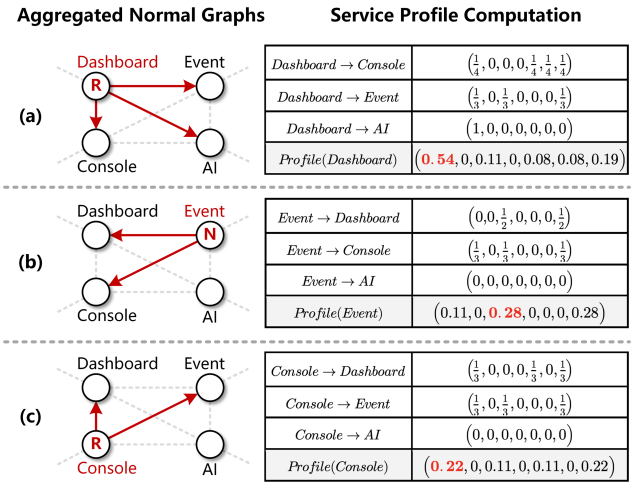


Figure 4: An example of service profile generation

TABLE 3: Typical categories of service profile

Service Profile	Dominant Metric	Typical Services
Representational	$\mathbf{M}_{\{lat,avl\}}$	Dashboard, Console
Computing	$\mathbf{M}_{\{thr,cpu\}}$	Spark, AI, NLP
Networking	$\mathbf{M}_{con}$	Message Hub, IoT center
Storage	$\mathbf{M}_{io}$	Cloudant, MongoDB, Hbase
Environmental	$\mathbf{M}_{mem}$	Container, Docker, Kubernetes

5.3 Subtraction Operation and Anomaly Profile

As a matter of fact, in real-world incidents, only a small number of services contribute to the anomaly propagation. Originally constructed behavior graph may contain redundant services and correlations due to a large part of service's irrelevance with the anomalies. Considering the fact that service profile only describes correlations in normal status, we remove these redundant correlations from the behavior graph. This makes the behavior graph more tailed to the description of anomalies. To do this, we define another operation - subtraction, and the concept of *anomaly profile*, to characterize the feature of root anomalies.

Subtraction operation. Subtraction is a operation on behavior graph, denoted as "−". Let $G(V,E,W) = G_a(V_a,E_a,W_a) - G_b(V_b,E_b,W_b)$, where $V = V_a \cap V_b$, $E = E_a \cap E_b$ and

$$[W]_{i,j,k} = \begin{cases} 0 \ if \ \|[W_a]_{i,j}\|_0 * [W_a]_{i,j,k} < \|[W_b]_{i,j}\|_0 * [W_b]_{i,j,k} \\ \frac{\|[W_a]_{i,j}\|_0 * [W_a]_{i,j,k} - \|[W_b]_{i,j}\|_0 * [W_b]_{i,j,k}}{\|[W]_{i,j}\|_1 * (\|[W_a]_{i,j}\|_0 + \|[W_b]_{i,j}\|_0)} \ else \end{cases}.$$

Given a time period set, which contains several time periods $\boldsymbol{w} = \{w_1, w_2, \cdots, w_k\}$, when the system is under normal status, and a behavior graph G_{ab} obtained from an anomaly dataset, we subtract behavior graphs generated on $\boldsymbol{w}$ from G_{ab}. Hence, $Profile(G_{ab}) = G_{ab} - \sum_{i=1}^{k} G_i$, where G_i is generated during w_i. We call the result as anomaly profile. It can better depict the nature of anomaly, as the sharing elements of the normal correlations from the anomaly behavior graph are removed.

Fig. 5 demonstrates an example of anomaly profile. In Fig. 5, **G1** is a normal behavior graph, where most of the services have inter-correlations. The anomaly behavior graph (**G2**) has a lot of sharing correlations compared to **G1**. We compute **G2**−**G1** to remove edges unrelated to the anomaly from **G2** (indicated by dashed arrow lines in **G2**). The solid red edges in **G2** represent the result of **G2**−**G1**, that is, abnormal correlations.

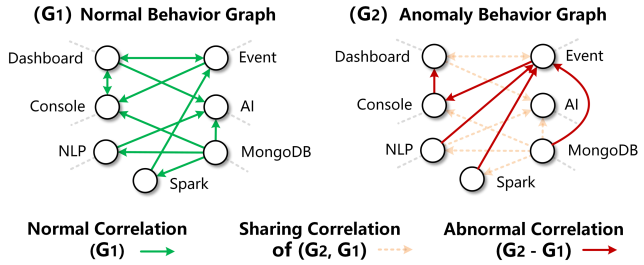


Figure 5: An example of behavior graph subtraction

6 AUTOMATED ROOT CAUSE DETECTION

6.1 Automated Metric Weight Learning

Traditionally, we choose latency to indicate the healthy condition of a short-term connection service, but it's not effective for long-term connection service monitoring. Load features can be used to indicate the health condition of IO-intensive services, but not suitable for computational-intensive services like AI and NLP. This tells that single type of metric cannot be applied to characterize the anomalies occurring in diverse services. In this section, we discuss how to find the most effective metric (or a combination of several metrics) for the service-specific anomalies.

