Front. Comput. Sci., 2025, 19(1): 191201
https://doi.org/10.1007/s11704-023-2701-0

REVIEW ARTICLE

Performance issue monitoring, identification and diagnosis of SaaS software: a survey

Rui WANG[1], **Xiangbo TIAN**[2], **Shi YING** (✉)[2]

1 College of Mining and Safety Engineering, Shandong University of Science and Technology, Qingdao 266590, China
2 School of Computer Science, Wuhan University, Wuhan 430072, China

Abstract SaaS (Software-as-a-Service) is a service model provided by cloud computing. It has a high requirement for QoS (Quality of Software) due to its method of providing software service. However, manual identification and diagnosis for performance issues is typically expensive and laborious because of the complexity of the application software and the dynamic nature of the deployment environment. Recently, substantial research efforts have been devoted to automatically identifying and diagnosing performance issues of SaaS software. In this survey, we comprehensively review the different methods about automatically identifying and diagnosing performance issues of SaaS software. We divide them into three steps according to their function: performance log generation, performance issue identification and performance issue diagnosis. We then comprehensively review these methods by their development history. Meanwhile, we give our proposed solution for each step. Finally, the effectiveness of our proposed methods is shown by experiments.

Keywords SaaS software, performance log generation, performance issue identification, performance issue diagnosis

1 Introduction

SaaS (Software-as-a-Service) provides software as service to consumers [1−5], whose key factor is QoS (Quality of Software). Among QoS [6−10], performance is the most important attribute, which directly affect the user experience and satisfaction for SaaS software. In the running environment provided by cloud computing, if the average time of SaaS software service response is too long, thus violating Service Level Objective (SLO) [11], it is believed that it is suffering a performance issue [12]. When this happens, it tends to cause the dissatisfaction of consumers, even causing the large economic loss [13,14]. Therefore, the operation and maintenance managers need to timely discover performance issues and take measures to ensure the normal operation of system. However, as the SaaS software is deployed in the cloud platform environment, applications or services at all levels interact frequently, resulting in massive data generated by various components of the system, and many of them are multidimensional data with noise, which not only increases the difficulty of identifying and diagnosing SaaS software performance problems in the traditional way, but also reduces the timeliness and accuracy of identification and diagnosis. Therefore, manual identification and diagnosis for performance issues is typically expensive and rely on empirical knowledge due to the complexity of the application software and the dynamic nature of the deployment environment. As a result, it is vital and valuable how to automatically identify and diagnose performance issues of SaaS software. In recent years, with the rapid development of machine learning and deep learning, more and more auto-learning methods are proposed, which are data-driven and suitable to automatically find the data distribution from a large amount of data. Based on this understanding, machine learning method is very suitable to solve performance identification and diagnosis problem. However, this problem is non-trivial because several challenges exist in automatically identifying and diagnosing performance issues of SaaS software.

(1) Insufficient information: there are many reasons leading SaaS software to performance issue, such as the problems of software, the lack of resource in the running environment, the user explosive requests and the third-party services. The performance issue identification and diagnosis need various information including the information of software and the running environment. However, the information is often insufficient, which can affect the accuracy and timeliness.

(2) Large data: the traditional log-based performance issue identification and diagnosis methods usually need domain-specific background knowledge obtained by the analysis of the managers for log. However, the frequent interaction between different compositions of SaaS software can generate millions of log data including lots of noise, which makes it difficult to analyze the log data.

(3) Complex interaction: with the rapid development of microservice architecture, the SaaS software is becoming more and more complex. The complex interaction brings the

Received November 18, 2022; accepted November 5, 2023

E-mail: yingshi@whu.edu.cn

challenges to performance issue identification and diagnosis methods.

(4) Muti-dimensional information: there are different dimensions of information collected from the SaaS software. These data describes the running state of the SaaS software from different angles. How to reasonably use these data is very challenging.

To tackle these challenges, a large number of researches have been made in this field, resulting in a rich papers and methods. They solve this problem from different angels. To the best of our knowledge, there are many researches summarized the related researches, such as [15−17]. However, most of researches only summarized a part of performance issue identification and diagnosis. For example, [15,16] concentrate on the performance issue identification while [17] focuses on the log parsing [18−20]. Little effort has been made to systematically summarize the difference and connection among them.

In this paper, we try to fill this gap by comprehensively reviewing performance issue identification and diagnosis method of SaaS software. Specifically, as shown in Fig. 1, we divided the performance issue identification and diagnosis method of SaaS software into three steps according to their function: performance log generation, performance issue identification and performance issue diagnosis. The detailed information of the three steps is summarized in Table 1.

The performance log generation provide the raw material for the analysis of SaaS software operation by log scraping or system monitoring. Performance issue identification judge if the SaaS software is suffering performance issue by metric-based method, log-based method and behavior-based method. Performance issue diagnosis aim to find out the reason of performance issue.Then, we review different methods of the three steps and their main characteristics by their development history. Meanwhile, we give our proposed solution for each step. Finally, the effectiveness of our proposed methods is shown by experiments.

The rest of this paper is organized as follows. Section 2 gives the preliminaries of this paper. Then, we review performance log generation, performance issue identification and performance issue diagnosis, and give our proposed solutions for each step in Section 3, 4 and 5, respectively. Afterwards, we give the experiments about our proposed methods in Section 6. Finally, we conclude this paper and give the future directions in Section 7.

2 Preliminaries

In this section, we list the notations and some basic definitions used in subsequent sections. Specifically, Table 2 illustrates the notations used in this paper.

Then, some definitions about log are given as follows.

Definition 1 Performance log. It refers to the time series log data used to support the performance issue identification and diagnosis.

Definition 2 Status log. It refers to the log recording the

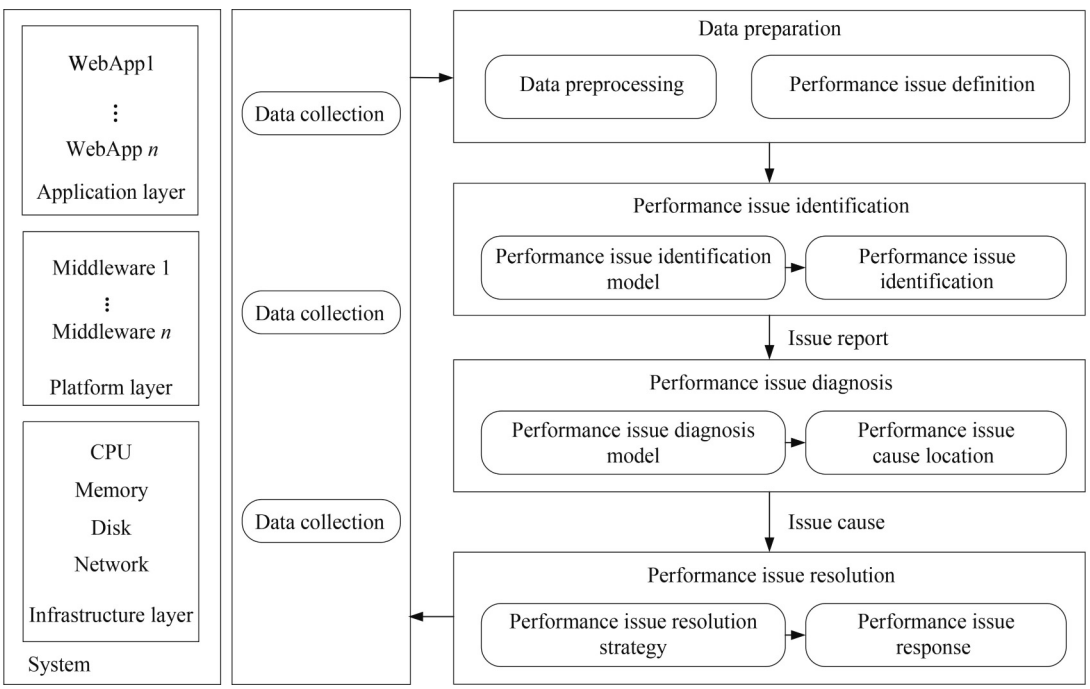

Fig. 1 The basic architecture of performance issue identification and diagnosis of SaaS software

Table 1 The detailed information of the three steps

Step	Techniques	Main functions
Performance log generation	Log scraping and system monitoring	Providing the raw material
Performance issue identification	Metric-based method, log-based method and behavior-based method	Judging performance issue
Performance issue diagnosis	Log-based analysis and process monitoring based approach	Classifying performance issue

Table 2 The notations used in this paper

Notations	Descriptions
S_t	The state of the SLO at time t.
m_i	The ith metric value of $\mathbf{M_t}$.
r	The requirement of user.
Δt	The time interval.
$T_{r\Delta t}$	The response time of the requirement r in time interval Δt.
t_{r_i}	The arrived time of ith response.
t_{s_i}	The response time of ith response.
$ART_{\Delta t}$	The average response time of SaaS software.
s_i	The ith service of SaaS software.
$Slow_{\Delta t}$	The number of slow requirements in time interval Δt.
$Normal_{\Delta t}$	The number of normal requirements in time interval Δt.
$SARatio_{\Delta t}$	The ratio of slow requirements.
Δ_l	The running time of service.
x_t	The metric vector of SaaS software at time t.
X	The observation set of $\mathbf{X_t}$.
χ	The concrete instance of X.
l_t	The system performance state related to X at time t.
ℓ	The system performance state sequence related to X.
τ	The time index set.
$V(\ell)$	The overall potential function of ℓ.
Z, Z_2	The normalizing constant.
N	The neighborhoods.
w_p	The normalizing weight for the total violations of constraint in the neighborhood.
v_i^t	The output of the ith neuron at time t.
$W_{ij}^t(\ell)$	The connection weight related to ℓ between ith and jth neurons at time t.
n	The number of samples.
m	The number of available features.
T	The number of snapshots.
F	The feature matrix.
F^i	The feature matrix of the ith sample.

related performance information of system in running time.

Definition 3 Event log. It refers to the log recording context execution information.

Definition 4 Error log. It refers to the log recording information about exceptions.

The specific information of different type of performance logs is shown in Table 3. Next, we give some definitions about performance issue.

Table 3 The data format of different type of performance logs

Log type	Attributes set	Attribute description
Status log	(LT,SID,ST)	LT: specific time for logging SID: ID of a recorded object ST: moment state of the recorded resource
Event log	(LT,SID,EID,ET,ED)	LT: specific time for logging SID: ID of a recorded object EID: event ID ET: event type ED: description information of the event
Error log	(LT,SID,LEVEL,ET, ED)	LT: specific time for logging SID: ID of a recorded object LEVEL: error level ET: specific type of error ED: error description information

Definition 5 Average Response Time (*ART*). It reflects the expected time of users for the response of software. The specific definition is given as follows:

$$ART_{\Delta t} = E[T_{r\Delta t}] = \sum_{i=1}^{n}(t_{r_i} - t_{s_i})/n, \tag{1}$$

where $T_{r\Delta t}$ represents the response time of the requirement r in time interval Δt; n denotes the number of user response in time interval Δt; t_{r_i} and t_{s_i} represent the arrived time and response time of ith response, respectively.

Definition 6 Slow-to-All-requests-ratio (*SARatio*). It reflects the current state of system, i.e., the response speed for users' requests. The specific definition is denoted as follows:

$$SARatio_{\Delta t} = \frac{|Slow_{\Delta t}|}{|Slow_{\Delta t}| + |Normal_{\Delta t}|}, \tag{2}$$

where $Slow_{\Delta t}$ and $Normal_{\Delta t}$ represents the number of slow requests and normal requests in time interval Δt, respectively. In this paper, we think that the system is suffering performance issue when *SARatio* is bigger than 5%.

3 Performance log generation

Log data is important recording material and analysis basis for software operation [21,22]. The log data is divided into two classes according to their format: text data and metric data. The former is often semi-structured text [23,24], such as tracing and the program log [25], while the latter is often the time series data including the value of a metric and its corresponding collection time. The two types of log data have different collecting method. Specifically, the former usually uses log scraping method to get data while the latter usually utilizes system monitoring method to obtain data. The two methods of obtaining log data are introduced as follows.

3.1 Log scraping

In the operation and maintenance of large-scale software, it is still a problem plaguing developers and managers to determine what to log, where to log and how to log [26–28]. In recent years, there have been some related researches. Figure 2 show the categories of these researches. Some studies paid attention to the macro development of logging. Specifically, Gujral et al. [29] analyzed the logging questions on six popular technical Q&A websites in 2021. This study demonstrate that the logging problem exists various fields, such as database, network and mobile computing. They utilized Latent Dirichlet Allocation (LDA) [30–36] to make semantic analysis, which

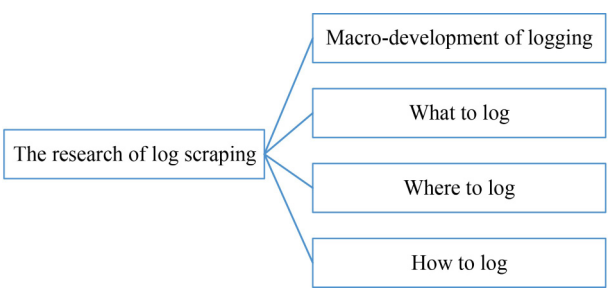

Fig. 2 The categories of log scraping researches

revealed the development trend of the log topics.

Some studies concentrated on what to log. Specifically, Yuan et al. [37] used the case-study method and described the results of log recording on five large-scale software in 2012. By analyzing 250 randomly sampled reported failures, the author found that even half of them could not be diagnosed using existing log data. However, most of these undiagnosed failures can be exposed through a common set of fault modes, such as system call return errors. This means that if relevant information is logged, it can significantly simplify the diagnosis of these failures. Fu et al. [38] systemically studied the real developing log and obtain six valuable findings in 2014, such as the type of recording log code snippet, the influence factors of recording log and the availability of automatically recording log.

Some studies focused on where to log. Zhu et al. [39] proposed a framework of "learning to log" to guide the log recording work in 2015. A logging advice tool was developed, called LogAdvisor, which can automatically learn where to do logging from existing logging instances and provide feasible suggestions for developers. It identifies the important influencing factors of where logging should be done, extracts structural features, textual features, and syntactic features from them, and then utilizes machine learning and noise processing techniques to accurately implement recommendations for logging. Li et al. [40] conducted a comprehensive study to uncover guidelines on code-block-level logging locations in 2020. They combined deep learning and manual investigation to analyze the logging statement and their surrounding code. In the experiments, both syntactic-based and semantic-based deep learning model achieve a better performance. However, on the cross system, syntactic-based model can still keep the good performance while the performance of semantic-based model is poor. The above experimental results demonstrate the existence of the hidden syntactic logging guidelines on the cross system. Gholamian et al. [41] proposed a method to predicting log statement by source code clones and natural language processing (NLP) in 2021. First, it utilizes source code clones to predict the location and code of a log statement. Then, it predicts the description and NLP model of a log statement. Finally, it automatically checks the other detailed information of a log statement. Experimental results show the effectiveness and good performance of log-aware clone detection for automated log location and description prediction.

Other studies paid attention to how to log. Specifically, Cinque et al. [42] argue that event logs have been widely used to analyze the failure behavior of various systems over the past thirty years. However, the implementation of the logging mechanism lacks a systematic approach, and the logs collected when reporting software failures are often inaccurate. This poses a threat to the effectiveness of log-based failure analysis. Therefore, they proposed a rule-based method in 2013 based on above limitation of the recent log mechanism. And experimental results show the effectiveness for analyzing invalidation problem of software. Li et al. [43] proposed to guide log revisions by learning from evolution history in 2019. This method can avoid the evolution of log code brought by bug fixing or feature updating. In addition, they think that the revisions are valuable since the log statement with similar context is common. Based on this idea, they designed and implemented an automatic tool: LogTracker. This tool learns log revision rules by mining the correlation between log context and modifications. Then, it recommends candidate log revisions by these learned rules. Experimental results show its effectiveness. Zhang et al. [44] quantitatively and qualitatively studied the production and test logging statement in 2022. Their study reveals the difference and relationship between the product and test logging statement and the reasons of using test log.

