HALO: Hierarchy-aware Fault Localization for Cloud Systems

Xu Zhang*, Chao Du*, Yifan Li*, Yong Xu*, Hongyu Zhang★, Si Qin*, Ze Li§, Qingwei Lin*, Yingnong Dang§, Andrew Zhou†, Saravanakumar Rajmohan†, Dongmei Zhang*
Microsoft Research*, Microsoft 365 †, Microsoft Azure§, The University of Newcastle★
{xuzhang2,chaodu,v-yifanli,yox,siqin,zeli,qlin,yidang,azhou,saravar,dongmeiz}@microsoft.com
hongyu.zhang@newcastle.edu.au

ABSTRACT

A typical cloud system has a large amount of telemetry data collected by pervasive software monitors that keep tracking the health status of the system. The telemetry data is essentially multi-dimensional data, which contains attributes and failure/success status of the system being monitored. By identifying the attribute value combinations where the failures are mostly concentrated (which we call *fault-indicating combination*), we can localize the cause of system failures into a smaller scope, thus facilitating fault diagnosis. However, due to the combinatorial explosion problem and the latent hierarchical structure in cloud telemetry data, it is still intractable to localize the fault to a proper granularity in an efficient way. In this paper, we propose HALO, a **h**ierarchy-aware f**a**ult **lo**calization approach for locating the fault-indicating combinations from telemetry data. Our approach automatically learns the hierarchical relationship among attributes and leverages the hierarchy structure for precise and efficient fault localization. We have evaluated HALO on both industrial and synthetic datasets and the results confirm that HALO outperforms the existing methods. Furthermore, we have successfully deployed HALO to different services in Microsoft Azure and Microsoft 365, witnessed its impact in real-world practice.

CCS CONCEPTS

• **General and reference → Reliability**; • **Information systems → Combination, fusion and federated search**.

KEYWORDS

Hierarchy-aware fault localization; cloud systems; telemetry data; hierarchy graph extraction

ACM Reference Format:
Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, Saravanakumar Rajmohan, Dongmei Zhang. 2021. HALO: Hierarchy-aware Fault Localization for Cloud Systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467190

*Qingwei Lin is the corresponding author of this work.

1 INTRODUCTION

Recent years have witnessed a rapidly growing demand for cloud services. As major cloud service providers, such as Amazon AWS, Microsoft Azure and Google Cloud, expand the capacities of their cloud computing platforms, guaranteeing the platforms' high availability has become the top priority [8]. Despite tremendous engineering efforts on cloud maintenance, faults are still inevitable and occur fairly frequently in large-scale cloud systems [10]. Slow fault mitigation process could result in unplanned interruptions of the services, which will not only incur significant financial losses due to the violation of Service-Level-Agreement (SLA), but also expose service providers to substantial reputation risk [20].

A typical cloud system is equipped with a large number of monitors, which are programs that track the health status of system components. These monitors generate telemetry data that could provide the first-hand information about the system faults. The raw telemetry data is usually organized as multi-dimensional tables containing the failure/success status of the monitored subjects (such as the responses of API requests) along with their attributes (such as the data center, cluster, software version, etc). The failure/success status in those tables can be further aggregated into counts by the attribute values. By identifying the attribute value combination where the failures are mostly concentrated, we may localize the root cause of the failure into a smaller scope and provide useful insights for fault diagnosis. Table 1 shows an illustrating example of telemetry data collected from our cloud system. Given the insight that the majority of failures occurred on the Cluster "PrdC01" when the API "GET-FILES" was called, engineers can proceed to check this API's call stacks on cluster "PrdC01". They could then quickly discover the root cause: the backend file system was crashed on Cluster "PrdC01", which in turn caused an unusually high failure rate on the "GET-FILES" API calls. This example illustrates the importance of correctly identifying the attribute value combination ({"Cluster": "PrdC01", "API": "GET-FILES"}) that indicates the fault, which we call *fault-indicating combination* in this paper.

Due to the scale and complexity of cloud systems, manual investigation on telemetry data is tedious and inefficient for identifying the fault-indicating combination. While ad-hoc aggregations on massive raw telemetry data could speed up this process, such approaches are difficult to generalize and error-prone. Recently, some automated methods [3, 7, 18, 21] analyze telemetry data and provide insights to the engineering teams for fault diagnosis. These methods identify fault-indicating attribute values through feature

Table 1: An example of aggregated telemetry data

Attribute Columns					Metric Columns	
Datacenter	OSVersion	Cluster	Node	API	Failures	Successes
DC1	V_1.1	PrdC01	N01	GET-FILES	180	220
DC1	V_1.1	PrdC01	N01	POST-RESET	10	30
DC1	V_1.1	PrdC01	N02	GET-FILES	150	160
DC1	V_1.1	PrdC02	N03	GET-FILES	5	20
DC1	V_1.2	PrdC02	N03	POST-RESET	0	20
DC2	V_1.2	PrdC03	N04	GET-PWD	0	125
DC2	V_1.3	PrdC03	N05	GET-FILES	5	120
DC2	V_1.3	PrdC04	N06	POST-RESET	2	100
DC2	V_1.4	PrdC04	N06	GET-PWD	20	220
......						

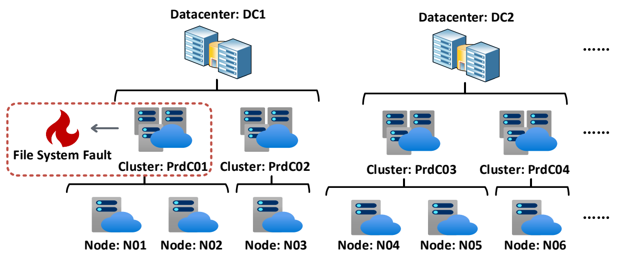

Figure 1: The strict hierarchical structure in Table 1

.

importance ranking or contrast pattern mining. Due to the notorious combinatorial explosion problem and the intrinsic hierarchy relationship in cloud telemetry data, existing methods often suffer from low efficiency or accuracy.

We observe that attributes in the telemetry data tend to exhibit *hierarchical relations* in which the values of one attribute (the *superior* attribute) are determined by the values of another attribute (the *inferior* attribute). Such phenomenon reflects the intrinsic hierarchical architecture of cloud systems and can be commonly found in many other large-scale distributed systems. For instance, the attributes representing the physical organization of cloud systems often form a strict hierarchical structure, such as the Node/Cluster/Datacenter structure shown in Fig.1. With the hierarchical relationship, failures concentrated on the value of a attribute can be also observed on the corresponding values of its inferior and superior attributes simultaneously. For example, in Table 1, "Datacenter" could be regarded as a superior attribute of "Cluster". The value {"Datacenter": "DC1"} could form a "good" indicator of the fault because most failures concentrated on "DC1". However, this "insight" would motivate engineers to look for non-existent issues on this large scope, which distracts the diagnosis process. On the other hand, while values from the inferior attribute Node {"Node":"N01"} and {"Node" : "N02"} can localize the fault "perfectly", little can be learned from this trivial "insight" and engineers may waste valuable time in troubleshooting those two nodes separately. Therefore, to guide the search for the fault-indicating combination, knowledge on the hierarchical relationships should be leveraged. Unfortunately, most of the existing work does not take the hierarchical property of telemetry data into account thus they may mislead the direction of fault diagnosis [3, 7, 21, 22].

