Generic and Robust Root Cause Localization for Multi-Dimensional Data in Online Service Systems

Zeyan Li[a,f], Junjie Chen[b], Yihao Chen[a], Chengyang Luo[a], Yiwei Zhao[a], Yongqian Sun[c], Kaixin Sui[d], Xiping Wang[a], Dapeng Liu[d], Xing Jin[e], Qi Wang[e], Dan Pei[a,f]

[a]*Tsinghua University, Beijing, China*
[b]*Tianjin University, College of Intelligence and Computing, Tianjin, China*
[c]*Nankai University, College of Software, Tianjin, China*
[d]*Bizseer, Beijing, China*
[e]*China Construction Bank, Beijing, China*
[f]*Beijing National Research Center for Information Science and Technology (BNRist), Beijing, China*

Abstract

Localizing root causes for multi-dimensional data is critical to ensure online service systems' reliability. When a fault occurs, only the measure values within specific attribute combinations (e.g., Province=Beijing) are abnormal. Such attribute combinations are substantial clues to the underlying root causes and thus are called root causes of multi-dimensional data. This paper proposes a generic and robust root cause localization approach for multi-dimensional data, PSqueeze. We propose a generic property of root cause for multi-dimensional data, generalized ripple effect (GRE). Based on it, we propose a novel probabilistic cluster method and a robust heuristic search method. Moreover, we identify the importance of determining external root causes and propose an effective method for the first time in literature. Our experiments on two real-world datasets with 5400 faults show that the F1-score of PSqueeze outperforms baselines by 32.89%, while the localization time is around 10 seconds across all cases. The F1-score in determining external root causes of PSqueeze achieves 0.90. Furthermore, case studies in several production systems demonstrate that PSqueeze is helpful to fault diagnosis in the real world.

Keywords: Root cause localization, online service system, ripple effect, multi-dimensional data

1. Introduction

Large online service systems (*e.g.*, online shopping platforms) serve millions of users and require high reliability to ensure user experience. Faults in large online service systems could cause enormous economic loss and damage user satisfaction [1]. For example, the loss of one-hour downtime for `Amazon.com` on Prime Day in 2018 (its biggest sale event of the year) is up to $100 million [2]. Therefore, it is in urgent demand to diagnose faults rapidly.

To ensure quality of software service, operators usually closely monitor some measures (*e.g.*, total dollar amount), which reflect the system status [3, 4]. When a fault occurs, the monitoring system can detect the abnormal measure values and raise alerts to operators. A measure record is associated with many attributes, and when a fault occurs, only the measure values of specific attribute combinations are abnormal [3, 4, 5]. For example, when the network service provided by *CMobile* in *Beijing* province fails, only the dollar amount of (Province=Beijing, ISP=CMobile) would decrease dramatically. Such *attribute combinations* can effectively indicate the fault location and serve as substantial clues to the underlying root causes [3, 5]. Thus, we call the set of such attribute combinations as the *root cause* of the multi-dimensional data [4, 6, 3, 7]. Therefore, following existing work [8, 3, 5, 9, 4, 6, 7], in this paper, we focus on *localizing root causes of multi-dimensional data* to help operators diagnose faults rapidly.

However, it is challenging due to the huge search space. On the one hand, there are many attributes (*e.g.*, dozens) and attribute values (*e.g.*, thousands) in large online service systems, leading to a combinatorial explosion. On the other hand, faults must be mitigated rapidly to reduce the impact on user experience, and thus it requires high efficiency for the localization.

Existing works [10, 11, 8, 4, 7, 5, 3, 9] apply various techniques to overcome the huge search space challenge, but they are not generic or robust enough due to some limitations or not efficient enough (see later in Table 11).

Email addresses: `zy-li18@mails.tsinghua.edu.cn` (Zeyan Li), `junjiechen@tju.edu.cn` (Junjie Chen), `chenyiha17@mails.tsinghua.edu.cn` (Yihao Chen), `luocy16@mails.tsinghua.edu.cn` (Chengyang Luo), `zhaoyw17@mails.tsinghua.edu.cn` (Yiwei Zhao), `sunyongqian@nankai.edu.cn` (Yongqian Sun), `suikaixin@bizseer.com` (Kaixin Sui), `wxp17@mails.tsinghua.edu.cn` (Xiping Wang), `liudapeng@bizseer.com` (Dapeng Liu), `jinxing.zh@ccb.com` (Xing Jin), `wangqi10.zh@ccb.com` (Qi Wang), `peidan@tsinghua.edu.cn` (Dan Pei)

For example, MID [5], iDice [8], and ImpAPTr [9] are only applicable to specific types of measures. Apriori [7, 3] and R-Adtributor [11] highly relies on parameter fine-tuning. Notably, all the previous approaches do not check *external root causes*, *i.e.*, root causes containing some unrecorded or unused attributes. The localization results are always incorrect when there are external root causes, which can mislead the direction of fault diagnosis and waste time [12].

This paper proposes *PSqueeze*, a generic and robust root cause localization approach for multi-dimensional data. Rather than impractical root cause assumptions or properties of specific kinds of measures, the search strategies of *PSqueeze* are based on a more generic property of root causes of multi-dimensional data, *generalized ripple effect* (GRE). GRE holds for different measures (see Section 3) and holds in real-world faults (see Section 6), enabling our search strategies' genericness. Based on GRE, we propose a "bottom-up&top-down" method to achieve high efficiency without much loss of genericness and robustness. Specifically speaking, in the bottom-up stage, we firstly group attribute combinations into different clusters, each of which contains those attribute combinations affected by the same root cause only, with a robust probabilistic clustering method based on GRE. In this way, *PSqueeze* firstly breaks down the problem into simpler sub-problems (*i.e.*, single root causes) and reduces search space. Then in the top-down stage, we propose a score function, generalized potential score (GPS), to evaluate how likely a set of attribute combinations is the root cause, and search from each cluster the attribute combinations maximizing it with a efficient heuristic search strategy. Finally, after the search, *PSqueeze* determines external root causes based on GPS.

To evaluate *PSqueeze*, we conduct extensive experimental studies based on two real-world datasets from two companies. Since the real-world faults are not enough for evaluation, we propose a fault simulation method and obtain 5400 simulated faults. The results show that *PSqueeze* outperforms all baselines by 32.89% in different situations while keeping high efficiency (costs about 10s consistently for each fault). We also inject 73 faults on an open-source benchmark system to prove the effectiveness of *PSqueeze* in real-world scenarios. For determining external root causes, the F1-score of *PSqueeze* achieves 0.90 on average. We also present several real-world success stories to demonstrate the efficacy of *PSqueeze* in real-world systems.

The major contributions are summarized as follows:

- We propose a novel property about root causes of multidimensional data, which is proved to hold in different situations and hold in real-world faults.
- We identify the importance of determining external root causes propose the first effective method for it.
- We propose a novel "bottom-up&top-down" localization method, *PSqueeze*, achieving high efficiency without much loss of genericness and robustness.
- We evaluate the effectiveness and efficiency *PSqueeze* in different situations based on 5400 simulated faults and xxx injected faults. We make our dataset and implementation public to help further studies in the field[1].

This paper extends our previous conference paper, Squeeze [6], in four aspects.

- New methods. First, this paper proposes the first external-root-cause-determining method in the field (Section 4.4). Second, to reduce the influence of noises, we propose a novel *probabilistic* clustering method (Section 4.2). Hence we name our new method as *PSqueeze* (probabilistic Squeeze).
- New experiment settings. First, we propose a more reasonable fault simulation strategy for evaluation (Section 5.1.1). Second, we implement and compare two more recent related works, MID [5] and ImpAPTr [9]. Third, we introduce two new datasets based on fault injection on an open-source benchmark system.
- New experiment results based on the new experiment settings (Section 5) and new real-world success stories (Section 6). The results show *PSqueeze* is effective and efficient and outperform the previous approaches including Squeeze.
- Enhancement to presentation. First, we clarify the definition of basic concepts formmaly (Section 2). Second, we present more details about our methodology, such as the proof of GRE for productions (Section 3), and the reasons for our methodology's design choices (Section 4). Notably, for a better understanding of GRE, we present a much more straightforward proof in Section 3.2.

2. Background

In this section, we first describe our problem intuitively. Then we introduce some necessary concepts, notations, and definitions. Finally, we define our problem formally.

2.1. Root Cause Localization for Multi-Dimensional Data

Table 1: Example structured logs for an online shopping platform

Order ID	Timestamp	Dollar Amount	Province	ISP
A001	2020.07.15 10:00:01	$16	Beijing	China Mobile
A002	2020.07.15 10:00:05	$21	Beijing	China Unicom

As introduced in Section 1, multi-dimensional data are essentially a group of structured logs generated by an online service system. Specifically speaking, by grouping the

[1]`https://github.com/NetManAIOps/PSqueeze`

Table 2: An example multi-dimensional data at a specific time point.

Province	ISP	real value	forecast value
Beijing	**China Mobile**	**5**	**10**
Beijing	**China Unicom**	**10**	**20**
Shanghai	China Unicom	30	31
Guangdong	China Mobile	10	9.8
Zhejiang	China Unicom	2	2
Guangdong	China Unicom	200	210
Shanxi	China Unicom	20	22
Jiangsu	China Unicom	200	203
Tianjin	China Mobile	41	43
Total		518	550.8

logs (*e.g.*, Table 1) by some *attributes* (*i.e.*, `Timestamp`, `Province` and `ISP`) and aggregating the *measure* values (*i.e.*, `Dollar Amount`), we transform the original logs into multi-dimensional data (*e.g.*, Table 2) [3].

To ensure service quality, operators closely monitor the overall measure values (*e.g.*, total dollar amount). When a measure value becomes abnormal (*e.g.*, in Table 2, the total dollar amount decreases from 550.8 to 518), a fault occurs in the online service system. A fault usually causes only the measure values under specific attribute combinations abnormal in practice [5]. For example, in Table 2, when a fault happens in the servers in Beijing, only the measure values of (Province=Beijing) (*i.e.*, the first two rows) are abnormal. Such attribute combinations indicate the scope of the fault, and thus, are substantial clues to the underlying root causes. We call such attribute combinations *root-cause attribute combinations* and the set of root-cause attribute combinations as the *root causes for the multi-dimensional data* [4, 3, 6]. For convenience, without other conflicts, "root causes" in this paper refer to root causes for multi-dimensional data. Localizing root causes of multi-dimensional data enables rapid fault diagnosis by directing the investigation.

Table 3: External root cause example.

ISP	real value	forecast value
China Mobile	**56**	**62.8**
China Unicom	**462**	**488**
Total	518	550.8

The exact root causes (*e.g.*, {(Province=Beijing)} in Table 3) may contain uncollected attributes (*e.g.*, `Province` in Table 3). We call such root causes *external root causes*, which are not rare in practice [12]. On the one hand, many attributes are seldom used in fault diagnosis because they contain many null values, are hard to understand, or are not informative. On the other hand, since the search space grows exponentially with the number of attributes, operators have to choose the most useful ones for root cause localization. When external root causes exist, the localization results would be wrong and misleading. However, all existing approaches do not check external root causes to our best knowledge.

2.2. Necessary Concepts and Notations

We denote the set of all *attributes* of the studied multi-dimensional data D as $A=\{a_1, a_2, ..., a_n\}$, where a_i is the i-th attribute, and n is the total number of attributes. Each attribute has a finite number of discrete feasible values, which are called *attribute values*. We denote the set of attribute values of a_i as $V_i=\{v_i^{(1)},v_i^{(2)},...,v_i^{(m_i)}\}$, where $v_i^{(j)}$ is the j-th attribute value of a_i, and m_i is the number of attribute values of a_i. An attribute a_i and one of its attribute values $v_i^{(j)}$ construct a *tuple*, $t = (a_i, v_i^{(j)})$. For each attribute a_i, we denote the set of its tuples as $T_i = \{a_i\} \times V_i$, where "$\times$" refers to Cartesian product. We denote the set of all tuples as $T = \cup_{i=1}^{n} T_i$. Then an *attribute combination* is a subset of T that contains at most one tuple from each T_i. Therefore *the set of all attribute combinations* can be denoted as $E=\{e \in \mathcal{P}(T) \mid \forall T_i, |e \cap T_i| \le 1\}$, where $\mathcal{P}(T)$ refers to the power set of T, and $|\cdot|$ refers to the cardinality of a set. In practice, a root cause of a fault can contain multiple root-cause attribute combinations, and a fault can have multiple root causes either. Hence *the set of all root cause candidates* is $\mathcal{P}(E)$ rather than E.