Based on the existing researches, there have been some available log scraping tools, such as the Fluentd tool, the Splunk tool [45] and the framework of Flume [46]. The Fluentd tool uses regular expressions to match the target tag and put the information obtained in the specified location. The Splunk tool assigns a specific token for each log file and utilizes it to search specified file. The framework of Flume is a good method of log scraping. Figure 3 illustrates it. As shown in Fig. 3, there are two channels: file-based channel and memory-based channel. The file-based channel can avoid the loss of data while its efficiency is low. Therefore, this channel usually uses the HBase [47−53] database to store. The efficiency of the memory-base channel is high while it is not sensitive for the loss of data. Therefore, it usually utilizes KafKa [54−57] to store log event and the overall framework of KafKa is shown in Fig. 4.

3.2 System monitoring

System monitoring aims to compensate the loss of the related performance information, which is helpful for the analysis of SaaS [58] software. To the best of our knowledge, some commercial cloud platforms have their own monitoring tools,

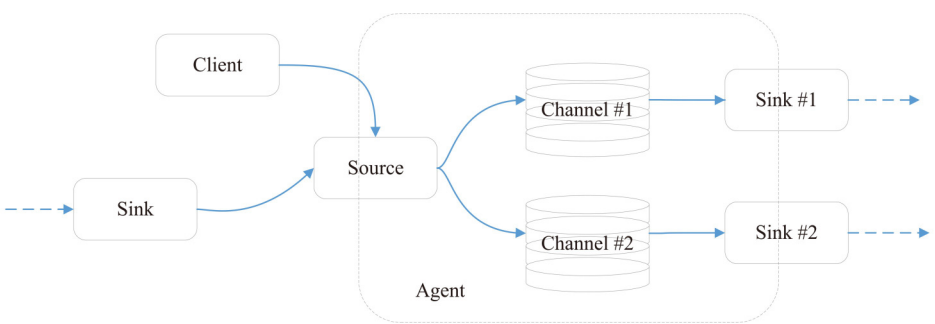

Fig. 3 The overall framework of fluentd

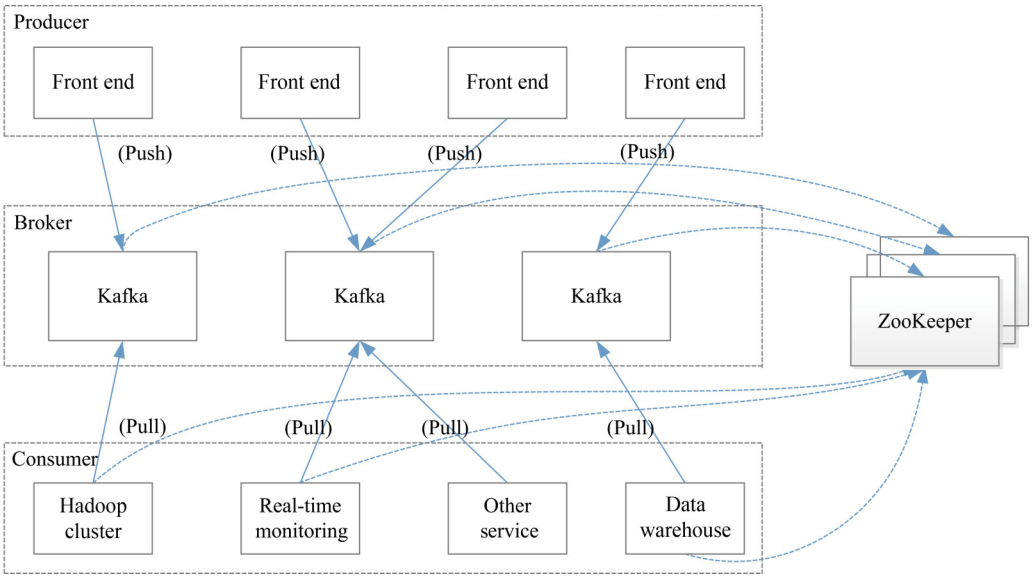

Fig. 4 The overall framework of KafKa

such as the CloudWatch tool of Amazon EC2 [59,60], the Ceilometer tool of Openstack and the AzureWatch [61] tool of Microsoft Azure. However, the above-mentioned monitoring systems of cloud platforms are not usually used to monitor other system since they are usually closely related to their cloud solution. Based on the above circumstance, some third-party tools have been developed to monitor the service compositions from different cloud platforms, such as Nimsoft [62], PCMONS [63], Ganglia [64], Nagios [65] and Zabbix [66].

In addition, some researches about cloud monitoring have been done. They concentrated on the monitoring technologies, platforms and frameworks. Next, we will introduce these researches.

Andreozzi et al. [67] proposed a grid system monitoring service GridICE in 2005, which can provide the fault monitoring reports, service level agreement violations and user-defined event mechanisms. However, it only can monitor the basic physical infrastructure with slow change. To solve this problem, Clayman et al. [25] proposed a monitoring framework Lattice for rapidly changing virtual resource environments. This framework can solve the problem of existing monitoring frameworks such as Ganglia, Nagios, and GridICE that can only monitor slowly changing physical infrastructure.

To solve the real-time monitoring problem of large-scale systems, König et al. [68] proposed a layered monitoring architecture based on distributed monitors in 2012, which has strong scalability. It can finish real time monitoring of large-scale systems with strong scalability to meet analysis needs.

To manage information of multi-tenant cloud platform in a reliable and scalable way, Povedano-Molina et al. [69] proposed a resource management monitoring framework for distributed cloud platform DARGOS in 2013. It can monitor the physical infrastructure resources and virtual resources for multi-tenant cloud platform.

To support performance-enhanced functionalities, Meng et al. [70] proposed a new monitoring model, called Monitoring as a Service (MaaS). This model supports the functional requirements and non-functional requirements. However, the above-mentioned researches neglect the information and control with regards to the customization of the monitoring metrics that the cloud customers have over the rented cloud resources. To solve this problem, Calero et al. [71] proposed a new monitoring architecture by integrating Nagios and OpenStack in 2015, called MonPaaS. It enables cloud providers to see a complete overview of infrastructure, and provides each cloud consumer with a monitoring Platform as a service, so that they can automatically see their leased cloud resources, and define other resources or services to be monitored.

Alhamazani et al. [72] proposed a new method to do efficient QoS monitoring and benchmarking of cloud applications hosted on multi-clouds environments in 2019. The highlight of this method is its capability of monitoring the ability of the single component of the target application. Experiments were conducted on several cloud platforms including Amazon EC2 and Microsoft Azure and experimental results show its effectiveness.

To solve the node failures and performance issues due to the increased work-load, Wang et al. [73] discussed some problems existing in the monitoring system of OpenStack cloud platform and proposed a new scheme for the monitoring of cloud service in 2022. This method allows users and managers to optimize the computing resources according to the changing business requirements.

However, the existing monitoring frameworks work under the control of service providers, which may create dissatisfaction among customers over Service Level Agreement (SLA) violations. Therefore, to solve this problem, Badshah et al. [74] proposed a reliable framework in 2022. This framework monitors the providers' services by adopting the third-party monitoring services with the SLA and penalties management. For violations, it can penalize the people who violate SLA. Simulation results demonstrate that this framework is able to satisfy the requirements of users. Mezni

et al. [75,76] provided an understanding of the lifecycle management phases of the emerging big service model, as well as the role of big data frameworks and multi-cloud strategies in provisioning these services in 2021 and 2023. This provides a reference for big data processing framework.

3.3 Our proposed performance log generation method

In this paper, a multilayer performance log generation framework is designed to record the related log information and the running environment information. Then, the information is processed to support the performance issue identification and diagnosis of SaaS software. Figure 5 gives the framework of performance log generation.

As shown in Fig. 5, the implement of the above framework is divided into four steps: performance log collection, performance log transformation, performance log processing and performance log storage. Specifically, the performance log from the SaaS application WebApp will be collected by some agents of Flume and then flow into the Flume sink node. In this process, two log event stream branches are generated. The first flows directly into HBase for storage; The second first flows into a Broker of Kafka [77] and then enter Spark Streaming for calculation. Finally, the results are written into MySQL [78]. The real-time monitoring for PaaS and IaaS will be implemented by different monitoring technologies and monitoring data is stored by MySQL. MySQL is capable of handling intricate data structures, providing robust support for complex queries, and possessing transactional capabilities. These are critical for analyzing and aggregating log data in the context of SaaS software performance issues, where data relationships and aggregations play a crucial role in troubleshooting and optimizing performance. In addition, HBase is adopted to store text log information including error information. Transferring data from MySQL to HBase allows for unified management of data, enabling advanced data analysis capabilities for global data storage. To ensure higher data real-time and reduce the communication overhead, the push & pull hybrid mode [79] is used as the data transmission strategy that switches between push and pull modes in real time according to the performance state of the system.

4 Performance issue identification

Performance issue identification, also known as performance anomaly detection, aims to judge whether the system is suffering performance issue according to the history observation data. The solutions of performance issue identification are divided into three classes: metric-based method, log-based method and behavior-based method.

4.1 Metric-based method

Metric-based method identifies the performance issue by some manually defined metrics. It is a black box approach without requiring to know the internal structure of the system. Until now, there are many studies about it, which are given as follows.

Lan et al. [80] proposed a method based on outliers to identify the node-level abnormity in the large-scale distributed system. In [80], a group of techniques was first utilized to build a unified data format and automatically analyze the collected data. Meanwhile, the feature extraction technique based on Independent Component Analysis (ICA) is used to reduce data size while the unsurprised techniques are utilized

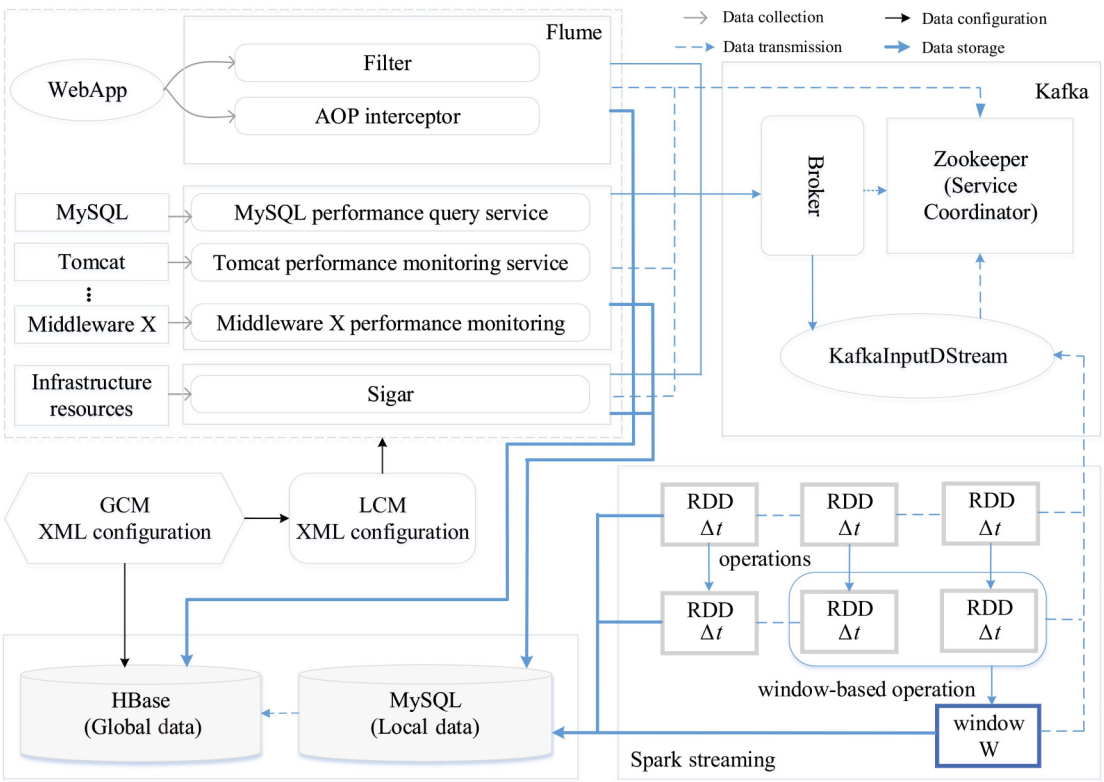

Fig. 5 The framework of performance log generation

to identify the outliers. Experimental results demonstrate that the proposed method has a high accuracy and low computation overhead. After that, they conducted further research in [81]. In [81], they proposed a scalable and nonparametric method to effectively identify performance abnormity of the large-scale system. This method can be used in various distributed parallel systems with peer-to-peer properties. Specifically, it takes a "divide and conquer" approach to solve scalability challenges, while nonparametric clustering and two-stage majority voting are used to improve detection flexibility and accuracy. In addition, they also designed a probabilistic model to quantitatively assess decentralized design options. And experiments were conducted on a suite of applications on production systems and experimental results demonstrate that this method outperforms existing methods in terms of accuracy with a negligible runtime overhead.

Odyurt et al. [82] introduced the concept of software passports based on Extra-Functional Behavior (EFB) metrics in 2019. Among these metrics, they concentrate on the CPU time, read and write communication event counts of different processes. Experimental results show its efficiency.

To applying deep learning in performance issue identification, Wang et al. [83] proposed a deep learning method based on Independent Component Analysis (ICA) and restricted Boltzmann machine (RBM) in 2020. This method adopts ICA to obtain features and RBM to identify the performance issue of SaaS software. Experimental results demonstrate that the proposed method outperform the classical shallow classification algorithms while keeping competitive efficiency.

However, all above-mentioned methods are not applicable for the large-scale, diverse and dynamically changing KPI [84−88] streams.

To solve this problem, Zhang et al. [89] proposed a new method based on PU learning [90−96] in 2021, called PUAD. By this method, the accurate KPI anomaly detection has been achieved. Specifically, it integrates clustering, PU learning and semi-surprised learning to improve the accuracy of anomaly detection. In addition, a new active learning method was proposed, which selects the samples most likely to be positive in each iteration to avoid false alarms. Experiments were conducted on 208 real world KPI streams and experimental results show its superior performance.

4.2 Log-based method

Log-based method identifies the performance issue based on the analysis for logging of SaaS software. It is a grey box approach requiring to know a part of the system execution path.

Some researches concentrate on the log processing and feature extraction, which are introduced as follows.

He et al. [97] proposed an online log parsing method based on the fixed-depth tree [98] in 2017. This method can accurately and efficiently parse the original logging in a streaming manner. In the logging parsing process, log templates can be automatically extracted. Experimental results on five real-world datasets show the superior performance of this method. However, this method not adaptive, which leads them to be unable to handle software/firmware upgrade.

After that, Du et al. [99] a log parsing method based on the longest common subsequence [100,101] in 2018. This method adopts the way of online stream processing to change unstructured system logging into structured data. Experiment results demonstrate that this method outperforms the state-of-the-art methods in terms of efficiency and accuracy. Similarly, it also is not adaptive and cannot handle software upgrade.

To solve this problem, Meng et al. [102] proposed an adaptive log parsing framework to support the intra-service and cross-service incremental template learning and update in 2020. Specifically, this method transforms the template generation problem into the word classification problem and learns the features of template words and variable words. Experimental results conducted on four public datasets demonstrate that the proposed method can support the accurate adaptive template update and new service's log parsing. However, this method cannot finish real-time monitoring and processing for SaaS software.

With the development of various software, more and more tasks require real-time monitoring. To meet this requirement, Vervaet et al. [103] proposed an online logging parsing method based on evolution tree [104−106], called USTEP. Experimental results on 13 real-world datasets show its superior effectiveness and robustness.

With the rapid increase of log data size, efficiency is more and more important for log parsing. Therefore, Dai et al. [107] proposed an automatic log parsing method, called Logram in 2022. It utilizes n-gram dictionary to achieve the efficient log parsing. Experiments were conducted on 16 public log datasets and Logram method was compared with five state-of-the-art methods. Experimental results demonstrate that this method outperforms all comparison methods in terms of efficiency and accuracy.

Other studies focus on the techniques of performance issue identification, which are given as follows.

Fu et al. [108] proposed a performance issue identification method based on non-structured log analysis in 2009. An algorithm was designed to convert free form text messages in log files into log keywords and construct log sequences. By training the log sequence, a finite state automaton FSA model was learned to represent the normal workflow of each component in the system. It has learned a performance measurement model that describes normal execution performance based on timing information in log messages. These models can automatically detect anomalies in newly inputted log files. And its effectiveness is shown by experiments. However, this method uses separate models for different types of data and build an ensemble to generate the final predictions, which neglects the correlations between data sources.