In this paper, we propose a **h**ierarchy-aware f**a**ult **lo**calization approach named **HALO** to identify the fault-indicating combination from telemetry data. It leverages the learnt knowledge about the hierarchical relationship among attributes to facilitate the fault localization process. HALO divides the failure localization process into two phases: 1) attribute-level search and 2) value-level search. In phase one, HALO identifies the hierarchical relationship among attributes and generates several *attribute search paths*. In phase two, HALO performs top-down search of *attribute values* along the hierarchically arranged search paths to form the fault-indicating combinations. Through exploiting the hierarchical relationship, our approach not only achieves high efficiency for large data volume, but also can localize the fault to a proper level of granularity.

The effectiveness and efficiency of HALO were evaluated by both experiments and industrial practice. Experiments on real-world offline production datasets shown that HALO can reach an average accuracy around 0.80, about 36% higher than those of the comparative methods. We also show that HALO can remain highly efficient for extremely large dataset thanks to its search space reduction achieved by hierarchy extraction. Finally, HALO has been successfully applied to Microsoft Azure Compute Service and Microsoft 365 Exchange Online Service. Both quantitative online A/B testing and qualitative case studies conducted in the past 8 months have confirmed HALO's effectiveness in industrial practice.

The major contributions of this paper are as follows:

- We propose HALO, an efficient automated hierarchical fault localization approach to locate the fault-indicating combinations from telemetry data effectively.
- We evaluated HALO on both real-world industrial and synthetic datasets and demonstrated that HALO can achieve much higher localization accuracy than the competing methods while remain efficient for large-scale data.
- We demonstrate the practical value of HALO on Microsoft Azure Compute Service and Microsoft 365 Exchange Service.

2 BACKGROUND AND CHALLENGES

2.1 Problem Formulation

In cloud systems, telemetry data are collected from the spatially distributed monitors and centralized to a remote database [2]. It can then be fetched for fault diagnosis. As shown in Table 1, a typical aggregated telemetry data $\mathcal{D}$ consists of multiple *attribute columns* $\mathcal{A} = \{A_1, A_2, ..., A_d\}$ representing the features of the monitored subject and *metric columns* $\mathcal{M} = \{M_F, M_S\}$ representing the aggregated number of failures and successes. Each row in $\mathcal{D}$ is called a *record*. The main objective of this paper is to identify a *fault-indicating combination* ω so that the failures in $\mathcal{D}$ are mostly concentrated on the *subspace* $\mathbb{S}$ determined by ω. Here $\mathbb{S}(\omega) = \sigma_\omega(\mathcal{D})$ where σ denotes a selection operator specified by the disjunctive normal formula $\omega = \{A_{i_1} = v_1 \wedge A_{i_2} = v_2 \wedge \cdots A_{i_j} = v_j\}$ in which each clause represents a value of an attribute column. We assume that all attribute columns are filled with discrete values. Continuous attributes, if encountered, are needed to discretized first.

2.2 Hierarchical Relationship in Cloud

The hierarchy among attributes of telemetry data reflects the inherent hierarchical architecture of a cloud system. For example, as a physical node always belongs to a cluster, records with the same node ID are expected to share the same cluster ID. Attributes also frequently appear as "peer" to others without showing clear hierarchical relation, such as "Node" and "API" attributes in Table 1, because an API is provided by multiple nodes and one node also supports different API functions. If knowing the values of attribute A_j would allow us to approximately deduce the values of another attribute A_i, we would say that there is strong hierarchical relationship between A_i and A_j, attribute A_i is superior to A_j, and attribute A_j is inferior to A_i. Otherwise we will regard A_i and A_j as peers. While the hierarchical relationships can be found frequently in cloud systems, such relationships could vary under different scenarios and context. Therefore, we need to infer those relationships from telemetry data systematically.

2.3 Challenges

We need to address the following challenges to find the fault-indicating combinations.

Firstly, fault localization in multi-dimensional telemetry data suffers from the combinatorial explosion problem, especially when facing an extremely large data volume. As the number of possible combinations increases exponentially with growing dimensions and cardinalities, it is increasingly more challenging to find the combination that can narrow down the problem scope precisely.

Secondly, in the presence of hierarchical relationship, how to localize the fault with an appropriate granularity presents another key challenge. As failures can be observed on different attribute values along the hierarchy, the fault could be localized to a less-optimal granularity. If the fault was localized to attribute values inferior to the optimal one, effort would be wasted in checking the fragmented components. If attribute values superior to the optimal one were chosen, little insight can be learnt from the board scope to guide the diagnosis process.

3 APPROACH

3.1 Overview

The overall framework of HALO (Fig.2) contains two major phases: an attribute-level search phase to narrow the searching space down to certain attributes; and a value-level search phase to further locate the fault-indicating attribute values. Specifically, the first phase detailed in Sections 3.2 and 3.3 aims at identifying search paths composed of hierarchically arranged *attribute columns*. HALO first identifies the relationships among attributes to construct the Attribute Hierarchy Graph (AHG), then generates *attribute search paths* by performing random walk on AHG. The second phase detailed in Sections 3.4 and 3.5 constitutes of top-down search along the hierarchically-arranged attribute paths for identifying the best attribute value combinations. This phase adopts the self-adaptive early-stopping mechanism to reduce the search space and enable locating the failures to a proper granularity. Reverse truncation is also applied to polish the obtained value combinations for improving their compactness.

3.2 Attribute Hierarchy Graph Extraction

To guide the search effectively and efficiently, we first need to organize all the attributes in telemetry data into an Attribute Hierarchy Graph (AHG). An AHG constitutes of multiple *levels* that form an ordered hierarchy. Each level contains one or multiple attributes. Within the same level, attributes are peers. Between different levels, attributes tend to exhibit hierarchical relationship.

As the core innovation of this paper, AHG is integrated into HALO to guide both attribute-level and value-level searches. The incorporation of attribute hierarchy allows us to handle both challenges in Sec.2.3. Firstly, the efficiency of fault localization is significantly improved. Searches guided by AHG not only helps avoid plenty of redundant and repetitive combination explorations, but also is conductive in ruling out the irrelevant values in inferior attributes beforehand. For example, there are very few failures emerging on Cluster "PrdC02" in Table 1. As attribute Cluster is superior to attribute Node, ruling out Cluster "PrdC02" would also eliminate all its subordinate nodes. Secondly, the attributes hierarchy is critical for localizing failures to an appropriate granularity. Searching through hierarchy ensures that we can progressively narrow down the problem space from a larger scope (superior level) to a smaller scope (inferior level), which offers us the chance to explore and stop at the scope that is most likely indicating the fault.

Still, to automatically extract AHG from multi-dimensional telemetry data, we need to determine both the number of levels and the allocations of attributes properly, which are non-trivial without any prior knowledge. In this section, we focus on the technical details of constructing AHG, including the pairwise relationship identification, the skeleton extraction, and the skeleton-based clustering.