A *leaf attribute combination* e (a.k.a. *leaf* for simplicity) is an attribute combination that contains tuples of every attribute, *i.e.*, $\forall T_i, |e \cap T_i|=1$. An attribute combination e_1 is *descended* from e_2 when $e_2 \subsetneq e_1$. For example, (Province=Shanghai$\wedge$ISP=China Unicom) is descended from (Province=Shanghai). The insight is that if e_1 is descended from e_2, then the slice of data represented by e_1 is a subset of that of e_2. We denote the set of all leaf attribute combinations descended from e as $LE(e)=\{e' \in E | e \subsetneq e' \wedge \forall T_i, |e' \cap T_i|=1\}$.

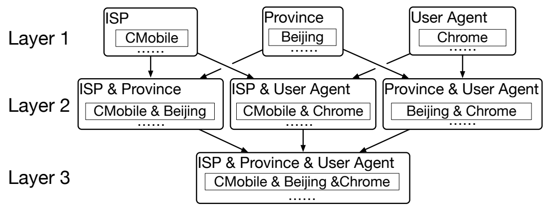


Figure 1: A graph of cuboids (rounded boxes) with 3 attributes

A *cuboid* is a set of attribute combinations enumerating all attribute values for the involved attributes, as shown in Fig. 1. Given a set of attributes $A' \subset A$, the corresponding cuboid is $Cuboid_{A'} = \{e \in E | \forall a_i \in A', |e \cap T_i|=1 \wedge |e|=|A'|\}$. We call $|A'|$ the *layer* of $Cuboid_{A'}$ (see Fig. 1).

Fundamental measures are those measures directly aggregated from raw logs and are additive [10]. For example, in Table 2, the total dollar amount of (Province=Beijing) is the sum of those of (Province=Beijing$\wedge$ISP=China Mobile) and (Province=Beijing$\wedge$ISP=China Unicom). *Derived measures* are derived from fundamental measures [10] and are typically non-additive [10]. For example, the overall average dollar amount is not the sum over those of all ISPs. The overall average dollar amount can be either greater than or less than that of (Province=Beijing). Some previous approaches [8, 4, 5] are not applicable on derived measures due to these characteristics.

The *real value* (denoted as $v(e)$) of an attribute combination e is the measure value that is actually observed based on the raw transaction logs, and the *forecast value* ($f(e)$) is its expected normal value. We calculate the forecast value by a time-series forecasting algorithm (see Section 4.1). Without loss of genericness, we assume both v and f are non-negative since most common measures are non-negative. Note that for a derived measure $M = h(M_1, M_2, ..., M_l)$, $v_M(e) = h(v_{M_1}(e), v_{M_2}(e), ..., v_{M_l}(e))$. Furthermore, for convenience, we extend the notations, v and f, to sets of attribute combinations: supposing that S is a set of attribute combinations, for fundamental measures, $v(S){=}\sum_{e'\in\bigcup_{e\in S} LE(e)} v(e')$ and so does f. For a derived measure $M = h(M_1, M_2, ..., M_l)$, $v_M(S) = h(v_{M_1}(S), v_{M_2}(S), ..., v_{M_l}(S))$ and so does f.

2.3. Problem Definition

The input of our problem is a snapshot of multi-dimensional data D (with both real and forecasting values) at the fault time. The forecasting values are obtained by an appropriate time-series forecasting algorithm, which is out of the scope of this paper. To better evaluate the robustness of *PSqueeze*, in this paper, we use MA (moving average, one of the simplest algorithms) for all scenarios (see later in Section 5.1). Our goal is to localize the *root cause of multi-dimensional data*, which refers to a set of attribute combinations that is:

- *Expressive.* An expressive root cause candidate S indicates the scope of faults in the multi-dimensional data accurately. In other words, the part of D specified by S is abnormal, and the other part is normal.
- *Interpretable.* An interpretable root cause candidate S is as concise as possible to make operators focus on the faulty attributes and attribute values.

Sometimes there are multiple root causes at the same time, which indicate different underlying failures and have different influence. In such cases, each root cause is a set of attribute combinations that is expressive and interpretable excluding the influence of the other root causes, and we aim to find the union of these root causes. Following existing works [10, 8, 4, 7, 11, 5, 3, 9], causal inference is also out of scope.

3. Generalized Ripple Effect

3.1. Background of Ripple Effect

Ripple effect, first empirically observed by [4] for *fundamental measures* only, captures the relationship of attribute combinations' abnormal magnitudes caused by the same root cause. The intuition is that all attribute combinations affected by the same root cause will change by the same proportion. We denote the set of attribute combinations affected by a root cause $S \in \mathcal{P}(E)$ as $\text{Aff}(S) = \{e \in E \mid \exists e_0 \in S,\ s.t.\ e_0 \subseteq e\}$. Then ripple effect can be expressed as

$$(f(e) - v(e))/f(e){=}(f(S) - v(S))/f(S), \forall e{\in}\text{Aff}(S) \quad (1)$$

For example, in Table 2, the root cause is $S = \{(\text{Province} = \text{Beijing})\}$ in cuboid C_{province}. Therefore, if $e_1 = (\text{Province} = \text{Beijing}\wedge\text{ISP} = \text{China Unicom})$, then $(f(S) - v(S))/f(S) = ((10{+}20) - (5{+}10))/(10{+}20) = 0.5$ and $(f(e_1) - v(e_1))/f(e_1) = (20 - 10)/20 = 0.5$. Note that for multiple-root-cause faults, S only denotes a single root cause rather than the union of multiple root causes.

3.2. Generalizing Ripple Effect for Derived Measures

First, we generalize ripple effect to derived measures. We aim to prove that (1) holds for a derived measure when (1) holds for all its underlying fundamental measures. Since most common derived measures are the quotient or product of two fundamental measures (*e.g.*, average dollar amount and success rate), without much loss of genericness, we provide the proof for such derived measures only. Compared with our previous conference version, we simplify the equations for clarity.

3.2.1. Quotient

Consider three measures, M_1, M_2, M_3, where M_1 and M_2 are fundamental measures and $M_3{=}M_1/M_2$. Because M_1 and M_2 are fundamental measures and they follow (1), for both M_i $(i = 1, 2)$, $v_{M_i}(e)/f_{M_i}(e) = v_{M_i}(S)/f_{M_i}(S)$. Therefore,

$$\begin{aligned}\frac{f_{M_3}(S) - v_{M_3}(S)}{f_{M_3}(S)} &= (\frac{f_{M_1}(S)}{f_{M_2}(S)} - \frac{v_{M_1}(S)}{v_{M_2}(S)})\frac{f_{M_2}(S)}{f_{M_1}(S)} \\ &= 1 - \frac{v_{M_1}(S)}{f_{M_1}(S)}\frac{f_{M_2}(S)}{v_{M_2}(S)} = 1 - \frac{v_{M_1}(e)}{f_{M_1}(e)}\frac{f_{M_2}(e)}{v_{M_2}(e)} \\ &= 1 - \frac{v_{M_3}(e)}{f_{M_3}(e)} = \frac{f_{M_3}(e) - v_{M_3}(e)}{f_{M_3}(e)}\end{aligned} \quad (2)$$

3.2.2. Product

Similarly, if M_1 and M_2 are fundamental measures and $M_3{=}M_1 \cdot M_2$, then for both M_i $(i = 1, 2)$, $v_{M_i}(e)/f_{M_i}(e) = v_{M_i}(S)/f_{M_i}(S)$. Therefore,

$$\begin{aligned}\frac{f_{M_3}(S) - v_{M_3}(S)}{f_{M_3}(S)} &= \frac{f_{M_1}(S)f_{M_2}(S) - v_{M_1}(S)v_{M_2}(S)}{f_{M_1}(S)f_{M_2}(S)} \\ &= 1 - \frac{v_{M_1}(S)}{f_{M_1}(S)}\frac{v_{M_2}(S)}{f_{M_2}(S)} = 1 - \frac{v_{M_1}(e)}{f_{M_1}(e)}\frac{v_{M_2}(e)}{f_{M_2}(e)} \\ &= 1 - \frac{v_{M_3}(e)}{f_{M_3}(e)} = \frac{f_{M_3}(e) - v_{M_3}(e)}{f_{M_3}(e)}\end{aligned} \quad (3)$$

The core idea of our proof is *finite difference* [13], and a similar method can be applied when dealing with other types of derived measures. Though our proof technique can hardly be applied on specific types of derived measures (*e.g.*, tail latency), our results can already cover most common measures (*e.g.*, the four golden signals from Google SRE [14]), including both fundamental measures and derived measures.

3.3. Generalizing Ripple Effect for Zero Forecast Values

Equation (1) does not work for zero forecast values (*i.e.*, $f(S) = 0$) and thus is not robust enough. To tackle it, we replace f with $\frac{f+v}{2}$. The intuition is to use $\frac{f+v}{2}$ to estimate f. Therefore, the formulation of GRE is

$$\frac{f(e) - v(e)}{f(e) + v(e)} = \frac{f(S) - v(S)}{f(S) + v(S)}, \forall e \in \text{Aff}(S) \tag{4}$$

Equation (4) is consistent with (1):

- If $f(e) = v(e) = 0$ or $f(S) = v(S) = 0$, the relationship between e and S is meaningless since there is actually no data for e or S. In other situations, (4) is always meaningful.
- If $f(e) \neq 0$ and $f(S) \neq 0$, it is obvious that (1) is equivalent to (4).
- If $f(e) = 0 \neq v(e)$ or $f(S) = 0 \neq v(S)$, then (1) does not hold. We extend the idea of ripple effect to more generic cases by modifying the calculation of anomaly magnitudes.

3.4. Deviation Score and Expected Abnormal Value

In this paper, we utilize GRE by *deviation scores* and *expected abnormal values*. According to (4), for any attribute combination e that is affected by the same root cause S, the value of $\frac{f(e)-v(e)}{f(e)+v(e)}$ keeps invariant. We define it as *deviation score* of e (denoted as $d(e)$). According to GRE,

$$\forall e \in \text{Aff}(S), d(e) = \big(f(S) - v(S)\big)/\big(f(S) + v(S)\big) \tag{5}$$

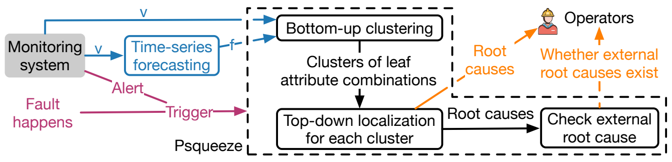
Therefore, given a root cause candidate S and any attribute combination $e \in \text{Aff}(S)$, if S is the correct root cause, then $d(e)=\frac{f(S)-v(S)}{f(S)+v(S)}$. As a result, the *expected abnormal value* of e should be

$$a(e) = f(e)\big(1 - d(e)\big)/\big(1 + d(e)\big) \tag{6}$$

If $a(e)$ differs from $v(e)$ a lot, then the candidate S breaks GRE and is not the correct root cause.

4. Methodology

4.1. Overview



v
Monitoring system
v
Time-series forecasting
f
Alert
Fault happens
Trigger
Bottom-up clustering
Clusters of leaf attribute combinations
Top-down localization for each cluster
Root causes
Root causes
Check external root cause
Operators
Whether external root causes exist
Psqueeze

Figure 2: The workflow of *PSqueeze*

The workflow of *PSqueeze* is illustrated in Fig. 2, where the dashed box highlights the scope of *PSqueeze*. When a fault happens (often indicated by alerts from the monitoring system), *PSqueeze* is triggered. *PSqueeze* takes the corresponding multi-dimensional data at the fault time (v) and its forecast values (f) as inputs. Then, *PSqueeze* reports the root causes to operators, and notifies operators whether there can be external root causes to avoid misleading.

PSqueeze contains three stages: 1) bottom-up clustering (Section 4.2), 2) top-down localization for each cluster (Section 4.3), and 3) external root cause determining (Section 4.4). Different from all previous work [10, 11, 8, 4, 7, 5, 3], *PSqueeze* employs a novel "bottom-up then top-down" searching strategy. In the bottom-up stage, *PSqueeze* groups leaf attribute combinations into different clusters, each of which contains the leaf attribute combinations affected by the same root cause. The bottom-up clustering enables the further design of our efficient in-cluster localization method by simplifying the problem from multiple-root-cause localization to single-root-cause localization. In the top-down step, *PSqueeze* uses a heuristic method based on our proposed generalized potential score (GPS) to efficiently search for the root cause in each cluster output by the bottom-up step. At the final stage, *PSqueeze* determines whether there are external root causes.