After that, Xu et al. [109] proposed a method of automatically mining console logs based on program analysis, information retrieval, data mining and machine learning. Specifically, it first analyzes the system source code to obtain a log template, which presents a list of variables and message types in the form of <name, type>. Match these templates with

the log messages to convert the log messages from unstructured text to structured data. By selecting variables to group log messages and extracting information from them, such as execution traces, a feature vector is constructed. Log messages of the same group are strongly correlated material, and there is correlation between features. Those deviating from the correlation mode are exception vectors. PCA was used to perform anomaly detection on feature vectors, identifying them as normal and abnormal, and converting the anomaly detection results into a form that is easy to understand. Decision trees are used to visualize them. However, it uses separate models for different types of data and build an ensemble to generate the final predictions. This processing method neglects the correlations between data sources.

To fill the gap, Nedelkoski et al. [110] proposed an unsurprised multimodal performance issue identification method in 2019. This method uses the LSTM neural networks and distributed tracing technique to identify the performance issue of software. Specifically, it utilizes the single mode and sequence text data to simulate the causality between services. Then based on this idea, the multi- modality structure is built by multi-modal tracing data. This method can not only model the normal system behavior, but also identify the performance issue. And experiments conducted on cloud data demonstrate that this method outperforms all baselines. However, there are overfitting problem in this method.

To solve the overfitting problem, Geiger et al. [111] proposed a performance issue identification method based on the unsurprised generative adversarial neural networks in 2020. Specifically, this method introduces a GAN [112−116] structure with the same cycle to map time series to time series. In addition, a new similarity measurement method was proposed to evaluate the similarity between two time series. Experimental results conducted on three famous datasets demonstrate this method outperform all baselines. However, this method is only suitable to the similar data distribution and its robustness is bad.

To increase the robustness of performance issue identification method, Wang et al. [117] proposed a robust log anomaly detection method based on contrastive learning and multi-scale masked sequence to sequence in 2021. Specifically, it combines the BERT [118−121] model with contrastive learning to extract features. Then, it utilizes a multi-scale masked sequence to sequence model to learn the context information from different scales. Finally, experimental results conducted on the public datasets demonstrate that the proposed method outperforms four baselines in terms of accuracy and robustness. However, it is a supervised learning method and needs a large number of labeled data, which is time-consuming and need a large amount of manual labeling.

After that, to reduce the reliance on labeled data, Yang et al. [122] proposed a novel anomaly detection method based on log, called PLELog. It is a semi-supervised method, which can get rid of time-consuming manual labeling by incorporating the knowledge on historical anomalies via probabilistic label estimation. In addition, an attention-based MU neural network was designed to effectively identify anomaly. Experiments conducted on two public datasets show its superior effectiveness and the application in real systems further show its practicability.

Recently, Farzad et al. [123] proposed a novel anomaly detection method, which uses the radius-based fuzzy C-means and a multilayer perceptron network to detect anomaly. The cluster centers and a radius are used to select reliable positive and negative loggings. This model was tested on three famous datasets, i.e., BGL, Openstack and Thunderbird. The experimental results demonstrate that this model outperform the state-of-the-art method.

In 2022, Zhang et al. [124] proposed a deep learning based microservice anomaly detection approach, called DeepTraLog. This method uses a unified graph representation to represent the complex structure of a trace and the related logs. Based on the unified graph representation above, a gated graph neural network model is trained where the loss function is constructed by the deep SVDD model. And experimental results show that the proposed method outperforms the state-of-the-art trace/log anomaly detection approaches. However, it neglects monitoring data of microservice system.

In the same year, a trace representation method [125] was proposed by them, called TraCRL. This method combines the graph neural network with the contrastive learning method to learn the representation of a trace. Specifically, it first constructs an operation invocation graph where nodes and edges represent service operations and operation invocations, respectively. Then, a graph neural network-based model for trace representation is trained based on the operation invocation graph of traces. And experimental results show that the proposed method can significantly improve the performance of trace anomaly detection and offline trace sampling. However, it cannot fully utilize the attribute information of invocation relationships.

4.3 Behavior-based method

Behavior-based method identifies the performance issue based on the analysis for behavior of SaaS software, such as component interactions and execution paths. It is a white box approach, which obtains data to reason the behavior of components by application instrumentation.

Until now, some researches about behavior-based performance issue identification have been made. Aguilera et al. [126] proposed a novel behavior-based performance issue identification method, which infers causal paths between application components and attributes the delay to a specific node.

After that, Chen et al. [127] utilized the probabilistic context free grammar to represent the execution paths. In their method, it is seen as the anomaly if the execution path can be parsed by grammar.

Then, Barham et al. [128] used the clustering to group paths. In this method, it is seen as the anomaly if the execution path does not match the clusters. Afterwards, Chen et al. [129] adopted statistic methods to periodically analyze the interaction between components using the χ^2 test.

Recently, Lim et al. [130] proposed a performance issue

identification method based on hidden markov random field (HMRF) [131,132]. This can identify the recurring and unknown performance issues by transforming the performance issue identification into a clustering problem based on HMRF. Specifically, this method utilizes history data to train the clustering model and the iteration algorithm based on EM to optimize the threshold and the clustering centers. Experimental results show its superior performance.

To unify traced response times and call trajectories in an interpretable way, Liu et al. [133] proposed a new unsupervised performance issue identification method based on service tracing and a fault root cause location algorithm in 2020. This method adds the posterior flows on basis of deep bayesian networks to achieve the nonlinear mapping. The test conducted on TrainTicket show the effectiveness of this method.

Performance metrics only can represent the average load of a system, which cannot help managers find the root-cause of fault. Therefore, Kohyarnejadfard et al. [134] proposed a framework of anomaly detection in 2021. This method first collects system calling flows during process execution by Linux Trace Toolkit Next Generation (LTTng). Then, it utilizes the machine learning to reveal the system calling anomaly. Finally, experiments conducted on real datasets show its effectiveness in different sceneries.

Micro-service is a structure of Web applications. Although it can improve the abstraction, modularity and extensibility of Web applications, it brings some challenges for fault root-cause localization. Therefore, Cai et al. [135] introduced the concept of service dependency graph (SDG) to describe the complex calling relationship among nodes and developed an anomaly detection and root-cause localization framework in 2021, called TraceModel. Experimental results on real-world datasets demonstrate that this model outperforms the state-of-the-art root-cause localization.

The tracing data can not only reflect the running time of the time of calling between services, but also express the calling relationship between services. Therefore, Li et al. [136] proposed a semi-surprised method to detect anomaly. It first uses semi-surprised method based on application tracing and performance monitoring data to obtain anomaly threshold. Then, it uses dynamic sliding window to detect anomaly and the sorting algorithm of tracing to locate the root cause. Finally, experimental results on the public dataset of AIOps Challenge 2020 show the superior performance.

4.4 Our proposed performance issue identification method

In fact, performance issue identification is binary classification problem. It identifies the state of SaaS software into two classes: performance anomaly and performance normality. In this subsection, a performance issue identification method is proposed, which is based on the HMRF [137]. Next, this method will be introduced as follows.

4.4.1 Performance issue identification formalization

In the performance issue identification problem, we try to calculate the current performance state of SaaS software by analyzing the observed performance metric data. Let $S_t = \{0, 1\}$ denote the states of the SLO ({compliance, violation}) at time t. Let $\vec{M}_t = [m_1, \ldots, m_i, \ldots, m_n]$ denote a vector of values for n collected metrics at time t, where m_i is the ith metric. In the complex running environment of SaaS software, the health state of the system is often unobservable before the occurrence of performance issues. But we can infer the current health state of the system through some other system characteristic parameters (low-level metrics, such as CPU, memory). This is consistent with the hidden state of Hidden Markov Random Field (HMRF), so the SLO performance state S of the system can be modeled as hidden state of HMRF. In addition, in order to reasonably infer the hidden state, it is necessary to analyze the changes of observable system characteristic parameters caused by it. This is also consistent with the structure of HMRF so these observations $\vec{M} = [m_1, \ldots, m_i, \ldots, m_n]$ can be modeled as parameters of HMRF. The following are formal definitions of performance issue identification:

Definition 7 X and L are two random fields whose state spaces are $\mathcal{X} = \{1, 2, \ldots, x\}$ and $\mathcal{L} = \{1, 2, \ldots, l\}$, respectively, $\mathcal{T} = \{1, 2, \ldots, T\}$ denotes the time epoch indices, for $\forall t \in \mathcal{T}$ we have $X_t \in \mathcal{X}$ and $L_t \in \mathcal{L}$.

Definition 8 Observation data set $X = (x_1, \ldots, x_i, \ldots, x_n)$ correspond to the set of measurements, random variable x_i correspond to the value of metric m_i at the tth epoch. Let χ be an observable instance of X, $\mathcal{X}$ be the set of all possible instances so that

$$\mathcal{X} = \{\chi = (x_1, \ldots, x_T) | x_t \in \mathcal{X}, t \in \mathcal{T}\},$$

where x_t denotes a vector of values for n collected metrics at time t.

Definition 9 Hidden label set $L = (l_1, \ldots, l_i, \ldots, l_n)$ is the hidden variable, hidden variable l_i denotes a SLO state of system related to m_i at the tth epoch. Let ℓ be a configuration of L, $\mathcal{L}$ be the set of all possible configurations so that

$$\mathcal{L} = \{\ell = (l_1, \ldots, l_T) | l_t \in \mathcal{L}, t \in \mathcal{T}\},$$

where l_t denotes a SLO state of system related to $\vec{M}_t$ at the tth epoch.

Definition 10 Temporal-neighboring constraint, which means that every pair of neighboring records ($\vec{M}$) tend to indicate the same SLO state (S), and are independent of past, non-neighboring records. Each hidden variable l_i is related only to its neighbors within an SLO state type. The hidden variables are mutually related via a neighborhood system $\mathcal{N}$.

The concept of HMRF is derived from Hidden Markov Model (HMM), which are defined as stochastic processes generated by a Markov chain whose state sequence cannot be observed directly, only through a sequence of observations. Each observation is assumed to be a stochastic function of the state sequence. Here, we consider a special case of an HMM in which the underlying stochastic process is a Markov Random Field (MRF) instead of a Markov chain, therefore, not restricted to one dimension. We refer to this special case as a hidden Markov random field model.

Performance issue identification is to determine whether the SaaS software is in a performance error state. The problem can

be formalized as an HMRF problem. SaaS software performance state can be modeled as hidden variables $L = \{l_t, t \in \mathcal{T}\}$, where $\mathcal{T} = \{1, 2, \ldots, T\}$ represents the snapshot time. The hidden variable l_t denotes the performance state associated with performance status log record at the tth snapshot. Assume that there are T performance status instances (performance status log record), including SLO violation instances and SLO compliance instances. Instances that are not adjacent may belong to different classes. The neighboring instances indicating the same performance state form a neighborhood, in which every pair of records is interconnected via **Definition 10**. Our objective is to train a performance issue identification model based on HMRF, which can judge the new performance state. If it is classified as SLO violation of the performance state, it means that SaaS software is in the performance error state, thereby achieving the purpose of identification.

4.4.2 Performance issue identification modeling by HMRF

We select a classifier based on the HMRF-MAP framework. The framework is to calculate the Maximum a Posterior (MAP) $P(L = \ell | X = \chi)$ that hidden state L of the system suffers a performance issue at the current moment t, through a sequence of observations $X = (x_{t,1}, \ldots, x_{t,i}, \ldots, x_{t,n})$. It has important practical advantages: it is interpretable and can combine expert knowledge and constraints.

We use HMRF to construct MAP estimation model, the process is as follows: 1) define a neighborhood system $\mathcal{N}_p$ and the potential $V_{\mathcal{N}_p}(\ell)$ of the neighborhood system; 2) define the prior potential function $V(\ell)$ to give $P(\ell)$; 3) derive the likelihood function $P(\chi|\ell)$; 4) multiply $P(\ell)$ and $P(\chi|\ell)$ to yield the posterior probability $P(L = \ell | X = \chi)$.

L represents the HMRF model of ℓ, according to Hammersley-Clifford theorem, the prior probability of ℓ can be expressed as a Gibbs distribution

$$P(L = \ell) = \frac{1}{Z_1} e^{-V(\ell)} = \frac{1}{Z_2} e^{-\sum_{\mathcal{N}_p \in \mathcal{N}} V_{\mathcal{N}_p}(\ell)}, \tag{3}$$

where Z_2 is a normalizing constant, $V(\ell)$ denotes the overall potential function, which can be expressed as a sum of potentials $V_{\mathcal{N}_p}(\ell)$ over all the neighborhoods $\mathcal{N}$ for the label configuration ℓ. According to **Definition 4**, every pair of records within each neighborhood $\mathcal{N}_p$ tends to be grouped into the same issue cluster. With this, $V_{\mathcal{N}_p}(\ell)$ can be defined as

$$V_{\mathcal{N}_p}(\ell) = w_p \cdot \sum_{\substack{\forall l_t, l'_t \in \mathcal{N}_p \\ t \neq t'}} \mathbb{I}[l_t \neq l'_t], \tag{4}$$

where the variable $\mathbb{I}$ denotes an indicator function ($\mathbb{I}[true] = 1, \mathbb{I}[false] = 0$) and w_p denotes the normalizing weight for the total violations of temporal-neighboring constraint in the neighborhood $\mathcal{N}_p$, gives more weights to neighboring records that are not grouped within the same cluster.

The likelihood function $P_\theta(X = \chi | L = \ell)$ models the conditional dependence and has Gaussian distribution $N(\mu_l, \sigma_l^2)$. Each class can be represented by its mean vector μ_l and variance σ_l^2. According to the characteristic (3) of the HMRF model, the conditional probability of ℓ, χ is

$$P_\theta(X = \chi | L = \ell) = \prod_{t \in \mathcal{T}} P(x_t | l_t) = \prod_{t \in \mathcal{T}} \frac{1}{\sqrt{2\pi\sigma_\ell^2}} exp\left(-\frac{(x-\mu_\ell)^2}{2\sigma_\ell^2}\right). \tag{5}$$

In Eq. (5), variance $\sigma_l^2 \in \{\sigma_v^2, \sigma_c^2\}$ represents the variance corresponding to the performance issue and the normal classes, respectively. Similarly, $\mu_l \in \{\mu_v, \mu_c\}$ represents the mean corresponding to the performance issue and the normal classes, respectively. Here, we consider $\theta = \{\mu_v, \mu_c, \sigma_v^2, \sigma_c^2\}$ as the parameter vector associated with the Probability Density Function (PDF).

ℓ cannot be obtained deterministically form x. Hence, $\hat{\ell}$ should be estimated from x. One way to estimate $\hat{\ell}$ is based on the statistical MAP criterion. The objective in this case is to have an estimation rule which yields $\hat{\ell}$ that maximizes the following posterior probability distribution

$$\hat{\ell} = \underset{\ell}{\operatorname{argmax}} \frac{P_\theta(X = \chi | L = \ell) P(L = \ell)}{P(X = \chi)}. \tag{6}$$

Considering the prior probability Eq. (3) and the likelihood function Eq. (5) of HMRF, we can deduce from Eq. (6)

$$\hat{\ell} = \underset{\ell}{\operatorname{argmin}} \{-\ln\left(\frac{1}{Z}\right) + \sum_{t \in \mathcal{T}} \frac{(x_t - \mu_{l_t})^2}{2\sigma_{l_t}^2} + \sum_{\mathcal{N}_p \in \mathcal{N}} V_{\mathcal{N}_p}(\ell)\}, \tag{7}$$

where $\frac{1}{Z}$ is a constant. By minimizing Eq. (7), we can optimize it and get $\theta = \{\mu_v, \mu_c, \sigma_v^2, \sigma_c^2\}$.

4.4.3 MAP estimation by Hopfield neural network

As the performance issue identification problem is viewed as a MAP estimation problem of the HMRF modeled performance issue with Hopfield Neural Network (HNN), it suffices to establish relation between Eq. (7) and the energy of an HNN, and to provide an updating rule so that the convergence is guaranteed. The clique potential function $V_{\mathcal{N}_p}(\ell)$ in Eq. (7) is considered as

$$V_{\mathcal{N}_p}(\ell) = -W_{sq}^t(\ell) v_s^t v_q^t, \tag{8}$$

where v_s^t and v_q^t represent the output of the sth and qth neurons, respectively and W_{sq}^t is the connection weight between them and depends on ℓ. Value of connection strengths are considered as

$$W_{sq}^t(\ell) = w_p \cdot \sum_{\substack{\forall l_t, l'_t \in \mathcal{N}_p \\ t \neq t'}} \mathbb{I}[l_t \neq l'_t], \tag{9}$$

where w_p is a parameter associated with the clique potential function and is a HMRF model parameter.