3.2.1 Pairwise Relationship Identification. To extract AHG, we need to measure the relationship between any two attributes. For instance, attribute "Cluster" is superior to "Node" as a node must belong to one and only one cluster and a cluster hosts multiple nodes. Thus, knowing the Node ID can determine the Cluster ID, but not vice versa. This fact inspires us to consider the *pairwise conditional entropy* defined in Eq.1:

$$H(A_m|A_n) = -\sum_{v_i\in A_m, v_j\in A_n} p\left(v_i, v_j\right)\log\left(\frac{p\left(v_i, v_j\right)}{p\left(v_j\right)}\right) \qquad (1)$$

where A_m, A_n represent attributes and v_i, v_j denotes their values. And we further define the Uncertainty Reduction (UR) in Eq.2.

$$UR\left(A_m|A_n\right) = 1 - \frac{H\left(A_m|A_n\right)}{H(A_m)}, \qquad (2)$$

where $\frac{H(A_m|A_n)}{H(A_m)}$ denotes the conditional entropy of A_m given A_n normalized by the entropy of A_m: $H\left(A_m\right) = \sum_{v_i\in A_m} p\left(v_i\right)\log\left(p\left(v_i\right)\right)$. Between 0 and 1, $UR\left(A_m|A_n\right)$ can be viewed as the percentage of uncertainty reduction in A_m given A_n. If A_m is superior to A_n, such as "Cluster" is to "Node", the uncertainty in A_m can be significantly eliminated given A_n, then $H\left(A_m|A_n\right) \approx 0$ and $UR\left(A_m|A_n\right) \approx 1$. On the contrary, $UR\left(A_m|A_n\right) \approx 0$ if A_m and A_n are peer.

While UR can be defined in both directions, the hierarchy relationship we interest in is essentially unidirectional. That is, we mainly focus on identifying cases whether the values of one attribute can be determined by the values of another attribute with finer granularity. As the granularity of an attribute can be roughly measured by its entropy, we will only consider the UR of attribute with lower entropy given attribute with higher entropy, and define

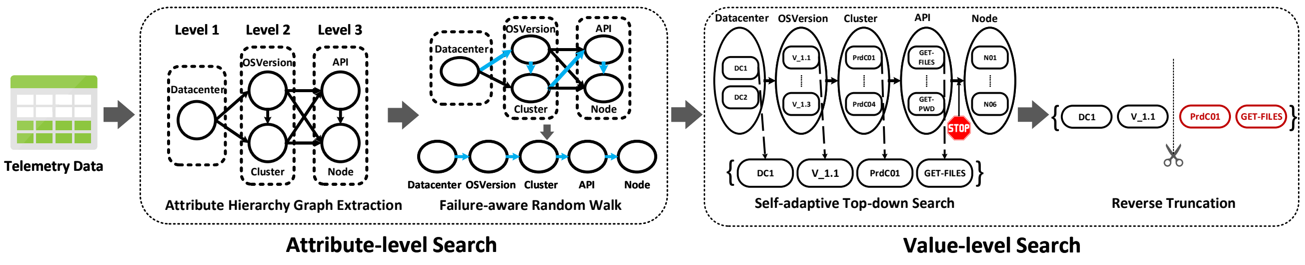

Figure 2: An overview of HALO.

the *Hierarchy Intensity* (*HI*) as Eq.3.

$$HI\left(A_m \rightarrow A_n\right)=\begin{cases} UR\left(A_m|A_n\right), & H\left(A_m\right)<H\left(A_n\right) \\ -\infty, & \text{otherwise.}\end{cases} \quad (3)$$

Here $-\infty$ denotes an invalid HI. Among valid $UR\left(A_m|A_n\right)$, high values close or equal to 1 would indicate strong hierarchical relationship while small values indicate peers relation. A Pairwise Relationship Graph (PRG) can be constructed using all valid HI as weighted directed edges, as shown in the first panel of Fig. 3.

3.2.2 Skeleton Extraction. To transfer the Pairwise Relationship Graph into AHG, the number of hierarchical levels has to be specified. For this purpose, we will firstly identify the hierarchy skeleton, a sequence of attributes forming a strict hierarchical chain. Each attribute in the skeleton forms the basis of a unique level. The rest attributes can then be clustered into those levels to complete AHG.

We define that there is a strict hierarchy relationship from attribute A_m to A_n if $HI\left(A_m \rightarrow A_n\right) \geqslant \tau$, where $\tau \in [0,1]$ is the threshold for identifying strict hierarchy relationship. In this paper, we found $\tau = 0.9$ is sufficient for most scenarios.

To extract the hierarchy skeleton, we start with the vertex with the highest in-degree in PRG because lower-level attributes (such as "Node" in Fig. 3) are always pointed to by higher-level attributes. We then traverse the graph to add new attributes to the skeleton iteratively given that a strict hierarchy relationship exists from the new vertex to the previous vertex. When multiple candidates are available, we will choose the attribute whose in-degree is closest to the current attribute to improve the chance of obtaining a long skeleton. In Fig. 3, after starting with "Node", both attributes "Cluster" and "DataCenter" are valid candidates as both $HI\left(\text{“}Cluster\text{”} \rightarrow \text{“}Node\text{”}\right)$ and $HI\left(\text{“}DataCenter\text{”} \rightarrow \text{“}Node\text{”}\right)$ exceed the threshold τ of strict hierarchy. Since the in-degree of "Cluster" is closer to "Node" than "DataCenter", it is more likely that "Cluster" is the *direct* superior of "Node". Thus, we will select the "Cluster" as the next attribute to be added to the skeleton. Afterward, "DataCenter" is added to complete the skeleton: $\text{“}DataCenter\text{”} \rightarrow \text{“}Cluster\text{”} \rightarrow \text{“}Node\text{”}$.

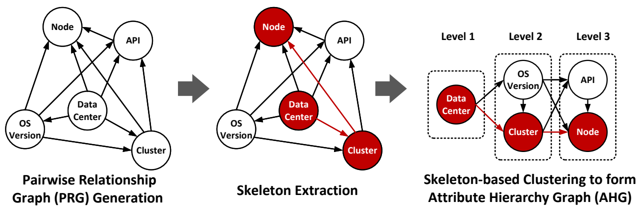
Node
API
Data Center
OS Version
Cluster
Pairwise Relationship Graph (PRG) Generation
Skeleton Extraction
Level 1
Level 2
Level 3
Skeleton-based Clustering to form Attribute Hierarchy Graph (AHG)

Figure 3: The Extraction of Attribute Hierarchy Graph.

3.2.3 Skeleton-based Clustering. Based on the extracted skeleton, we initialize the *hierarchy levels*, each of which is filled up with an attribute from the skeleton. We aim to cluster the remaining attributes into these hierarchy levels so that the attributes in the same level are roughly peers and the attributes between different levels exhibit significant hierarchy structure. This target can be formulated into the following optimization problem:

$$\mathcal{H}^{*}=\underset{\mathcal{H}}{\arg\min}\sum_{i=1}^{n} I_i-\sum_{i=1}^{n-1} C_i. \quad (4)$$

In Eq.4, the optimal AHG $\mathcal{H}^*$ is associated with small internal hierarchy score I_i within each hierarchy level and large cross hierarchy score C_i between adjacent hierarchy levels. n denotes the number of levels. Both I_i and C_i are defined below:

$$\begin{aligned} I_i &= \frac{\sum_{j=1}^{|\mathcal{H}_i|}\sum_{k=1}^{|\mathcal{H}_i|} HI\left(\mathcal{H}_i^j \rightarrow \mathcal{H}_i^k\right)}{|\mathcal{H}_i| * \left(|\mathcal{H}_i|-1\right)/2}, \\ C_i &= \frac{\sum_{j=1}^{|\mathcal{H}_i|}\sum_{k=1}^{|\mathcal{H}_{i+1}|} HI\left(\mathcal{H}_i^j \rightarrow \mathcal{H}_{i+1}^k\right)}{|\mathcal{H}_i| * |\mathcal{H}_{i+1}|}. \end{aligned} \quad (5)$$

where $|\mathcal{H}_i|$ denotes the number of attributes in the i-th level and $\mathcal{H}_i^j$ denotes the j-th attribute in the i-th level. To compute I_i, only valid HI ($> -\infty$) is used. For C_i, the invalid HI is considered to penalize objective score and avoid cases where the cross hierarchy level direction of a pair of attributes violates their HI direction.

