IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received September 4, 2019, accepted September 16, 2019, date of publication September 30, 2019, date of current version October 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2944456

A Real-Time Trace-Level Root-Cause Diagnosis System in Alibaba Datacenters

ZHENGONG CAI[1], WEI LI[1], WANYI ZHU[2], LU LIU[1], AND BOWEI YANG[1]
[1]Zhejiang University, Hangzhou 310012, China
[2]Alibaba Inc., Hangzhou 310027, China

Corresponding author: Wanyi Zhu (wanyi.wyz@alibaba-inc.com)

This work was supported by the Alibaba-Zhejiang University Joint Institute of Frontier Technologies.

ABSTRACT Root-cause analysis (RCA) for service performance degradation can be a challenging exercise given the increasingly complex, inter-related, distributed infrastructure environment in today's enterprises. Many approaches have been applied into enterprise datacenters to improve the maintenance efficiency. A novel graph-level RCA approach is introduced in this paper, including tracing, weighted graph matching and suspicion ranking. The approach is developed based on performance profiling, tracing, and logging systems in Alibaba datacenters to speed up the real-time root-cause diagnosis. Our system allows the discovery of normative patterns and the corresponding key graph properties, which are stored and updated offline as a knowledge base for subsequently being used in trace-level risk estimation and identification of transitions that are unexpected deviations from the normative patterns. Through testing in production, we show the effectiveness of applying the graph-level RCA to discover the origins of problems and generate real-time operational support. It greatly decreases the workload for locating the root-cause of the anomaly.

INDEX TERMS Graph-level root-cause analysis, performance anomaly, performance profiling and tracing, relative importance analysis.

I. INTRODUCTION

In recent years, data center management has become complicated with the increasing velocity, volume and dynamic nature of data. It is well known that modern enterprise applications and systems sometimes experience unexpected and undesirable performance anomalies, which are associated with lost sales and man-hours spent on recovery, and can have significant cost implications. Similarly, performance bottlenecks may eventually lead to system failure and outages spanning minutes to hours, if they were left unattended. These implications show not only the importance but also the potential economic value of robust and automated solutions for detecting performance problems in real time.

Failing to timely detection and diagnosis of performance issues is critical for service providers before they trigger unforeseen service downtime. There is a growing need for robust, reliable anomaly detection and root-cause analysis (RCA) systems [1]–[4]. There have been considerable efforts in academia and industry towards the real-time anomaly detection field. While doing a reasonably good job in responsively detecting anomalies, most of the existing detection techniques still face additional challenges of scalable data center monitoring in explaining the anomalies in the post-detection phase. This involves uncovering the deepest root cause of an anomaly and reasons for identified concerns, providing a coherent picture for the 'why' and 'how' of the anomaly, or presenting the results in a user-friendly form for data engineers to take corrective actions on.

In this paper, we introduce a real-time trace-level root-cause detection system that has been integrated with EagleEye, which is Alibaba's production distributed tracing system. EagleEye is deployed across virtually all of Alibaba's systems, and has allowed the vast majority of our largest work-loads to be traced without need for any application-level modifications, and with a minimum noticeable performance impact. It achieves a high degree of application-level transparency, and ubiquitous deployment of a tracing platform with an acceptable performance overhead. Although it is effective in determining which part of a system is experiencing slowdowns, it is not always sufficient for pin-pointing the root causes. For example, a request may be slow not caused by its own behavior, but because other requests were queued ahead of it. Graph mining provides a powerful machinery for

The associate editor coordinating the review of this manuscript and approving it for publication was Xingwang Li.

addressing this issue, which can effectively capture the long-range correlations among inter-dependent data objects.

Finally, we developed an online diagnosis system through the graph-structured data provided by EagleEye and graph data mining techniques to: 1) discover the workload-specific normative patterns; 2) classify the traces based on graph properties and identify the determinant subsequences or strongly connected structures; 3) discover the anomalous transitions that are significant deviations from its corresponding normative pattern; 4) uncover the origins of performance issues/anomalies thereby determining potential remedies or mitigation methods.

By comparing with other existing identification methods using production data, our proposed system shows the effectiveness in correctly detecting anomalous transitions and determining root-causes within a reasonable amount of time. Our trace-level RCA achieves automated troubleshooting, which has been demonstrated to be considerably faster and reliable and less labor-intensive than manual bugging or searching through the trace data or consulting the domain experts of different applications.

By testing on the real production system in Alibaba datacenters, our RCA system can help system administrators to correctly identify the true problems rather than detecting symptoms. This will foster true examination of causes, and allows application owners to identify potential performance bottlenecks in their application implementations and revise them accordingly. This is an important step in reaching the root causes of problems impacting the improvement in daily operations and management in data centers.

Thus, the contributions of this paper are summarized:

- An empirical approach for RCA based on graph matching is proposed for automatically identifying what applicatoin service brings the performance anomaly in nearly real time without requiring instrumentation of application code.
- Relative importance weight is introduced into the graph to locate the critical nodes and lines. This weight is dynamically generated through the graph analysis.
- The effectiveness of this approach has been estimated in Alibaba's large-scale production distributed systems, which have thousands of traces and tens of thousands of servers.

The rest of this paper is organized as follows. Section II introduces the related work and background of our research. Section III introduces our proposed empirical approach. Section IV details the proposed system for analyzing anomalies in real-time monitoring data. Section V presents an evaluation of our approaches with existing detection approaches in a real-time production system in Alibaba datacenters. Lastly, conclusions and future work are presented in Section VI.

II. RELATED WORKS

Our research focuses on real-time RCA based on graph data mining techniques and trace data from the production environment. Tracing system activities via tagging incoming requests has been implemented in several previous works such as X-Traces [1], Pinpoint [2] and X-ray [3]. X-Trace records network flow across protocols and layers, but does not support RCA [1]. Pinpoint is currently limited to the single-machine tracing of interactions among J2EE middleware components to localize faults rather than performance anomalies [2]. X-ray, a performance summarization system deployed in production software, attributes performance costs to each basic block executed by a software application and computes a list of root causes ranked by performance costs. However, it is limited by binary instrumentation and additional user input of application configurations [3].