In AutoMAP, our basic idea of choosing appropriate diagnosis metric is drawn from the way of manual troubleshooting by experienced SRE. That is, if the confirmed root cause in historical incidents shows high correlation with a front-end service in terms of certain metric, in similar situations, the candidate root cause service should also have high correlation using the same metric. To this end, AutoMAP introduces a metric weight learning mechanism based on profile similarity. We define the measurement function for service correlation and result precision:

Service Correlation. If a behavior graph $G(V,E,W)$ is generated from $\mathbb{M}$, let $\mathbb{C}$ be the multi-metric correlation matrix for each service pair $v_i, v_j \in \boldsymbol{V}$, where $\mathbb{C}_{\boldsymbol{k}} = [\mathbf{C}_{\boldsymbol{k}}]_{\boldsymbol{n}\times\boldsymbol{n}}$ records the correlation scores based on $\mathbb{M}_{\boldsymbol{k}}$. The correlation $c_{i,j,k} = [\mathbf{C}_{\boldsymbol{k}}]_{\boldsymbol{i,j}}$ scores the relevance of v_i to v_j in terms of $\mathbb{M}_{\boldsymbol{k}}$. It is defined as

$$c_{i,j,k} = [\mathbf{C}_k]_{i,j} = \left| \frac{\sum_{p=1}^{w}([\mathbf{M}_k]_{i\times p} - \overline{[\mathbf{M}_k]_i})([\mathbf{M}_k]_{j\times p} - \overline{[\mathbf{M}_k]_j})}{\sqrt{\sum_{p=1}^{w}([\mathbf{M}_k]_{i\times p} - \overline{[\mathbf{M}_k]_i})^2} \sqrt{\sum_{p=1}^{w}([\mathbf{M}_k]_{j\times p} - \overline{[\mathbf{M}_k]_j})^2}} \right|.$$

We calculate the covariance of two metric series divided by the product of their standard deviations, and use the absolute value of the result as the score. This score measures the strength of either positive or negative linear correlation between two services. The value of $c_{i,j,k}$ is in [0,1]. $c_{i,j,k} = 1$ implies completely correlation, whereas $c_{i,j,k} = 0$ means that there is no correlation between them.

Result Precision. In order to quantify the confidence of metric, we introduce a performance measurement to evaluate the precision of result, denoted as $P(G)$, where G is the anomaly

behavior graph been used. The higher result precision indicates that the algorithm can identify the root cause more accurately, resulting in less wrong candidates to investigate. More specifically, given an anomaly behavior graph G and metrics $\mathbb{M}$, let $R_{n\times 1}$ be the candidate results and $\boldsymbol{v}_{rc}$ be the root cause services, the precision score is generally calculated as $|R \cap \boldsymbol{v}_{rc}|/|\boldsymbol{v}_{rc}|$.

Fig. 6 demonstrates the procedure of automated metric weight learning in AutoMAP. Given a target anomaly G_A, we search top-k similar profiles to G_A in historical records, and denote the result as $\{G_1, G_2, \cdots, G_k\}$. Therefore, for G_A, the suggested metric weight is calculated by the correlation score of root cause and front-end service, considering from $\mathbb{M}_1$ to $\mathbb{M}_m$ separately. Let $\boldsymbol{w}$ be a $m \times 1$ matrix, where $w_i \in \boldsymbol{w}$ represents the confidence weight of $\mathbb{M}_k$ when diagnosing G. $\sum \boldsymbol{w} = 1$. Let v_{rc}, v_{fe} denote the root cause and front-end service, and $c_{rc,fr,k}$ be their correlation score in terms of $\mathbb{M}_k$. We calculate $\frac{1}{k}\sum_{j=1}^{k} P(G_j)\, c_{rc,fr,i}$ as the precision-weighted voting result from $\{G_1, G_2, \cdots, G_k\}$ for $\mathbb{M}_k$. We use normalized value as suggested weight of $\mathbb{M}_k$. Particularly, if there is no historical anomaly profile, we use a default metric $\mathbf{M}_{lat}$ to start this process.