To find the solution to Eq.4, we adopt a heuristic approach to cluster the remaining attributes into existing levels one by one. The pseudo-code is summarized in Algorithm 1. To alleviate the impact of the order of attributes to be clustered on the final solution, we will cluster "low impact" attributes first. Specifically, at each step, we evaluate the influence on the objective score of every potential move (clustering one remaining attribute into any level, line 5 ~ line 9) and choose the one with the lowest impact to update the hierarchy (line 10). This process is repeated till all attributes are clustered. Finally, all edges crossing multiple levels are removed to generate the final Attribute Hierarchy Graph, as shown in Fig.3.

3.3 Failure-aware Random Walk

The goal of the attribute-level search phase is to extract sequences composed of multiple attributes as the search paths, which is to be used in the next phase. These attributes should be not only arranged hierarchically according to the AHG but also strongly correlated with the failures. In this way, it can offer the value-level search process a better chance of localizing the fault correctly. Motivated

Algorithm 1: Skeleton-based clustering

```
Require: Skeleton, $\mathcal{S}$
         Remaining Attributes Set, $\mathcal{R}$
         Objective score function Eq.4, Obj
Ensure : Attribute Hierarchy Graph, $\mathcal{H}$
1  $\mathcal{H}$ is initialized using $\mathcal{S}$
2  while $\mathcal{R} \neq \emptyset$ do
3      $min\_obj = \infty$; $add\_attr = None$; $add\_level = None$
4      for $A_i \in \mathcal{R}, level \in \mathcal{H}.size()$ do
5          $\mathcal{H}' = \text{copy}(\mathcal{H})$
6          $\mathcal{H}'_{level} = \mathcal{H}'_{level} \cup A_i$
7          $s = \text{Obj}\,(\mathcal{H}')$
8          if $s < min\_obj$ then
9              $min\_obj = s$; $add\_attr = A_i$; $add\_level = level$
10     $\mathcal{H}_{add\_level} = \mathcal{H}_{add\_level} \cup add\_attr$
11     $\mathcal{R}$.remove($add\_attr$)
12 return $\mathcal{H}$
```

by both targets, we devise a failure-aware random walk on AHG to generate attribute search paths. This random walk obeys two basic rules: 1) skipping hierarchy levels is prohibited. 2) walking towards the high failure-correlated attributes is preferred.

We use the information gain ratio γ_i between attribute column A_i and the failure/success metrics columns as the *failure-correlation score* to guide the random walk. A higher γ_i implies that failures are more concentrated on A_i, thereby it ought to have a higher search priority. To generate a walking path, we firstly sample one attribute from the first hierarchy level. We then iteratively sample its **neighboring** attributes (from the same level or its inferior level) on AHG as the next hop, until no neighboring attributes are available. The sampling probabilities follow the distribution of the normalized failure-correlation scores of the current attribute's neighbors. To guarantee the diversity of search paths, the procedures above will be executed multiple times to obtain several walking paths.

3.4 Self-adaptive Top-down Search

In the second phase, HALO searches for the value combinations along each hierarchy-arranged attribute search path in parallel. Such combinations should cover most failures while keeping an appropriate granularity, neither too broad nor too trivial. To meet these requirements, two fundamental problems should be addressed. Firstly, which attribute values should be incorporated into the fault-indicating combination? Secondly, how to form a proper stop criterion to terminate the search at the right moment and avoid excessively deeper or shallower search along the path?

3.4.1 *Self-adaptive Top-down Search Framework.*

In the value-search phase of HALO, we propose a self-adaptive top-down search framework summarized in the pseudo-code of Algorithm 2. HALO first initializes a candidate combination collector Φ using all values from the first attribute columns in path $\mathcal{P}$ (line 1) to trace the growth of each candidate combination. Then, HALO sequentially scans all attribute columns according to the attribute search path order, i.e. from the higher-level attributes to the lower-level attributes (line 3). For each attribute column, HALO merges its every value into existing combinations in Φ and evaluates the *election score* of these *tmp_candidates* (line 6~7). The election score is used for measuring the failure concentration on these *tmp_candidates.* Those promising candidates with high election scores are reserved in the collector Φ for continuing to incorporate values of the subsequent attributes.

Algorithm 2: Self-adaptive Top-down Search Framework

```
Require: An Attribute Path $\mathcal{P} = [A_1, A_2, ..., A_{|\mathcal{P}|}]$
         Election Scoring Function $\text{E}\,(*)$
         Damping Scoring Function $\text{D}\,(*)$
         Stop Search Threshold $\delta$
Ensure : Fault-indicating Combinations Result Set $\Omega$ of $\mathcal{P}$
1  Initialize the combinations collector $\Phi = \{v_i | v_i \in A_1\}$
2  Initialize the combinations result set $\Omega = []$
3  for $A_j \in [A_2, ..., A_{|\mathcal{P}|}]$ do
4      $election\_score\_list = []$, $tmp\_candidates = []$
5      for $v_i \in A_j, \omega_k \in \Phi$ do
6          $tmp\_candidates.append(\omega_k \wedge \{A_j = v_i\})$
7          $election\_score\_list.append(\text{E}\,(\omega_k \wedge \{A_j = v_i\}))$
8      $thr = \text{OTSU}\,(election\_score\_list)\,; \Phi = [];$
9      for $\omega' \in tmp\_candidates$ do
10         if $\text{D}\,(\omega') \geqslant \delta$ then
11             $\Omega = \Omega \cup \omega'$ // stop searching
12         else if $\text{E}\,(\omega') \geqslant thr$ then
13             $\Phi = \Phi \cup \omega'$ //reserve candidates to keep searching
14 return $\Phi$
```

The promising candidate combination selection is achieved by an automatic thresholding method named OTSU [24] stemming from image segmentation. The basic idea of OTSU is looking for a threshold *thr* for splitting the election scores of the candidate combinations into two groups to minimize intra-group variances (line 8). The *tmp_candidates* with the election scores higher than *thr* are picked to renew the collector Φ (line 12 ~ 13) and others are pruned away. This process is similar to the beam search [23]. However, the general beam search requires a fixed search width but in our approach, it is self-adaptively decided by OTSU.

Each candidate combination in the collector keeps merging the attribute values iteratively along the attribute search path until the termination condition is triggered. For each candidate combination, a *damping score* is calculated along with the election score. If the damping score is higher than the threshold δ, HALO would determine that current candidate combination has reached an appropriate fault-indicating granularity, stop searching on the current combination, and output it to the result list (line 10 ~ 11).