Our proposed trace-based RCA is a relatively unexplored area. The traditional RCA is still a human-centric, business-aligned process that is considered to be outside the scope of analytics and machine learning (ML) techniques to begin with [4]–[6]. However, a production distributed system can be described as a large and complex network, where nodes in the graph represent individual applications, servers, locales and other entities, whereas lines in the graph represent relationships between pairs of entities. These data objects cannot always be treated as points lying in a multidimensional space independently. Application-specific performance data may exhibit inter-dependencies which should be accounted for during the root-cause detection process. To leverage for the increasing multidimensionality and interdependency of performance issues in data centers, many data-driven techniques have been developed in the past decades, especially for automatically spotting outliers and causes in unstructured collections of multidimensional data points [5]. The machine learning process involves modeling system behavior during normal operations and the trained model will be applied in real-time to find unexpected behaviors [4]. The common phases of ML-assisted RCA include normative/critical process discovery, process clustering/classification, and process similarity comparison [6], [7].

Process discovery is the task of deriving a process model from process execution data that are typically stored in event logs, which in turn are generated by information systems that support the process execution [6], [7]. In [6], process event log data are enriched with relevant contextual information and normalized attributes to explain the possible cause(s) of risk incidents. Other ML approaches have been mentioned to extract critical process in [7]–[9]. To deal with nontrivial constructs and/or the presence of noise in the event logs, Aalst et al introduced a genetic algorithm which is a robust global search method to mine process models. The approach used a causal matrix to discover Petri net models [8]. To incorporate multiple process metrics across multiple data sources, Alaeddini and Dogan proposed a Bayesian network approach for root cause diagnosis of process variations and provided a probabilistic confidence level of the diagnosis [9]. Multiple sensor metrics have been identified with statistical correlations to the process frauds of interest. Then the sequential data from the sensors can be combined through a cause belief network framework to provide a probabilistic diagnosis of root cause. However, these approaches focus on statistical correlation analysis of interdependent events instead of the system-wide performance tracing, so some relationships may not correctly detected.

For process classification, supervised machine learning approaches such as support vector machines (SVM) have been mentioned in research to detect the known anomalies/faults [10], [11]. For unsupervised algorithms, Julisch proposed an alarm-clustering method to extract alarm groups that share common structural properties and are generally indicative of root causes. Then, these alarm groups will be presented to a human expert for confirmation and can be removed to reduce the future alarm loads once identified [12]. Almanmory and Zhang introduced a clustering-based approach to identify the root-causes in the false alarms generated by the intrusion detection systems, which greatly helps the security analyst in identifying the root causes and then reduces the alarm load by over 70% in the future [13]. These approaches calculated the similarity between events or alarms to identify the reasons for performance issues, but they ignored the causal dependency in a business process.

In addition to process discovery and classification used to identify the dominant process, the heterogeneous nature of a distributed system with dynamical dependency applications determines the roles of each node in the production system that can be quite different. Identifying the most critical node or ranking their relative importance of each transition via quantitative analysis in large scale networks is thus very significant, which allows us to better control the potential cascading failures and spreading initiated by the so-called important nodes. This has been an active area of research in many different fields, primarily in the study of social networks and web graph analysis [14]. A number of different quantitative approaches have been developed to characterize the importance of a node in a graph relative to all other nodes. For example, a variety of "centrality" measures have been proposed by sociologists to characterize the quantitative properties of social network [15]. Other statistical-based quantitative graph modeling approaches have been widely used, such as the embedding of social network data in latent Euclidean spaces [16]. The process discovery approaches were used to guide the design of our approach. However, none of the above systems have been designed for or tested in a large-scale data centers.

Our work is heavily inspired by the past literature that details the graph-based anomaly detection analysis [17]–[20]. The graph based approaches can be categorized as unsupervised vs semi-supervised approaches, static vs dynamic graphs, and attributed vs plain graphs. Graph scoring and matching is used to find anomalies through comparing with normal graph patterns [19]. Several statistical methods (relative importance and quantile analysis) were employed to rank the importance of applications and the corresponding transitions in the production system, to determine the most likely candidate or bottleneck responsible for the detected anomaly [21], [22]. In this paper, we provide a practical solution to root-cause diagnosis in a large-scale production environment, via collecting trace data online from production distributed system, running anomaly detection online, determining the critical and normative patterns as offline processes, and then triggering RCA in a responsive manner.

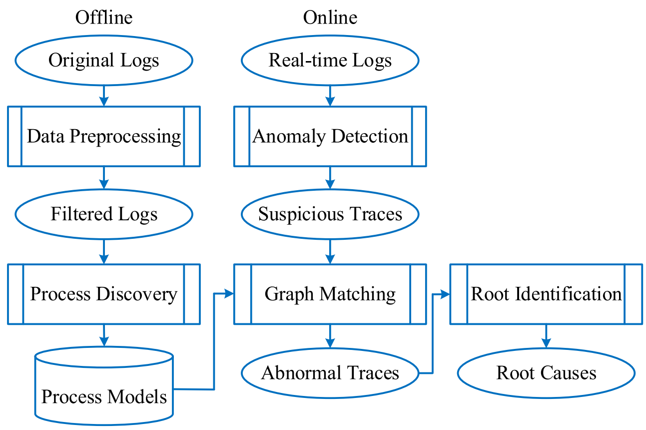

FIGURE 1. Trace-level RCA framework.

III. EMPIRICAL APPROACH OVERVIEW

This paper proposes an empirical approach to RCA in a large datacenter. The approach introduces process modeling for locating suspicious traces that bring anomaly. It includes two parts: offline modeling and online analysis. Offline modeling is to build normative patterns and their performance metrics from massive monitoring data, whereas online analysis is to find the suspicious traces through graph matching, as shown in Fig. 1.

A. OFFLINE MODELING

Offline modeling is to generate normal business processes from historical execution logs. It has two critical steps: data preprocessing and process discovery. Data preprocessing is composed of a set of filters which remove the incomplete traces, noises and business independent data. Process discovery rebuilds the execution traces and finds the normative patterns.

1) DATA PREPROCESSING

Data preprocessing is an important step in analyzing the trace data and the associated performance metrics, but some data that might affect the data integrity should be removed first to facilitate process analysis. Statistical methods (nonparametric Tukey's method) combined with domain knowledge are used to eliminate artifacts, normalize the data, and automatically discard counters that significantly violate assumptions or business rules. Business rules are defined as the trading logic in e-commerce. For example, the error codes, tagged with the remote procedure call (RPC) information to indicate the RPC status, are classified into various types by application owners to determine if the trace information is valid for further performance analysis. Issues such as missing/incomplete trace information, expired trace IDs, and wrong RPC types with no business meanings will be removed before further analysis. These data issues are generally occurred at the beginning or the end of each trace, therefore the deletion does not affect the integrity of process mining analysis.