Substituting Eq. (9) in Eq. (8) we get

$$V_{\mathcal{N}_p}(\ell) = -w_p \cdot \left(\sum_{\substack{\forall l_t, l'_t \in \mathcal{N}_p \\ t \neq t'}} \mathbb{I}[l_t \neq l'_t]\right) v_s^t v_q^t. \tag{10}$$

Considering Potts model, i.e., a generalization of Ising mode, we can rewrite Eq. (3) as

$$P(L = \ell) = \frac{1}{Z_2} e^{-\sum_{\mathcal{N}_p \in \mathcal{N}} -w_p \cdot \left(\sum_{\substack{\forall l_t, l'_t \in \mathcal{N}_p \\ t \neq t'}} \mathbb{I}[l_t \neq l'_t]\right) v_s^t v_q^t}. \tag{11}$$

For implementing the HMRF model with Hopfield's network, we may interpret x as the initialization of the network (each metric is considered as a neuron). Similarly μ_l can be interpreted as the present state of the network. Thus we may rewrite Eq. (7) as

$$\hat{\ell} = \underset{\ell}{\operatorname{argmin}}\{\sum_s \frac{(I_s - v_s^t)^2}{2\sigma^2} - \sum_{(s,q),s\neq q} W_{sq}^t(\ell) v_s^t v_q^t\}. \quad (12)$$

Now the problem is reduced to the minimization of Eq. (12). We can establish a relation between Eq. (12) and the energy function E of HNN and provide an updating rule so as to reach a minimum of E. In the high gain limit it can be excluded and written as

$$E(v^t) = -\sum_{s=1}^{r}\sum_{q=1,q\neq s}^{r} W_{sq}^t(\ell) v_s^t v_q^t - \sum_{s=1}^{r} I_s v_q^t + \frac{1}{2R}\sum_{s=1}^{r}(v_s^t)^2. \quad (13)$$

By proper adjustment of coefficients, Eq. (12) can be made equivalent to minimize $E(v^t)$.

4.4.4 Model parameters estimation by EM algorithm

In order to obtain the optimal parameters of the HRMF model from a set of metrics data, we design **Algorithm 1** based on the Expectation Maximum (EM). HMRF model parameters are estimated recursively in EM framework. In the HMRF-MAP framework, x is considered as the observed data and ℓ as the unobserved data to be estimated. For estimation of ℓ, the observed data x is modeled with HMRF. The aim of **Algorithm 1** is to estimate $\theta = \{\mu_v, \mu_c, \sigma_v^2, \sigma_c^2\}$ based on the observed data x. The algorithm begins with an initial arbitrary value θ^0 at time 0, and the performance record labels are estimated using the parameter θ^t at time t.

Algorithm 1 EM algorithm

Input: the training set $D = \{(X_1, l_1), \ldots, (X_i, l_i), \ldots, (X_n, l_n)\}$
Output: $\hat{\ell}$, $\theta^t = \{\mu_v, \mu_c, \sigma_v^2, \sigma_c^2\}$

1. Select an arbitrary initial parameter set θ^0;
2. Initialize the class labels using $\text{SARatio}_{\Delta t}$ metric;
3. E-step: Calculate the likelihood distribution $P^t(X = x|L = \ell)$, and the MAP estimate as

$$\hat{\ell}^t = \underset{\ell}{\operatorname{argmax}} P_\theta(L = \ell | X = x)$$

by using HNN;
4. M-step: Compute the posterior probability distribution and update the parameters

$$\mu_l^t = \frac{\sum_{i\in L} P^{t-1}(\ell|x_i) x_i}{\sum_{i\in L} P^{t-1}(\ell|x_i)}$$

$$(\sigma_l^t)^2 = \frac{\sum_{i\in L} P^{t-1}(\ell|x_i)(x_i - \mu_\ell)^2}{\sum_{i\in L} P^{t-1}(\ell|x_i)}$$

5. Repeat steps 3-5 until the stopping criterion is met, i.e., the changes of the parameter values lie within a predefined positive constant ϵ in consecutive iterations.

4.4.5 Performance issue identification algorithm

Figure 6 depicts all the steps required for the implementation of our performance issue identification method. In this section, we will explain every step taken. The first step is to get the performance issue identification model by training the historical data, which is the system state data collected by our performance log generation system, and meets the input requirements of the model after processing; The second step is to continuously monitor the system status and dynamically update the model according to the latest data during the system running to make the model more consistent with the current running status of the system; In the third step, the probability of the current state of the system is calculated and output. If the MAP estimation value obtained by the HNN outputs "1", the current performance log record can be judged as a performance issue. After outputting the results, we update the historical performance log record space.

Algorithm 2 gives a complete description of performance issue identification algorithm through the analysis and design of key parts such as the construction, solution and parameter estimation of performance issue identification model.

5 Performance issue diagnosis

The performance issue diagnosis aims to find the reason of SaaS software performance issue. In general, this problem can be solved from the text mining or analysis of resources and these solutions are divided into two classes based on the above

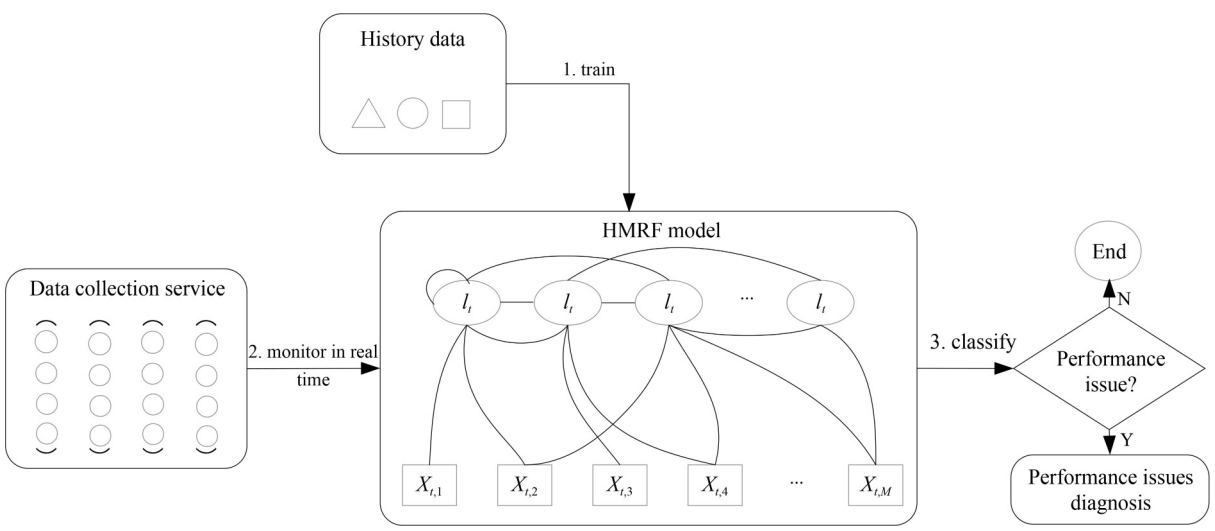

Fig. 6 The steps of our proposed performance issue identification method

Algorithm 2 Performance issue identification algorithm

Input: $X_t = (x_{t,1}, \ldots, x_{t,i}, \ldots, x_{t,n})$
Output: "1" or "0"

1. Model the performance record X using HMRF;
2. Conduct **Algorithm 1**;
3. Obtain I_s, the initial bias using the threshold for the SARatio metric, using 95th percentile

$$I_s = \begin{cases} +1, & \text{if} \quad SARatio > 5\% \\ -1, & \text{if} \quad SARatio \leq 5\% \end{cases}$$

4. x_t are fed as the input to each neuron of HNN;
5. Compute the energy value using Eq. (13);
6. Iterate the HNN until all the neurons are stabilized;
7. Obtain the output of the HNN as ℓ;
8. Update the performance record space.

angels: log-based method and process-monitoring-based method.

5.1 Log-based method

When the system suffers performance issue, the log-based method can be used to solve this problem [138]. This method first obtains the related information from the log data. Then, it adopts the data mining and machine learning to diagnose the performance issue.

At present, many researches about log-based method have been done. Fu et al. [139] proposed designed an event correlation mining and prediction system based on log to predict the invalidation of system in 2012, called LogMaster. The LogMaster parses the log data into the event sequence, where each event is represented as a 9-tuple of information. Specifically, An algorithm was designed to mining event rules, called Apriori-LIS and a system invalidation prediction system based on event correlation graph.

Zou et al. [140] implemented a fault log analysis system, called UiLog, which can help the system managers to know the real-time status of the entire system. Specifically, it first conducts fault log analysis and quantifies the fault classification results of manual learning using a fault keyword matrix to confirm the fault type. Then, it performs fault log correlation, combines the fault log analysis results with the time window, performs fault cause analysis, and then clusters the relevant logs using the same fault cause as the clustering criterion to find the root cause of the fault. Finally, a fault tree is generated based on the cause and type of the fault in order to diagnose the new log.

In 2019, Guo et al. [141] proposed a graph-based trace analysis method to help to understand microservice architecture and diagnose performance problem, named GMTA, which supports various needs such as visualizing service dependencies, making architectural decisions, analyzing the changes of service behaviors, detecting performance issues and locating root causes. This method can efficiently process traces produced on the fly. Specifically, it divides traces into different paths and further groups them into business flows. And the authors used a case study conducted in eBay's monitoring team and Site Reliability Engineering team to demonstrate its substantial benefits in industrial-scale microservice systems. However, there is a plenty of useless logging, which makes it difficult for the mining of event correlation.

To solve this problem, Huo et al. [142] proposed an association rules mining and self-updating method in 2020, called IWApriori. This method contains two steps: log preprocessing and the association rules mining and updating based on IWApriori. It can effectively improve rule integrity. Experiment results conducted on real-world log dataset that this method outperforms other methods in terms of time performance, space performance and the effectiveness of mining rules.

After that, Wang et al. [143] proposed a novel root-cause metric localization method by incorporating log anomaly detection. It is believed that the root-cause metric value should change with the anomaly score of the system fault in [143]. Specifically, this method collects anomaly scores by log anomaly detection algorithms and identify root-cause metric by the robust correlation analysis. Experimental results on public micro-service systems show its superior performance.

Considering that the use of the tracing data can make the localization of root-cause more accurate, Li et al. [136] proposed a new anomaly detection algorithm in 2021. It uses semi-supervised learning to obtain the anomaly threshold and the sorting algorithm based on tracing to locate the root-cause. They tested this method on public datasets and the superior performance is shown by experiments.

Similarly, Liu et al. [144] proposed a high-efficient root cause localization approach for availability issues of microservice systems in 2021, called MicroHECL. Based on a dynamically constructed service call graph, the MicroHECL analyzes possible anomaly propagation chains and ranks candidate root caused by correlation analysis. Specifically, combining machine learning with statistical methods, the authors designed a customized models for the detection of different types of service anomalies. Meanwhile, this method uses a pruning strategy to eliminate irrelevant service calls in anomaly propagation chain analysis. And experimental results show its superior performance. However, the above approaches rely on empirical techniques.

To solve this problem, Gan et al. [145] proposed a novel machine learning-driven root cause analysis method in 2021, which focuses on practicality and scalability. Specifically, this method uses unsupervised machine learning method to reduce the cost of trace labeling and determine the root cause of unpredictable performance. The experiments were conducted on both local clusters and large clusters on Google Compute Engine and the experimental results show the superior performance of the proposed method. Similarly, Ma et al. [146] proposed a novel framework for anomaly detection and root cause identification in the microservice system in 2022. This method first uses the history data to train an anomaly detector without pre-defined thresholds. Then, it designs a causal relationship extraction approach to construct impact graphs. Finally, a heuristic investigation algorithm based on random walk is designed to identify the root cause of the anomaly.

To improve the efficiency of the identification of root cause, Li et al. [147]. proposed a root cause analysis method based

on intervention recognition. This method first constructs a causal relationship graph based on the knowledge of microservice system architecture and some causal assumptions. Then, the linear regression method is used to infer the root cause. Finally, the experiments were conducted on a real-world dataset and the experimental results demonstrate that the proposed method outperform all comparison methods.

5.2 Process-monitoring-based method

Process-monitoring-based method diagnoses the performance issue by online monitoring and analysis. It always keeps an eye on its running circumstance of the target system. Until now, many related researches have been made. The difference of these works is very large since their aims, analysis methods and information resource are different.

Zhao et al. [148] designed an analyzer for non-intrusive distributed service request flow. It first infers how to parse logs by the static analysis for the binary code of applications. Then, it integrates different logging and associates them with a specific requirement. Note that, it infers the requirement flow by running log data without the modification of source code. LProf is a profiling tool that helps diagnose actual performance anomalies. Experiments show its effectiveness for the performance issue diagnosis.

Bare et al. [149] proposed an online diagnosis framework in 2009, called ASDF. This framework can monitor and analyze the data resource with time and locate the performance issue into a specific node or a node set. Its flexible structure allows the system managers to customize the data sources and analysis modules. And the authors show the effectiveness of ASDF on performance issue according to the file.

Attariyan et al. [150] proposed a performance aggregation technology of automatic performance issue diagnosis in 2012. It first takes the performance cost as the attribute of each basic block. And then, dynamic information flow tracing is used to evaluate the possibility of each hidden root-cause being executed. Finally, the cost of each hidden root-cause is collected. Based on the idea, author developed a tool, called X-ray to help users solve the performance issue of software without the support of developers.

To help performance analyst to effectively compare load test results, Malik et al. [151] proposed a supervised method and three unsupervised methods in 2013. These methods can effectively identify the performance issue violating SLA. Meanwhile, it can utilize a collection of smaller manageable important performance counters to analyze the root-cause of performance issue.

With the size and complexity of high-performance computing systems, the influence of performance change is becoming more and more important. To reduce the influence of performance change as more as possible, Tuncer et al. [152] proposed a new framework based on machine learning to automatically diagnose the performance issue. This framework utilizes history data to obtain the anomaly features. Specifically, it first transforms the collected time series data into statical features. Then, the available anomaly features are obtained. Afterwards, these features are used to diagnose anomaly. Finally, experimental results on real-world HPC systems demonstrate that this framework is better than the existing anomaly diagnosis techniques.

In the cloud platforms, although performance monitoring metrics can entirely describe the current status of service, empirical alarm thresholds alone often fail to identify root cause in real time. Therefore, Li et al. [153] proposed a common anomaly detection algorithm. This algorithm uses the offline and online learning methods to dynamically modify the feature matrix and the anomaly threshold of metric. Then, deviation degree, scoring and sorting algorithm are used to identify the root cause of fault. This algorithm was tested on the dataset of AIOps challenge (2021) and the experimental results show its good performance.

Although there are data monitoring infrastructures, the lack of labels describing the state of the system is a pervasive problem. To fill this gap, Borghesi et al. [154] proposed a method obtaining labels from service monitoring tools. Specifically, this method is used to obtain labels. Then, these labels are used to train a deep learning model. Finally, experimental results demonstrate that this method can accurately diagnose the real faults.

5.3 Our proposed performance issue diagnosis method

The problem to be solved in this part is to diagnose the performance issues during the SaaS software running and to help the operation and maintenance manager to analyze the performance issues more finely and find out the causes of the performance issues. In this section, an automatic performance issue diagnosis method is proposed. This method further diagnoses the identified performance issues and constructs RBM classification model. This model classifies the current performance issues of the system and finds the causes of the issues.

Based on the above reasons, a performance issue diagnosis method based on RBM is proposed. First, we build a performance issue diagnosis model and train its parameters to get a maximum likelihood estimation (MLE) that can correctly distinguish the type of performance issue; then, we use the model to calculate a new data source to achieve the purpose of diagnosing the running state of the system; finally, after the operation and maintenance managers get the type of the running state of the system, they take corresponding effective measures.

5.3.1 Performance issue diagnosis formalization

Performance issue diagnosis can be regarded as a multi-classification problem in supervised learning. Suppose that there exist T instances of performance issue and each of which belongs to one of the l issue types ($l \leqslant T$). The T issue instances (i.e., violation instances) and the non-issue instances (i.e., compliance instances) can be grouped into $l+1$ clusters (l issue types plus 1 non-issue type). Each cluster consists of performance issues of the same type. The SLO compliance records can be treated as a special non-issue type. The problem is to automatically group the record space into $l+1$ clusters, and assign a new record to a correct cluster. If the new record can be well classified into a cluster, then the cause of the performance issue is discovered.