3.4.2 *Scoring Function Instantiation.*

We emphasize that our proposed self-adaptive top-down search is a flexible framework as the election and damping scoring function can be customized for different scenarios. They can be specified by the customer through the call-back functions. In this section, we introduce the default instantiation of both scoring functions used in HALO.

Election Scoring Function. If more failures concentrate on a certain combination, we are more inclined to choose it as the fault-indicating one. Inspired by this idea, we define the failure score S_F on a combination ω as the harmonious average of $S_A(\omega)$ and $S_I(\omega)$, i.e. $S_F(\omega) = \frac{2 \cdot S_A(\omega) \cdot S_I(\omega)}{S_A(\omega) + S_I(\omega)}$:

$$S_I(\omega) = \frac{\sum M_F(\omega)}{\sum M_F}; S_A(\omega) = \frac{\sum M_F(\omega)}{\sum M_F(\omega) + \sum M_S(\omega)} \quad (6)$$

where S_I is the *impact score* denoting the percentage of failures on the combination ω to the total failures. S_A, the *availability score*, measures the ratio of failures against the total number on ω.

Damping scoring function. Failures in cloud environment tend to spread evenly over the records on the true fault-indicating combination. In Table 1, the failures are uniformly spread across all the records (line 1 and line 3) on {"Cluster": "PrdC01" "API": "GET-FILES"}. On the contrary, the distribution of failures over corresponding superior attribute is far from uniform. In this example, failure rates are subject to large variety among the records from datacenter "DC1" (line 1 ∼ line 5). It implies the true epicenter of the failures lies on its inferior attributes, not on "DC1". Motivated by this observation, we design the stopping rule based on whether the availability scores of those records under the candidate combination are balanced. In particular, for the current *tmp_candidate* ω', we retrieve all related records and calculate **each record's** availability score as in Eq.6. The difference score S_d between the estimated availability scores distribution and the standard uniform distribution with the same dimension is measured by their JS-divergence. A lower S_d indicates that the distribution of records' availability scores under the current ω' are more close to uniform and we should stop the search instead of exploring the lower-level attributes.

3.5 Reverse Truncation

Each obtained fault-indicating combination is composed of values from all attributes in the search path before the termination. Still, values from superior attributes are redundant if the values from inferior attribute values have pinpointed the fault location precisely. For example, if HALO has already localized the fault to a cluster, ID of associated data center and other superior attributes would be unnecessary. To make the combinations more succinct, we perform reverse truncation to polish the combination.

To be specific, for an obtained combination $\omega = \{A_1 = v_1 \wedge A_2 = v_2 \wedge \cdots \wedge A_{|\omega|} = v_{|\omega|}\}$, we calculate its election score $\mathrm{E}(\omega)$. After that, we enumerate its sub-combination in reverse order, $\omega_1^* = \{A_{|\omega|} = v_{|\omega|}\}, \omega_2^* = \{A_{|\omega|-1} = v_{|\omega|-1} \wedge A_{|\omega|} = v_{|\omega|}\}, \cdots, \omega_{|\omega|}^* = \{A_{|\omega|} = v_{|\omega|} \wedge \cdots \wedge A_1 = v_1\}$. For each sub-combination, we evaluate its election score $\mathrm{E}(\omega^*)$ and compare it with the original $\mathrm{E}(\omega)$. If the difference for a certain ω_i^* is smaller than a threshold α, it means the current sub-combination is representative enough and those preceding higher-level attribute values contribute little to raising election score. Thus, we replace the original ω by ω_i^*. Finally, all truncated combinations from different attribute search paths are put together and sorted according to their election scores to form a ranking list. In practice, we recommend top of it to users.

4 EXPERIMENT

In this section, we conduct extensive experiments to evaluate the effectiveness and efficiency of our proposed approach.

4.1 Experimental Setup

Datasets. We use different types of datasets to evaluate the effectiveness and efficiency of HALO.

Production Datasets: The production datasets are collected from three services in Microsoft named FuseBot, RescueBox and Gandalf during the second half of 2020. Both FuseBot and RescueBox are affiliated with Microsoft 365 Exchange. FuseBot is responsible for the safe deployment of Microsoft 365 Exchange Online through identifying the problematic Exchange build versions. RescueBox is a fast recovery service in Microsoft 365 Exchange and is used for automated mitigation actions. Gandalf [17] is a suite of AIOps [10] solution in Microsoft Azure, a large-scale cloud computing service. A total of 47 datasets were collected from these 3 services. The number of raw records ranges from 779 to 67 million and the number of attribute columns varies from 5 to 62. The ground truth of fault-indicating combinations are all manually labeled by the on-call engineers based on the postmortems of the incidents. We organize those datasets into three groups by the service names.

Synthetic Datasets: We also created 5 groups of synthetic datasets to evaluate HALO under a controlled environment. The first three groups include simulated datasets with a fixed number of attribute values in the true fault-indicating combination. The rest two groups are composed with datasets in which all attributes exhibit pure hierarchy/peer structure to explore HALO's performance under extreme circumstances. Each of the five groups consists of 8 datasets[1], where the number of raw records varies from 5000 to 1.2 million, and the number of attribute columns varies from 6 to 10.

Comparative Methods. We compared HALO with three existing methods proposed for the fault localization problem in large-scale software systems. **Tree Model based Predicate Extraction (TM-PE)** is a machine-learning based approach proposed in [3]. It trains a random forest classifier to predict the success/failure status from attribute values and extracts value combinations from the learnt models. **Fast Dimensional Analysis (FDA)** [18] is a frequent pattern mining method. It uses FP-Growth algorithm [6] to combine the attribute values with a high support ratio among records with failure status and relies on the association rule learning to select the combination that is most unique to the failures. **Lumos** [21] is a library for diagnosing regressions in the context of A/B testing. Its fault localization component leverages the hazard scores to evaluate individual attribute values' relevance to the failures. All methods including HALO report a ranking list of fault-indicating attribute values or combinations. For the sake of evaluation, only the results on the top of the ranking list is considered[1].

Metrics. For each dataset $\mathcal{D}_i$, we evaluate the model accuracy with the Sørensen–Dice coefficient [25] between the ground truth combination $\hat{\omega}_i$ and the combination ω_i reported by models. This metric allows us to penalize both redundant and missing attribute values. The average Sørensen–Dice coefficient across all datasets within the same group is used as the final metric as shown in Eq.7,

[1]The details of datasets and implementation can be found in Supplementary Materials.

where N is the number of datasets in a dataset group. This metric has been widely used in [14, 16] for the fault localization problem.

$$SDC = \sum_{i=0}^{N} \left(\frac{2 \cdot \omega_i \cap \hat{\omega}_i}{|\omega_i| + |\hat{\omega}_i|} \right) / N \tag{7}$$

4.2 Effectiveness

We first evaluated the effectiveness of HALO and all three comparative methods on the production datasets and summarized the experimental results in Table 2. Those results suggest that HALO performs much better than other methods. It achieved about 0.80 SDC-Score, about 36% higher than other methods on average. Among the comparative methods, FDA and TM-PE directly extract fault-indicating attribute value combinations without accounting for the hierarchy structure. Lacking mechanism for localizing failures into a proper granularity, those two methods may pick irrelevant attribute values, which lower the SDC-Scores. In contrast, as Lumos only evaluates the attribute value individually while ignoring their interactions, it tends to form incomplete solutions.