With pre-cleaned trace data, we group the data by its associated business annotations, which are marked by domain experts. Doing so enables us to focus on the normal traces flowing through our target applications. The performance metrics that we collect for analysis include the inbound request size and response time (the time duration of sub-process transition). The non-parametric statistical analysis (Tukey's method) is employed to remove any significant outliers from these performance metrics, no need for marked data.

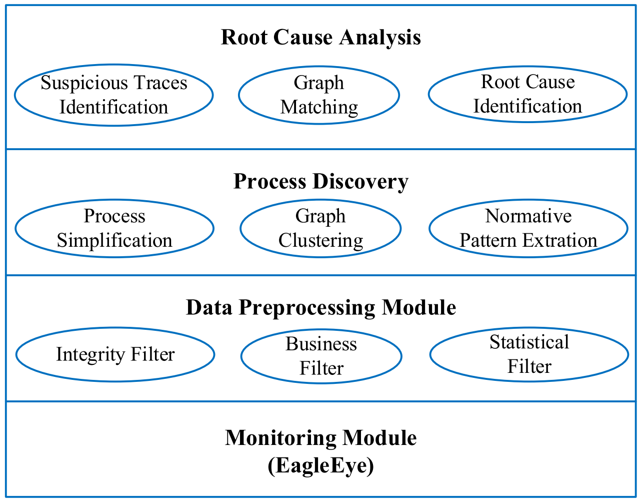

FIGURE 2. Trace-level RCA implementation framework.

2) PROCESS DISCOVERY

A process is defined as an execution path of the program. Process discovery is to find the normative performance patterns in variety of trace data. For the difference of business requests, even the same request with different parameters may have different execution paths and performance metrics. This paper proposes a novel solution for building the normative patterns using the Run-Length Encoding (RLE) which is widely used in sequence analysis, as in Algorithm 1.

The algorithm is designed in five steps:

Step 1: The filtered traces can be formulated as a set trace set T. Each trace is composed of nodes and edges. Nodes represent services, and edges represent service transitions. C is defined as the trace cluster, initially null.

Algorithm 1 The Process Discovery Algorithm

1. T -> traces; C ->trace clusters
2. preprocess the traces with RLE
3. for each trace t_i in T,
   initial trace clusters C with $c_i = \{t_i\}$;
   t_i is used as normative trace NT_i for c_i;
   calculate the distance between c_i and c_j;
4. trace clustering
   4.1 merge two clusters with nearest distance as a new cluster c_n;
   4.2 generate normative trace NT_n from the new trace cluster;
   4.3 recalculate the distance between c_n and others;
   4.4 repeat 4.1 - 4.3 until the nearest distance less than given threshold;
5. for each trace cluster
   use statistical models to generate the normative patterns

Step 2: For each trace, it can be compressed through merging the cycles with repeated nodes and edges. In this way, the trace can be simplified.

Step 3: Initialize the trace clusters set C and the normative traces. Calculate the distance between each clusters using the graph similarity of their normative traces. The distance of traces is defined using the combined distance of optimal matching technique (OM) and substitution cost matrix, detailed in Section IV C.

Step 4: The Hierarchical Clustering Algorithm (HCA) is introduced to group similar traces into a cluster. From the historical data, the traces are execution paths, which are not the same even for the same request. So clustering is one of the common ways to find similar traces.

Step 5: The common parts of the traces and their performance metrics are used as the normative patterns.

Through the process discovery, the normative patterns and their performance metrics can be obtained from the massive monitoring data. This is a practical way to build offline models, using statistical analysis. The normative patterns are weighted graphs and stored as .xml files for further application.

B. ONLINE ANALYSIS

In the production environment, the online analysis of RCA mainly includes the following four steps illustrated in Fig. 2:

1) Sampling and identifying the suspicious traces. The suspicious traces are identified by anomaly detection using our proposed approach in [28].

2) Comparing the suspicious traces with normative patterns which are computed and stored in the offline modeling phase. This is a graph matching approach to find the difference between suspicious traces and normative patterns generated in section IIIA. The difference between suspicious traces and normative patterns will be treated as candidate causes.

3) Identifying the deepest root of issues via relative importance analysis. Through graph matching, there may be over one suspicious node or edge that is different from normative patterns. The deeper the node/edge in the graph, the more

probability it can be root cause. The relative importance of node or edge is considered for locating the root cause.

4) Linking with other event information to confirm if the problem identified is the root-cause and valid for subsequent data operations.

Online analysis is a real-time processing stage to report the potential root causes from the running traces. The identified root causes may not be single for each trace. For over one cause in a trace, the ranking algorithm based on relative importance analysis is used and the top-k strategy is selected.

IV. THE ROOT CAUSE DIAGNOSIS SYSTEM

In this section, we introduce the implementation of the RCA approach and detection system, including the trace data collection module, data preprocessing module, process mining module and real-time implementation module. This suggested real-time RCA system integrates with Alibaba large-scale distributed tracing infrastructure: collects and analyzes the trace data across different interdependent services; identifies the normative application-specific patterns; classifies traces by critical performance metrics; compares the similarity between the sampled suspicious traces and the normative patterns; derives the risk score for each transition in the trace via integrating the relative importance of each application in the network; determines the sources of a performance anomaly and sends alerts to the datacenter administrators if there are any significant deviations in the process patterns of the production data. Fig. 2 illustrates the infrastructure of our trace-level RCA running in the Alibaba production environment.

A. TRACE DATA COLLECTION-EAGLEEYE

EagleEye is the ubiquitous tracing platform for distributed services in Alibaba, which records all the distributed control paths with near-zero intervention from application developers by relying on the instrumentation of common libraries. This provides a comprehensive view of the process flow of incoming requests. It includes four main modules as shown in Fig. 3: trace data generation SDK; real-time ETL data processing system; offline computing cluster and web-based user interface.

The trace data generation SDK is used to generate the trace log. It can easily integrate with each application and record RPC trace data into a log file. The real-time ETL system is the large distributed JStorm clusters to extract, transform and store the trace information into our internal data store for further persisting and querying monitoring data. The historical trace data and the associated application-specific performance metrics are sampled and stored in the offline computing cluster, which enables further offline analyses. Finally, all the trace-level performance monitoring can be exhibited in the front-end web interface. Because most of the applications in Alibaba rely on the unified middleware, the integration with EagleEye requires minimum intervention from application owners.