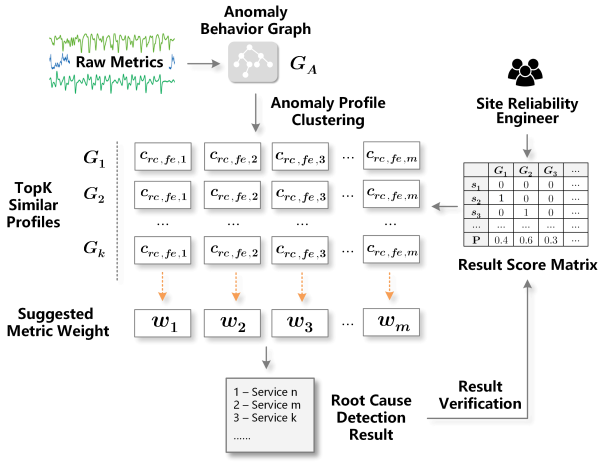


Figure 6: Metric weight learning in AutoMAP

6.2 Profile Similarity

In this section, we propose a similarity function to enable the search for similar anomaly profiles. Traditional measurements usually compare the distance of graph topology [26]. However, in AutoMAP, we aim to measure the similarity between profiles in terms of their root anomaly pattern. Therefore, this function should be able to measure the similarity of profile's topology, service profile and edges in the same time. We define our similarity function as follows:

Profile Similarity. Let $sim(G_i, G_j)$ be the similarity score of anomaly profile G_i, G_j, $sim(G_i, G_j) \in [0,1]$. If $sim(G_i, G_j) = 1$, G_i and G_j are identical. Otherwise, if $sim(G_i, G_j) = 0$, G_i and G_j share no common feature. The calculations of similarity between G_i and G_j consist of four steps:

Step 1. Let $G(V, E, W)$ be the sharing anomaly profile between G_i and G_j, where $G = (G_i - G_j) + (G_j - G_i)$;

Step 2. Compute the vertex overlap score, $sim_{VO}(G_i, G_j) = \frac{|V|}{|V_i|+|V_j|} sim_V(G_i - G_j, G_j - G_i)$, where sim_V is the service profile distance between two graphs;

Step 3. Compute the edge overlap score, $sim_{EO}(G_i, G_j) = \frac{|E|}{|E_i|+|E_j|} sim_E(G_i - G_j, G_j - G_i)$, where sim_E is the edge weight distance between two graphs;

Step 4. Compute the overall similarity score, using the harmonic mean of vertex and edge overlap score: $sim(G_i, G_j) = 2 * \frac{sim_{VO}(G_i,G_j) * sim_{EO}(G_i,G_j)}{sim_{VO}(G_i,G_j) + sim_{EO}(G_i,G_j)}$.

In Step 2, we convert services into corresponding profile categories according to their dominant metric:$\{(\mathbf{M}_{\{lat,avl\}} \rightarrow \mathbf{R}), (\mathbf{M}_{\{thr,cpu\}} \rightarrow \mathbf{C}), (\mathbf{M}_{con} \rightarrow \mathbf{N}), (\mathbf{M}_{io} \rightarrow \mathbf{S}), (\mathbf{M}_{mem} \rightarrow \mathbf{E})\}$. The vertex similarity $sim_V(G_i - G_j, G_j - G_i)$ is defined as the number of same vertexes between $G_i - G_j$ and $G_j - G_i$. For instance, if G_1 is a correlation sequence **R**→**C**→**S**→**E**, G_2 is a sequence **C**→**C**→**S**→**N**. We obtain the sharing anomaly profile by intersecting $G_1 - G_2$ and $G_2 - G_1$, which is **C**→**S**. Thus, $sim_V(G_1 - G_2, G_2 - G_1) = 2$. We have $sim_{VO}(G_1, G_2) = \frac{|V|}{|V_1|+|V_2|} sim_V(G_1 - G_2, G_2 - G_1) = \frac{2}{4+4} * 2 = \frac{1}{2}$. In Step 3, we define $sim_E(G_1 - G_2, G_2 - G_1) = dist(W_{G1-\text{Share}}, W_{G2-Share})$, where $W_{G1-Share}$ and $W_{G2-Share}$ represent the weight matrix of sharing anomaly profile in G_1 and G_2. Using the same example, $W_{G1-Share}$ and $W_{G2-Share}$ are the weight vectors of edge **C**→**S** in G_1 and G_2, respectively. We calculate $dist(W_{G1-\text{Share}}, W_{G2-Share})$ using Euclidean distance of vectors.

6.3 Root Cause Detection

Fig. 7 shows a constructed behavior graph (part) using the same incident shown in Fig. 3 and Fig. 4. Note that most user-domain APIs and normal services are removed from this example due to limited space. In the graph, we see most of the abnormal services are connected, which means they probably participate in the propagation. We notice that some of them (i.e. S_1, S_{12} and S_{27}) are not connected with any other services. Besides, although (S_3, S_{19}) and (S_4, S_{11}, S_{23}, S_{25}) are inter-connected, they are isolated from the front-end service - S_{18}, so they will not be taken into account in root cause detection. We are curious about it and seek advice to SRE. They found that these services belong to HTML5-based WebSocket server and NLP module. They usually have very long connections and their metrics make no sense in this anomaly. In other words, they have no correlation with other abnormal services. Hence, some irrelevant services can be removed from consideration in behavior graph construction phase.