Notably, this paper extends our previous conference version with respect to methodology in two aspects. First, we employ probabilistic clustering for robustness (Section 4.2). Second, we introduce external root causes checking to avoid misleading (Section 4.4). Besides, we fix the issue in (9) and enhance the presentation of the methodology for clarity.

4.2. Bottom-Up Searching through Clustering

In this stage, we determine the cluster boundaries with a probabilistic cluster method based on the leaf attribute combinations with large abnormal changes.

4.2.1. Forecast Residual-Based Filtering

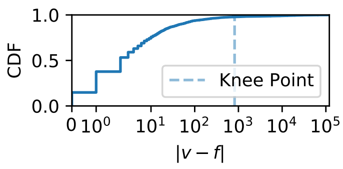


Figure 3: An illustration example of our forecast residual-based filtering

In order to make the following clustering step focus on abnormal leaf attribute combinations (*i.e.*, affected by root causes), we need to detect abnormal leaf attribute combinations. Following existing work [15, 16, 17, 4], we use **forecast residuals** (*i.e.*, the difference between real and forecast values) to indicate the extent of the changes of attribute combinations and apply a threshold to decide whether the change is abnormal or not. We apply knee-point method on the cumulative distribution function (CDF) of the forecast residuals of leaf attribute combinations for automated threshold selection. It is because given a large number of leaf attribute combinations, the number of abnormal leaf attribute combinations is usually much less than the normal ones. A knee point refers to a point where the increase of filtered-out leaf attribute combinations is no longer worth the increase of the threshold. In Fig. 3, we present an example CDF of an online service system fault and its knee point (the vertical dashed line). We define a knee point as the point with maximum curvature rather than other ad-hoc definitions for genericness and robustness following existing work [18]. The advantage of the knee-point method is that it is simple, efficient, and completely automated.

4.2.2. Calculating the Distribution of Deviation Scores

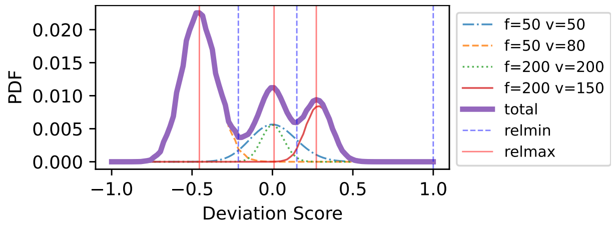


Figure 4: Illustration of *PSqueeze*'s probabilistic clustering.

As introduced in Section 3.4, leaf attribute combinations affected by the same root causes have the same deviation scores, based on which we design our clustering method. First, we estimate the distribution of all leaf attribute combinations' deviation scores. Squeeze (our previous conference version [6]) presumes the observed deviation score is correct. However, deviation scores can be significantly affected by noises from natural variation or inaccurate forecasting, especially when the real and forecast values are small. For example, given an attribute combination where $v \sim Pois(5)$ and $f = 5$, its deviation score is supposed to be 0 since the expectation of its real value is equal to its forecast value. However, its deviation score could be ± 0.222 or ± 0.182 ($v = 4, 6$ due to noise or natural variation, or $f = 4, 6$ due to inaccurate forecast), and thus it can be mistakenly grouped into incorrect clusters. Thus it is required to estimate the distribution of deviation scores in a more robust manner. Since the variation and forecasting errors can hardly be eliminated, we try to explicitly model the noises. Unlike Squeeze, *PSqueeze* considers what if the deviation scores of an attribute combination are biased and determines the probability that the leaf attribute combination should be grouped into each cluster. As a result, when the deviation score of an attribute combination is largely biased due to noises and thus is grouped into an incorrect cluster by Squeeze, *PSqueeze* groups it into the correct cluster with a certain probability. In this way, we make *PSqueeze* more robust to such noises than Squeeze.

Specifically speaking, we firstly calculate the probability density function (PDF/PMF) for each abnormal attribute combination's deviation score and then average them to obtain the overall distribution. The choice of PDF and PMF depends on our measure: we calculate PDF for continuous measures (*e.g.*, average response time) and PMF for discrete measures (*e.g.*, the number of orders). To calculate the PDF/PMF, we need to assume the distribution family according to domain knowledge about the nature of the measure. For example, considering the measure of the number of total orders, we can assume that the number of total orders v follows *Poisson distribution*, *i.e.*, $v \sim Pois(\lambda = v')$, because the speed of order arrivals keeps stable in a short duration. Then the real deviation score is supposed to be $ds = 2\frac{f-v'}{f+v'}$. Therefore, the probability mass function of ds is $P(ds = 2\frac{f-v-k}{f+v+k}) = Pois(v; \lambda = v+k)$ where v denotes the observed real value. Compared with that Squeeze considers that ds follows $P(ds = 2\frac{f-v-k}{f+v+k}) = \begin{cases} 1, & k = 0 \\ 0 & k \neq 0 \end{cases}$, it models the probability that the observed deviation score is biased due to noises. In this way, we calculate all the PDF/PMFs of leaf attribute combinations' deviation scores (*e.g.*, the light curves in Fig. 4). Then, by averaging all these probability density/mass functions together, we get the overall distribution of deviation scores of all attribute combinations (*e.g.*, the bold solid curve in Fig. 4).

In this paper, we use *Poisson distribution* for all fundamental measures, including #orders and #page views. It is because *Poisson distribution* is suitable for describing the number of event occurrences with a constant mean rate. For derived measures, we do not use probabilistic clustering, given the difficulty in finding appropriate distribution families for derived measures.

4.2.3. Determining the Cluster Boundaries

Algorithm 1 Deviation Score Based Probabilistic Clustering

```
1: procedure DensityCluster($PDF$)
2:     $centers \leftarrow$ argrelmax($PDF$)
3:     $boundaries \leftarrow$ argrelmin($PDF$)
4:     $clusters \leftarrow []$
5:     for $center$ in $centers$ do
6:         $l \leftarrow$ maximum in $boundaries$ s.t. $l < center$
7:         $r \leftarrow$ minimum in $boundaries$ s.t. $r > center$
8:         $clusters$.append($\{e \in LE(\emptyset) | d(e) \in [l, r]\}$)
    return clusters
```

Based on the distribution of deviation scores, we determine the number of clusters and the cluster boundaries. Since the deviation scores of the leaf attribute combinations affected by the same root cause should crowd around a small area, the low-density areas separate the clusters. It is a 1-dimensional clustering problem, which is special because there is no saddle point in 1-dimensional spaces, and thus *relative maximums* of the PDF/PMF represent all high-density areas. Hence applying a high-dimensional clustering method (*e.g.*, DBSCAN [19]) would be unnecessarily costive. Instead, we design a simple clustering method intuitively, as shown in Algorithm 1. First, we select the points where deviation scores crowd, *i.e.*, relative maximums of the PDF/PMF (*e.g.*, the three solid red vertical lines in Fig. 4), as the centroids of clusters.

Hence the number of clusters is equal to the number of relative maximums of the PDF/PMF. Then, since low-density areas separate the clusters, we select the nearest low-density points, *i.e.*, *relative minimums* of the PDF (*e.g.*, the dashed purple vertical lines in Fig. 4), as the boundaries of the clusters. Then, the probability of an attribute combination in a cluster is equal to the probability that its deviation score locates between the boundaries. In other words, a cluster contains the leaf attribute combinations whose PDF/PMF intersects with the area between boundaries, and the intersection area denotes the probability that the attribute combination is in the cluster.

4.3. Top-Down Localization in Each Cluster

Algorithm 2 Localization in Each Cluster

```
1: procedure InClusterLocalization($cluster$)
2:     $root\_causes \leftarrow []$
3:     for $cuboid$ in all cuboids from top to bottom do
4:         $AC \leftarrow$ sorted attribute combinations in $cuboid$ by $r_{descended}$ in descending order
5:         for $split$ in all valid splits do
6:             $score[split] \leftarrow \mathrm{GPS}(AC[:split])$
7:         $root\_cause \leftarrow AC[:\mathrm{argmax}_{split} score]$
8:         $root\_causes \leftarrow root\_causes + [root\_cause]$
9:         if $root\_cause$'s $score \geq \delta$ then
10:            Stop search next layer
11:    sort $root\_causes := \{S_i\}$ by $GPS(S_i) * C - I(S_i)$ in descending order
12:    return $root\_causes[0]$
```

The output of the bottom-up search is a list of clusters, each of which is a set of leaf attribute combinations that are affected by the same root cause. For convenience, we denote a cluster as $Cluster$. At this stage, we aim to localize root causes for each cluster. The root cause of a cluster is defined as a set of attribute combinations that is expressive when considering $Cluster$ and the normal leaf attribute combinations only and is interpretable. To overcome the challenges of huge search space, we propose an efficient heuristic search method for the in-cluster root cause localization, which contains three key techniques. 1) a cuboid-wise top-down search strategy to narrow down the search space, 2) a heuristic strategy to search a cuboid efficiently, and 3) a robust objective function to evaluate the expressiveness and interpretability of a root cause candidate. A summary of our method at this stage is presented in Algorithm 2.

4.3.1. Cuboid-Wise Search Strategy

We search for each cluster's root cause in each cuboid layer by layer (line 3). Taking Fig. 1 as an example, we would search cuboid ISP, Province, and User Agent first, then search cuboid ISP&Province, ISP&User Agent, and Province&User Agent, and finally search cuboid ISP & Province&User Agent. On the one hand, the cuboid-wise search strategy is motivated by the assumption that the root cause of a cluster is only a subset of a cuboid. The assumption is practical due to the following intuitions: 1) the attribute combinations in the cluster are affected by the same root cause; 2) in practice one root cause rarely requires more than one cuboid to describe it according to our analysis on many real-world production faults. On the other hand, we search shallow cuboids first because root cause candidates in shallower cuboids are more interpretable (see Section 2.3) than those in deeper cuboids, and thus we call it a top-down search method.

4.3.2. Heuristic Method to Search a Cuboid

If an attribute combination e is part of the root cause, then according to GRE, all of its descent leaf attribute combinations (*i.e.*, $LE(e)$) should have similar deviation scores, *i.e.*, they should all in the cluster $Cluster$. We call the ratio of descended leaf attribute combinations in the cluster of e as the *descended ratio* of e. It is denoted as

$$r_{descended}(e) = \frac{\sum_{e' \in LE(e) \cap Cluster} p(e' \in Cluster)}{\sum_{e' \in LE(e)} p(e' \in Cluster)} \tag{7}$$

, where $p(e' \in Cluster)$ denotes the probability that $Cluster$ contains e' (see Section 4.2.3). For example, in Table 2, supposing that the cluster contains the first two rows in the table, and we are searching cuboid Province now, the descended ratio of ($Province = Beijing$) is 1 and those of others are 0. We sort the attribute combinations of a cuboid by their descended ratios in descending order (line 4). In this way, the attribute combinations at the front of the list are more likely to be part of the root cause than those at the back.

4.3.3. Generalized Potential Score

Then, we aim to find the top-k items in the sorted attribute combination list of a cuboid as the root cause candidate of the cuboid (line 7). For this purpose, we propose *generalized potential score* (GPS), to evaluate how likely a set of attribute combination (S) is the root cause in the following aspects: 1) it is expressive, *i.e.*, the real and forecast values of its descended leaf attribute combinations (*i.e.*, $LE(S) = \bigcup_{e \in S} LE(e)$) should be different, and those of other leaf attribute combinations should be close; 2) it follows GRE, *i.e.*, the real values of $LE(S)$ should be close to the corresponding expected abnormal values (see Section 3.4). We do not evaluate interpretability here because in the same cuboid, adding extra attribute combinations reduce expressiveness and interpretability simultaneously. Comparing real and expected abnormal values helps to filter the false expressive candidates caused by inaccurate forecasting and noise and make *PSqueeze* more robust.