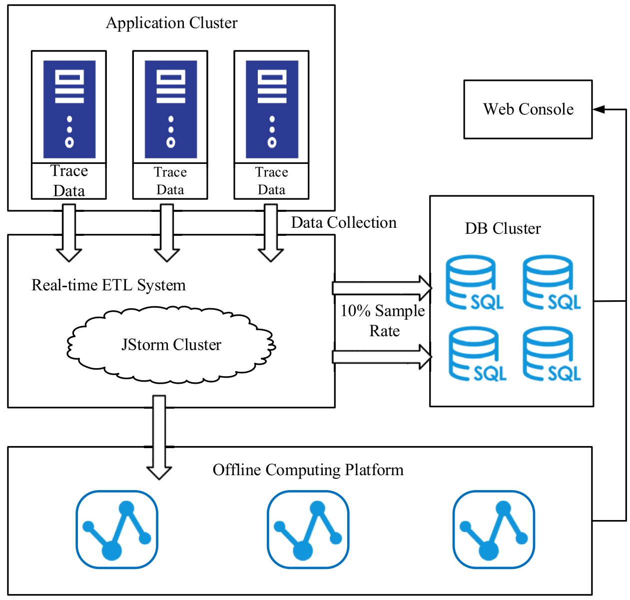

FIGURE 3. Architecture of trace collection system.

Each application will log the trace data in each RPC by default. The trace data consists of three parts: trace information, RPC information and business information. Trace Id is the unique identifier for each trace, and RPC ID indicates the location and level of each RPC. The RPC information includes RPC type (represented by different codes), service name, result code, response time and etc. It can indicate the status of each RPC and help the developer to identify issues. We use the annotation-based schemes to explicitly tag each record with a global identifier (trace Id and RPC Id) that links these message records back to the originating request, thereby reconstructing the causal relationships between the individual spans in a single distributed trace. Meanwhile, application owners are allowed to augment the trace with their own business annotations. For instance, application owners can include tags related to the data operations of their specific interests, such as stress testing, grey deployment, business-classified services, and other user-defined service information.

We illustrate how a tracing infrastructure for our distributed services records information about all the work done in a system with a given initiator in Fig. 4. In a typical EagleEye trace, tree nodes/vertexes are basic units of work that each request passes through. The edges indicate a causal relationship between a node and its parent node. Independent of its place in a larger trace tree, though, a node is a simple log of timestamped events which encode the start and end time of a transition, any RPC timing data, and zero or more application-specific annotations. For example, Fig. 4 shows a service with 5 application-specific servers: a front-end (A), three middle-tiers (B, C, and D), one messaging application and two backends (DBs). When an incoming user request (the initiator in this case) arrives at the front-end, it sends three RPCs to servers B, C and one messaging application.

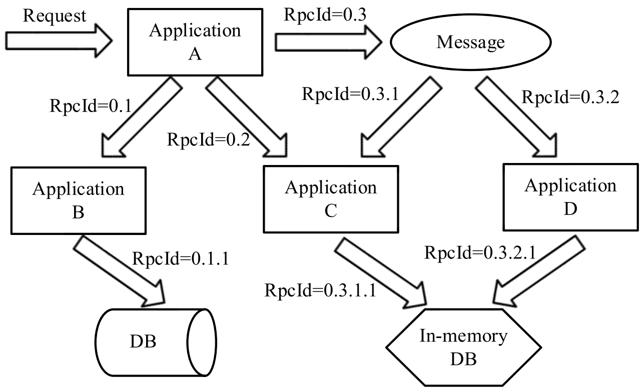

FIGURE 4. The path taken through a simple serving system on behalf of user request. The letter-labelled nodes represent processes/application-ns in a distributed system. The RpcId represents the sequence of event taken through the initial user request to the back-end database.

B can respond right away and the data will be logged into the backend database, but C and D, which are the subscribers of the messaging application, also receive RPC via messaging and access the information from the in-memory database through RPC calls.

The overhead associated with trace generation, collection and storage is the most critical segment of EagleEye's performance footprint. To ensure a low performance overhead, a dynamic sampling approach is utilized to ensure the representativeness of datacenter traffic flow profiles and the stability of application performance. The sampling rate is parameterized by the inbound traffic/loads per unit time. For instance, workloads with low traffic automatically increase their sampling rate while those with very high traffic will lower it so that overheads remain under control. Users can also define the sampling rate according to the specific IT operations such as stress testing, business promotions and etc. The default sampling rate is one-tenth of incoming requests to be traced and logged into a local disk for subsequent analysis, given no historical information exists for the new workload.

With negligible overhead, these four modules collect continuous, real-time, and representative trace data at the datacenter scale, thereby providing system-wide performance insights for administrators and developers.

B. TRACE DATA PREPROCESSING

Data preprocessing is implemented as a set of filters, including integrity filter, business filter and statistical filter, which are used to preprocess the incomplete traces and noises.

The integrity filter is to remove the monitoring data without a complete trace. Some execution data may be lost during the monitoring phase. For example, there are some invocations that are not started from the application entrances. It is difficult to analyze these invocations and their performance metrics with no contexts. The filter is executed automatically.

The business filter is performed with predefined business rules, which are provided by domain experts according to the business annotations. The business rule is represented as business terms and execution actions on these business terms, like order, trade, payment in e-commerce. The execution traces that cannot be mapped to the business rules are removed. For example, wrong RPC types, re-connection messages, etc.

The statistical filter is to remove the low frequency execution traces in the given time windows. The proposed approach in this paper is based on statistical analysis to build normative patterns. The traces that are few executed can be accurately modeled.

C. TRACE PROCESS DISCOVERY

With the preprocessed trace data filtered by the specific business annotations, we use graph data mining techniques to extract the process model for each critical business process. It involves the following steps: processing and simplifying the traces; extracting the critical/dominant processes; grouping the processes based on clustering techniques.

1) Trace Simplification: To simplify the long, redundant trace information, the same type of middleware applications will be replaced with the same encoding code. We also use a lossless data compression run-length encoding algorithm to replace large sequences of repeating events with the only items of this event followed by a counter (also called run-length) showing the number of times this event is repeated. Thus, we introduce a new metric 'duration' to represent the run-length of each event.