Our report indicates that the root cause of this incident is that several event components run out of memory. It then propagates in the infrastructure and slow down the console, finally freezes the dashboard. Starting from S_{18}, we observe that a service chain "18→13→6" represents the causal correlations "R→C→E". However, if we strictly follow the directions in behavior graph, only a small number of services can be confirmed (marked using bold green line in Fig. 7), while most of other services are out of the scope of investigation.

The feature of anomaly will change as the hierarchical service calling and propagation. Thus, it is inaccurate to directly use service correlation score to traverse for the root cause (please refer

to our experimental results for TBAC [27] - a typical algorithm based on hierarchical service correlation). In real production systems, large amount and highly-diverse microservices make the diagnosis extremely challenging. Without any automated tools, SRE team spend 3 hours on average to identify the root cause for each incident.

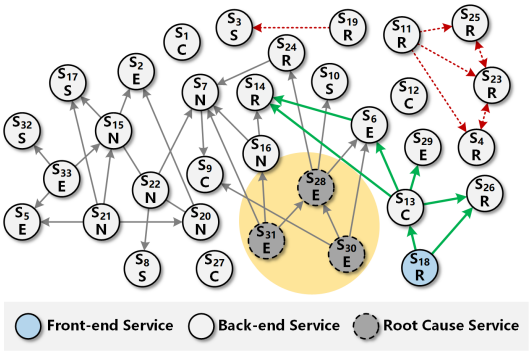


Figure 7: Part of the constructed behavior graph for a sample incident

6.4 Heuristic Random Walk Algorithm

AutoMAP employs a dynamic and heuristic algorithm to identify the root cause. It simulates our operations in traditional troubleshooting. More specifically, we start from the front-end service, follow the service-calling topology and gradually investigate each service based on its metric correlation to the anomaly. A node that is targeted by more investigation paths is more likely to be the root cause. Based on this intuitive observation, we propose a random walk algorithm with three types of transitions, namely *forward*, *self* and *backward transition*.

Forward Transition. Basically, a random visitor walks from service v_i to v_j with probability $p_{i,j}$ if $e_{ij} = 1$. According to the basic idea of root cause diagnosis, the visitor visits v_j proportionally to its correlation score to v_{fe}, i.e. $c_{j,fe,k}$. We also consider the confidence of all types of metrics. Hence, the forward transition probability matrix $\mathbf{P}$ is defined as follows,

$$[\mathbf{P}]_{i,j} = p_{i,j} = \sum_{k=1}^{m} w_k W_{i,j,k} \frac{c_{j,fe,k}}{\sum_{l \in O_i} c_{l,fe,k}}, \forall e_{ij} = 1,$$

where O_i is set of out-neighbor nodes of node v_i in G_A.

Self-Transition. The self-transition encourages the visitor to stay longer on its currently-visiting service, in case that none of its in- and out-neighbors show high correlation. Let p_i^s denote the self-transition probability for the visiting node v_i. p_i^s is determined by the difference of $p_{i,i}$ and the maximum transition probability of in- and out-neighbors of v_i. Hence, we have

$$p_i^s = \max\left(0, p_{i,i} - \max_{l \in I_i \cup O_i} p_{i,l}\right),$$

where I_i is set of in-neighbor nodes of node v_i in G_A.

Backward Transition. Another case is that when the visitor is visiting a particular service with low correlation score, it may find no way to leave if all its out-neighbor services are of low correlation to the given anomaly. Hence, we design the backward transition to resolve this issue, making the random walk more dynamic. Specifically, let $p_{i,j}^b$ be the backward transition probability from v_i to $v_j \in I_i$ in G_A, it is defined as follows,

$$p_{i,j}^b = \rho \frac{p_{i,j}}{\sum_{l \in I_i} p_{i,l}},$$

where $\rho \in [0, 1)$ is a strength parameter. ρ controls the degree of faithfulness of the random walk path to the original edge directions. If we set ρ lower, the visitor is more constrained to the original direction. Conversely, if we set ρ higher, it is more encouraged to walk backward when needed.

Random Walk. Given an anomaly behavior graph G_A, the random walk algorithm starts from v_{fe}, calculates the probability of forward, backward and self-transitions, and randomly select one of them. Services are visited in sequence by the visitor's randomly choosing the next target among its currently vising node and its neighbors. We record how many times of each service being visited and output the list descending as the result. Note that we conduct the random walk based on original behavior graph G_A instead of its anomaly profile (i.e. remove normal correlations from G_A). It increases the probability of detecting more candidates. We summarize this process in Algorithm 2.