Restricted Boltzmann Machine is a bipartite graph model, which means that it contains two layers, the visible layer and the hidden layer. The visible layer is composed of visible units (corresponding to visible variables) $v = [v_1, v_2, \ldots, v_D]$ representing the observable data, the hidden layer is composed of some hidden units (corresponding to hidden variables) $h = [h_1, h_2, \ldots, h_N]$. And each unit of hidden layer is connected with all the units of visible layer and vice versa, while there is no connection within each layer. Both v and h are binary vectors.

In performance issue diagnosis problem, we try to infer the performance issues by analyzing the observed metric data. This problem can be formulated as a RBM problem. The performance issues can be modeled as hidden variables $L = \{l_t, t \in \mathcal{T}\}$, where $\mathcal{T} = \{1, 2, \ldots, T\}$ denotes the snapshot times. The hidden variable l_t represents the performance issue associated with record at the tth snapshot. Our objective is to train a RBM-based performance issue diagnosis model to categorize the performance issues in the entire performance status log record space into several issue types (including the non-issue type). Based on this model, new performance status records can be classified. If they are classified into a certain type of issue, it indicates that the cause of the performance issue is obtained, thereby achieving the purpose of diagnosis.

5.3.2 Performance issue diagnosis based on RBM

Similarly, performance issue diagnosis is also a pattern classification problem. It is to induce or learn a classifier function $\mathcal{F}'$ mapping the current system state to a known system performance state class. The training dataset of the classifier is derived from the multi-category labeled observation log, providing preclassified instances for the analysis to learn from. Since supervised learning is superior to unsupervised learning, we still choose supervised learning to train the model.

5.3.2.1 Feature matrix generation

We define any individually measurable variable of the node being observed as a feature, with the goal of constructing a feature matrix for data analysis. Let m be the number of features collected from n nodes. In order to capture the trend and the correlation of these features, each node samples T snapshots. Therefore, there are n matrices $F^i (i = 1, 2, \ldots, n)$, each representing the feature matrix collected from the ith node. In each matrix F^i, the element $f^i_{h,j}$ represents the value of the feature h collected at the jth snapshot, where $1 \leqslant j \leqslant T$ and $1 \leqslant h \leqslant m$. In order to facilitate the data analysis, we reorganize each matrix F^i into a $(m \times T)$ vector $f^i = [f^i_{1,1} f^i_{1,2} \cdots f^i_{m,k}]^T$. Finally, we construct a single large-scale matrix as a feature matrix

$$F_{(m \times T) \times n} = [f^1, f^2, \ldots, f^n]. \tag{14}$$

We think all the features are equally important. However, the data collected may have different scales. In order to convert the data into uniform scale, the matrix F is first normalized to F' so that the feature values can be controlled between 0.0 and 1.0. Then it is necessary to adjust F' to F'' so that its columns have zero mean. This ensures that the following feature extraction capture the true variance, thus avoiding distorting results due to the mean difference. For the data matrix F'', each column represents a node and each row gives the values of a particular feature.

5.3.2.2 ICA-based feature extraction

We analyze the overall performance of the SaaS software by measuring the correlation between performance metrics, which requires extensive calculations. We try to learn features by unsupervised learning, both Principal Component Analysis(PCA) and ICA are important choice for feature learning. In this paper, we also prove this point. We apply ICA to the matrix F'', and treat each column of F'' as a data point of $IR^{m \times T}$. First calculate the covariance matrix of F'':

$$cov = \frac{1}{n} F'' F''^T. \tag{15}$$

then calculate nonzero Eigenvalues of cov in a descent order: $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_r$. Let $V = diag(\lambda_1, \lambda_2, \ldots, \lambda_r)$ and $E = [e_1, e_2, \ldots, e_r]$, where e_i is the Eigenvector corresponding to λ_i. The whitened data of F'' are defined as

$$X = V^{-1/2} E^T F'', \tag{16}$$

where X is a $r \times n$ matrix, and $r \leqslant m \times T$. After whitening, ICA projects the data point $x_i \in IR^r$ to a data point $y_i \in IR^s$, as follows:

$$y_i = M^T x_i. \tag{17}$$

The dimensionality of y_i is set to s. The goal is to find an optimal projection matrix M such that y_i are maximally independent. The process of finding M is the feature extraction process, as shown in **Algorithm 3**.

5.3.2.3 RBM-based performance issues classification

After recognizing the performance issue features, a Boltzmann machine classifier for the features is established, and the classifier is trained. The training process is mainly to use the sample set of performance issues to train the RBM with visible units with Gibbs distribution function.

RBM is an Energy-based model, the energy of the joint configuration of visible variable **v** and hidden variable **h** is

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\mathbf{h}^T W \mathbf{v} - b^T \mathbf{v} - a^T \mathbf{h}, \tag{18}$$

Algorithm 3 ICA-based feature extraction

Input: feature matrix F''
Output: new feature matrix Y

1. Calculate the covariance matrix of F'' according to Eq. (15).
2. Whiten the feature matrix F'' according to Eq. (16).
3. Project (iteratively searching for M):
   1). Choose a random initial matrix $M = [m_1, m_2, \ldots, m_s]$, where $\|m_i\| = 1$.
   2). For $i = 1$ to s
   $$m_i^+ = \frac{1}{n} \sum_{i=1}^n x_i (m_j^T x_i)^3 - 3 m_j$$
   $$m_i = m_i^+ / \|m_i^+\|$$
   Here, the first calculation is to ensure that the iteration approaches the maximal independence and the second one ensures that $\|m_i\| = 1$.
   3). $M = M(M^T M)^{-1/2}$, which is to ensure that $m_i \neq m_j$ when $i \neq j$.
   4). If $(1 - |m_i^T m_i|) < \delta$, where δ is a small constant, the algorithm converges;
   Otherwise, go to step 2).
4. Get the new feature matrix $Y = \{y_1, y_2, \ldots, y_i, \ldots, y_n\}$, row vectors are independent of each other.

where the model parameter $\theta = \{W, b, a\}$, W is the weight of the connection between the visible unit and the hidden unit, b and a are the bias of the visible unit and the hidden unit, respectively. Boltzmann distribution is the distribution of this energy configuration.

Because of the special structure of RBM (without connection within the layer and with connection between the layers), the activation state h_j of each hidden unit is independent when $\mathbf{v}$ is given. On the contrary, given $\mathbf{h}$, the activation state v_i of each visible unit is also independent of each other, that is

$$P(h_j = 1|\mathbf{v}) = \sigma\left(\sum_i W_{ij} v_i + a_j\right), \tag{19}$$

$$P(v_i = 1|\mathbf{h}) = \sigma\left(\sum_j W_{ij} h_j + b_i\right), \tag{20}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the logistic sigmoid function.

An RBM can be used to model the distribution of the observed data $P_\theta(\mathbf{v})$ by learning from the training data $\mathbf{v}_t$. The training is done in an unsupervised manner. In fact, RBM is a Markov random field problem, and it belongs to a structured prediction, involving an exponential combination of outputs. So the calculation is very large, it is difficult to solve with gradient descent method. In order to solve the above problem, G. E. Hinton et al. proposed an efficient learning algorithm, Contrastive Divergence (CD).

One way was first proposed by G. E. Hinton et al. and explicitly discussed by Hugo Larochelle was to use an RBM directly as a classifier called the Classification Restricted Boltzmann Machine (ClassRBM).

ClassRBM has the similar structure with the ordinary RBM. It models the joint distribution of an input $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ and target class $l \in \{0, 1, 2, \ldots, C\}$ using a hidden layer of binary stochastic units $\mathbf{h} = [h_1, h_2, \ldots, h_N]$. This is done by first defining an energy function

$$E(\mathbf{y}, \mathbf{h}, l; \Theta) = -\mathbf{h}^T W \mathbf{v} - b^T \mathbf{v} - a^T \mathbf{h} - d^T \mathbf{e}_l - \mathbf{h}^T U \mathbf{e}_l, \tag{21}$$

with parameters $\Theta = \{W, b, a, d, U\}$, U is the weight of the connection between the target class and the hidden unit. $\mathbf{e}_l = (1_{i=l})_{i=0}^C$ is the "one hit of C" representation of l. From the energy function, we assign probabilities to values of l, $\mathbf{y}$ and $\mathbf{h}$ as follows:

$$p(\mathbf{y}, \mathbf{h}, l) = \frac{exp(-E(\mathbf{y}, \mathbf{h}, l))}{Z}, \tag{22}$$

where Z is a normalization constant (also called partition function) which ensures that Eq. (22) is a valid probability distribution. (y, l) is the visual layer. The $\mathbf{y}$ part is used for input of sample feature information, and the l part is used for input of class label information, which is represented by binary vector. When the sample belongs to the k class, the k-th node in the l part takes the value 1, and the other nodes take the value 0, i.e., $l(k) = 1, l(i) = 0, i \neq k$.

Through the energy function of the ClassRBM model, the posterior activation probability of visible units in the model can be deduced, as shown in Eq. (23).

$$p(\mathbf{h}|l, \mathbf{y}) = \prod_j p(h_j|l, \mathbf{y}), \tag{23}$$

with $p(h_j = 1|l, \mathbf{y}) = \sigma(a_j + U_{jl} + \sum_i W_{ij} y_i)$.

Similarly, the posterior activation probability of hidden units in this model, as shown in Eq. (24):

$$p(\mathbf{y}|\mathbf{h}) = \prod_i p(y_i|\mathbf{h}), \tag{24}$$

with $p(y_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j W_{ij} h_j)$.

ClassRBM can be trained in two ways. One is called the generative training objective which maximizes the joint probability over training data $\mathbf{y}_t$ and its label l_t. The other is called the discriminative training objective which maximizes the probability of the label l_t of training data conditioned on the training data $\mathbf{y}_t$. The discriminative training objective is more suitable for relatively smaller data, the following training objective is used:

$$f(\Theta, D_{train}) = -\frac{1}{|D_{train}|} \sum_{t=1}^{|D_{train}|} \ln p(l_t|\mathbf{y}_t), \tag{25}$$

where $D_{train} = \{(\mathbf{y}_t, l_t)\}$ is the set of training examples.

Minimizing the objective function Eq. (25) is equivalent to maximizing $p(l_1, \ldots, l_{|D_{train}|}|\mathbf{y}_1, \ldots, \mathbf{y}_{|D_{train}|})$, as we assume that the training examples are independent from each other. It is worthy to notice that, $\mathbf{y}_t$ does not have to be binary in this case, as the distribution of $\mathbf{y}_t$ is not modeled.

Calculate the gradient of $f(\Theta, D_{train})$ over parameters Θ:

$$\frac{\partial}{\partial\theta} f(\Theta, D_{train}) = \frac{1}{|D_{train}|} \sum_{t=1}^{|D_{train}|} (\mathbb{E}_{p(\mathbf{h}|l_t, \mathbf{y}_t)}[\frac{\partial}{\partial\theta} E(l_t, \mathbf{y}_t, \mathbf{h})] - \mathbb{E}_{p(l, \mathbf{h}|\mathbf{y})}[\frac{\partial}{\partial\theta} E(l, \mathbf{y}, \mathbf{h})]), \tag{26}$$

where $\mathbb{E}_{p(y)}[f(y)]$ represents the expectation of $f(y)$ over distribution $p(y)$. The pseudo code of model parameter update procedure is given by Algorithm 4.

Given the training set $D_{train} = \{(y_1, l_1), (y_2, l_2), \ldots, (y_i, l_i), \ldots, (y_n, l_n)\}$ and the label set $L = \{0, 1, 2, \ldots, C\}, l \in L$. Training ClassRBM model based on historical data. Our goal is to further classify the current issued state of system identified by the performance issue identification model and obtain specific label information of the state data. After the model is trained, a new identified sample data $\mathbf{y}_{new}$ is assigned to the class l^* according to the following decision rule:

Algorithm 4 Discriminative training update of the ClassRBM using CD

Input: training pair $(\mathbf{y}_t, l_t)$ and learning rate η

Positive phase
$l^0 \leftarrow l_t, \mathbf{y}^0 \leftarrow \mathbf{y}_t$
$\hat{\mathbf{h}}^0 \leftarrow \sigma(a + W\mathbf{y}^0 + U\mathbf{e}_{l^0})$
Negative phase
$\mathbf{h}^0 \sim p(\mathbf{h}|l^0, \mathbf{y}^0), l^1 \sim p(l|\mathbf{h}^0), \mathbf{y}^1 \sim p(\mathbf{y}|\mathbf{h}^0)$
$\hat{\mathbf{h}}^1 \leftarrow \sigma(a + W\mathbf{y}^1 + U\mathbf{e}_{l^1})$
# Update
for $\theta \in \Theta$ **do**
    $\theta \leftarrow \theta - \eta[\frac{\partial}{\partial\theta} E(l^0, \hat{\mathbf{h}}^0|\mathbf{y}_t) - \frac{\partial}{\partial\theta} E(l^1, \hat{\mathbf{h}}^1|\mathbf{y}_t)]$

Notation: $x \leftarrow y$ means x is set to value y
Notation: $x \sim p$ means x is a sample from distribution p

$$l^* = \arg \max_{l \in \{1,2,\ldots,C\}} p(l|\mathbf{y}_{new}). \quad (27)$$

Obviously, the data vector $\mathbf{y}_{new}$ is classified into the class where $p(l|\mathbf{y}_{new})$ is the largest.

5.3.2.4 Performance issues diagnosis algorithm

The above supervised learning is applied to analyze performance issues automatically. In this paper, the performance issue diagnosis problem is regarded as a multi-classification problem. The multi-classification method for performance issue is divided into three stages: feature matrix generation, feature extraction and classification. All the steps required for the performance issue diagnosis method are shown in Fig. 7. Through data transformation, the generated performance status log data set is integrated into a feature matrix (usually high-dimensional). Here, a feature is defined as any individually measurable variable of the node being observed, such as CPU utilization, available memory size, I/O, network traffic, etc. Through feature extraction, the feature matrix is reduced in dimension while retaining the most relevant information in the data. This can not only speed up the training of the performance issue diagnosis model by reducing the data dimensionality, but also improve the diagnosis ability of the model by eliminating the inherent data dependency. The newly arrived system performance state (identified performance issue) is input into the trained RBM-based performance issue diagnosis model, and the likelihood estimate of each performance issue type to which the performance issue belongs is calculated, and which type l has the largest likelihood estimate, the performance issue will be judged to this type l^*. The above diagnosis process is to subdivide the identified performance issues into specific classes to find the causes of the performance issues.

Algorithm 5 gives a complete description of the performance issue diagnosis algorithm by analyzing and designing the key parts of the performance issue diagnosis model, such as data input, training, solution and parameter estimation.

Algorithm 5 Performance issue diagnosis algorithm

Input: training set $D_{train} = \{(y_1, l_1), (y_2, l_2), \ldots, (y_i, l_i), \ldots, (y_n, l_n)\}$, new identified performance state $\mathbf{y}_{new}$;
Output: the trained ClassRBM model, l^*.

1. Initialization: learning rate η, number of Gibbs steps to do in CD, number of hidden units;
2. Construct feature matrix using Eq. (14);
3. Feature extraction using **Algorithm 3**;
4. Training the ClassRBM model using **Algorithm 4**;
5. Classification:
   $\mathbf{y}_{new}$ are fed as the input to the trained ClassRBM model;
   Classify the performance issue based on the trained ClassRBM model using Eq. (12).

6 Experiment design and results analysis

To verify the effect of our proposed method, the experiment was conducted on PC with Windows 10, 1 CPU and 8 GB memory. The experimental dataset was collected from a real Integrated Disaster Reduction Application System (IDRAS) [138]. This system relies on cloud service platform and the SOA idea, which contains many web components with independent functions. This system is developed by Java and deployed on Cloud Stack. The proposed method is implemented by python. 20 metrics are collected from the PaaS and IaaS layers to represent the observed state of the system, as shown in Table 4. We collected 2862 performance log records (i.e., 2862 snapshots), including 491 performance issues, and these 491 problem instances, plus non-problem instances, total 19+1 performance problem types. Experiments were conducted by the 5-fold cross-validation to avoid overfitting.