2) Process Clustering: With the simplified and cleaned trace data including the process structure and the associated edge properties (response size, response time, and duration), we group the traces with similar patterns/characteristics via process clustering techniques and graph similarity measurement. Hierarchical agglomerative clustering used here requires the specification of both a dissimilarity/distance matrix and a linkage algorithm, e.g., single-link, complete-link or Ward's method [23].

The distance matrix between traces includes the difference in process sequences and the edge properties. (1) The optimal matching technique (OM) is used to generate edit distance that is the minimal cost, in terms of insertions, deletions and substitutions, for transforming one sequence into another [24], represented as $dist\_1_{ij}$. (2) In addition to the string edit distance, a substitution cost matrix is computed according to the edge properties-a weighted average of three key measurement values, represented as $dist\_2_{ij}$. The distance of traces $dist_{ij}$ are defined as in (1):

$$dist_{ij} = w_1 * dist\_1_{ij} + w_2 * dist\_2_{ij} \tag{1}$$

where w_1 and w_2 are weights for single distance, satisfying $w_1 + w_2 = 1$.

Given the various magnitudes of different performance metrics, each measurement is firstly normalized by means of hyperbolic tanh function as in (2), which is a rescaling of the logistic sigmoid and the $[-1, 1]$ out-put range tends to be more suitable for graph clustering. The cost matrix includes two components: the sum of square difference in edge properties of the longest common subsequence (LCS) between traces; the difference in subsequence beyond LCS, which is the sum of squares of each edge property. The square root of the sum of these two components is used to estimate the cost matrix of graph properties.

$$Tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}) \tag{2}$$

For our trace clustering with unlabeled data, we select using hierarchical algorithms because: (1) hierarchical algorithms often require less parameters, (2) parameters of non-hierarchical algorithms are often not trivial and very context-specific, and (3) end-users of our RCA approach have more freedom in choosing a selection of clusters according to the level of abstraction they pursue. To determine the optimal number of clusters in partitioning the traces, three methods- elbow, silhouette and gap statistic methods are used for comparisons [23].

3) Normative Process Extraction: Once traces are partitioned into groups by similar patterns, the normative processes for each group can be retrieved through the following methods: (1) the most relevant sequential patterns are extracted via Interesting Sequence Miner (ISM), which makes use of a structural expectation-maximization framework and ranks them using an associated probability measure of interestingness [24], [25]; (2) the graph properties associated with the most relevant subsequence are estimated by statistical approach. The confidence intervals for each edge property can be estimated by the modified Z-statistic based on the median absolute deviation (MAD). The statistical upper and lower bound should be located within a range where the modified Z-scores with an absolute value which is no greater than 3.5 to achieve 95% confidence, which are an empirical values in industry.

Now, we build a normative pattern for each trace group with the dominant subsequence pattern and its corresponding edge properties (the expected value and the 95% confidence levels). This data is used for path comparison to determine if the sampled traces are significantly different from the normative patterns. The R package TraMineR [26], a toolbox for categorical sequence data originally designed for life trajectory modeling in the social sciences, is used here to generate trace clustering and extract frequent subsequence.

D. REAL-TIME RCA IMPLEMENTATION

In the production environment, the real-time implementation of trace-level RCA is shown in Fig. 5, which mainly includes the following steps.

1) Sampling and identifying the suspicious traces;

2) Comparing the suspicious traces with its normative patterns which are computed and stored in offline analysis;

3) Identifying the deepest root of issues via relative importance analysis;

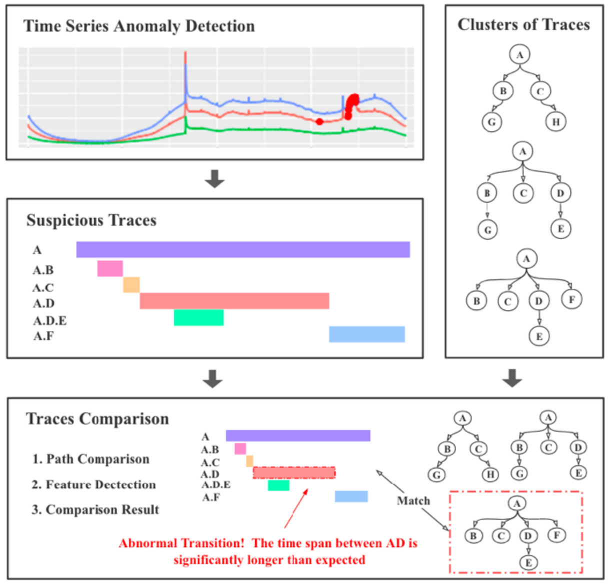

FIGURE 5. Real-time RCA Implementation.

4) Linking with other event information to confirm if the problem identified is the root-cause and valid for subsequent data operations.

Firstly, when a built-in anomaly detector finds an anomaly in the application performance monitoring system, it sends an event to a collection of anomaly handlers. The event consists of a unique anomaly identifier, event timestamp, application name and the detector's time sliding window corresponding to the anomaly. Furthermore, this anomaly handler will trigger the random selection of suspicious traces that occurred within the same time constraint. Doing so enables us to focus exclusively on comparing the traces with issues rather than comparing all traces. This ensures the path comparison and root-cause determination can be performed at a reasonable computation cost.

Secondly, the selected suspicious traces will be compared against the normative pattern in the corresponding trace group. This is done by comparing the graph properties of their most relevant sequential patterns. The anomalous transition(s) will be flagged if they fall outside the 95% statistical confidence limits.

Thirdly, we employ the regression-based relative importance analysis in terms of the explained variance of the total response time to identify the most relevant cause of the anomaly [18]. Here, the total response time (T_{total}) of a typical inbound request is regressed against the latency of each transition (T_{xi}). For a time window during a normal status, a linear additive model is fitted over the per-transition latency (T_{xi}), as in (3), where N is the transition number in the request.

$$T_{total} = \sum_{i=1}^{N} T_{x_i} \tag{3}$$

All predictors will be ranked based on their contribution to the variance in T_{total} using the following algorithms including:

- Lindemann Merenda and Gold (LMG) algorithm- partition R-square value of the regression fit over the relative contribution of each predictor;
- First algorithm: compare the R-square values from regression models with one regressor only;
- Genizi algorithm: decompose R-square in multiple regression with correlated regressors;
- Last algorithm: what each regressor is able to explain in addition to all other regressors that are available;
- Betasq algorithm: assessing individual contributions consists in considering scale-invariant standardized coefficients;
- Pratt algorithm: decompose the R-square values by the product of the standardized coefficient and the correlation;
- Car algorithm-R-square value decomposition according to squares of scores [27].