Algorithm 2. Root Cause Detection Algorithm

Input. G_A, v_{fe}.
new transition probability $\mathbf{P}$, result array $R[n]$, v_s=v_{fe}, v_p=v_{fe}
repeat n rounds
  $v_p \leftarrow v_s$
  for each $l \in O_s$ calculate $p_{s,l}$
  for each $l \in I_s$ calculate $p_{s,l}^b$
  calculate p_s^s, row normalize $[\mathbf{P}]_{(p,s)}$
  $v_s \leftarrow$ randomly choose from $O_s \cup I_s \cup \{v_s\}$
  $R[s] \leftarrow R[s] + 1$
end
$R[n] \leftarrow Sort(R[n])$
Output. $R[n]$

7 EXPERIMENTS AND EVALUATIONS

7.1 Dataset and Benchmarks

We use both simulated and real-world production environment to evaluate AutoMAP. The simulated environment is implemented using a microservice-based system (the project has been uploaded to GitHub, but due to anonymous review requirements, we cannot provide the link in this version). It is composed of 16 microservices following a pre-configured topology running in Docker containers and managed by Zookeeper. In order to simulate anomalies, in each round of test, we randomly select one back-end services and inject fault into it, i.e. shut down the container or perform DoS (Denial of Service) attacks.

Our real-world datasets consist of 20 incidents occurred in our cloud platform. These incidents are internally reported, collected and verified which services are the root cause by SRE team. For each incident, we have about 15 million metrics, collected during 7200 seconds (1 hour before and 1 hour after the anomaly was

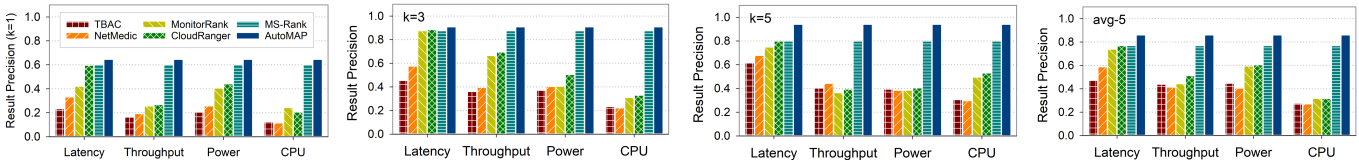


Figure 8: Top-1, 3, 5 and avg-5 precision of AutoMAP and different algorithms using real-world incidents

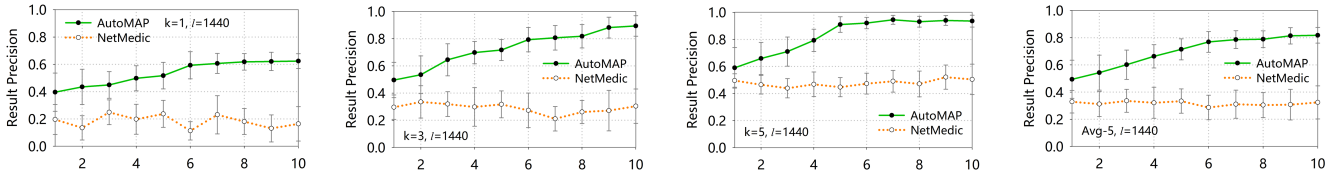


Figure 9: Precision of AutoMAP and NetMedic with different rounds of test using real-world incidents

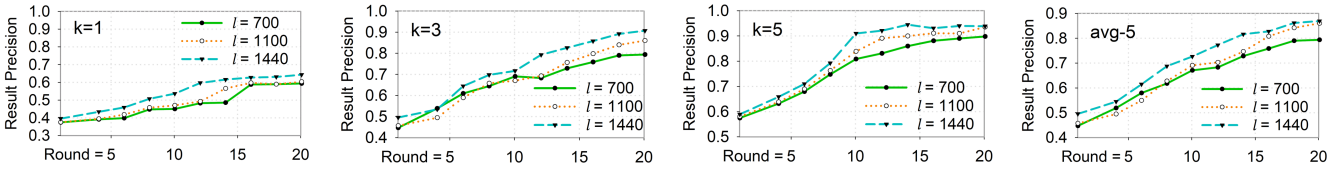


Figure 10: Top-1, 3, 5 and avg-5 precision of AutoMAP with different rounds of test and ℓ, using real-world incidents

TABLE 4: Precision of different algorithms in simulated environment

		TBAC	Monitor Rank	Cloud Ranger	Net Medic	MS-Rank*	Auto MAP*
Top-1	$\mathbf{M}_{Lat}$	23.1%	25.4%	59.4%	22.7%	59.4%	**65.7%**
	$\mathbf{M}_{Thr}$	16.2%	41.9%	40.1%			
Top-3	$\mathbf{M}_{Lat}$	45.3%	87.4%	89.5%	37.8%	89.5%	**91.2%**
	$\mathbf{M}_{Thr}$	35.9%	66.3%	68.2%			
Top-5	$\mathbf{M}_{Lat}$	61.3%	89.7%	93.3%	54.3%	93.3%	**93.5%**
	$\mathbf{M}_{Thr}$	40.1%	72.1%	73.4%			
Avg-5	$\mathbf{M}_{Lat}$	47.0%	73.7%	85.2%	49.7%	85.2%	**89.7%**
	$\mathbf{M}_{Thr}$	43.7%	64.1%	68.8%			

* Note: Result after 10 rounds of detection with multiple metrics.