We measure the difference between the real and forecast values of $LE(S)$ by normalized Manhattan distance, *i.e.*, $d_1(\boldsymbol{v}(S), \boldsymbol{f}(S)) = \frac{1}{|LE(S)|} \sum_{e \in LE(S)} |v(e) - f(e)|$, and the difference between the real and expected abnormal values by $d_1(\boldsymbol{v}(S), \boldsymbol{a}(S)) = \frac{1}{|LE(S)|} \sum_{e \in LE(S)} |v(e) - a(e)|$. Similarly, for other leaf attribute combinations (denoted as

$LE(S'))$, the difference between the real and forecast values is $d_1(\boldsymbol{v}(S'), \boldsymbol{f}(S')) = \frac{1}{|LE(S')|}\sum_{e \in LE(S')}|v(e) - f(e)|$. Considering that we are conducting in-cluster localization and there can be other root causes, $LE(S')$ does not contain the leaf attribute combinations in other clusters. Based on these, we define GPS as follows:

$$GPS = 1 - \frac{d_1(\boldsymbol{v}(S), \boldsymbol{a}(S)) + d_1(\boldsymbol{v}(S'), \boldsymbol{f}(S'))}{d_1(\boldsymbol{v}(S), \boldsymbol{f}(S))) + d_1(\boldsymbol{v}(S'), \boldsymbol{f}(S'))} \tag{8}$$

GPS robustly indicates S's expressiveness, even if the anomaly magnitude of S is insignificant. Considering *potential score* [4] and *explanation power* [10, 11], the forecast residuals of $LE(S')$ would accumulate as the size of $LE(S')$ increases, and thus they cannot reflect the expressiveness when the anomaly magnitude of S is insignificant. For example, in Table 2, the GPS of the first two rows in bold is 0.743, while the potential score is 0.303 and the explanation power is 0.457.

4.3.4. Selecting among Candidates from Different Cuboids

Finally, we select the root cause with both high expressiveness and interpretability from the candidates found in the cuboids. For this purpose, we quantitively define the interpretability of S as $I(S) = \sum_{e \in S}|e|^2$. For example, $I(\{(\text{Province=Beijing})\}) = 1$ and $I(\{(\text{Province=Beijing} \wedge \text{ISP=CUnicom}), (\text{Province=Beijing} \wedge \text{ISP=CMobile})\}) = 8$. We use a weight C to trade off expressiveness (measured by GPS) and interpretability (line 11). To calculate C automatically, we employ an empirical formula, which can achieve good effectiveness:

$$\begin{aligned} g_{cluster} &= \log(num\_cluster + 1)/num\_cluster \\ g_{attribute} &= num\_attr/\log(num\_attr + 1) \\ g_{coverage} &= -\log(\text{coverage of abnormal leaves}) \\ C &= g_{cluster} \times g_{attribute} \times g_{coverage} \end{aligned} \tag{9}$$

The intuition is that if there are fewer clusters or more attributes, or the cluster contains fewer abnormal leaf attribute combinations, then GPS is more important than interpretability. It is refined based on that in our previous conference version [6], and the original version can be negative in some cases and causes errors.

Furthermore, for efficiency, if the candidates' GPS scores exceed a given threshold δ at a certain layer of cuboids, *PSqueeze* would stop searching deeper layers (line 9). We set $\delta = 0.9$ by default and discuss its impact in Section 5.5.

4.4. Determine External Root Cause

PSqueeze determines external root causes by examining whether the localized root causes are expressive enough, *i.e.*, have high GPS scores. When there are external root causes, since it is impossible for algorithms to localize the real root causes, the GPS scores of the localized root causes would be relatively low. Specifically speaking, we check whether the GPS scores of the clusters is less than

Algorithm 3 Determine External Root Cause

```
1: procedure DETERMINE_EXRC($rc\_list$)
2:     $minGPS \leftarrow +\infty$
3:     for $S$ in $rc\_list$ do
4:         $minGPS \leftarrow \min(GPS(S), minGPS)$
       return $minGPS \geq \delta_{ExRC}$
```

an automated threshold (denoted by δ_{ExRC}). We denote the minimum GPS scores of all per-cluster root cause as min_GPS. If $min\_GPS < \delta_{ExRC}$, there is at least one cluster where *PSqueeze* is not able to find a good enough root cause, which indicates that there probably exist external root causes. Then the operators will be informed that the results of *PSqueeze* can be misleading due to external root causes. If $min\_GPS \geq \delta_{ExRC}$, then *PSqueeze* localizes good enough root causes for all selected clusters, then there are not external root causes, and thus the final result is reliable.

The threshold δ_{ExRC} is automatically selected by historical data. The key idea is that for those faults without external root causes, their min_GPS, which represents the minimum GPS of all per-cluster root causes, should be near 1. It is because a good root cause should be localized for each cluster, and thus they would be grouped into a cluster by the density-based clustering method in Section 4.2. Therefore, given those min_GPS (defined in Algorithm 3) of historical faults, we firstly cluster them with Algorithm 1, and then use the lower boundary of the cluster with the largest centroid as δ_{ExRC}. Note that we **do not** need to know which faults encounter external root causes, *i.e.*, the automated threshold selection is unsupervised. If there are not enough historical faults, we use a default value 0.8 for δ_{ExRC}.

5. Evaluation

In this section, we conduct extensive experiments to evaluate the localization accuracy and efficiency of *PSqueeze* based on both simulated and injected faults.

5.1. Experiment Settings

5.1.1. Datasets

We have two real-world datasets collected from two production systems of two companies in several weeks. One is from an online shopping platform (denoted as $\mathcal{I}_1$), and the other one is from an Internet company (denoted by $\mathcal{I}_2$). Although these are real-world data, it is hard to obtain enough real-world anomalies and the corresponding root causes as the ground truth. Therefore, we generate simulated faults according to GRE based on these real-world datasets to evaluate *PSqueeze* as follows:

1. We select a time point from the time series and add different Gaussian noises to the real values of all leaves. It is used to emulate different forecast residuals.

2. We randomly choose $n\_element \in \{1, 2, 3\}$ (a.k.a. n_ele for short) cuboids *with replacement* in layer $cuboid\_layer$, which is randomly chosen from $\{1, 2, 3\}$.

3. We randomly choose $n\_element$ different attribute combinations from every selected cuboid, which are the root-cause attribute combinations for this simulated fault. Note that the datasets with only different forecast residuals (*i.e.*, $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$) share the same time points but have different root-cause attribute combinations.

4. For each selected root-cause attribute combination, we modify the real values of its descended leaf attribute combination by GRE with a random magnitude. Particularly, for $\mathcal{D}$, the measure of which is success rate, we firstly modify their success rates according to GRE. Then, we randomly generate the total order numbers and successful order numbers according to the success rates.

5. We add extra Gaussian noises ($\mathcal{N}(0, 5\%)$) to these descended leaf attribute combinations since GRE would not perfectly hold in practical faults.

6. We drop invalid "faults" when

   (a) there exists another attribute combination that shares the same (or very similar) set of descended leaf attribute combinations with a selected root-cause attribute combinations;

   (b) the Gaussian noises added in normal leaf attribute combinations (*i.e.*, those leaf attribute combinations that are not descended from any selected root-cause attribute combination) are so large that the overall measure value of them is abnormal.

The root cause attribute combinations with different deviation scores are simulating independent multiple root causes. Note that the deviation score of different root-cause attribute combinations could be the same sometimes. In such cases, the root-cause attribute combinations with the same deviation score would be considered as one root cause containing multiple root-cause attribute combinations. In all datasets, we add anomalies with random magnitudes, and thus, the anomalies are not guaranteed to be significant.

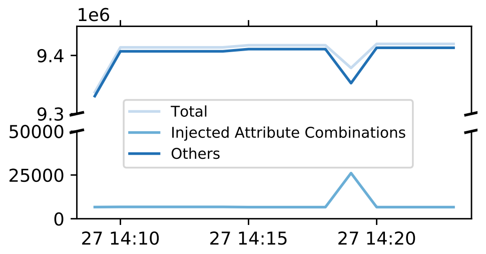


Figure 5: The measure value of attribute combinations that are not intended to simulate faults on is significantly abnormal.

This paper extends the simulation process in our previous version [6] by adding step 6. The process described in step 1∼5 could generate inappropriate faults for evaluation. For example, consider that we randomly choose (A=a1∧B=b1∧C=c1) as the root-cause attribute combination, and b1 is related to c1. In such cases, (A=a1∧B=b1∧C=c1) (A=a1∧B=b1) and (A=a1∧C=c1) have almost the same set of leaf attribute combinations. Therefore, it would be unreasonable to take (A=a1∧B=b1∧C=c1) as the root cause rather than (A=a1∧C=c1) or (A=a1∧B=b1). Nevertheless, as we add Gaussian noises in all leaf attribute combinations, thus the attribute combinations that are not intended to simulate faults on would become abnormal as well, as shown in Fig. 5. To tackle this problem, unlike our previous version [6], we remove invalid faults by step 6.

Table 4: Summary of Datasets

	n	$\lvert LE(\emptyset)\rvert$	$\lvert\mathcal{P}(E)\rvert$	**Source**	**Measure**	**Residual**	**EP**
$\mathcal{A}$	5	15324	2^{75888}	$\mathcal{I}_1$	F: #orders	3.92%	90.3%
$\mathcal{B}_1$	4	21600	2^{31338}	$\mathcal{I}_2$	F: #page views	3.97%	74.2%
$\mathcal{B}_2$	4	21600	2^{31338}	$\mathcal{I}_2$	F: #page views	7.96%	60.6%
$\mathcal{B}_3$	4	21600	2^{31338}	$\mathcal{I}_2$	F: #page views	11.9%	56.1%
$\mathcal{B}_4$	4	21600	2^{31338}	$\mathcal{I}_2$	F: #page views	15.9%	53.7%
$\mathcal{D}$	4	13806	2^{21534}	$\mathcal{I}_2$	D: success rate	3.99%	59.3%
$\mathcal{E}$	9	373	2^{373}	-	D: average latency	37.3%	89.5%
$\mathcal{F}$	9	373	2^{373}	-	D: stall rate	45.8%	86.9%

Simulated fault datasets. By the new simulation method, we get 6 new simulated fault datasets with different simulation parameters. In Table 4, we describe some basic statistics of our datasets. For each combination of n_ele and $cuboid\_layer$, we simulated 100 faults in each dataset. Hence there are 5400 faults in total. In Table 4, n denotes the number of attributes, $|LE(\emptyset)|$ denotes the number of all leaf attribute combinations, and $|\mathcal{P}(E)|$ denotes the number of all root cause candidates. These datasets contain three different measures, including both fundamental (denoted as F in Table 4) and derived measures (D), and all of them are of common golden signals [14]. In practice, we do not count the attribute combinations that never occur in data, and thus $|LE(\emptyset)|$ and $|\mathcal{P}(E)|$ seem lower than theoretical results. Residual in Table 4 denotes the average forecasting residuals in percent of all normal leaf attribute combinations. EP (explanation power [10, 11]) denotes the fraction of total forecast residual of all abnormal leaf attribute combinations over that of both normal and abnormal ones.

Injected fault datasets. Furthermore, compared with the previous conference version, this paper also two new datasets, named $\mathcal{E}$ and $\mathcal{F}$, based on more realistic fault injection (rather than directly adjusting measure values). More specifically, we deploy Train-Ticket [20], which is one of the largest open-source microservice benchmark systems and is widely used in literature [21, 22, 20, 23, 24], on a Kubernetes cluster with five servers. We utilize Istio [25] and ChaosMesh [26] to inject the following types of faults onto the system: delaying or dropping packets or HTTP requests/responses sent to specific pods/APIs/services or containing specific parameters. We injected 73 faults in total. Then, we collect detailed information on every HTTP request between the microservices with Jaeger and Istio, including client/server service name, URL, response time, status code, etc. Based on the request details, we collect two types of derived measures to build the two datasets, i.e., average latency (=total latency of all requests / the number of requests) and stall rate (=the total number of stall requests/the number of requests). In both datasets, the attributes and number of distinct attribute values are as follows: client service name (17), pod (64), method (3), URL prefix (71), station name (10), train type (6), start station (9), end station (10). Compared with the simulated faults (i.e., $\mathcal{A}$, $\mathcal{B}_*$, and $\mathcal{D}$), the diversity of root causes is limited with respect to *n_elements* and *cuboid_layer*, but the injected faults do not rely on any assumption and are much more representative of real-world system failures.