Table 2: Evaluation results on production datasets

Methods	FuseBot	RescueBox	Gandalf
TM-PE	0.404	0.510	0.414
FDA	0.476	0.688	0.333
Lumos	0.404	0.458	0.333
HALO	0.844	0.760	0.771

To further validate the effectiveness of HALO systematically under controlled conditions, we applied all four methods on synthetic datasets and summarized results in Table 3. Here "DG-N" denotes a dataset group in which the ground truth combination contains exactly N attribute values. "Peer-Only" and "Hierarchy-Only" refers to the groups with pure peer/hierarchy relationship among all attribute columns. From this table, we found that the accuracy of all methods decreases gradually as the length of the ground truth combination increases, due to the fact that the longer combination is more difficult to be perfectly identified. Even though, HALO had the least accuracy degradation and still outperformed other methods. In the "Hierarchy-Only" group with perfect hierarchy structure, the advantage of HALO can be fully exploited to obtain perfect result. In the "Peer-Only" dataset group, while HALO cannot extract valid hierarchy structure, it can still leverage the failure-aware random walk to guide the value-level search towards the more promising attributes. Under this extreme circumstance, HALO exhibited robust performance and received scores better than those of other methods.

4.3 Efficiency

4.3.1 Performance comparsion under different data volume. Operators and developers are expected to immediately locate the failures so that they can take actions in time. Thus, the efficiency of the localization algorithm is crucial for accelerating the diagnosis process. We studied the running time of HALO and other three methods on datasets with different data scales. We plotted the results along with corresponding log-log regression lines in Fig.4. The horizon axis denotes the records number in raw telemetry data and the vertical axis denotes the running time. Both axes are in log scale.

Table 3: Evaluation results on synthetic datasets

Methods	DG-1	DG-2	DG-3	Peer Only	Hierarchy Only
TM-PE	0.850	0.846	0.750	0.553	0.533
FDA	1.000	0.938	0.842	0.738	1.000
Lumos	0.821	0.821	0.790	0.646	0.400
HALO	1.000	0.975	0.915	0.856	1.000

We can conclude from this figure that the running time of HALO increases slowly with the rising volume of telemetry data because the regression line of HALO approximate horizontal. In TM-PE and FDA, the majority of the running time is spent on the model fitting and pattern extraction, or frequent pattern mining, which all become much slower for large volume of data. As a single attribute value evaluation method, the computation in Lumos does not involve complicated combination generation procedure and is much faster than other methods. However, the results in Table 2 show that the accuracy of Lumos is considerably inferior to HALO.

4.3.2 Performance comparison under different settings. In HALO, the hierarchy extraction in the attribute-level search and the early stop in the value-level search are both conducive to reducing the search space. To validate the usefulness of both components, we summarized the number of attribute-value combinations to be searched under different settings on a large-scale dataset (67,504,971 records and 6 attribute columns) in Fig .5. We can see that both components contribute significantly to the search space reduction. Without hierarchy extraction and early stop, we need to search through about 457K possible combinations, a huge number for practical application. If we only enable either hierarchy extraction or early stopping, the size of search space can be reduced to 63K and 14K, respectively. With both components invoked as in HALO, the total search space can be narrowed down to only 6K.

5 INDUSTRIAL PRACTICE

So far, HALO has already successfully applied to multiple scenarios in Microsoft 365 and Azure Cloud, including safe deployment, API availability regression investigation, live migration performance bottleneck analysis, and VM abnormal rebooting diagnosis, etc. In this section, we will present the online evaluation of HALO and then share its impact in production and our experience from real-world deployment.

5.1 Online Evaluation in Production

In Microsoft 365 Exchange online service, FuseBot is a safe deployment service, which is responsible for diagnosing whether a certain Exchange software deployment should be responsible for system incidents. Due to the Progressive Delivery process, there would be multiple build versions co-existing in the production environment. Traditional solution to this task was called as Version Comparison, which ranks these different build versions according to their aggregated failure rates. However, the diagnosis results did

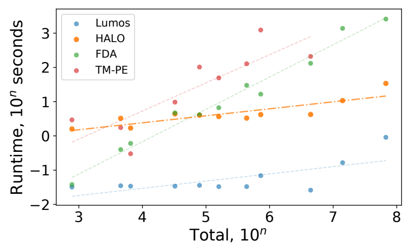

Figure 4: Running time comparison under different data volume.

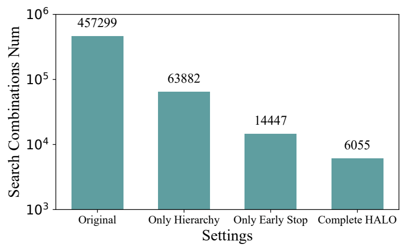

Figure 5: Combinations to be searched under different settings

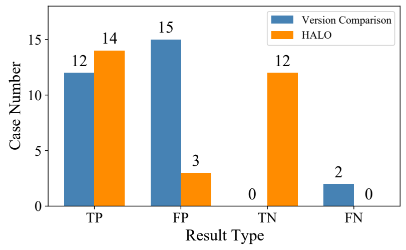

Figure 6: HALO A/B testing results in Microsoft 365 Exchange Online.

not meet the expectations due to the existence of *ambient faults*, which are caused by non-software issues, such as disk corrupt, network timeouts, and power loss [12, 15]. These ambient faults were often misidentified by the Version Comparison solution as the software-related problems, which led to many false positive results. We built a shadow running service equipped with HALO for better fault localization. We performed an online A/B testing from March 2020 to June 2020. The goal of this online A/B testing is to verify how many software faults can be correctly identified and whether ambient faults can be successfully eliminated by HALO, in comparison to the traditional Version Comparison solution. The results are summarized in Fig. 6.

Fig. 6 shows that Version Comparison wrongly categorized 15 ambient faults as software-related issues (i.e., False Positives, FP). It is because that "software version" is generally a high-level attribute in the hierarchy structure of the telemetry data, the failures caused by ambient faults usually concentrate at lower-level attribute values and accumulate on corresponding superior build versions. Therefore, traditional Version Comparison is prone to report false-positive cases. On the contrary, HALO exploits the hierarchy relationship to localize the fault to attribute values of a proper level and thus successfully eliminated 12 of 15 false alarms in online A/B testing. We also found that the numbers of true software faults captured by Version Comparison and HALO are close to each other. The True Positive (TP) cases are 12 vs. 14. It shows that HALO is also capable of identifying software-related issues.

HALO has been shipped into Microsoft 365 FuseBot service since June 2020. It has successfully identified tens of problematic Exchange Online build versions, and achieved 86.9% precision and 93.0% recall. It is estimated to save about hundreds of Time-to-Mitigate (TTM) hours, thanks to its fast fault localization process. The identified cases included dozens of high severity problems, which triggered urgent rollback mitigation actions.

5.2 Case Study

5.2.1 Cross Dependency in Safe Deployment . Deployment safety is important for the maintenance of Microsoft 365. Once a problematic software rollout is confirmed, on-call engineers are required to roll back the bad deployment and fix the problem as soon as possible. Besides the ambient issue presented above, complex cross dependency among micro-services in Microsoft 365 is another intractable problem for identifying the bad deployment. It is common to find out that a burst of failures in a certain service is caused by a deployment event of another service, rather than by bugs in its own software. This leads to a big challenge for rapid fault diagnosis.