TABLE 5: Avg-5 precision when introducing different domain knowledge into AutoMAP, using real-world incidents

AutoMAP + Knowledge		$\ell = 1440$			$\ell = 200$		
		avg-5	IMP*	STD	avg-5	IMP*	STD
AutoMAP +API	Link API with same source	89.9%	+0.2%	0.05	61.7%	+4.9%	0.29
AutoMAP -Normal	Remove normal services (<5%)	**92.6%**	**+3.2%**	0.05	58.9%	+0.1%	0.21
AutoMAP +Linked	Add known calling correlations	90.7%	+1.1%	0.04	**63.6%**	**+8.2%**	0.19
AutoMAP -Unlinked	Remove edges without dependencies	91.8%	+2.3%	0.04	60.1%	+2.2%	0.15
AutoMAP +Direction	Orient direction of edges (<5%)	89.7%	+0.0%	0.04	62.0%	+5.4%	0.15

* Improvement compared to original AutoMAP after 10 rounds of test.

detected) from 1732 microservice APIs. This dataset only recorded key system APIs and we removed most user-domain APIs.

Several baseline approaches are selected: TBAC [27], MonitorRank [23], CloudRanger [30], NetMedic [8] and MS-Rank [32]. We use same incidents to run those selected algorithms. For algorithms rely on single-metric: TBAC, MonitorRank and CloudRanger, we examine their precision in terms of $\mathbf{M}_{lat}$ and $\mathbf{M}_{thr}$. For algorithms relying on pre-defined topology, we use behavior graph constructed on $\mathbf{M}_{lat}$ to run them, as the ground truth of this topology is unrecorded. We compare the algorithms using top-k and average top-1 to k result (denoted as avg-k) precision. In each round of test, we choose the avg-5 precision as the score of different metrics and update the score matrix in AutoMAP.

Our prototype system is implemented using Python and Pcalg package (https://pypi.org/project/pcalg/). In Algorithm 1, we use χ^2-test [25] to determine the conditional independence. The metric sampling interval we used for simulated platform is 2 seconds, and 5 seconds for our real-world cloud system. Experiments are conducted in a workstation with Intel Xeon 2.4GHz CPU, 16GB RAM running 64-bit Windows Server 2008. All experimental results are obtained by averaging 20 different rounds of tests.

7.2 Experiments and Results Analysis

7.2.1 Root cause identification. The first experiment compares the precision of selected algorithms, when k =1/ 3/ 5, $\ell = 1440$. For AutoMAP and MS-Rank, we record theirs result precision after 10 rounds of optimization. Fig. 8 summarizes the experimental results using real-world incidents. In general, we see that AutoMAP outperforms other algorithms in terms of result precision. Specifically, it has 64.4% result precision in top-1 and 93.9% in top-5. In terms of other algorithms, the result precision is usually less than 50% without selecting an appropriate metric. Table 4 summarizes the experimental results obtained from the simulated system, which are similar as those obtained from real-world incidents. We found that the root cause detection is inaccurate if it's only based on correlation score (TBAC). Also, compared to static algorithms (TBAC, NetMedic, MonitorRank, CloudRanger), random walk scheme can identify the root cause with higher accuracy. Besides, compared with the MS-Rank algorithm,

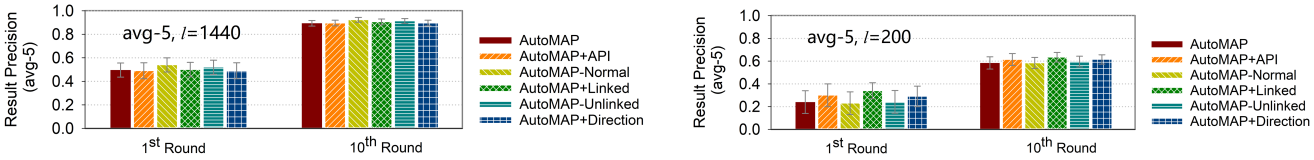


Figure 11: Avg-5 precision of AutoMAP augmented with different types of knowledge

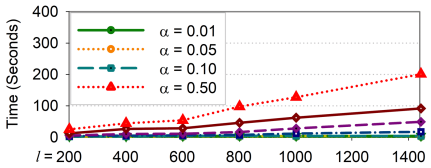


Figure 12: Time cost when executing AutoMAP with different α

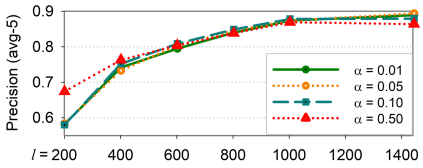


Figure 13: Avg-5 precision with different α

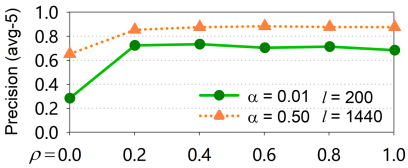


Figure 14: Avg-5 precision with different ρ

the introduction of anomaly profile in AutoMAP can effectively improve the result accuracy, especially for top-1 results.