5.1.2. Evaluation Metrics

F1-score is used in this paper to evaluate root cause localization for multi-dimensional data. We denote the root cause reported by the algorithm as S and the ground truth as $\hat{S}$. Then $tp=|S \cap \hat{S}|$ denotes the number of true positive root-cause attribute combinations, $fp=|S-\hat{S}|$ denotes the number of false positives, and $fn=|\hat{S}-S|$ denotes the number of false negatives. Then F1-score is defined as:

$$F1\text{-}Score=(2 \times tp)/(2 \times tp + fp + fn) \tag{10}$$

To extensively study the performance of *PSqueeze* under various situations, following existing work [4], we evaluate F1-scores separately for different root cause settings, *i.e.*, different *n_elements* and *cuboid_layer*.

To evaluate *PSqueeze* on determining external root causes, we also use the widely-used F1-score. We denote the set of faults with external root causes as $\hat{F}$ and the set of faults that are reported by our algorithm to have external root causes as F. Then ExRC_F1-score (**ex**ternal **r**oot **c**ause F1-score) is calculated as follows:

$$ExRC\_F1\text{-}Score = 2 \times \frac{precision \times recall}{precision + recall}$$

where $precision = |F \cap \hat{F}|/|F|$ denotes the probability that a determined external root cause is true and $recall = |F \cap \hat{F}|/|\hat{F}|$ denotes the fraction of external root cause cases that have been determined.

Finally, we also evaluate the time efficiency of *PSqueeze*. In the following experiments, we present the average running time of all cases in the corresponding setting.

5.1.3. Baseline Approaches

We compare *PSqueeze* with the following baseline approaches, which are summarized in Table 11:

- Squeeze [6] (SQ), our previous conference version.
- Adtributor [10] (ADT) assumes root causes involve only single attributes and mines all attribute combinations with high *explanation power* and then sorts them by *surprise*.
- R-Adtributor [11] (RAD) recursively calls Adtributor to localize multi-attribute root causes.
- Apriori (APR) is a popular frequent pattern mining algorithm [27]. Ahmed et al. [7] and Lin et al. [3] take association rules of abnormal leaf attribute combinations as root causes, and they use Apriori and *confidence* to mine association rules.
- HotSpot+GRE [4] (HS) uses Monte Carlo tree search (MCTS) to search the set of attribute combinations with the highest *potential scores*. We adapt the original HotSpot for derived measures according to GRE.
- MID [5] searches for the attribute combinations that maximize their objective function, which is similar to that in iDice [8], and uses a heuristic based on entropy to speed up the search. As we found that their objective function is limited to their scenario (# issue reports) and performs poorly with general multi-dimensional data, we replace their objective function with our GPS.
- ImpAPTr [28, 9] (IAP) search for attribute combinations that maximize *impact factor* and *diversity factor* with BFS (breath-first search). Since ImpAPTr only ranks attribute combinations rather than decide which are the root-cause attribute combinations, we take the top-*n_ele* ranked attribute combinations as root-cause attribute combinations. Besides, the original impact factor works for decreasing measure values, and thus, we modify it by deciding the sign of impact factor adaptively for each fault.

We do not compare with iDice [8] due to its inferior performance in our scope according to our previous version [6] and MID [5]. We set $\delta = 0.9$ for all cases. The parameters of other algorithms are set following the original papers. Note that all approaches except ImpAPTr have no idea about *n_ele* or *cuboid_layer* of the faults.

Table 5: Time Usage Comparison of Forecast Methods

Algorithm	MA	Period [29]	ARIMA [10]
Time Usage (μs)	6.21($\pm$2.13)	28.7($\pm$3.95)	38672($\pm$32312)

In our evaluation, we always apply MA (moving average) for forecasting. Specifically speaking, we calculate the forecast value of a leaf attribute combination e at a specific time point t_0 by averaging the real values of e at $t_{-10}, t_{-9}, ..., t_{-1}$. We choose MA because MA is one of the simplest forecast algorithms and costs little time. We present the time usage for a single leaf attribute combination of several algorithms used by existing work in Table 5.

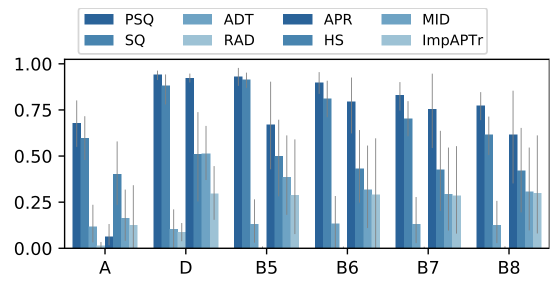


Figure 6: The F1-scores on simulated datasets.

Table 6: The p-value (by t-test) and effect size (by Cohen's d [30]) of every baseline across all datasets

	SQ	ADT	RAD	APR	HS	MID	IAP
p-value	7.9e-03	1.1e-41	6.2e-65	3.8e-04	1.9e-12	1.7e-18	3.3e-18
effect size	0.52	4.28	7.55	0.71	1.53	2.05	2.03

5.2. RQ1: Effectiveness in Root Cause Localization

As shown in Fig. 6, on average of different *n_element* and *cuboid_layer* settings, *PSqueeze* achieves the highest performance in all datasets and outperforms the baselines significantly. By further calculation, the F1-score of *PSqueeze* outperforms the baselines (excluding Squeeze) by 32.89% at least on average of all settings. The improvement is significant according to Table 6. Moreover, *PSqueeze* is more robust and performs well consistently in different situations. As shown in Table 7 and Table 8, despite of different *n_elements*, *cuboid_layer*, and dataset settings *PSqueeze* achieves good performance, and *PSqueeze* outperforms all the other baselines in most (31 out of 54) settings.

In Fig. 7, we present the F1-scores with different numbers of root causes on the simulated fault datasets. The results show that *PSqueeze* can achieve consistently good performance even if there is more than one root cause.

Notably, *PSqueeze* performs well consistently regarding different *anomaly magnitudes*. Formally speaking, the anomaly magnitude of a fault is $\frac{|\sum_{e\in LE(\emptyset)}(v(e)-f(e))|}{|\sum_{e\in LE(\emptyset)} f(e)|}$, and it denotes the relative magnitude of the abnormal fluctuation on the overall measure value. As shown in Fig. 8,

Table 7: Overall performance comparison on $\mathcal{A}$ and $\mathcal{D}$.

F1-Score		**n_element,cuboid_layer**									
	Algo.	**total**	**1,1**	**1,2**	**1,3**	**2,1**	**2,2**	**2,3**	**3,1**	**3,2**	**3,3**
$\mathcal{A}$	PSQ	**0.68**	**0.97**	**0.93**	**0.70**	0.73	**0.67**	**0.67**	0.57	0.60	0.28
	SQ	0.60	0.94	0.75	0.52	0.68	0.65	0.57	0.37	**0.60**	**0.29**
	ADT	0.12	0.37	0.00	0.00	0.36	0.00	0.00	0.32	0.00	0.00
	RAD	0.01	0.06	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.05
	APR	0.06	0.31	0.03	0.00	0.05	0.02	0.07	0.06	0.03	0.00
	HS	0.40	0.71	0.35	0.24	**0.88**	0.27	0.01	**0.58**	0.32	0.25
	MID	0.16	0.69	0.32	0.03	0.00	0.23	0.01	0.00	0.19	0.01
	IAP	0.12	0.87	0.00	0.00	0.02	0.00	0.00	0.22	0.00	0.00
$\mathcal{D}$	PSQ	**0.94**	0.95	**0.96**	**0.97**	**0.97**	**0.97**	**0.95**	0.84	**0.94**	**0.95**
	SQ	0.88	0.95	0.95	0.94	0.95	0.93	0.90	0.48	0.93	0.90
	ADT	0.10	0.16	0.00	0.00	0.32	0.00	0.00	0.46	0.00	0.00
	RAD	0.09	0.14	0.12	0.00	0.06	0.18	0.00	0.09	0.20	0.01
	APR	0.92	0.96	0.93	0.90	0.95	0.96	0.86	**0.96**	0.91	0.89
	HS	0.51	**0.99**	0.91	0.01	0.75	0.70	0.00	0.64	0.58	0.02
	MID	0.51	0.95	0.68	0.25	0.67	0.57	0.27	0.51	0.44	0.28
	IAP	0.30	0.41	0.43	0.04	0.50	0.34	0.04	0.64	0.26	0.01

Table 8: Overall performance comparison on $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$.

F1-Score		**n_element,cuboid_layer**									
	Algo.	**total**	**1,1**	**1,2**	**1,3**	**2,1**	**2,2**	**2,3**	**3,1**	**3,2**	**3,3**
$\mathcal{B}_1$	PSQ	**0.93**	**1.00**	**1.00**	**0.98**	**0.99**	**0.90**	0.89	0.96	0.80	0.85
	SQ	0.91	0.94	1.00	0.96	0.97	0.88	**0.91**	0.89	0.79	**0.87**
	ADT	0.13	0.29	0.00	0.00	0.43	0.00	0.00	0.45	0.00	0.00
	RAD	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.01
	APR	0.67	1.00	0.78	0.09	0.98	0.87	0.18	**0.96**	**0.93**	0.23
	HS	0.50	0.97	0.72	0.14	0.77	0.50	0.10	0.78	0.47	0.05
	MID	0.38	0.94	0.82	0.27	0.01	0.56	0.23	0.00	0.43	0.19
	IAP	0.29	1.00	0.00	0.00	0.86	0.00	0.00	0.71	0.00	0.00
$\mathcal{B}_2$	PSQ	**0.90**	1.00	**0.99**	**0.93**	**1.00**	0.91	**0.80**	0.93	0.80	**0.74**
	SQ	0.81	0.91	0.97	0.93	0.99	0.78	0.75	0.78	0.53	0.66
	ADT	0.13	0.28	0.00	0.00	0.44	0.00	0.00	0.49	0.00	0.00
	RAD	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01
	APR	0.79	0.97	0.91	0.36	1.00	**0.93**	0.49	**0.96**	**0.96**	0.58
	HS	0.43	0.97	0.60	0.18	0.73	0.43	0.03	0.68	0.27	0.00
	MID	0.32	0.96	0.86	0.26	0.01	0.42	0.11	0.01	0.18	0.05
	IAP	0.29	**1.00**	0.00	0.00	0.86	0.00	0.00	0.76	0.00	0.00
$\mathcal{B}_3$	PSQ	**0.83**	0.88	**0.93**	**0.86**	0.95	0.85	**0.74**	0.93	0.74	**0.59**
	SQ	0.70	0.61	0.93	0.84	0.83	0.75	0.69	0.66	0.49	0.53
	ADT	0.13	0.28	0.00	0.00	0.41	0.00	0.00	0.50	0.00	0.00
	RAD	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.00
	APR	0.75	**1.00**	0.88	0.24	**0.99**	**0.93**	0.38	**0.97**	**0.96**	0.44
	HS	0.43	0.98	0.62	0.08	0.76	0.39	0.02	0.71	0.27	0.00
	MID	0.29	0.93	0.83	0.22	0.01	0.39	0.09	0.00	0.14	0.03
	IAP	0.29	0.99	0.00	0.00	0.84	0.00	0.00	0.74	0.00	0.00
$\mathcal{B}_4$	PSQ	**0.77**	0.69	**0.89**	**0.76**	0.91	**0.80**	**0.72**	0.92	0.68	**0.58**
	SQ	0.62	0.32	0.86	0.75	0.69	0.70	0.65	0.58	0.46	0.54
	ADT	0.13	0.27	0.00	0.00	0.37	0.00	0.00	0.51	0.00	0.00
	RAD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.01
	APR	0.61	**1.00**	0.72	0.05	**1.00**	0.78	0.09	**0.96**	**0.81**	0.12
	HS	0.42	1.00	0.71	0.10	0.71	0.30	0.02	0.70	0.26	0.00
	MID	0.31	0.97	0.84	0.25	0.00	0.41	0.09	0.00	0.15	0.05
	IAP	0.30	1.00	0.00	0.00	0.93	0.00	0.00	0.74	0.00	0.00

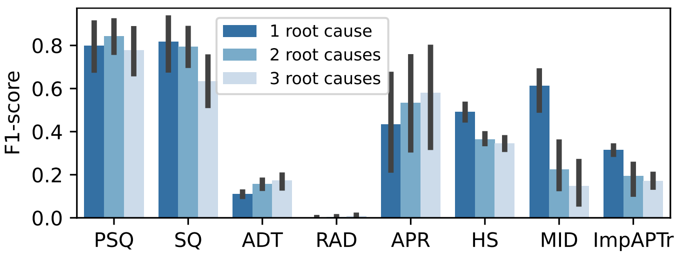


Figure 7: The F1-scores with different numbers of root causes

PSqueeze and Squeeze achieve high performance in spite of anomaly magnitudes. However, the performance of Adtributor, HotSpot, and ImpAPTr varies largely with different anomaly magnitudes. Though the performance of Apriori, R-Adtributor, and MID is relatively stable, their F1-scores are not high enough.