Therefore, the component with the highest ranked contribution can be selected as the candidate of the origins of the anomaly. With the relative importance, the health indicator of each suspicious trace can be measured via its deviation from the normative pattern, which is the weighted average of the numeric difference in graph proprieties between each transition. The weighted deviation is the arithmetic mean in which each transition is multiplied by its corresponding weight, based on its relative importance and is summed and divided by the total weight.

Lastly, with the list of candidate components for the anomaly and performance bottleneck, we will also relate these candidates with other application-level information such as the log of any recent change in workload, and the exception/error log of the application. We can finalize and assign different priorities to application owners according to their health indicators. This allows the application owners and administrators to uncover the causes of a performance anomaly or potential bottleneck in the data centers, and take corrective action in a responsive manner.

V. RESULT

In this section, to evaluate the efficacy of our RCA system in the distributed production environment, we quantify the accuracy of identifying components that cause performance anomaly in Alibaba's production scenarios. In these scenarios, there are tens of thousands physical servers, on which complex systems are running. When a fault occurs, it takes several hours of tens of engineers to locate the root cause. Through this experiment, the performance and scalability of the RCA system was proved.

A. NORMATIVE PROCESS EXTRACTION

We use RCA system to diagnose performance problems identified by an online anomaly detector in EagleEye production system. The normative process is extracted based on the normal trace data collected in the offline analysis phase. The paths in the distributed system are highly variated.

TABLE 1. The ranking of paths based on frequency counts of the sampled traces.

Path id	Path	Path Freq
1	1-33	0.7355
2	1-33->1-1->1-33	0.0416
3	1-33->1-1	0.0155
4	1-33->1-1->1-33->1-33->1-33->1-33->1-4->1-33	0.0041
5	1-33->1-1->1-33->1-4->1-33	0.0041

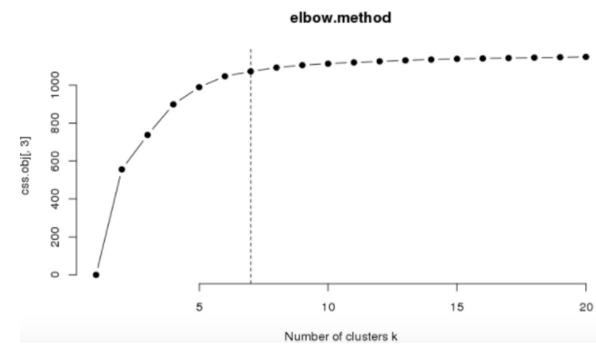

FIGURE 6. The result of optimal number of clusters selected by elbow method.

Table 1 shows the top frequent paths in the production environment. Therefore, we employed the hierarchical clustering techniques to group similar paths and narrow down the scope for further process comparisons. The number-labelled nodes represent applications in a distributed system.

With the performance metrics (response size, response time and duration) associated with the edge transition of each trace and the normalized distance matrix among different traces, we employed three methods (elbow, silhouette and gap statistic methods) to determine the optimal number of clusters. The result of the elbow method showed the most appropriate cluster numbers for subsequent hierarchical clustering (Fig. 6).

For each cluster, we extract the critical sub-sequences that best represent the process characteristics using the ISM algorithm. Fig. 7 gives an example of process clusters. It shows how a complex business process in Alibaba production environment can be best compressed into the most relevant subsequence (highlighted in different colors) for process comparison. In addition to the critical process mined during this process, the associated graph properties are estimated with the expected values of three key performance metrics and its 95% confidence limits based on the sampled trace data.

In addition to normative process extraction and process clustering, we employ the regression-based relative importance analysis to determine the most critical nodes and transitions in large-scale networks. Fig. 8 shows the relative importance of applications and transitions measured by the selected algorithms (LMG, First, Genizi, Last, Betasq, Pratt and Car), which have been described in Section IV D. The line width is proportional to the importance score of each transition in a network. Taking one network as one example,

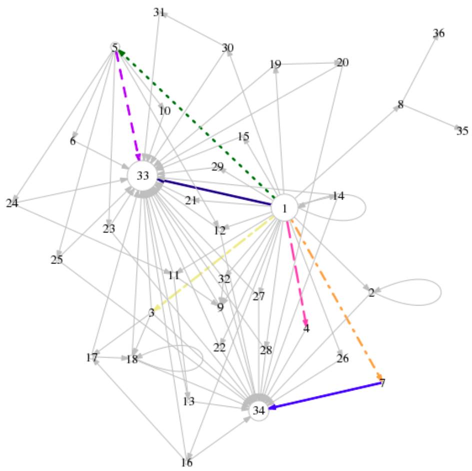

FIGURE 7. Critical transitions in the normal trace data. The number-labelled nodes represent applications in a distributed system; Each directional line represents the transition between applications; The lines highlighted in different colors represent the critical transition mined through Interesting Sequence Miner.

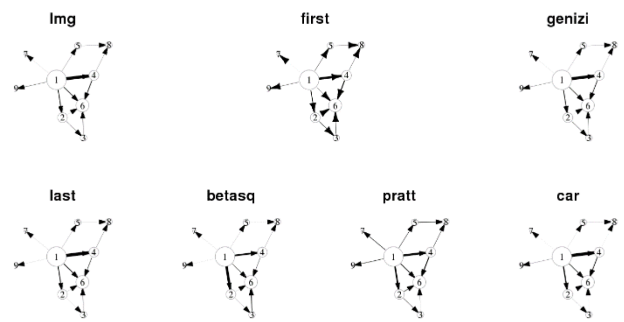

FIGURE 8. Relative importance of each transition in a network based on several algorithms. The number-labelled nodes represent each applicat-ion; Each directional line represents the transition between application-ns; The line width represents the relative importance of transition in the distributed system.

the flow from application 1 to application 4 has been consistently identified as the most important transition via all the methods used. Each application and its transition can be finalized and assigned with different priorities according to their ranks of relative importance in a large network. This will reflect on the measurement of the health indicator for each suspicious trace. The transition or application with the largest deviation score will be considered as the origins of performance issues.