In our practice, we once encountered an increasing of failures on the REST API availability signal of the Messages Ingestion service. Intuitively, this service's deployment version was marked as the prime suspect. However, 3 hours after the investigation was kicked off, engineering teams still could not find anything wrong with this service and fell into confusion. At the same time, HALO reported an attribute value combination composed of the Messages Ingestion service and a newly deployed Exchange software version, which is depended by the former. More than 97% failures are concentrated on this combination. Noting the potential correlation between those two services, on-call engineers inspected activities logs from both services and realized that the store protocol team from Exchange service should be involved in the investigation. Only 5 minutes later after this team was informed, the root cause, a class casting failure in an event handler, was found. This event handler was incompatible with the Messages Ingestion service and caused failures on the REST API availability. This alert was quickly escalated to a higher severity and this Exchange deployment version was rolled back at once. Thanks to the HALO's engagement, the potential break out of this bad rollout was successfully avoided in an early stage.

5.2.2 Hardware Failure Analysis using off-the-shelf HALO. As a general solution, HALO has been applied to Microsoft 365 RescueBox and Azure Gandalf services as a third-party off-the-shelf diagnosis tool for different customized scenarios. In this way, engineers from both services are able to query telemetry data and seamlessly invoke HALO to conduct analysis. Especially during the hardware failure analysis process, the diverse fault root causes, heterogeneous telemetry data schema and diverse metrics require on-call engineers to execute the diagnosis tool according to their own different site conditions.

In September 2020, some monitors reported that CPU Internal ERRor (IERR) failures in Azure Compute Service increased significantly. The incident manager called up both the OS teams and the infrastructure teams to investigate this problem. They queried CPU and VM telemetry data and invoked HALO API. HALO reported an attribute value combination, including a microcode version and a hardware SKU (Stock Keeping Unit) ID. It was a critical finding as it provided crucial insight into the dependency between software and hardware. This alert was bumped to the higher priority and the engineering team started to investigate along the clues given by HALO. Finally, it was concluded that this issue was caused by a bug in CPU microcode. It was interesting that only the machines with power capping enabled, such as the hardware SKU we found, are sensitive to this bug (which would be triggered by the frequent throttling when the power capping limit is breached). The engineering team decided to temporally disabled power capping on the impacted machines of this SKU. This issue was eventually resolved after a microcode patch from CPU vendors was applied.

6 RELATED WORK

Much work has been proposed for fault diagnosis with telemetry data. For example, Raff et.al [22] ran a two sample Z-test on each column and rank all the columns by the p-values. While this approach can be used to find relevant attribute columns, it can not identify the fault-localizing attribute value. Pool et. al [21] proposed Lumos as an A/B testing approach capable of fault diagnosis. It applied Bias Normalization to reduce false alarm rate, and hypothesis tests on one-hot encoded attribute columns to narrow down the problem scope to attribute values. Castelluccio et. al [7] developed a modified version of STUCCO, a contrast set mining algorithm [4, 5], to find attribute-value pairs that are significantly different between software crash reports. Similarly, Lin et. al [18] utilized FP-Growth, an association rule mining algorithm [13], to find frequent sets among the failed records.

Other prior work utilized machine learning algorithms for fault diagnosis with telemetry data. For example, the Fa system [11] used margin-based agglomerative clustering [19] and a Partition-Check-Merge algorithm to match the new failed records with previously annotated failed reasons. However, the fact that it requires pre-labeled records as input data makes it unsuitable for our use case. Chen et. al [9] presented an approach to diagnose the failures by training a decision tree, and used the split rule of the tree nodes as the diagnosis result. Based on this method, Bansal et. al [3] proposed TM-PE, which relies on random forest for rule extraction and improvement on precision and robustness.

7 CONCLUSION

Rapid fault diagnosis is vital for the maintenance of cloud systems. In this paper, we propose HALO, an efficient hierarchical fault localization approach to locating the fault-indicating combinations from telemetry data. Our approach divides the telemetry data based failure localization process into two phases: 1) attribute-level search, where hierarchical relationship among attributes are identified, and 2) value-level search, where the fault-indicating combinations are identified. The exploit of the hierarchical relationship allows HALO to localize fault to a proper level of granularity with high efficiency. Experiments on real-world production datasets shown that HALO can reach an average accuracy of about 0.80, which is about 36% higher than that of the comparative methods. HALO has also been successfully applied to Microsoft Azure and Microsoft 365, and its effectiveness is confirmed in industrial practice.

ACKNOWLEDGEMENT

We thank our partners Cathy Chen, Keyan Zhang, Weikun Zhong, Qijun Hong, Junbo Qin, Botao Zhang, Victor Rühle, Xukun Li for the close collaboration, thank Robert Gu, Jim Kleewein, Murali Chintalapati, Melur Raghuraman, Lidong Zhou, Hsiao-Wuen Hon for their great support. Hongyu Zhang is supported by ARC DP200102940.

REFERENCES

[1] Kusto. https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/.

[2] Observability, telemetry, and monitoring. https://cloud.ibm.com/docs/java?topic=cloud-native-observability-cn.

[3] C. Bansal, S. Renganathan, A. Asudani, O. Midy, and M. Janakiraman. Decaf: diagnosing and triaging performance issues in large-scale cloud services. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020.

[4] S. Bay and M. Pazzani. Detecting change in categorical data: mining contrast sets. In *KDD '99*, 1999.

[5] S. Bay and M. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5:213–246, 2004.

[6] C. Borgelt. An implementation of the fp-growth algorithm. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 1–5, 2005.

[7] M. Castelluccio, C. Sansone, L. Verdoliva, and G. Poggi. Automatically analyzing groups of crashes for finding correlations. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017.

[8] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 111–120. IEEE, 2019.

[9] M. Y. Chen, A. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. 2004.

[10] Y. Dang, Q. Lin, and P. Huang. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.

[11] S. Duan, S. Babu, and K. Munagala. Fa: A system for automating failure diagnosis. *2009 IEEE 25th International Conference on Data Engineering*, pages 1012–1023, 2009.

[12] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)*, 14(3):1–26, 2018.

[13] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8:53–87, 2006.

[14] B. Hofer, A. Perez, R. Abreu, and F. Wotawa. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, 22(1):47–74, 2015.

[15] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 150–155, 2017.

[16] J. Kim and E. Lee. Empirical evaluation of existing algorithms of spectrum based fault localization. In *The International Conference on Information Networking 2014 (ICOIN2014)*, pages 346–351. IEEE, 2014.

[17] Z. Li, Q. Cheng, K. Hsieh, Y. Dang, P. Huang, P. Singh, X. Yang, Q. Lin, Y. Wu, S. Levy, et al. Gandalf: An intelligent, end-to-end analytics service for safe deployment in large-scale cloud infrastructure. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 389–402, 2020.

[18] F. Lin, K. Muzumdar, N. Laptev, M.-V. Curelea, S. Lee, and 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*, 4:1 – 23, 2020.

[19] K. Munagala, R. Tibshirani, and P. Brown. Cancer characterization and feature set extraction by discriminative margin clustering. *BMC Bioinformatics*, 5:21 – 21, 2003.

[20] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, S. Keerthi, S. Herbert, and S. Dhulipalla. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2029–2038, 2015.