6.1 Evaluation metrics

We choose the most common classification metrics, recall, precision and $F1$, as an indicator to test the effectiveness of the proposed performance issue identification method (bi-classification method) and the proposed performance issue diagnosis method (multi-classification method). For each issue

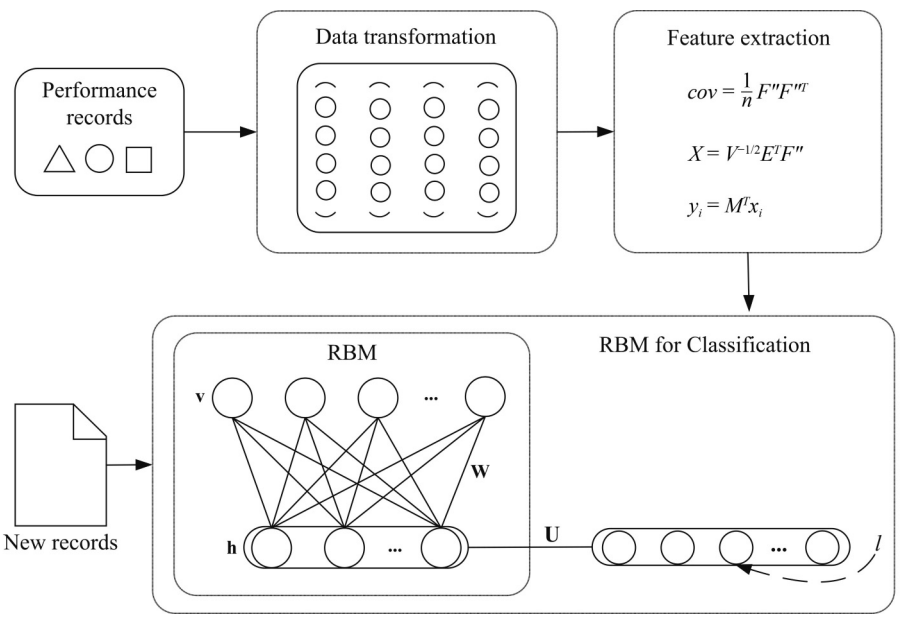

Fig. 7 The steps of our proposed performance issue diagnosis method

Table 4 Performance metrics

Layers	Objects	Metrics	Descriptions
SaaS	Service	Response Time	Response Time of service (ms)
PaaS	MySQL	Threads_connected	No. of MySQL server connected threads (counts)
		Threads_running	No. of MySQL server running threads (counts)
	Tomcat	JVM_Free	Size of free JVM (MB)
		Tomcat_requestCount	Total No. of Tomcat requests (counts)
		Tomcat_Thread	No. of Tomcat threads (counts)
IaaS	CPU	CPU_Cores	No. of CPU Cores (counts)
		CPU_utilization	Percentage of CPU utilization (%)
	Memory	Mem_totalSize	Size of Memory (GB)
		Mem_usagePercent	Percentage of Memory utilization (%)
	Disk	Disk_avail	Size of free disk (GB)
		IO_Read	Rate of read operation (Kbps)
		IO_Write	Rate of write operation (Kbps)
	Network	NetCard_Receive	Rate of Network card receive bytes (Kbps)
		NetCard_Send	Rate of Network card send bytes (Kbps)
Physical resources	CPU	CPU_Allocated	Allocated CPU (GHz)
		utilization	Percentage of CPU utilization (%)
	Memory	Mem_Allocated	Allocated Memory (GB)
		utilization	Percentage of Memory utilization (%)
	Primary storage	Storage_Allocated	Allocated Primary Storage (TB)
		utilization	Percentage of storage utilization (%)

type, recall measures the fraction of the relevant records whose issue types are successfully classified. Precision measures the fraction of classified records that are indeed relevant. $F1$ is the balance of precision and recall metrics. Classification results are expressed in a confusion matrix [155], as shown in Table 5.

More specifically, recall is the proportion of performance issues correctly classified to the number of actual issue records:

$$Recall = \frac{TP}{TP + FN}. \tag{28}$$

Precision is the proportion of performance issues correctly classified to the number of records classified:

$$Precision = \frac{TP}{TP + FP}, \tag{29}$$

where TP (True Positive) is the number of correctly classified records (i.e., the testing records that are assigned to the correct issue type). FP (False Positive) is the number of classified records that are irrelevant (i.e., the irrelevant testing records that are assigned to the target issue type). FN (False Negative) is the number of relevant records that are failed to be classified (i.e., the relevant testing records that are not assigned to the target issue type).

Recall and *Precision* are a contradictory measure, so it is not appropriate to use only one of them to evaluate the performance of the identification model. $F1$ can combine them to evaluate models, it is widely used in performance evaluation of prediction model and other software engineering research fields [156]. The calculation formula of $F1$ is the harmonic mean of recall and precision:

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision}. \tag{30}$$

Table 5 Evaluation metrics

		Actual record	
		Relevant	NonRelevant
Classified result	NonClassified classified	True positives (TP)	Fault positives (FP)
		Fault negatives (FN)	True negatives (TN)

6.2 The experiments of our proposed performance issue identification method

Our proposed performance issue identification method was evaluated from system cost and effectiveness by experiments, which is introduced as follows.

6.2.1 System cost experiments

This section mainly introduces the system cost experiments. The proposed performance issue identification method needs some system cost since it needs to monitor the system state in real-time. To verify the system cost of our proposed method, the experiment was conducted when the number of concurrent requests increases from 1 to 100. Figure 8 gives the influence of performance issue identification for service performance.

As shown in Fig. 8, the response time of service gradually increases with the number of concurrent requests. When the number of concurrent requests is larger than 100, the response time of service rapidly increases. This phenomenon illustrates that the system cannot process these concurrent requests. The service response time increase of performance issue identification is about 5–15 ms. These experimental results demonstrate that the system cost of our proposed method is very small.

6.2.2 Effectiveness experiments

This section mainly introduces the effectiveness experiments. These experiments were used to verify the effectiveness of our proposed method. To verify the effectiveness of the HMRF-MAP, we compared it with 7 comparison models, such as *Support Vector Machine* (SVM) [157−159], *Naive Bayes*

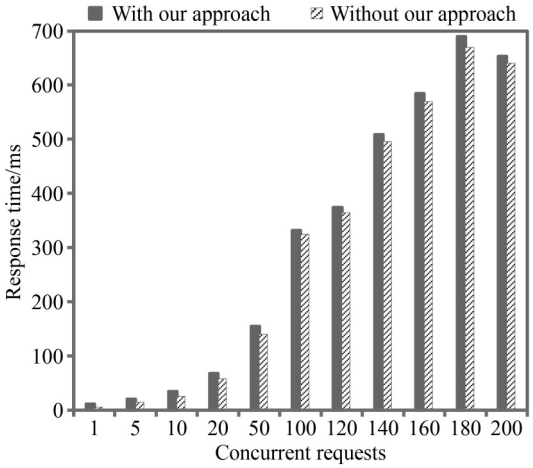

Fig. 8 The influence of performance issue identification for service performance

Classifier (NBC) [160], *KNeighbors Classifier* (KNC) [161], *Nearest Centroid Classifier* (NCC) [162], *Logistic Regression* (LR) [163]. Table 6 gives the experimental results.

As shown in Table 6, the HMRF-MAP outperforms all comparison models in terms of $F1$. Specifically, it is 0.91% better the ranking second model (SVM) while it is 32.05% better than the worst model (MNB). This further show the effectiveness of HMRF-MAP.

To verify the effectiveness of our proposed method for response time, we compared the response time of using performance issue method and manual identification when the system is suffering performance issue. Figure 9 gives the response time of using performance issue method and manual identification.

As shown in Fig. 9, the average response time rapidly increases when the system is suffering performance issue. From the figure we can draw the conclusion: the proposed performance issue identification method can obviously reduce the response time of system compared with manual identification when the system is suffering performance issue. The experimental results demonstrate that our proposed performance issue identification method can effectively reduce response time in the period of system performance issue and restore the service capability of the system in time.

6.3 The experiments of our proposed performance issue diagnosis method

Our proposed performance issue diagnosis method was evaluated from system cost and effectiveness by experiments, which is introduced as follows.

6.3.1 System cost experiments

This section mainly introduces the system cost experiments where $\eta = 0.06$, $iterations = 30$ and the number of hidden units is 100. The proposed performance issue diagnosis method needs some system cost since it needs to diagnose the performance issue. To be able to quickly find the reasons of performance issue, the system cost of our proposed method should be as small as possible. To verify the system cost of our proposed method, the experiment was conducted when the number of concurrent requests increases from 1 to 200. Figure 10 gives the influence of our proposed performance issue diagnosis method for service performance.

As shown in Fig. 10, the response time increases with the number of load concurrent requests. Specifically, the response time increase of service due to performance issue diagnosis method is about 6–37 ms. This demonstrate that the system cost of our proposed method is very small, which is totally acceptable.

Table 6 $F1$ of the HMRF-MAP and comparison models

Dataset	NB			SVM	KNC	NCC	LR	HMRF
	GaussianNBC	MultinomialNBC	BernoulliNBC					
IDRAS1	0.86	0.69	0.77	**0.90**	0.87	0.88	0.89	0.89
IDRAS2	0.85	0.65	0.78	0.87	0.85	0.84	0.85	**0.90**
IDRAS3	0.86	0.66	0.78	0.89	0.84	0.85	0.86	**0.91**
IDRAS4	0.84	0.70	0.79	**0.88**	0.83	0.84	0.85	**0.88**
IDRAS5	0.83	0.67	0.77	**0.87**	0.85	0.86	**0.87**	**0.87**
AVG.	0.848	0.674	0.778	0.882	0.848	0.854	0.864	**0.89**

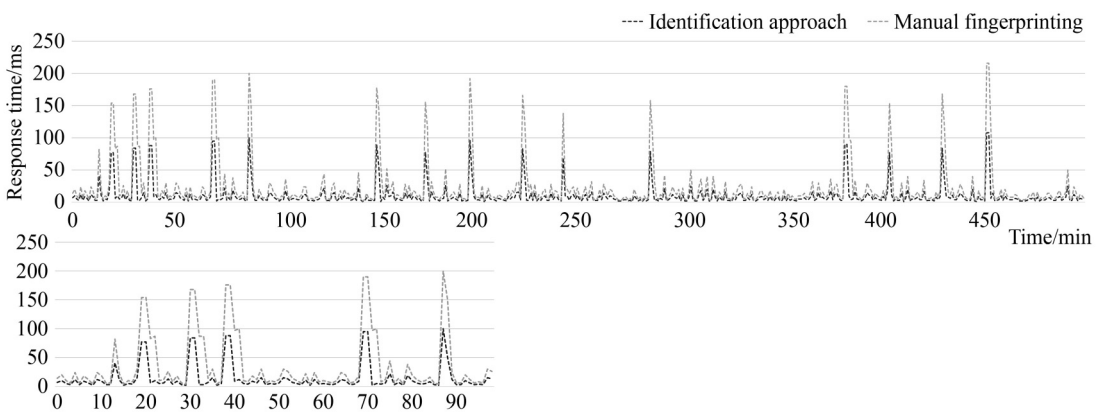

Fig. 9 The response time of using performance issue method and manual identification

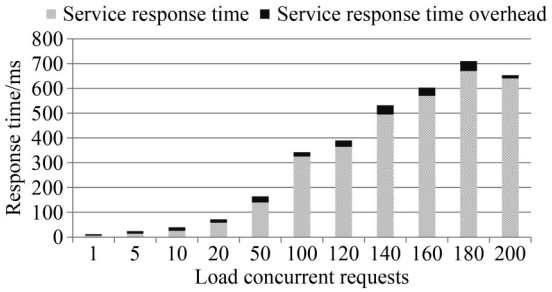

Fig. 10 The influence of our proposed performance issue diagnosis method for service performance

6.3.2 Effectiveness experiments

This section mainly introduces the effectiveness experiments. These experiments were used to verify the effectiveness of our proposed method. To verify the effectiveness of the RBM with ICA, we compared it with 5 comparison models, such as *Gaussian Naive Bayes Classifier* (GaussianNBC) [164], *Decision Tree* (DT) [165], Boosting [166], *Maximum Entropy* (ME) [167], and RBM without ICA. Table 7 gives the experimental results.

As shown in Table 7, the performance issue diagnosis model constructed by RBM is superior to the other four shallow classification algorithms in average *Precision*, average *Pecall* and average *F*1 value. This further show the effectiveness of RBM and ICA.

To verify the effectiveness of our proposed method for response time, we compared the response time of using performance issue identification method, performance issue diagnosis method and manual identification when the system is suffering performance issue. Figure 11 gives the response time of using performance issue identification method, performance issue diagnosis method and manual identification.

Table 7 *F*1 of the RBM with ICA and comparison models

Models	Metrics		
	Precision	*Recall*	*F*1
RBM with ICA	0.88	0.91	0.89
RBM without ICA	0.87	0.90	0.89
GNB	0.86	0.87	0.87
DT	0.84	0.86	0.85
Boosting	0.84	0.87	0.86
ME	0.83	0.86	0.86

As shown in Fig. 11, the average response time rapidly increases when the system is suffering performance issue. Specifically, the performance issue diagnosis method is more effective in reducing response time in the period of system performance issue compared with the performance issue identification method and manual identification method. The experimental results demonstrate that our proposed performance issue diagnosis method can effectively reduce response time in the period of system performance issue and restore the service capability of the system in time.

7 Conclusion and future directions

In this paper, we first divide the performance issue identification and diagnosis method of SaaS software into three steps according to their function: performance log generation, performance issue identification and performance issue diagnosis. Then, we comprehensively review these methods by their development history and describe their characteristics and difference among them. Meanwhile, we give our proposed solution for each step. Finally, the effectiveness of our proposed methods is shown by experiments on a real Integrated Disaster Reduction Application System (IDRAS).

With the rapid development of SaaS patterns, identifying and diagnosing performance issues during the operation of SaaS software systems has become increasingly important. The identification and diagnosis methods studied in this article have certain application value. However, with the development of the SaaS model, software has become increasingly complex and diverse. In response to this reality, the method proposed in this article can be improved in the following aspects.

(1) The method proposed in this article mainly targets certain specific attributes in performance logs and applies them to a specific SaaS software application system. However, when identifying and diagnosing performance issues with different types of SaaS software, these attributes or combinations of attributes may no longer be suitable. Therefore, in future work, we will apply our method to various application systems and search for methods to automatically construct classification features.

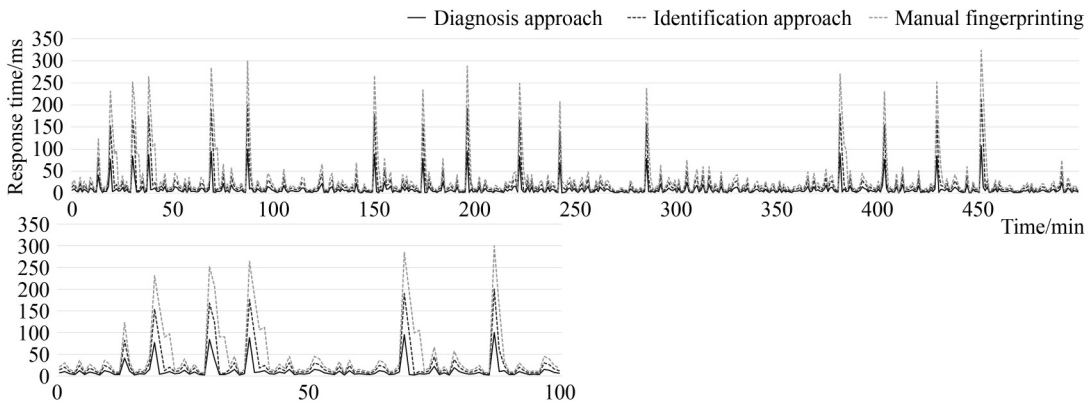

Fig. 11 The response time of using performance issue identification method, performance issue diagnosis method and manual identification

(2) The methods proposed in this article respectively use HMRF and RBM to establish recognition and diagnostic models. When establishing the model, all training data will be stored in memory. However, as the model undergoes more and more model updates, more and more performance status logs will be saved. This situation will result in the updating and construction of the model taking longer and longer, affecting the efficiency of recognition and diagnosis. Therefore, in future work, parallel approaches can be considered to construct the model.