B. ROOT CAUSE DIAGNOSIS ACCURACY

In the following study, by comparing with the common fixed threshold methods on the trace data collected from Alibaba production datacenters, we evaluate the efficiency of our root cause diagnosis model. Table 2 shows the fixed threshold model for each key performance metrics, including response time, response size and request size. These are predefined by domain experts.

We firstly sample the traces with the performance issues (high latency) identified by the online anomaly detector of EagleEye with a sampling rate of one trace out of ten total incoming web requests within a window size of 1 hour. The total sample size for the normal and abnormal traces are illustrated in Table 3. Our root-cause detection system will automatically identify the corresponding normative pattern for each suspicious trace, compare their graph properties against the 95% confidence intervals pre-determined offline for normative patterns, and flag a transition where the key performance score is beyond 3.5 median absolute deviations over the median performance statistics of a normative pattern. Each detection model is trained separately on each sampled trace; i.e., learned parameters do not carry over. Since this is data collected from the production system, we have no control of the anomalies that manifested, nor do we have knowledge about them. Thus, in our evaluations, we will compare the roots of anomalies diagnosed by the various techniques with the anomalies examined and confirmed by the domain experts.

TABLE 2. The fixed thresholds for each key performance metric defined by domain experts.

Metric	Normal range	Unit of measurement
Response Time	[0,107]	ms
Request Size	[57,81]	byte
Response Size	[12,731]	byte

TABLE 3. The trace data collected for model evaluation.

Data set	Path number
Abnormal path	30
Normal path	24

To assess the efficiency of our detection models with the traditional methods, we compute three standard metrics: Recall, Precision, and F-measure (or the balanced F-score). Precision (also called positive predictive value) is a measure of how many instances are correctly classified among all positive predictions including true and false positives. Recall (also known as sensitivity, true positive rate) is a measure of a proportion of all real positive observations that are correct. F-measure is the weighted harmonic mean of precision and recall. Recall and precision are two widely used performance metrics in classification. Precision permits to measure the fidelity of the classification model regarding each particular class, whereas recall measures the per-class accuracy. Table 4 shows the outcome analysis of recall, precision and F-measure relating to the root-cause detection schema and the other three static threshold methods. The first three results represent the fixed threshold approaches for key performance metrics. Our RCA method is tested with the ensemble metrics of response time, request and response size. Our RCA method has a highest F-score 83%, with the high recall rate of 93%

TABLE 4. Root cause diagnosis result.

Metric	Precision	Recall	F1-score
Fixed Response Time Threshold	0.55	0.20	0.29
Fixed Request Size Threshold	0.68	1.00	0.81
Fixed Response Size Threshold	0.19	0.10	0.13
RCA method	0.72	0.93	0.83

and a highest 72% precision. In several cases, multiple root causes contribute to the problem, our RCA correctly ranked the true root cause(s), missing one test subject due to the tightness of confidence bounds defined for the normative process. Our approach outperforms the other static threshold techniques. High F-scores achieved by testing on three typical performance counters, demonstrate that our proposed approach is precise and robust in identifying the anomalous behaviors.

In the data centers, the response time and response size vary considerably as the number of clients, message size, request frequency change and the application performance. There is not a one-size-fits-all threshold approach in determining the best sensitivity-to-specificity tradeoff for anomaly detection and root-cause diagnosis. With multiple testing on the high latency of incoming requests sampled from the production system, our overall results are very promising. Further, the computation time and resource needed is quite reasonable when compared to the time and effort of manual inspection and analysis: our RCA method took less than 1 minute on average to identify the root cause, while the traditional manual inspection of a performance anomaly usually required more than 30 minutes of analysis time across multiple teams.

The proposed system is demonstrated to perform root-cause detection in real time, which is much more efficiently with significantly improving the performance of the anomaly detection in the data centers compared to the other available state of the art solutions. However, currently RCA addresses the diagnosis of performance issues on an ad-hoc basis. It's limited by the following aspects: (i) the nature and granularity of data we can collect from EagleEye system. For instance, the performance data collected is at the minute level and the taint inside the OS kernels are not tracked so our RCA cannot infer causal dependencies; (ii) the accuracy of RCA detection is sensitive to the graph properties estimated for normative processes using statistical learning approach. Since our approach involves unsupervised learning instead of supervised learning (i.e., no training or labelling is involved at the current stage). We hope to address these limitations more thoroughly in the future development, either by instrumenting the kernel, or by collecting more detailed performance data from other real-time monitoring system, or by developing a reinforcement learning module via continuous sampling or labelling to improve the accuracy and sensitivity of the detection.

VI. CONCLUSION

Troubleshooting performance issues is notoriously challenging in distributed systems. Based on large-scale tracing and logging systems in Alibaba distributed environment, a trace-level and graph mining based system to automate root-cause determination is developed in this paper. Our system allows the discovery of normative patterns and the corresponding key graph properties which are stored and updated offline as a knowledge base for subsequently being used in trace-level risk estimation and identification of anomalous events. It does not require system administrators and end users with the intimate understanding of application-level source code and detailed application knowledge. Our system is ideal for causal users and system administrators to focus on troubleshooting software systems on correcting the potential candidates of causes identified and ranked by our RCA system.

There are still some works that can be improved, including how to automatically generate business processes from as-is applications, the scientific enhancement of the empirical approach and more production scenarios for evaluation. In the future, some supervised approaches may be introduced to improve the precision and recall of the RCA.

REFERENCES

[1] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, ''X-trace: A pervasive network tracing framework,'' in *Proc. 4th USENIX Conf. Netw. Syst. Design Implement.*, Cambridge, MA, USA, Apr. 2007, p. 20.

[2] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, ''Pinpoint: Problem determination in large, dynamic Internet services,'' in *Proc. Int. Conf. Dependable Syst. Netw.*, Jun. 2002, pp. 595–604. doi: 10.1109/DSN.2002.1029005.

[3] M. Attariyan, M. Chow, and J. Flinn, ''X-ray: Automating root-cause diagnosis of performance anomalies in production software,'' in *Proc. 10th USENIX Conf. Operating Syst. Design Implement.*, Oct. 2012, pp. 307–320.

[4] H. Lal and G. Pahwa, ''Root cause analysis of software bugs using machine learning techniques,'' in *Proc. 7th Int. Conf. Cloud Comput., Data Sci. Eng.–Confluence*, Jan. 2017, pp. 105–111. doi: 10.1109/CONFLUENCE.2017.7943132.