PSqueeze, as well as many most other baselines, relies on forecast values. As shown in Table 8, *PSqueeze* performs worse as the forecast residual goes large (from $\mathcal{B}_1$ to $\mathcal{B}_4$, as shown in Table 4), but *PSqueeze* consistently outperforms others even in the worst case. The performance of Apriori is not monotonic to the forecasting residuals because its parameter setting is sensitive to the forecasting residual. Some approaches like HotSpot seem affected less by forecasting residuals, but they perform poorly. It worth noting that *PSqueeze* is much more insensitive to forecasting residuals than our previous version, Squeeze,

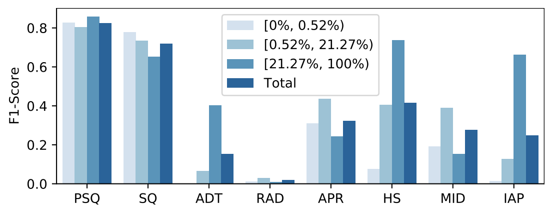


Figure 8: The F1-scores of faults with different anomaly magnitudes

which further demonstrates the effectiveness of our extension (the probabilistic clustering method in Section 4.2).

In those settings where *PSqueeze* does not achieve the best performance, either Squeeze (3 out of 54), Apriori (14 out of 54), HotSpot (3 out of 54), or ImpAPTr (2 out of 54) achieve the best performance. However, the baseline approaches are not general and robust enough to perform well in different situations. Adtributor only localizes root causes of the first-layer cuboids, and thus, it cannot work at all settings where *cuboid_layer* $\neq 1$. Its *explanation power* is sensitive to the impact of attribute combinations (*i.e.*, how much data is specified by attribute combinations), and thus it performs badly when anomaly magnitudes are small. Note that faults with root causes in deeper cuboids often have smaller anomaly magnitudes. Hence the performance of Adtributor decreases as *cuboid_layer* increases. Although R-Adtributor localizes root causes in any cuboids, it is hard for R-Adtributor to decide when the recursion should terminate, and thus it works poorly in most settings. Although Apriori achieves the best performance in several settings, it also works extremely badly in some settings (*e.g.*, all settings on $\mathcal{A}$, *cuboid_layer* $= 3$ on $\mathcal{B}_1$) due to its sensitivity to parameters. The *hierarchical pruning strategy* could wrongly prune the right search path, and thus it performs poorly when *n_ele* or *cuboid_layer* is large. Though both MID and ImpAPTr are designed for and limited to particular measures (the number of issue reports and success rate respectively), we have adapted them in our experiments. However, they still suffer from other limitations. The searching strategies of MID and ImpAPTr do no consider multiple root causes, and thus, they perform poorly when *n_ele* > 1. Moreover, MID's heuristic strategy is limited to their scenario, and thus, when root-cause attribute combinations are in deeper cuboids, it is harder for MID to search it. The *contributor power* in ImpAPTr is sensitive to the impact of attribute combinations like *explanation power*, and thus it is inappropriate for faults with small anomaly magnitudes. While the baseline approaches suffer from these limitations, *PSqueeze* is able to consistently achieve good performance in different situations.

Finding 1 *PSqueeze* is more general and robust to perform consistently well in different situations.

PSqueeze underperforms other baselines, especially Apriori, in some settings when the forecasting residuals are large (*i.e.*, in datasets $\mathcal{B}_3$ and $\mathcal{B}_4$), or *n_elements* and *cuboid_layers* are large. It is mainly because, in such cases, the bottom-up clustering step is affected by the noises. For example, some normal leaf attribute combinations could be grouped into a cluster due to the large forecasting residuals. To reduce the influence of noises, we apply probabilistic clustering in *PSqueeze*. As a result, as shown in Table 8, the larger the forecasting residuals are, the more *PSqueeze* outperforms Squeeze. In general, *PSqueeze* outperforms Squeeze in most settings (50 out of 54) and the F1-score of *PSqueeze* outperforms Squeeze by over 30% in 11 out of 54 settings, and the improvement can be up to 113.7%. As the forecasting residuals increase from $\mathcal{B}_1$ to $\mathcal{B}_4$, the improvement of *PSqueeze* over Squeeze increases from 1.96% to 25.22%, and the effect sizes (Cohen's d [30]) are 0.27, 0.67, 0.95 and 1.10 respectively. Moreover, according to Table 6, the improvement is significant. By further calculation, the F1-score of *PSqueeze* outperforms Squeeze by 11.73% on average.

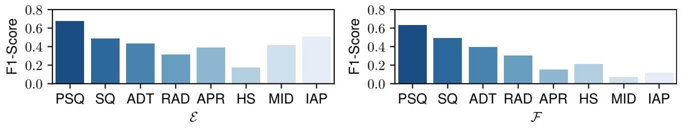


Figure 9: The F1-scores on datasets $\mathcal{E}$ and $\mathcal{F}$

Finding 2 *PSqueeze* significantly outperforms our previous version by reducing the influence of noises.

In Fig. 9, we present the F1-scores on $\mathcal{E}$ and $\mathcal{F}$. The results show that *PSqueeze* outperforms the baselines, including Squeeze, and achieves the best performance. Compared with the simulated faults (i.e., $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$), the performance of *PSqueeze* slightly degrades on the inject faults. It is probably because the data generation of $\mathcal{E}$ and $\mathcal{F}$ do not assume GRE holds at all.

Finding 3 *PSqueeze* can achieve good performance and outperforms the baselines in realistic faults.

5.3. RQ2: Effectiveness in Determining External Root Causes

Table 9: Effectiveness on Determining External Root Cause

ExRC_F1-Score			**n_element,cuboid_layer**								
	d_{ex}	**total**	**1,1**	**1,2**	**1,3**	**2,1**	**2,2**	**2,3**	**3,1**	**3,2**	**3,3**
$\mathcal{A}$	1	0.78	0.65	0.73	0.95	0.91	0.82	0.92	0.61	0.84	0.62
	2	0.82	0.67	0.97	0.95	0.78	0.84	0.93	0.72	0.89	0.60
	3	0.92	0.97	0.96	0.96	0.88	0.88	0.96	0.75	0.95	0.94
$\mathcal{B}_1$	1	0.91	1.00	0.97	0.96	0.92	0.72	0.97	0.86	0.82	0.93
	2	0.99	1.00	1.00	0.99	1.00	0.99	1.00	0.99	0.94	0.98
	3	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00
$\mathcal{B}_2$	1	0.89	1.00	0.99	0.94	0.87	0.85	0.89	0.80	0.73	0.90
	2	0.99	1.00	0.99	1.00	1.00	0.99	0.98	0.97	0.98	1.00
	3	0.98	1.00	0.99	1.00	0.97	0.99	0.97	0.95	0.98	1.00
$\mathcal{B}_3$	1	0.85	0.81	0.97	0.89	0.87	0.84	0.95	0.78	0.68	0.86
	2	0.93	0.81	1.00	0.94	0.88	0.96	0.95	0.88	1.00	0.97
	3	0.95	0.91	1.00	0.95	0.91	0.99	0.95	0.91	0.99	0.98
$\mathcal{B}_4$	1	0.77	0.74	0.91	0.93	0.76	0.67	0.77	0.68	0.70	0.75
	2	0.89	0.71	0.94	0.93	0.79	0.96	0.93	0.78	0.98	0.95
	3	0.89	0.71	0.96	0.93	0.75	1.00	0.96	0.81	0.97	0.95

The number of eliminated attributes is denoted as d_{ex}.

Our method of determining external root causes works well on different datasets. We randomly select and eliminate some attributes in the original datasets to emulate external root causes. If a root cause contains any attribute that is eliminated, it becomes an external root cause. As shown in Table 9, *PSqueeze* successfully determines external root causes with high F1-scores in almost all settings. In 110 out of 135 settings, the $ExRC\_F1{-}Score$ of *PSqueeze* achieves over 0.80. By further calculation, the $ExRC\_F1{-}Score$ of *PSqueeze* achieves 0.90 on average. To the best of our knowledge, there is not any existing approach determining external root causes, and thus we do not compare with existing approaches. In some cases, the F1-scores are relatively low (*e.g.*, 0.60). The main reason is that there could be high-GPS attribute combinations even if the exact root causes are external due to the correlation among attributes.

Finding 4 *PSqueeze* can effectively determine external root causes.

5.4. RQ3: Efficiency

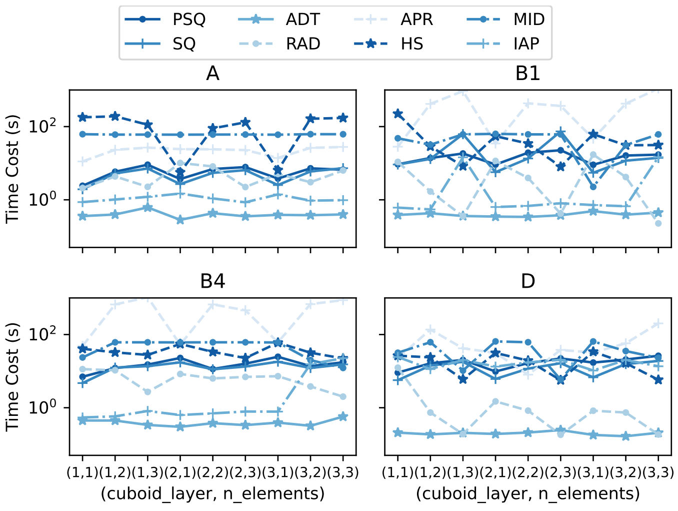


Figure 10: Running time comparison (for individual faults) of $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_4, \mathcal{D}$.

We evaluate the efficiency of *PSqueeze* by comparing its average time cost of each fault case with that of others. We run every experiment on a server with 24 × Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz (2 sockets) and 64G RAM. All algorithms are implemented with Python utilizing matured libraries like Pandas and NumPy. Experiments of all algorithms are conducted under the same condition. In Fig. 10 we present the running times of $\mathcal{A}, \mathcal{B}_1, \mathcal{B}_4, \mathcal{D}$. We do not present results of all datasets because all of $\{\mathcal{B}_i, i = 1, 2, 3, 4\}$ have similar results. *PSqueeze* costs only about ten seconds even in the worst cases. It is efficient enough since measures are usually collected every minute or every five minutes. HotSpot is sometimes as efficient as *PSqueeze*, but sometimes it would cost more time. Apriori costs hundreds of seconds, which is so slow that impractical. Others can be fast, but they do not effectively localize root causes.

Finding 5 *PSqueeze* is efficient enough in practice to localize root causes for multi-dimensional data.

5.5. RQ4: Performance under Different Configurations

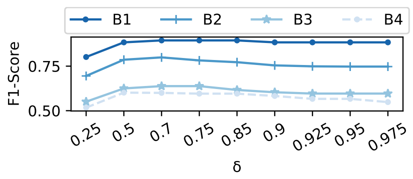


Figure 11: F1-Scores over different δ with $cuboid\_layer = 3, n\_ele = 3$.

All the parameters in Section 4 are automatically configured except δ, the GPS threshold. We present the F1-scores of *PSqueeze* under different δ in Fig. 11 with $cuboid\_layer = 3, n\_ele = 3$ in $\mathcal{B}_i$, $i = 1, 2, 3, 4$. We only choose this setting because it is the hardest setting. *PSqueeze*'s performance does not change a lot as δ changes. Results in Table 7, Table 8 and Section 5.4 also show that $\delta = 0.9$ leads to good enough effectiveness and efficiency. Since δ is the GPS threshold for early stopping and the distributions of GPS are not supposed to be related to datasets, it is reasonable to set δ near 0.9 regardless of the specific dataset.

Finding 6 *PSqueeze* is robust to different configiurations.