7.2.2 Self-Optimization. In order to verify the effectiveness of self-optimization mechanism, we examine AutoMAP using multiple rounds of test, and compare its precision with traditional NetMedic solution from round 1 to 10. The experiment is conducted when k =1/ 3/ 5, $\ell = 1440$, using real-world incidents. The results are illustrated in Fig. 9, which shows that the precision of AutoMAP increases significantly when we conduct more rounds of test. For example, the precision increases from 59.7% ($k = 5$) to 93.9% at 10th round. AutoMAP shows significant advantages over NetMedic. As NetMedic does not support self-optimizing, so it is unstable in the rapidly changing system architecture.

7.2.3 Algorithm parameter - ℓ. The third experiment evaluates the impact of different parameters on AutoMAP, i.e., ℓ - incident period length. We compare the precision of AutoMAP when ℓ is increased from 700 to 1440 for real-world incidents. When we input more metrics into AutoMAP, which means parameter ℓ is set larger, the results show more accurate correlations between the services, as shown in Fig. 10. It is worth noticing that a small ℓ results in a significant impact on the precision, e.g., the precision drops below 40% when $\ell = 700$. We further compare the growth rate of precision with different ℓ. Fig. 10 shows that the precision increases more quickly when ℓ is higher, as we have more data to support multi-metric optimization.

7.2.4 Domain knowledge. This experiment verifies the effect of domain knowledge. We propose five enhancement methods using different types of knowledge in AutoMAP (See Table IV). Our experiment compares the result of these enhanced versions with the original AutoMAP in the 1st and 10th round. Table V and Fig. 11 summarize the results obtained from the experiments ran with real-world incidents. We observe the algorithms enhanced with domain knowledge have different impacts on the original AutoMAP. For instance, when $\ell = 200$, domain knowledge can better help to improve the algorithm precision, especially for the three algorithms that add information to behavior graph (AutoMAP+API 4.9% improvement, AutoMAP+Linked +8.2%, AutoMAP+Direction +5.4%). However, when $\ell = 1440$, although the enhanced algorithms can still improve the precision, the derived behavior graph is sufficient to be used for heuristic diagnosis. That clearly says, the role of domain knowledge gets weaker when more sampling data are available.

7.2.5 Algorithm parameters - α and ρ. The last experiment conducted using real-world incidents analyzes the impact of two important parameters - α and ρ (See Fig. 12 and 13). To observe the impact of α, we increased it from 0.01 to 0.50, and find the overall execution time increases linearly (see Fig. 12). Even when $\alpha = 0.5$, it takes less than 4 minutes to process the complete dataset. We then developed additional experiments to analyze the effect of α and ρ on precision (see Fig. 13). When we increase α from 0.01 to 0.50, and compare the precision under different ℓ, we find that when the dataset is sparser (for instance, when $\ell =$ 200), higher α makes the result more precise, because AutoMAP does not have enough metrics to characterize the correlations. When we set α lower, the number of edges in the behavior graphs may become too few to ensure the accessibility of visitor, leading to low result precision. Meanwhile, when we use more metric records to run AutoMAP, the impact of using different α is not obvious. As a result, we can choose a relatively small α, making the correlation graph more consistent with ground truth.

Lastly, to evaluate the impact of the backward transition parameter - ρ, we set up two environments (α = 0.01, $\ell = 200$ versus α = 0.50, $\ell = 1440$) and compare the precision using different ρ. As the results indicated in Fig. 14, when ρ is smaller, a higher α is needed to ensure that the random walk algorithm has more paths to be chosen. Besides, the precision is not significantly improved when ρ is set close to 1. Therefore, it is recommended to choose a moderate backward transition parameter, e.g., ρ =0.2.

8 CONCLUSIONS

This paper presents AutoMAP, a system that enables automated anomaly diagnosis for microservice-based web applications. Our experiments conducted in both real-world and simulated environment show that AutoMAP outperforms other methods in precision. It offers fast identification of root cause especially in large-scale microservice architecture. AutoMAP treats the microservice-based applications as "grey box" and make no assumption about pre-knowledge. Therefore, it could fit into most application scenarios because it does not require a pre-defined topology, given a large part of legacy systems only have basic performance monitoring metrics. Besides, it is easy to introduce expert experiences into AutoMAP. Our future work will be focused on the application of AutoMAP to other complex systems, such as social and biological networks.