[5] A. Chigurupati and N. Lassar, ''Root cause analysis using artificial intelligence,'' in *Proc. Annu. Rel. Maintainability Symp.*, Orlando, FL, USA, Jan. 2017, pp. 1–5. doi: 10.1109/RAM.2017.7889651.

[6] S. Suriadi, C. Ouyang, W. M. Aalst, and A. H. Hofstede, ''Root cause analysis with enriched process logs,'' *Business Process Management Workshops* (Lecture Notes in Business Information Processing), vol. 132. Berlin, Germany: Springer, 2012, pp. 174–186. doi: 10.1007/978-3-642-36285-9_18.

[7] M. Leoni, W. M. Aalst, and M. Dees, ''A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs,'' *Inf. Syst.*, vol. 56, pp. 235–257, Mar. 2016. doi: 10.1016/j.is.2015.07.003.

[8] W. M. Aalst, A. K. Medeiros, and A. J. Weijters, ''Genetic Process Mining,'' in *Proc. Int. Conf. Appl. Theory Petri Nets*, G. Ciardo and P. Darondeau, Eds. Berlin, Germany: Springer, 2005, pp. 48–69. doi: 10.1007/11494744_5.

[9] A. Alaeddini and I. Dogan, ''Using Bayesian networks for root cause analysis in statistical process control,'' *Expert Syst. Appl.*, vol. 38, no. 9, pp. 11230–11243, 2011. doi: 10.1016/j.eswa.2011.02.171.

[10] W. Hu, Y. Liao, and V. R. Vemuri, ''Robust support vector machines for anomaly detection in computer security,'' in *Proc. Int. Conf. Mach. Learn. Appl.*, Los Angeles, CA, USA, 2003, pp. 168–174.

[11] Z. J. Abbaszadeh, ''Supervised fault detection using unstructured server-log data to support root cause analysis,'' M.S. thesis, Tampere Univ. Technol., Tampere, Finland, 2014.

[12] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 443–471, Nov. 2003. doi: 10.1145/950191.950192.

[13] S. O. Al-Mamory and H. Zhang, "Intrusion detection alarms reduction using root cause analysis and clustering," *Comput. Commun.*, vol. 32, no. 2, pp. 419–430, 2009. doi: 10.1016/j.comcom.2008.11.012.

[14] Y. Wang, Z. Di, and Y. Fan, "Identifying and characterizing nodes important to community structure using the spectrum of the graph," *PLoS ONE*, vol. 6, no. 11, 2011, Art. no. e27418. doi: 10.1371/journal.pone.0027418.

[15] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge, U.K.: Cambridge Univ. Press.

[16] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *J. Amer. Stat. Assoc.*, vol. 97, no. 460, pp. 1090–1098, 2001. doi: 10.1198/016214502388618906.

[17] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data Mining Knowl. Discovery*, vol. 29, no. 3, pp. 626–688, 2015. doi: 10.1007/s10618-014-0365-y.

[18] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *J. Internet Services Appl.*, vol. 1, no. 1, pp. 19–30, May 2010. doi: 10.1007/s13174-010-0003-x.

[19] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Appl. Math. Lett.*, vol. 21, no. 1, pp. 86–94, Jan. 2008. doi: 10.1016/j.aml.2007.01.006.

[20] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2003, pp. 631–636. doi: 10.1145/956750.956831.

[21] J. P. Magalhães and L. M. Silva, "Root-cause analysis of performance anomalies in web-based applications," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, Mar. 2011, pp. 209–216. doi: 10.1145/1982185.1982234.

[22] J. P. Magalhaes and L. M. Silva, "Detection of performance anomalies in Web-based applications," in *Proc. 9th IEEE Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, USA, Jul. 2010, pp. 60–67. doi: 10.1109/NCA.2010.15.

[23] V. Aalst and M. P. Wil, *Process Mining*, Berlin, Germany: Springer.

[24] A. Abbott and A. Tsay, "Sequence analysis and optimal matching methods in sociology: Review and prospect," *Sociol. Methods Res.*, vol. 29, no. 1, pp. 3–33, 2011. doi: 10.1177/0049124100029001001.

[25] J. Fowkes and C. Sutton, "A subsequence interleaving model for sequential pattern mining," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, 2016, pp. 835–844. doi: 10.1145/2939672.2939787.

[26] A. Gabadinho, G. Ritschard, N. S. Mueller, and M. Studer, "Analyzing and visualizing state sequences in R with TraMineR," *J. Stat. Softw.*, vol. 40, no. 4, pp. 1–37, 2011. doi: 10.18637/jss.v040.i04.

[27] U. Gömping, "Relative importance for linear regression in R: The package relaimpo," *J. Stat. Softw.*, vol. 17, no. 1, pp. 925–933, 2006. doi: 10.18637/jss.v017.i01.

[28] W. Y. Zhu, M. Xia, H. F. Wang, J. D. Chen, L. Liu, and Z. G. Cai, "A dynamic ensemble learning approach for online anomaly detection in Alibaba datacenters," in *Proc. 18th Ind. Conf. Data Mining*,

ZHENGONG CAI received the B.S. and Ph.D. degrees from the School of Computer Science and Engineering, Zhejiang University, China, in 2006 and 2011, respectively. He is currently with the College of Software Technology, in 2011. His research interests include software engineering, data mining, and cloud computing.

WEI LI received the B.S. degree in information security from North China Electric Power University, China, in 2018. He is currently pursuing the M.S. degree with the School of Aeronautics and Astronautics, Zhejiang University, China. His research interests include artificial intelligence, UAV ADHOC networks, and software-defined networks.

WANYI ZHU received the Ph.D. degree in operations research and statistics from Pennsylvania State University. She is currently a Full Stack Data Scientist experienced in optimization, machine learning, and big data analysis. Her research interests include e-commerce, risk management, and finance. She is a Fellow of the Society of Actuaries.

LU LIU is currently pursuing the M.S. degree in software engineering with Zhejiang University, China. Her research interests include data mining and machine learning.

BOWEI YANG received the Ph.D. degree in computer science and technology from Zhejiang University, China, in 2011. He is currently an Associate Professor with the School of Aeronautics and Astronautics, Zhejiang University, in 2016. His main research interests include software-defined satellite networking and UAV ADHOC networks, AI-assisted wireless communications technology, and AI-Big data joint optimization in cellular networks and D2D communications.