[21] J. Pool, E. Beyrami, V. Gopal, A. Aazami, J. Gupchup, J. Rowland, B. Li, P. Kanani, R. Cutler, and J. Gehrke. Lumos: A library for diagnosing metric regressions in web-scale applications. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 2020.

[22] P. Raff and Z. Jin. The difference-of-datasets framework: A statistical method to discover insight. *2016 IEEE International Conference on Big Data (Big Data)*, pages 1824–1831, 2016.

[23] D. R. Reddy et al. Speech understanding systems: A summary of results of the five-year research effort. department of computer science, 1977.

[24] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–165, 2004.

[25] V. Verma and R. K. Aggarwal. A comparative analysis of similarity measures akin to the jaccard index in collaborative recommendations: empirical and theoretical perspective. *Social Network Analysis and Mining*, 10:1–16, 2020.

A SUPPLEMENTARY MATERIALS

A.1 Experimental Setting

We carried out our experiments on a server which runs on 64-bit Windows Server 2016. It owns two Inter(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz 2.59GHz processors and 512 GB of memory. HALO is implemented with Python 3.7.8 and data is queried with Kusto [1].

A.2 Implementation Details of Comparative Methods

A.2.1 TM-PE. TM-PE is proposed in a diagnosis and triaging system for performance issues in large-scale cloud services [3]. In the diagnosis component, TM-PE treats the success/failure status as the target labels and the other attributes as the features to train a random forest classifier for fitting the telemetry data. After that, TM-PE used a parser to extract *correlated predicates* and *scope predicates* from the trained model to capture the attribute values that are directly correlated to failures or indicate their scope. These predicates are composed of either positive form $A_i = v \longrightarrow \{True\}$ or negative form $A_i = v \longrightarrow \{False\}$. For adapting to our scenarios, we only collected all positive form predicates and regarded them as the possible fault-indicating combinations.

A.2.2 FDA. Fast Dimensional Analysis (FDA) is a framework that automates the root cause analysis on structured logs in a large-scale service environment. FDA firstly uses the well-known frequent-set mining algorithm FP-Growth to find the frequent combinations that coexist with failures repeatedly. Secondly, the *lift* and *support* metrics are used to filter out some irrelevant patterns. We utilized the Python package *pyfpgrowth*[2] to perform the FP-Growth algorithm and implemented the filtering procedure described in their paper by ourselves. For each frequent combination found and survived from the filtering process, we consider it as a possible fault-indicating one.

A.2.3 Lumos. Lumos is a python library for diagnosing metrics regressions in web-scale applications. Different from regression diagnosis, any attributes in telemetry data, including some static ones (such as the device type or geographical location), could be helpful for fault localization. Therefore, we regard all attributes in our scenarios as the *hypothesis features* [21]. We do not consider bias check and normalization process for *invariant features* and only took advantage of the hypothesis feature ranking component in Lumos. As introduced in their paper, we calculate the *hazard score* for all attribute values. This score is defined as the difference between the control and treatment datasets of the conditional probability of feature occurrence given that a failure occurred [21]. We selected those attribute values whose hazard score is larger than a self-adaptive threshold and treated them as the fault-indicating ones.

A.3 Parameter Tuning

We fine-tuned the related parameters of all algorithms, specifically:

- TM-PE: threshold for score difference between left child and right child, $diff\_threshold = 0.3$
- FDA: minimum support in FP-Growth $min\_support = 0.5 * \sum_D M_F$, thresholds used in pruning, $H_{lift} = 1.1$, $H_{supp} = 1.1$
- Lumos: threshold for hazard score $score\_threshold$ is set adaptively to $\mu_s + 3\sigma_s$, where s represents all hazard scores and μ, σ denote the mean and standard deviation, respectively
- HALO: stop search threshold $\delta = 0.1$, strict hierarchy threshold $\tau = 0.9$, reverse truncation threshold $\alpha = 0.8$, number of search paths = 20.

[2]https://pypi.org/project/pyfpgrowth/

A.4 Details in Evaluation

The output of either TM-PE, FDA, or HALO is a ranking list of possible fault-indicating combinations. From the practical point of view, engineers and operators tend to focus on the top result in the diagnosis report. Therefore, we measured Sørensen–Dice coefficient by taking the top-1 combination. The raw output of Lumos is a ranked list of fault-indicating attribute values, not combinations. For fairness, we took those top ranked values whose hazard score is higher than the fine-tuned threshold.

In addition, we set a timeout limit of 7200 seconds (i.e., 2 hours). If the runtime of an algorithm on a dataset exceeds this, we will terminate it and output the empty result. In our practice, if a diagnosis algorithm requires more than 2 hours to complete, it is impractical as engineers may have already localized the problem manually.

A.5 Synthetic Datasets Generation

The synthetic datasets DG-1, DG-2, DG-3 and Hierarchy Only are generated in the following steps:

(1) Generate a tree to emulate the hierarchical structure among attributes
(2) Transform the tree into a table of attributes
(3) Choose a row randomly, and pick some of its attribute values as the ground truth combination
(4) Assign the metrics to each row based on whether the row matches the true failure-indicating combination

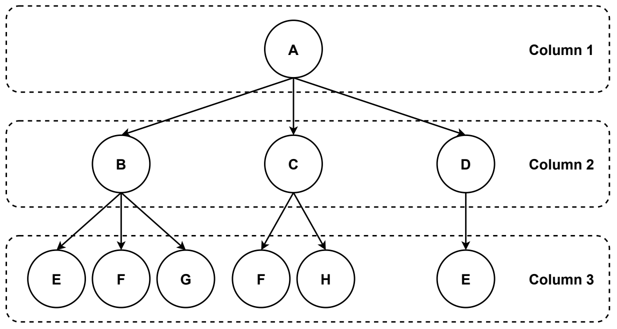

Figure A1: The generated tree structure

An example of the tree structure can be found in Fig. A1. Each layer represents am attribute column and each node represents a value. At each level, a *reuse rate* is determined. When a new node is to be mounted, it may use a previous value at the probability of reuse rate, or generate a new one. If the reuse rate is zero, all values in the next layer will be unique, leading to a pure hierarchy relationship. If the reuse rate is non-zero, certain values may appear again under different previous-column values.

Table A1: Transformed attribute table

Column 1	Column 2	Column 3
A	B	E
A	B	F
A	B	G
A	C	F
A	C	H
A	D	E

Table A1 shows the attribute table converted from the tree. After that, we randomly select some attribute values as the ground truth combination. For example, if the first row and Column 2, Column 3 are chosen, then the failure-indicating combination will be {"Column 2": "B", "Column 3": "E"}. For the records satisfying this combination, M_S are sampled from a distribution with small mean value, and M_F are sampled as such with large mean value. For the other records, the distribution for M_S has a large mean while that of M_F has a small one.

When generating DG-1, DG-2 and DG-3, we use various reuse rates in step (1) and select the length of ground truth combination respectively in step (3). For Hierarchy Only, the reuse rate is always set to zero.

The way we generate "Peer Only" dataset is slightly different because the procedure above inevitably introduces hierarchical relationship to a certain degree. We make sure that all columns are peer by making them all independent. When synthesizing the table of attributes, a value pool is generated for each attribute column in advance, and all values are selected i.i.d. from its corresponding pool. Then we conduct step (3) and step (4) as previously stated to assign the metric columns.