(3) The method proposed in this article involves two steps in SaaS software performance analysis: identifying performance problems and identifying the causes of performance problems. The next step should be to select corresponding measures from performance methods to solve performance problems and evaluate performance. Identifying performance issues and identifying their causes can accelerate the speed of restoring service levels, and appropriate problem-solving solutions or strategies can better address the situation and stop losses in a timely manner. Therefore, in future work, we will address the performance issues of SaaS software.

Acknowledgments The work was supported by the National Key R&D Program of China (2022YFB3304300), the Humanities and Social Sciences Youth Foundation, Ministry of Education (23YJCZH221) and the Natural Science Foundation of Shandong Province (ZR2023QE030).

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

Data availability statement The data that support the findings of this study are available from the National Disaster Reduction Center of China but restrictions apply to the availability of these data, which were used under license for the current study, and so are not publicly available. Data are however available from the authors upon reasonable request and with permission of the National Disaster Reduction Center of China.

References

1. Chen Z, Kim M, Cui Y. SaaS application mashup based on high speed message processing. KSII Transactions on Internet and Information Systems (TIIS), 2022, 16(5): 1446–1465
2. De León Guillén M Á D, Morales-Rocha V, Fernández Martínez L F. A systematic review of security threats and countermeasures in SaaS. Journal of Computer Security, 2020, 28(6): 635–653
3. Soni D, Kumar N. Machine learning techniques in emerging cloud computing integrated paradigms: a survey and taxonomy. Journal of Network and Computer Applications, 2022, 205: 103419
4. Li W, Zhang Y, Guo Z, Liu L. Study on SaaS cloud service development for telecom operators. Telecommunications Science, 2012, 28(1): 132–136
5. Ju J, Wang Y, Fu J, Wu J, Lin Z. Research on key technology in SaaS. In: Proceedings of 2010 International Conference on Intelligent Computing and Cognitive Informatics. 2010, 384−387
6. O'Dywer R, Neville S W. Assessing QoS consistency in cloud-based software-as-a-service deployments. In: Proceedings of 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM). 2017, 1−6
7. He Q, Han J, Yang Y, Grundy J, Jin H. QoS-driven service selection for multi-tenant SaaS. In: Proceedings of the 5th IEEE International Conference on Cloud Computing. 2012, 566−573
8. Varshney S, Sandhu R, Gupta P K. QoS based resource provisioning in cloud computing environment: a technical survey. In: Proceedings of the 3rd International Conference on Advances in Computing and Data Sciences. 2019, 711−723
9. Park J, Jeong H Y. The QoS-based MCDM system for SaaS ERP applications with social network. The Journal of Supercomputing, 2013, 66(2): 614–632
10. Luo H, Shyu M L. Quality of service provision in mobile multimedia-a survey. Human-centric Computing and Information Sciences, 2011, 1(1): 5
11. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. Concurrency and Computation: Practice and Experience, 2005, 17(2−4): 323−356
12. Berman F, Fox G, Hey A J G. Grid Computing: Making the Global Infrastructure A Reality. New York: John Wiley & Sons, 2003
13. Gao J, Pattabhiraman P, Bai X, Tsai W T. SaaS performance and scalability evaluation in clouds. In: Proceedings of the 6th International Symposium on Service Oriented System (SOSE). 2011, 61−71
14. Wang R, Ying S. SaaS software performance issue identification using HMRF-MAP framework. Software: Practice and Experience, 2018, 48(11): 2000–2018
15. Munshi M, Shrimali T, Gaur S. A review of enhancing online learning using graph-based data mining techniques. Soft Computing, 2022, 26(12): 5539–5552
16. Batool I, Khan T A. Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review. Computers and Electrical Engineering, 2022, 100: 107886
17. El-Masri D, Petrillo F, Guéhéneuc Y G, Hamou-Lhadj A, Bouziane A. A systematic literature review on automated log abstraction techniques. Information and Software Technology, 2020, 122: 106276
18. Zhong Y, Guo Y, Liu C. *FLP*: a feature-based method for log parsing. Electronics Letters, 2018, 54(23): 1334–1336
19. Zhang C, Meng X. Log parser with one-to-one markup. In: Proceedings of the 3rd International Conference on Information and Computer Technologies (ICICT). 2020, 251−257
20. Fang L, Di X, Liu X, Qin Y, Ren W, Ding Q. QuickLogS: a quick log parsing algorithm based on template similarity. In: Proceedings of the 20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). 2021, 1085−1092
21. Zeng L, Xiao Y, Chen H, Sun B, Han W. Computer operating system logging and security issues: a survey. Security and Communication Networks, 2016, 9(17): 4804–4821
22. Chen B, Jiang Z M. A survey of software log instrumentation. ACM Computing Surveys, 2022, 54(4): 90
23. Behera A, Panigrahi C R, Pati B. Unstructured Log Analysis for System Anomaly Detection—A Study. Advances in Data Science and Management: Proceedings of ICDSM 2021. Singapore: Springer Nature Singapore, 2022, 497-509, 149−158
24. Fu Q, Lou J G, Lin Q, Ding R, Zhang D, Xie T. Contextual analysis of program logs for understanding system behaviors. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR). 2013, 397−400
25. Clayman S, Galis A, Mamatas L. Monitoring virtual networks with lattice. In: Proceedings of 2010 IEEE/IFIP Network Operations and Management Symposium Workshops. 2010, 239−246
26. Yao K, De Pádua G B, Shang W, Sporea C, Toma A, Sajedi S. Log4perf: Suggesting and updating logging locations for web-based systems' performance monitoring. Empirical Software Engineering, 2020, 25(1): 488–531
27. Rong G, Zhang Q, Liu X, Gu S. A systematic review of logging practice in software engineering. In: Proceedings of the 24th Asia-

Pacific Software Engineering Conference (APSEC). 2017, 534−539
28. He S, He P, Chen Z, Yang T, Su Y, Lyu M R. A survey on automated log analysis for reliability engineering. ACM Computing Surveys, 2022, 54(6): 130
29. Gujral H, Lal S, Li H. An exploratory semantic analysis of logging questions. Journal of Software: Evolution and Process, 2021, 33(7): e2361
30. Schwarz C. Ldagibbs: A command for topic modeling in Stata using latent dirichlet allocation. The Stata Journal: Promoting Communications on Statistics and Stata, 2018, 18(1): 101–117
31. Joung J, Kim H M. Automated keyword filtering in latent dirichlet allocation for identifying product attributes from online reviews. Journal of Mechanical Design, 2021, 143(8): 084501
32. Li H, Liu J, Zhang S. Hierarchical latent dirichlet allocation models for realistic action recognition. In: Proceedings of 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2011, 1297−1300
33. Fu J, Liu N, Hu C, Zhang X. Hot topic classification of microblogging based on cascaded latent dirichlet allocation. ICIC Express Letters, Part B: Applications, 2016, 7(3): 621–625
34. Wu J, Son G, Wang S. A competency mining method based on latent dirichlet allocation (LDA) model. Journal of Physics: Conference Series, 2020, 1682: 012059
35. Liu Y, Jin Z. A text classification model constructed by latent dirichlet allocation and deep learning. In: Proceedings of the 4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering. 2015
36. Rus V, Niraula N, Banjade R. Similarity measures based on latent dirichlet allocation. In: Proceedings of the 14th International Conference on Computational Linguistics and Intelligent Text Processing. 2013, 459−470
37. Yuan D, Park S, Huang P, Liu Y, Lee M M, Tang X, Zhou Y, Savage S. Be conservative: enhancing failure diagnosis with proactive logging. In: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation. 2012, 293−306
38. Fu Q, Zhu J, Hu W, Lou J G, Ding R, Lin Q, Zhang D, Xie T. Where do developers log? An empirical study on logging practices in industry. In: Proceedings of the 36th International Conference on Software Engineering. 2014, 24−33
39. Zhu J, He P, Fu Q, Zhang H, Lyu M R, Zhang D. Learning to log: helping developers make informed logging decisions. In: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering. 2015, 415−425
40. Li Z. Studying and suggesting logging locations in code blocks. In: Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering: Companion Proceedings. 2020, 125−127
41. Gholamian S. Leveraging code clones and natural language processing for log statement prediction. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2021, 1043−1047
42. Cinque M, Cotroneo D, Pecchia A. Event logs for the analysis of software failures: a rule-based approach. IEEE Transactions on Software Engineering, 2013, 39(6): 806–821
43. Li S, Niu X, Jia Z, Liao X, Wang J, Li T. Guiding log revisions by learning from software evolution history. Empirical Software Engineering, 2020, 25(3): 2302–2340
44. Zhang H, Tang Y, Lamothe M, Li H, Shang W. Studying logging practice in test code. Empirical Software Engineering, 2022, 27(4): 83
45. Zadrozny P, Kodali R. Big Data Analytics Using Splunk: Deriving Operational Intelligence from Social Media, Machine Data, Existing Data Warehouses, and Other Real-Time Streaming Sources. Berkeley: Apress, 2013
46. Patel H A, Meniya A D. A survey on commercial and open source cloud monitoring. International Journal of Science and Modern Engineering (IJISME), 2013, 1(2): 42–44
47. George L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data. Sebastopol: O'Reilly Media, Inc., 2011
48. Serrano D, Han D, Stroulia E. From relations to multi-dimensional maps: towards an SQL-to-HBase transformation methodology. In: Proceedings of the 8th IEEE International Conference on Cloud Computing. 2015, 81−89
49. Bhupathiraju V, Ravuri R P. The dawn of big data-Hbase. In: Proceedings of 2014 Conference on IT in Business, Industry and Government (CSIBIG). 2014, 1−4
50. Saloustros G, Magoutis K. Rethinking Hbase: design and implementation of an elastic key-value store over log-structured local volumes. In: Proceedings the 14th International Symposium on Parallel and Distributed Computing. 2015, 225−234
51. Zhang C, Liu X. HBaseMQ: a distributed message queuing system on clouds with HBase. In: Proceedings of 2013 Proceedings IEEE INFOCOM. 2013, 40−44
52. Hou Y, Yuan S, Xu W, Wei D. Transformation of an E-R model into HBase tables: a data store design for IHE-XDS document registry. In: Proceedings of the 12th IEEE International Conference on Ubiquitous Intelligence and Computing and the 12th IEEE International Conference on Autonomic and Trusted Computing and the 15th IEEE International Conference on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom). 2015, 1809−1812
53. Bao X, Liu L, Xiao N, Liu F, Zhang Q, Zhu T. HConfig: resource adaptive fast bulk loading in HBase. In: Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing. 2014, 215−224
54. Giblin C, Rooney S, Vetsch P, Preston A. Securing Kafka with encryption-at-rest. In: Proceedings of 2021 IEEE International Conference on Big Data (Big Data). 2021, 5378−5387
55. Wang Z, Dai W, Wang F, Deng H, Wei S, Zhang X, Liang B. Kafka and its using in high-throughput and reliable message distribution. In: Proceedings of the 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS). 2015, 117−120
56. Wu H. Research proposal: reliability evaluation of the apache Kafka streaming system. In: Proceedings of 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). 2019, 112−113
57. Zhang H, Fang L, Jiang K, Zhang W, Li M, Zhou L. Secure door on cloud: a secure data transmission scheme to protect Kafka's data. In: Proceedings of the 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS). 2020, 406−413
58. Tsai W, Bai X, Huang Y. Software-as-a-service (SaaS): perspectives and challenges. Science China Information Sciences, 2014, 57(5): 1–15
59. Liu D, Pei D, Zhao Y. Application-aware latency monitoring for cloud tenants via CloudWatch+. In: Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop. 2014, 73−81
60. Stephen A, Benedict S, Kumar R P A. Monitoring IaaS using various cloud monitors. Cluster Computing, 2019, 22(5): 12459–12471
61. Da Silva Rocha É, Da Silva L G F, Santos G L, Bezerra D, Moreira A, Gonçalves G, Marquezini M V, Mehta A, Wildeman M, Kelner J, Sadok D, Endo P T. Aggregating data center measurements for availability analysis. Software: Practice and Experience, 2021, 51(5):

868–892

62. Tasquier L, Venticinque S, Aversa R, Di Martino B. Agent based application tools for cloud provisioning and management. In: Proceedings of the 3rd International Conference on Cloud Computing. 2012, 32−42
63. De Chaves S A, Uriarte R B, Westphall C B. Toward an architecture for monitoring private clouds. IEEE Communications Magazine, 2011, 49(12): 130–137
64. Massie M L, Chun B N, Culler D E. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, 2004, 30(7): 817–840
65. Nagios X. The industry standard in it infrastructure monitoring. See Logon-int.com/nagios/ website, 2011
66. Mardiyono A, Sholihah W, Hakim F. Mobile-based network monitoring system using Zabbix and telegram. In: Proceedings of the 3rd International Conference on Computer and Informatics Engineering (IC2IE). 2020, 473−477
67. Andreozzi S, De Bortoli N, Fantinel S, Ghiselli A, Rubini G L, Tortone G, Vistoli M C. GridiCE: a monitoring service for grid systems. Future Generation Computer Systems, 2005, 21(4): 559–571
68. König B, Calero J M A, Kirschnick J. Elastic monitoring framework for cloud infrastructures. IET Communications, 2012, 6(10): 1306–1315
69. Povedano-Molina J, Lopez-Vega J M, Lopez-Soler J M, Corradi A, Foschini L. DARGOS: a highly adaptable and scalable monitoring architecture for multi-tenant clouds. Future Generation Computer Systems, 2013, 29(8): 2041–2056
70. Meng S, Liu L. Enhanced monitoring-as-a-service for effective cloud management. IEEE Transactions on Computers, 2013, 62(9): 1705–1720
71. Calero J M A, Aguado J G. MonPaaS: An adaptive monitoring platformas a service for cloud computing infrastructures and services. IEEE Transactions on Services Computing, 2015, 8(1): 65–78
72. Alhamazani K, Ranjan R, Jayaraman P P, Mitra K, Liu C, Rabhi F, Georgakopoulos D, Wang L. Cross-layer multi-cloud real-time application QoS monitoring and benchmarking as-a-service framework. IEEE Transactions on Cloud Computing, 2019, 7(1): 48–61
73. Wang H, Zhang X, Ma Z, Li L, Gao J. An microservices-based openstack monitoring system. In: Proceedings of the 11th International Conference on Educational and Information Technology (ICEIT). 2022, 232−236
74. Badshah A, Jalal A, Farooq U, Rehman G U, Band S S, Iwendi C. Service level agreement monitoring as a service: an independent monitoring service for service level agreements in clouds. Big Data, 2023, 11(5): 339–354
75. Mezni H, Sellami M, Aridhi S, Charrada F B. Towards big services: a synergy between service computing and parallel programming. Computing, 2021, 103(11): 2479–2519
76. Mezni H. Web service adaptation: a decade's overview. Computer Science Review, 2023, 48: 100535
77. Kumar R, Jain K, Maharwal H, Jain N, Dadhich A. Apache CloudStack: open source infrastructure as a service cloud computing platform. International Journal of Advancement in Engineering Technology, Management & Applied Science, 2014, 1(2): 111–116
78. Schwartz B, Zaitsev P, Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication. Sebastopol: O'Reilly Media, Inc., 2012
79. Sun W, Zhang X, Guo C J, Sun P, Su H. Software as a service: configuration and customization perspectives. In: Proceedings of 2008 IEEE Congress on Services Part II (Services-2 2008). 2008, 18−25
80. Lan Z, Zheng Z, Li Y. Toward automated anomaly identification in large-scale systems. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(2): 174–187
81. Yu L, Lan Z. A scalable, non-parametric method for detecting performance anomaly in large scale computing. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(7): 1902–1914
82. Odyurt U, Meyer H, Pimentel A D, Paradas E, Alonso I G. Software passports for automated performance anomaly detection of cyber-physical systems. In: Proceedings of the 19th International Conference on Embedded Computer Systems. 2019, 255−268
83. Wang R, Ying S. SaaS software performance issue diagnosis using independent component analysis and restricted boltzmann machine. Concurrency and Computation: Practice and Experience, 2020, 32(14): e5729
84. Zhao N, Han B, Cai Y, Su J. SeqAD: an unsupervised and sequential autoencoder ensembles based anomaly detection framework for KPI. In: Proceedings of the 29th IEEE/ACM International Symposium on Quality of Service (IWQOS). 2021, 1−6
85. Chaturvedi A. Method and system for near real time reduction of insignificant key performance indicator data in a heterogeneous radio access and core network. In: Proceedings of 2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). 2020, 1−7
86. Kusrini E, Safitri K N, Fole A. Design key performance indicator for distribution sustainable supply chain management. In: Proceedings of 2020 International Conference on Decision Aid Sciences and Application (DASA). 2020, 738−744
87. Hinderks A, Schrepp M, Mayo F J D, Escalona M J, Thomaschewski J. Developing a UX KPI based on the user experience questionnaire. Computer Standards & Interfaces, 2019, 65: 38–44
88. Fotrousi F, Fricker S A, Fiedler M, Le-Gall F. KPIs for software ecosystems: a systematic mapping study. In: Proceedings of the 5th International Conference of Software Business. 2014, 194−211
89. Zhang S, Zhao C, Sui Y, Su Y, Sun Y, Zhang Y, Pei D, Wang Y. Robust KPI anomaly detection for large-scale software services with partial labels. In: Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering (ISSRE). 2021, 103−114
90. Jiang Y, Haihong E, Song M, Zhang K. Research and application of newborn defects prediction based on spark and PU-learning. In: Proceedings of the 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS). 2018, 657−663
91. Shu S, Lin Z, Yan Y, Li L. Learning from multi-class positive and unlabeled data. In: Proceedings of 2020 IEEE International Conference on Data Mining (ICDM). 2020, 1256−1261
92. Chen X, Chen W, Chen T, Yuan Y, Gong C, Chen K, Wang Z. Self-PU: Self boosted and calibrated positive-unlabeled training. In: Proceedings of the 37th International Conference on Machine Learning. 2020, 141
93. Han K, Chen W, Xu M. Investigating active positive-unlabeled learning with deep networks. In: Proceedings of the 34th Australasian Joint Conference on Advances in Artificial Intelligence. 2022, 607−618
94. Hu W, Le R, Liu B, , Ji F, Ma J, Zhao D, Yan R. Predictive adversarial learning from positive and unlabeled data. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2021, 7806-7814
95. Qiu J, Cai X, Zhang X, Cheng F, Yuan S, Fu G. An evolutionary multi-objective approach to learn from positive and unlabeled data. Applied Soft Computing, 2021, 101: 106986