6. Success stories

We have successfully applied *PSqueeze* in several large commercial banks and a top Internet company. The results show that *PSqueeze* can do great help for operators by rapidly and accurately localizing root causes. In this section, we present some representative success stories. For confidential reasons, some details are omitted or anonymized. Compared with our previous version, we apply *PSqueeze* in more production systems from different companies and collect more cases (*e.g.*, case 4 and 5).

6.1. Case 1: Insignificant Anomaly Magnitude

One day night, a fault occurred at a top Internet company. The HTTP error counts suddenly burst, as shown in Fig. 12. The attributes and the number of distinct values are listed as follows: data center (11), province (7), ISP (6), user agent(22). A potential root cause is manually found by the operators, consisting of only one attribute combination ($AC1$ in Fig. 12), which took them one hour.

We retrospectively ran *PSqueeze* over this system's logs, and in several seconds, we found more root causes: $AC1$ and $AC2$, as shown in Fig. 12. It is obvious that $AC2$ also has severe error bursts. $AC2$ is mistakenly ignored by the operators because it occupied only a small fraction of the total error counts. Manual analysis apparently has difficulties in localizing root causes of anomalies with insignificant magnitudes. *PSqueeze* would help the operators to notice such root causes efficiently.

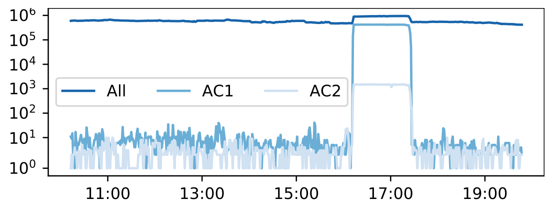


Figure 12: Measure values (in log-scale) along time of *Case 1*. $AC1$ is (user agent=uc$\wedge$idc=ih$\wedge$province=other), and $AC2$ is (user agent=uc$\wedge$idc=is$\wedge$province=other).

6.2. *Case 2: Intra-System localization*

One day from 9:00 to 11:00, the operators of a bank received many tickets and alerts and noticed that the API call success rate of a system suffered a severe drop. The search space is large, and the attributes and the number of distinct values are listed as follows: province (38), agency (815), server group (16), channel (4), server (339), code (4), status (2), service type (3). After two hours of fruitless manual root cause localization, the operators decided to just roll back the entire system to the last version, which happened to actually fix the issue. After the roll-back, it took another 2 hours for an inexperienced operator on duty to eventually find the root cause (there was a bug in the newly deployed version of the software for Service Type 0200020) based on the 2-hour logs during the fault.

Upon the request of the operators, we retrospectively ran *PSqueeze* over this system's logs during the fault. *PSqueeze* took a few seconds to report the root cause (Service Type=020020), which indicates exactly the software with a buggy version update. Had *PSqueeze* been used immediately after the fault happened, operators could have localized the root cause much faster.

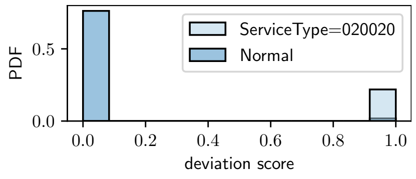


Figure 13: The histogram of deviation scores in *Case 2*.

Fig. 13 shows this case's results of deviation-score-based clustering. We can see that the deviation scores of all descended leaf attribute combinations of the root cause (Service Type=020020) are very close to each other. This to some extent confirms the generalized ripple effect.

6.3. *Case 3: Inter-System localization*

One day, there was a burst of failures in a bank's transaction system. There are many subsystems that communicate with each other by API (application programming interface) calls. The search space is also large, and the attributes and the number of distinct values are listed as follows: source (13), source IP (66), destination (7), destination IP (10), interface (135). The operators located the root cause (destination=ic in Fig. 14) in ten minutes by manually analyzing many faulty traces (a series of API calls on different services to realizing on transaction is a trace [31, 32, 33]).

Again, upon the request of the operators, we retrospectively ran *PSqueeze* over this system's API call logs during the fault. *PSqueeze* localized the root causes, as shown in Fig. 14, in just several seconds. It also confirms the generalized ripple effect because deviation scores of leaf attribute combinations that are descended from the same root-cause attribute combination are close to each other. Note that *PSqueeze* reports more root-cause attribute combinations than what the operators find. The operators confirm that these additional root-cause attribute combinations are indeed valid: they are abnormal but are just affected by the *ic* service due to the dependency among services. Had *PSqueeze* been actually used immediately after the issue, operators could have localized the root cause much faster (seconds vs minutes) and more accurately. We also run some other algorithms on Case 3 (see Table 10).

Table 10: Qualitative comparison on industrial cases: whether the algorithm can find the true root cause. Some cases are missing due to deployment issues.

true RC?	*PSqueeze*	SQ	HS	APR	ADT	RAD	MID	IAP
Case 1	Yes	Yes	No	No	No	No	No	No
Case 3	Yes	Yes	Yes	Yes	No	No	No	Yes
Case 4	Yes	No	No	Yes*	Yes	No	No	No
Case 5	Yes	No	No	Yes*	Yes	No	No	No

* Reporting too many false positives to be really helpful.

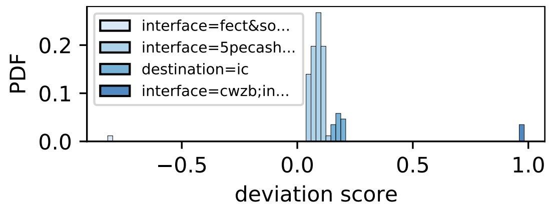


Figure 14: The root-cause attribute combinations and the histogram of deviation scores of their descent leaves in *Case 3*.

6.4. *Case 4 and 5: External Root Causes*

One day at about 00:30, a fault occurred, and thus, a lot of transactions suffered long response latency at a system of a large commercial bank. There are many attributes and attribute values: transaction status (4), host IP (75), return code (275), transaction code (636), and MQ name (5). Therefore, manually localizing root cause attribute combinations is very challenging.

We retrospectively ran *PSqueeze* on this system's logs and successfully localize (return code = 0014) as the root cause (see Fig. 15). Then the operator manually confirms this attribute combination directly indicates the exact underlying root cause. We also compare *PSqueeze* with other baselines in these cases and present the comparison in Table 10. Note that although Apriori successfully localizes the root cause, it also localizes 55 non-root-cause attribute combinations.

Initially, we did not take the two attributes, return code and transaction code, into consideration since there are too many attribute values of them. Then *PSqueeze* reported there might be an external root cause as *min_GPS*

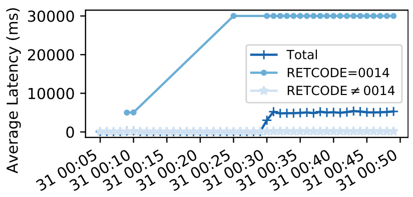


Figure 15: Case 4. If the return code is 0014, then the average latency increases a lot; otherwise the average latency keeps steady.

(refer to Algorithm 3) was only 0.81. Moreover, all these approaches give invalid root causes: *PSqueeze*, HotSpot, Adtributor, MID and ImpAPTr localize almost all MQ names, R-Adtributor also localizes most MQ names combined with *transaction status* = *S*, and Apriori gives dozens of root causes. Operators are not able to infer the underlying root cause from such results, and actually they can be misled. Thus, we took all available attributes into consideration and then found the exact root cause.

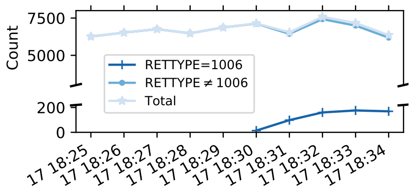


Figure 16: Case 5. If the return type is 1006, then the transaction count increases a lot; otherwise the average latency keeps steady.

At another system of the same bank, another fault caused the transaction count to increase slightly and deviated from its normal state. We also retrospectively ran *PSqueeze* and other baselines, and the comparison is presented in Table 10. *PSqueeze* accurately localizes the root cause attribute combination, "RETTYPE=1006" (see Fig. 16). Note that Apriori localizes too many (17) non-root-cause attribute combinations again. For both case 4 and 5, Squeeze did not localize the exact root-cause attribute combinations due to the large forecasting residuals. Similar to case 4, we did not found the exact root cause until we were notified that there might be external root causes by *PSqueeze* and considered more attributes.

7. Discussion

7.1. Threats to Validity

The major threat to validity lies in the lack of real-world datasets. However, given the difficulty in accessing real-world datasets, all the closely related works use simulated faults as well as our previous conference version, except iDice [8] and MID [5], which are both from Microsoft. We think there are two reasons for the infeasibility of real-world datasets. First, on the one hand, there are a limited number of faults in real-world production systems. Among them, the faults whose root causes can be indicated by root-cause attribute combinations is even less. On the other hand, in many cases, due to the lack of automated root-cause attribute combinations localization tools, the faults are diagnosed with the help of other monitoring data, such as logs and traces, and thus, the root-cause attribute combinations are missed in the failure tickets. Therefore, valid real-world cases that have ground-truth root-cause attribute combinations are somewhat rare. It could take years for engineers to collect hundreds of valid cases. Second, multi-dimensional data are usually highly confidential. Unlike infrastructural metrics, such as CPU utilization and network throughput, the fields in the request logs that we use to build multi-dimensional data are usually sensitive, such as the dollar amount and the user's location. The high risk also hinders companies from sharing real-world multi-dimensional data publicly.

To mitigate this threat to validity, first, we provide several real-world fault cases. Second, compared with the previous conference version, we added two new datasets (i.e., $\mathcal{E}$ and $\mathcal{F}$), which were generated by realistic fault injection.

7.2. Limitations

Root causes of multi-dimensional data are not exact root causes of faults but only clues to them. Nevertheless, localizing root causes for multi-dimensional data is important and helpful since it can direct further investigation right after faults occur.

PSqueeze, as well as the previous approaches, relies on time-series forecasting. To reduce the influence brought by inaccurate forecasting, especially when the real value is small, we introduce probabilistic clustering. The experiment result shows that *PSqueeze* is robust to forecasting residuals.

PSqueeze focuses on only categorical attributes and cannot leverage numerical attributes directly, as well as most previous approaches [7, 4, 9, 11, 8, 5, 10, 3]. We observe that numerical attributes are much less prevalent in practice (*e.g.*, in the five companies studied in this paper). According to our interviews with some engineers, they choose not to record them because they are not sure how to use them for diagnosis. We will work on supporting numerical attributes in the future.

PSqueeze focuses on only numerical measures, as well as most previous approaches [10, 11, 8, 4, 5, 9] except Apriori [7, 3]. However, operators usually aggregate only numerical measures as time-series for monitoring and fault discovery in current industrial practice since categorical measures are not suitable for this. Thus operators are concerned about only numerical measures.

8. Related Work

Recently, many approaches focus on fault diagnosis in various contexts. Most are different from ours [34, 35, 36, 1, 37, 31, 38, 39, 33, 40, 21]. On the one hand, we focus on root cause localization on multi-dimensional data. On the other hand, some of these works use intuitive domain-knowledge-based empirical methods, while we propose a generic algorithm.

There are also several approaches focusing on localizing root causes for multi-dimensional data [10, 11, 8, 4,

Table 11: Comparison of root cause localization approaches for multi-dimensional data Section 5

Approach	Genericness		Robustness			Efficiency	Search Strategy
	general scenario	measure	any anomaly magnitude	determine external root cause	relying on parameter fine-tuning		
Adtributor [10]	no	fundamental&derived	no	no	no	good	top-down
iDice [8]	no	#issue reports	no	no	no	depends	top-down
Apriori [7, 3]	yes	fundamental&derived	yes	no	yes	depends	bottom-up
R-Adtributor [11]	yes	fundamental&derived	yes	no	yes	good	top-down
HotSpot [4]	yes	fundamental	no	no	no	depends	top-down
Squeeze [6]	yes	fundamental&derived	yes	no	no	good	bottom-up&top-down
ImpAPTr [9]	no	success rate	no	no	no	good	top-down
MID [5]	no	#issue reports	yes	no	no	depends	top-down
PSqueeze	yes	fundamental&derived	yes	yes	no	good	bottom-up&top-down

7, 5, 3, 9]. We compare them in Table 11 in three aspects, *i.e.*, genericness, robustness, and efficiency. Some approaches are focusing on a specific scenario rather than generic multi-dimensional data. For example, Adtributor [10] only cares about single-attribute root causes. iDice [8], MID [5] and ImpAPTr [9] utilize the special properties of specific types of measures (*i.e.*, #issue reports and success rate) and thus are not generic. Many approaches [4, 8, 9] filter attribute combinations by their impact (*e.g.*, the percent of issue reports or transactions under them) and thus cannot handle insignificant anomalies. Due to the design of termination conditions or pruning strategies, some approaches rely on parameter fine-tuning [11, 7, 3], and the running time of some approaches [4, 8, 5, 7, 3] varies in different faults.