REFERENCES

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, pp. 42-52, 2016.

[2] S. Newman, *Building Microservices*. " O'Reilly Media, Inc.", 2015.

[3] H. Wu, A. N. Tantawi, and T. Yu, "A self-optimizing workload management solution for cloud applications," in *IEEE 20th International Conference on Web Services (ICWS)*, 2013, pp. 483-490: IEEE.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1-58, 2009.

[5] N. V. D. Shahir Daya, et al., "Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach," *IBM Redbooks*, 2015.

[6] J. Gertler, *Fault detection and diagnosis in engineering systems*. CRC press, 1998.

[7] Q. Wang *et al.*, "Detecting transient bottlenecks in n-tier applications through fine-grained analysis," in *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, 2013, pp. 31-40: IEEE.

[8] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 243-254, 2009.

[9] M. Ługorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," *Science of computer programming*, vol. 53, no. 2, pp. 165 -194, 2004.

[10] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments," in *International Conference on Service-Oriented Computing*, 2018, pp. 3-20: Springer.

[11] X. Zhou, X. Peng, T. Xie, et al., "Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study," *IEEE Transactions on Software Engineering*, 2018.

[12] T. Ahmed, B. Oreshkin, and M. Coates, "Machine learning approaches to network anomaly detection," in *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, 2007, pp. 1-6: USENIX Association.

[13] Y. Liu, L. Zhang, and Y. Guan, "A distributed data streaming algorithm for network-wide traffic anomaly detection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, pp. 81-82, 2009.

[14] R. Jiang, H. Fei, and J. Huan, "Anomaly localization for network data streams with graph joint sparse PCA," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 886-894: ACM.

[15] J. Gao, G. Jiang, H. Chen, and J. Han, "Modeling probabilistic measurement correlations for problem determination in large-scale distributed systems," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009, pp. 623-630: IEEE.

[16] C. Wang *et al.*, "VScope: middleware for troubleshooting time-sensitive data center applications," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2012, pp. 121-141: Springer.

[17] G. Jiang, H. Chen, and K. Yoshihira, "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 312-326, 2006.

[18] A. Jalali and S. Sanghavi, "Learning the Dependence Graph of Time Series with Latent Factors," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012, pp. 473-480.

[19] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626-688, 2015.

[20] G. Jiang, H. Chen, and K. Yoshihira, "Efficient and scalable algorithms for inferring likely invariants in distributed systems," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 11, pp. 1508-1523, 2007.

[21] J. Thalheim *et al.*, "Sieve: actionable insights from monitored metrics in distributed systems," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 14-27: ACM.

[22] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *ACM SIGCOMM Computer Communication Review*, 2007, vol. 37, no. 4, pp. 13-24: ACM.

[23] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," in *ACM SIGMETRICS Performance Evaluation Review*, 2013, vol. 41, no. 1, pp. 93-104: ACM.

[24] K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, no. 2, pp. 158-181, 1990.

[25] M. Ding, Y. Chen, S. Bressler. Granger causality: basic theory and application to neuroscience. *Handbook of time series analysis: recent theoretical developments and applications*, pp. 437-460, 2006.

[26] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 19-30, 2010.

[27] N. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring, "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation," in *13th European Conference on Software Maintenance and Reengineering, CSMR'09*, 2009, pp. 47-58: IEEE.

[28] H. Shan, Y. Chen, H. Liu, Y. Zhang, X. Xiao, X. He, M. Li, and W. Ding, "ϵ-Diagnosis: Unsupervised and Real-time Diagnosis of Small- window Long-tail Latency in Large-scale Microservice Platforms," *The World Wide Web Conference (WWW)*, 2019, pp. 2659-2665: ACM.

[29] Y. Chen, X. Yang, Q. Lin, et al., "Outage Prediction and Diagnosis for Cloud Service Systems," *The World Wide Web Conference (WWW)*, 2019, pp. 2659-2665: ACM.

[30] P. Wang, J-M. Xu, M. Ma*, W-L. Lin, D-S. Pan, Y. Wang and P-S. Chen, "CloudRanger: Root Cause Identification for Cloud Native Systems," in *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (*CCGrid*), 2018, pp. 492-502.

[31] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments," in *Proceedings of the International Conference on Service-Oriented Computing* (*ICSOC*), 2018, pp. 3-20: Springer.

[32] M. Ma, W-L. Lin, D-S. Pan, P. Wang, "MS-Rank: Multi-Metric and Self-Adaptive Root Cause Diagnosis for Microservice Applications," in *Proceedings of the IEEE International Conference on Web Services* (*ICWS*), 2019, pp. 60-67.