96. Gong C, Liu T, Yang J, Tao D. Large-margin label-calibrated support vector machines for positive and unlabeled learning. IEEE Transactions on Neural Networks and Learning Systems, 2019, 30(11): 3471–3483
97. He P, Zhu J, Zheng Z, Lyu M R. Drain: an online log parsing approach with fixed depth tree. In: Proceedings of 2017 IEEE International Conference on Web Services (ICWS). 2017, 33−40
98. Plaat A, Schaeffer J, Pijls W, De Bruin A. Best-first fixed-depth minimax algorithms. Artificial Intelligence, 1996, 87(1-2): 255–293
99. Du M, Li F. Spell: online streaming parsing of large unstructured system logs. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(11): 2213–2227
100. Wang B, Yang X, Li J. Locating longest common subsequences with limited penalty. In: Proceedings of the 22nd International Conference on Database Systems for Advanced Applications. 2017, 187−201
101. Weems B P, Bai Y. Finding longest common increasing subsequence for two different scenarios of non-random input sequences. In: Proceedings of 2005 International Conference on Foundations of Computer Science. 2005, 64−72
102. Meng W, Liu Y, Zaiter F, Zhang S, Chen Y, Zhang Y, Zhu Y, Wang E, Zhang R, Tao S, Yang D, Zhou R, Pei D. LogParse: making log parsing adaptive through word classification. In: Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN). 2020, 1−9
103. Vervaet A, Chiky R, Callau-Zori M. USTEP: unfixed search tree for efficient log parsing. In: Proceedings of 2021 IEEE International Conference on Data Mining (ICDM). 2021, 659−668
104. Chakrabarti A, Striegel A, Manimaran G. A case for tree evolution in QoS multicasting. In: Proceedings of the 10th IEEE International Workshop on Quality of Service (Cat. No.02EX564). 2002, 116−125
105. Li K. A random-walk-based dynamic tree evolution algorithm with exponential speed of convergence to optimality on regular networks. In: Proceedings of the 4th International Conference on Frontier of Computer Science and Technology. 2009, 80−85
106. Tomer A, Schach S R. The evolution tree: a maintenance-oriented software development model. In: Proceedings of the 4th European Conference on Software Maintenance and Reengineering. 2000, 209−214
107. Dai H, Li H, Chen C S, Shang W, Chen T H. *Logram*: efficient log parsing using *n*-gram dictionaries. IEEE Transactions on Software Engineering, 2022, 48(3): 879–892
108. Fu Q, Lou J G, Wang Y, Li J. Execution anomaly detection in distributed systems through unstructured log analysis. In: Proceedings of the 9th IEEE International Conference on Data Mining. 2009, 149−158
109. Xu W, Huang L, Fox A, Patterson D, Jordan M I. Detecting large-scale system problems by mining console logs. In: Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles. 2009, 117−132
110. Nedelkoski S, Cardoso J, Kao O. Anomaly detection from system tracing data using multimodal deep learning. In: Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD). 2019, 179−186
111. Geiger A, Liu D, Alnegheimish S, Cuesta-Infante A, Veeramachaneni K. TadGAN: Time series anomaly detection using generative adversarial networks. In: Proceedings of 2020 IEEE International Conference on Big Data (Big Data). 2020, 33−43
112. Luo W, Wang P, Wang J, An W. The research process of generative adversarial networks. Journal of Physics: Conference Series, 2019, 1176(3): 032008
113. Tran N T, Tran V H, Nguyen N B, Nguyen T K, Cheung N M. On data augmentation for GAN training. IEEE Transactions on Image Processing, 2021, 30: 1882–1897
114. Liu Z, Sabar N, Song A. Improving evolutionary generative adversarial networks. In: Proceedings of the 34th Australasian Joint Conference on Artificial Intelligence. 2022, 691−702
115. Sinha R, Sankaran A, Vatsa M, Singh R. AuthorGAN: Improving GAN reproducibility using a modular GAN framework. 2019, arXiv preprint arXiv: 1911.13250
116. Xia W, Zhang Y, Yang Y, Xue J H, Zhou B, Yang M H. GAN inversion: a survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023, 45(3): 3121–3138
117. Wang X, Cao Q, Wang Q, Cao Z, Zhang X, Wang P. Robust log anomaly detection based on contrastive learning and multi-scale mass. The Journal of Supercomputing, 2022, 78(16): 17491–17512
118. Zhang Z, Wu S, Jiang D, Chen G. BERT-JAM: boosting BERT-enhanced neural machine translation with joint attention. 2020, arXiv preprint arXiv: 2011.04266
119. Trang N T M, Shcherbakov M. Vietnamese question answering system from multilingual BERT models to monolingual BERT model. In: Proceedings of the 9th International Conference System Modeling and Advancement in Research Trends (SMART). 2020, 201−206
120. Shi L, Liu D, Liu G, Meng K. AUG-BERT: an efficient data augmentation algorithm for text classification. In: Proceedings of the 8th International Conference in Communications, Signal Processing, and Systems. 2020, 2191−2198
121. Praechanya N, Sornil O. Improving Thai named entity recognition performance using BERT transformer on deep networks. In: Proceedings of the 6th International Conference on Machine Learning Technologies. 2021, 177−183
122. Yang L, Chen J, Wang Z, Wang W, Jiang J, Dong X, Zhang W. Semi-supervised log-based anomaly detection via probabilistic label estimation. In: Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE). 2021, 1448−1460
123. Farzad A, Gulliver T A. Log message anomaly detection with fuzzy c-means and MLP. Applied Intelligence, 2022, 52(15): 17708–17717
124. Zhang C, Peng X, Sha C, Zhang K, Fu Z, Wu X, Lin Q, Zhang D. DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning. In: Proceedings of the 44th International Conference on Software Engineering. 2022, 623−634
125. Zhang C, Peng X, Zhou T, Sha C, Yan Z, Chen Y, Yang H. TraceCRL: contrastive representation learning for microservice trace analysis. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2022, 1221−1232
126. Aguilera M K, Mogul J C, Wiener J L, Reynolds P, Muthitacharoen A. Performance debugging for distributed systems of black boxes. ACM SIGOPS Operating Systems Review, 2003, 37(5): 74–89
127. Chen Y Y M. Path-Based Failure and Evolution Management. Berkeley: University of California at Berkeley, 2004
128. Barham P, Donnelly A, Isaacs R, Mortier R. Using magpie for request extraction and workload modelling. In: Proceedings of the 6th Symposium on Operating System Design & Implementation. 2004, 18
129. Chen H, Jiang G, Ungureanu C, Yoshihira K. Failure detection and localization in component based systems by online tracking. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. 2005, 750−755
130. Lim M H, Lou J G, Zhang H, Fu Q, Teoh A B J, Lin Q, Ding R, Zhang D. Identifying recurrent and unknown performance issues. In: Proceedings of 2014 IEEE International Conference on Data Mining. 2014, 320−329
131. Fischer A, Igel C. Training restricted Boltzmann machines: an introduction. Pattern Recognition, 2014, 47(1): 25–39
132. Carreira-Perpinan M Á, Hinton G E. On contrastive divergence

learning. In: Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics. 2005, 33−40

133. Liu P, Xu H, Ouyang Q, Jiao R, Chen Z, Zhang S, Yang J, Mo L, Zeng J, Xue W, Pei D. Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks. In: Proceedings of the 31st IEEE International Symposium on Software Reliability Engineering (ISSRE). 2020, 48−58
134. Kohyarnejadfard I, Aloise D, Dagenais M R, Shakeri M. A framework for detecting system performance anomalies using tracing data analysis. Entropy, 2021, 23(8): 1011
135. Cai Y, Han B, Su J, Wang X. TraceModel: an automatic anomaly detection and root cause localization framework for microservice systems. In: Proceedings of the 17th International Conference on Mobility, Sensing and Networking (MSN). 2021, 512−519
136. Li M, Tang D, Wen Z, Cheng Y. Microservice anomaly detection based on tracing data using semi-supervised learning. In: Proceedings of the 4th International Conference on Artificial Intelligence and Big Data (ICAIBD). 2021, 38−44
137. Liu J, Hu Y, Wu B, Wang Y, Xie F. A hybrid generalized hidden markov model-based condition monitoring approach for rolling bearings. Sensors, 2017, 17(5): 1143
138. Wang R, Ying S, Sun C, Wan H, Zhang H, Jia X. Model construction and data management of running log in supporting saas software performance analysis. In: Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE 2017). 2017, 149−154
139. Fu X, Ren R, Zhan J, Zhou W, Jia Z, Lu G. LogMaster: mining event correlations in logs of large-scale cluster systems. In: Proceedings of the 31st IEEE Symposium on Reliable Distributed Systems. 2012, 71−80
140. Zou D, Qin H, Jin H, Qiang W, Han Z, Chen X. Improving log-based fault diagnosis by log classification. In: Proceedings of the 11th IFIP International Conference on Network and Parallel Computing. 2014, 446−458
141. Guo X, Peng X, Wang H, Li W, Jiang H, Ding D, Xie T, Su L. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020, 1387−1397
142. Huo Y, Dong J, Ge Z, Xie P, An N, Yang Y. IWApriori: an association rule mining and self-updating method based on weighted increment. In: Proceedings of the 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). 2020, 167−172
143. Wang L, Zhao N, Chen J, Li P, Zhang W, Sui K. Root-cause metric location for microservice systems via log anomaly detection. In: Proceedings of 2020 IEEE International Conference on Web Services (ICWS). 2020, 142−150
144. Liu D, He C, Peng X, Lin F, Zhang C, Gong S, Li Z, Ou J, Wu Z. MicroHECL: High-efficient root cause localization in large-scale microservice systems. In: Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2021, 338−347
145. Gan Y, Liang M, Dev S, Lo D, Delimitrou C. Sage: practical and scalable ML-driven performance debugging in microservices. In: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021, 135−151
146. Ma M, Lin W, Pan D, Wang P. ServiceRank: root cause identification of anomaly in large-scale microservice architectures. IEEE Transactions on Dependable and Secure Computing, 2022, 19(5): 3087–3100
147. Li M, Li Z, Yin K, Nie X, Zhang W, Sui K, Pei D. Causal inference-based root cause analysis for online service systems with intervention recognition. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2022, 3230−3240
148. Zhao X, Zhang Y, Lion D, Ullah M F, Luo Y, Yuan D, Stumm M. lprof: a non-intrusive request flow profiler for distributed systems. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation. 2014, 629−644
149. Bare K, Kavulya S P, Tan J, Pan X, Marinelli E, Kasick M, Gandhi R, Narasimhan P. ASDF: an automated, online framework for diagnosing performance problems. In: Casimiro A, Lemos R, Gacek C, eds. Architecting Dependable Systems VII. Berlin: Springer, 2010, 201−226
150. Attariyan M, Chow M, Flinn J. X-ray: automating root-cause diagnosis of performance anomalies in production software. In: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation. 2012, 307−320
151. Malik H, Hemmati H, Hassan A E. Automatic detection of performance deviations in the load testing of large scale systems. In: Proceedings of the 35th International Conference on Software Engineering (ICSE). 2013, 1012−1021
152. Tuncer O, Ates E, Zhang Y, Turk A, Brandt J, Leung V J, Egele M, Coskun A K. Online diagnosis of performance variation in hpc systems using machine learning. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(4): 883–896
153. Li M, Tang D, Wen Z, Cheng Y. Universal anomaly detection method based on massive monitoring indicators of cloud platform. In: Proceedings of 2021 IEEE International Conference on Software Engineering and Artifcial Intelligence (SEAI). 2021, 23−29
154. Borghesi A, Molan M, Milano M, Bartolini A. Anomaly detection and anticipation in high performance computing systems. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(4): 739–750
155. Stehman S V. Selecting and interpreting measures of thematic classification accuracy. Remote Sensing of Environment, 1997, 62(1): 77–89
156. Powers D M W. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2020, arXiv preprint arXiv: 2010.16061
157. Hsieh W W. Machine Learning Methods in the Environmental Sciences: Neural Networks and Kernels. Cambridge: Cambridge University Press, 2009
158. Qin J, He Z S. A SVM face recognition method based on Gabor-featured key points. In: Proceedings of 2005 International Conference on Machine Learning and Cybernetics. 2005, 5144−5149
159. Hearst M A, Dumais S T, Osuna E, Platt J, Scholkopf B. Support vector machines. IEEE Intelligent Systems and Their Applications, 1998, 13(4): 18–28
160. Rish I. An empirical study of the naive Bayes classifier. In: Proceedings of IJCAI 2001Workshop on Empirical Methods in Artificial Intelligence. 2001, 41–46
161. Larose D T, Larose C D. k-nearest neighbor algorithm. In: Discovering Knowledge in Data: An Introduction to Data Mining. Wiley, 2014, 149–164
162. Manning C, Raghavan P, Schütze H. Vector space classification. In: An Introduction to Information Retrieval. 2009, 289–317
163. Freedman D A. Statistical Models: Theory and Practice. Cambridge, England: Cambridge University Press, 2009
164. Lam P, Wang L, Ngan H Y, Yung N H, Yeh A G. Outlier detection in large-scale traffic data by naïve Bayes method and Gaussian mixture model method. 2015, arXiv preprint arXiv: 1512.08413
165. Loh W Y. Classification and regression trees. WIREs: Data Mining and Knowledge Discovery, 2011, 1(1): 14–23

166. Freund Y, Schapire R E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: Proceedings of the 2nd European Conference on Computational Learning Theory. 1995, 23−37

167. Phillips S J, Anderson R P, Schapire R E. Maximum entropy modeling of species geographic distributions. Ecological Modelling, 2006, 190(3-4): 231–259

Rui Wang is currently a lecturer with the College of Energy and Mining Engineering, Shandong University of Science and Technology, China. She received the PhD degree from Wuhan University, China in 2019. Her research interests include software engineering and data mining.

Xiangbo Tian received the BE degree from the Shandong University of Science and Technology, China in 2018. He is currently working toward the PhD degree with the School of Computer Science, Wuhan University, China. His current research interests include service computing and microservices.

Shi Ying is currently a professor in the School of Computer Science, Wuhan University, China. His main research interests include software engineering and artificial intelligence.