Time series forecasting has been extensively studied, and there are many approaches. Statistical approaches [41, 42, 29, 43] make some statistical assumptions on the time series. Supervised ensemble approaches [16] try to ensemble statistical approaches in a supervised manner. Recently, unsupervised deep-learning-based approaches [15, 17] are making great progress. The selection of appropriate forecasting algorithms is usually based on the nature of data [16].

There are many other studies on the analysis of multi-dimensional data. A series of studies [44, 45, 46] focus on identifying interesting patterns (a.k.a. insights) in multi-dimensional data. Lumos [47] diagnoses metric regressions by ranking attributes by their importance after regression. Castelluccio et al. [48] focus on mining contrasting sets by statistical tests, which are attribute combinations with distinctive supports in different groups, to find attribute combinations related to a specific group of crashes. Liu et al. [38] debug high response time by mining distinctive conditions for high response time with the help of the decision tree algorithm.

9. Conclusion

Given the importance of root cause localization for multi-dimensional data, many approaches are proposed recently. However, they are not generic or robust enough due to some limitations. In this paper, we propose a more generic and robust approach, *PSqueeze*. *PSqueeze* employs a novel "bottom-up&top-down" searching strategy based on our proposed generalized ripple effect to achieves high efficiency without much loss of genericness and robustness. Notably, this paper further extends our previous studies by a probabilistic clustering method and a method for determining external root causes. We conduct extensive experiments on both simulated and injected faults. The results show that the F1-score of *PSqueeze* outperforms previous approaches by 32.89% on average and consistently costs only about 10s. Besides, the F1-score in determining external root causes reaches 0.90 on average. Furthermore, case studies in several large commercial banks and an Internet company show that *PSqueeze* can localize root causes much more rapidly and accurately than traditional manual analysis in practice.

10. Acknowledgment

The authors gratefully thank Suning.com Co., Ltd. for providing parts of the data used in this paper's evaluation. We thank Juexing Liao and Chuanxi Zheng for proofreading this paper. This work is supported by the National Key R&D Program of China 2019YFB1802504, and the State Key Program of National Natural Science of China under Grant 62072264.

References

[1] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, D. Zhang, An empirical investigation of incident triage for online service systems, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, 2019. doi:10.1109/ICSE-SEIP.2019.00020.

[2] Amazon Prime Day issues estimated to cost $72 million to $99 million - Business Insider (Jul. 2018).
URL `https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7`

[3] F. Lin, K. Muzumdar, N. P. Laptev, M.-V. Curelea, S. Lee, S. Sankar, Fast Dimensional Analysis for Root Cause Investigation in a Large-Scale Service Environment, Proceedings of the ACM on Measurement and Analysis of Computing Systems 4 (2) (Jun. 2020). doi:10.1145/3392149.

[4] Y. Sun, Y. Zhao, Y. Su, D. Liu, X. Nie, Y. Meng, S. Cheng, D. Pei, S. Zhang, X. Qu, X. Guo, HotSpot: Anomaly Localization for Additive KPIs With Multi-Dimensional Attributes, IEEE Access 6 (2018). doi:10.1109/ACCESS.2018.2804764.

[5] J. Gu, C. Luo, S. Qin, B. Qiao, Q. Lin, H. Zhang, Z. Li, Y. Dang, S. Cai, W. Wu, Y. Zhou, M. Chintalapati, D. Zhang, Efficient incident identification from multi-dimensional issue reports via meta-heuristic search, in: Proceedings of the 28th

ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020. doi:10.1145/3368089.3409741.

[6] Z. Li, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, Q. Wang, D. Pei, Generic and Robust Localization of Multi-dimensional Root Causes, in: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), 2019. doi:10.1109/ISSRE.2019.00015.

[7] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, H. Yan, Detecting and Localizing End-to-End Performance Degradation for Cellular Data Services Based on TCP Loss Ratio and Round Trip Time, IEEE/ACM Transactions on Networking 25 (6) (Dec. 2017). doi:10.1109/TNET.2017.2761758.

[8] Q. Lin, J.-G. Lou, H. Zhang, D. Zhang, iDice: Problem identification for emerging issues, in: Proceedings of the 38th International Conference on Software Engineering, 2016. doi:10.1145/2884781.2884795.

[9] G. Rong, H. Wang, Y. You, H. Zhang, J. Sun, D. Shao, Y. Xu, Locating the Clues of Declining Success Rate of Service Calls, in: 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 2020. doi:10.1109/ISSRE5003.2020.00039.

[10] R. Bhagwan, R. Kumar, R. Ramjee, G. Varghese, S. Mohapatra, H. Manoharan, P. Shah, Adtributor: Revenue Debugging in Advertising Systems, in: 11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14), 2014.

[11] M. Persson, L. Rudenius, Anomaly Detection and Fault Localization An Automated Process for Advertising Systems, Master's thesis, Chalmers University of Technology and University of Gothenburg, SE-412 96 Gothenburg (2018).
URL `https://odr.chalmers.se/bitstream/20.500.12380/256407/1/256407.pdf`

[12] M. Kim, R. Sumbaly, S. Shah, Root cause detection in a service-oriented architecture, in: M. Harchol-Balter, J. R. Douceur, J. Xu (Eds.), ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA, June 17-21, 2013, 2013. doi:10.1145/2465529.2465753.

[13] C. Jordan, K. Jordán, Calculus of Finite Differences, American Mathematical Soc., 1965.

[14] N. R. Murphy, B. Beyer, C. Jones, J. Petoff, Site Reliability Engineering: How Google Runs Production Systems, 1st Edition, O'Reilly Media, Beijing ; Boston, 2016.

[15] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, H. Qiao, Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications, in: Proceedings of the 2018 World Wide Web Conference, 2018. doi:10.1145/3178876.3185996.

[16] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, M. Feng, Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning, in: Proceedings of the 2015 Internet Measurement Conference, 2015. doi:10.1145/2815675.2815679.

[17] Z. Li, W. Chen, D. Pei, Robust and Unsupervised KPI Anomaly Detection Based on Conditional Variational Autoencoder, in: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), 2018. doi:10.1109/PCCC.2018.8710885.

[18] V. Satopaa, J. Albrecht, D. Irwin, B. Raghavan, Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior, in: 2011 31st International Conference on Distributed Computing Systems Workshops, 2011. doi:10.1109/ICDCSW.2011.20.

[19] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, X. Xu, DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN, ACM Transactions on Database Systems 42 (3) (Jul. 2017). doi:10.1145/3068335.

[20] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, D. Ding, Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study, IEEE Transactions on Software Engineering (2018). doi:10.1109/TSE.2018.2887384.

[21] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, D. Pei, Practical Root Cause Localization for Microservice Systems via Trace Analysis, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), 2021. doi:10.1109/IWQOS52092.2021.9521340.

[22] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, X. Li, MicroRank: End-to-end latency issue localization with extended spectrum analysis in microservice environments, in: Proceedings of the Web Conference 2021, 2021. doi:10.1145/3442381.3449905.

[23] Y. Chen, M. Yan, D. Yang, X. Zhang, Z. Wang, Deep Attentive Anomaly Detection for Microservice Systems with Multimodal Time-Series Data, in: 2022 IEEE International Conference on Web Services (ICWS), 2022. doi:10.1109/ICWS55610.2022.00062.

[24] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, L. Cao, W. Zhang, K. Sui, Y. Wang, X. Du, G. Duan, D. Pei, Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems, in: Proceedings of the 2022 30th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022.

[25] Istio.
URL `https://istio.io/latest/`

[26] Chaos-mesh/chaos-mesh, Chaos Mesh (Feb. 2022).
URL `https://github.com/chaos-mesh/chaos-mesh`

[27] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011.

[28] H. Wang, G. Rong, Y. Xu, Y. You, ImpAPTr: A Tool For Identifying The Clues To Online Service Anomalies, in: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020.

[29] S.-B. Lee, D. Pei, M. Hajiaghayi, I. Pefkianakis, S. Lu, H. Yan, Z. Ge, J. Yates, M. Kosseifi, Threshold compression for 3G scalable monitoring, in: 2012 Proceedings IEEE INFOCOM, 2012. doi:10.1109/INFCOM.2012.6195498.

[30] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, Lawrence Erlbaum Associates, 1988.

[31] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, C. He, Latent error prediction and fault localization for microservice applications by learning from system trace logs, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019. doi:10.1145/3338906.3338961.

[32] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, D. Pei, Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks, in: 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 2020. doi:10.1109/ISSRE5003.2020.00014.

[33] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, L. Su, Graph-based trace analysis for microservice architecture understanding and problem diagnosis, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020. doi:10.1145/3368089.3417066.

[34] R. Wu, M. Wen, S.-C. Cheung, H. Zhang, ChangeLocator: Locate crash-inducing changes based on crash reports, Empirical Software Engineering 23 (5) (Oct. 2018). doi:10.1007/s10664-017-9567-4.

[35] Q. Lin, H. Zhang, J. Lou, Y. Zhang, X. Chen, Log Clustering Based Problem Identification for Online Service Systems, in: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016. doi:10.1145/2889160.2889232.

[36] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, L. Zhang, An Empirical Study of Fault Localization Families and Their Combinations, IEEE Transactions on Software Engineering (2019).

doi:10.1109/TSE.2019.2892102.

[37] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, D. Zhang, Continuous Incident Triage for Large-Scale Online Service Systems, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019. doi:10.1109/ASE.2019.00042.

[38] D. Liu, Y. Zhao, K. Sui, L. Zou, D. Pei, Q. Tao, X. Chen, D. Tan, FOCUS: Shedding light on the high search response time in the wild, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016. doi:10.1109/INFOCOM.2016.7524413.

[39] D. Liu, MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems, in: ICSE 2021 Software Engineering in Practice, 2020. doi:10.1109/ICSE-SEIP52600.2021.00043.

[40] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, D. Zhang, Identifying impactful service system problems via log analysis, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018. doi:10.1145/3236024.3236083.

[41] Y. Chen, R. Mahajan, B. Sridharan, Z.-L. Zhang, A provider-side view of web search response time, ACM SIGCOMM Computer Communication Review 43 (4) (Aug. 2013). doi:10.1145/2534169.2486035.

[42] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, Z. Zang, X. Jing, M. Feng, FUNNEL: Assessing Software Changes in Web-Based Services, IEEE Transactions on Services Computing 11 (1) (Jan. 2018). doi:10.1109/TSC.2016.2539945.

[43] M. Ma, S. Zhang, D. Pei, X. Huang, H. Dai, Robust and Rapid Adaption for Concept Drift in Software System Anomaly Detection, in: 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), 2018. doi:10.1109/ISSRE.2018.00013.

[44] B. Tang, S. Han, M. L. Yiu, R. Ding, D. Zhang, Extracting Top-K Insights from Multi-dimensional Data, in: Proceedings of the 2017 ACM International Conference on Management of Data, 2017. doi:10.1145/3035918.3035922.

[45] Q. Lin, W. Ke, J.-G. Lou, H. Zhang, K. Sui, Y. Xu, Z. Zhou, B. Qiao, D. Zhang, BigIN4: Instant, Interactive Insight Identification for Multi-Dimensional Big Data, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018. doi:10.1145/3219819.3219867.

[46] R. Ding, S. Han, Y. Xu, H. Zhang, D. Zhang, Quick-Insights: Quick and Automatic Discovery of Insights from Multi-Dimensional Data, in: Proceedings of the 2019 International Conference on Management of Data, 2019. doi:10.1145/3299869.3314037.

[47] J. Pool, E. Beyrami, V. Gopal, A. Aazami, J. Gupchup, J. Rowland, B. Li, P. Kanani, R. Cutler, J. Gehrke, Lumos: A Library for Diagnosing Metric Regressions in Web-Scale Applications, in: Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), 2020. doi:10.1145/3394486.3403306.

[48] M. Castelluccio, C. Sansone, L. Verdoliva, G. Poggi, Automatically analyzing groups of crashes for finding correlations, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017. doi:10.1145/3106237.3106306.