Computers, Materials & Continua CMC, vol.62, no.3, pp.1161-1185, 2020

A Performance Fault Diagnosis Method for SaaS Software Based on GBDT Algorithm

Kun Zhu[1], Shi Ying[1,*], Nana Zhang[1], Rui Wang[1], Yutong Wu[1], Gongjin Lan[2] and Xu Wang[2]

Abstract: SaaS software that provides services through cloud platform has been more widely used nowadays. However, when SaaS software is running, it will suffer from performance fault due to factors such as the software structural design or complex environments. It is a major challenge that how to diagnose software quickly and accurately when the performance fault occurs. For this challenge, we propose a novel performance fault diagnosis method for SaaS software based on GBDT (Gradient Boosting Decision Tree) algorithm. In particular, we leverage the monitoring mean to obtain the performance log and warning log when the SaaS software system runs, and establish the performance fault type set and determine performance log feature. We also perform performance fault type annotation for the performance log combined with the analysis result of the warning log. Moreover, we deal with the incomplete performance log and the type non-equalization problem by using the mean filling for the same type and combination of SMOTE (Synthetic Minority Oversampling Technique) and undersampling methods. Finally, we conduct an empirical study combined with the disaster reduction system deployed on the cloud platform, and it demonstrates that the proposed method has high efficiency and accuracy for the performance diagnosis when SaaS software system runs.

Keywords: GBDT algorithm, SaaS software, performance log, performance fault diagnosis.

1 Introduction

In recent years, with the rapid development of Internet technology, application software has evolved from the single, closed, and static form to the distributed, open, and dynamical from. An innovative application mode Software-as-a-Service (SaaS) begins to rise [Zawoad, Dutta and Hasan (2013)].

SaaS software has to face not only more challenges than the traditional mode in development, such as multi-users, high concurrency, and large data volume [Wen, Zhang and Zhu (2015)], but also more performance problems. SaaS software will suffer the declination of software service quality and even the software performance degradation due to a variety of factors. On one hand, the performance degradation may be caused by the software architecture and code design flaws, so we need to improve its architecture design

[1] School of Computer Science, Wuhan University, Wuhan, 430072, China.

[2] Department of Computer Science, Vrije University Amsterdam, Amster-dam, 1081HV, The Netherlands.

[*] Corresponding Author: Shi Ying. Email: yingshi@whu.edu.cn.

from the software constructing process [Ding, Fu, Lou et al. (2012)]. On the other hand, the performance degradation may be caused by SaaS software running in a large-scale, high-complexity and unpredictable dynamic cloud environment [Jin, Liu, Zheng et al. (2018)]. Possible situations are as follows: (1) Insufficient resources on the virtual machine or physical node. For instance, the CPU utilization, memory and disk space resources reach the threshold; (2) Service requests to the server are too frequent. For example, many users send a large number of service requests at the same time within a certain period of time, which results in consequence that the server cannot process a huge amount of requests in time, so average response time is excessively long for user service requests; (3) Dynamic changes in hard- ware resource operating conditions. For instance, the power interruption or the downtime of the virtual machine, the disconnected network connection, and the unreliable host [Wu, Garg, Versteeg et al. (2014); Nagaraj, Killian and Neville (2012); Mavridis and Karatza (2017); Malik, Hemmati and Hassan (2013)]. The above situations may cause application software in SaaS mode to suffer from software performance degradation such as the long response time, the reduced resource utilization or the throughput rate, and even the loss of availability.

In the modern computer system, logs are used to record the operation status of the system, the event occurring in the system, and the anomaly behavior in the system [Ju, Wang, Fu et al. (2010)]. Therefore, the log data (such as the running log, the warning log, the debug log, etc.) can be considered to be a primary information source for diagnosing the performance fault. The traditional method of diagnosing the performance fault based on logs is to rely on system maintenance personnel to extract the information related to performance faults from a large number of complex logs [Gill, Jain and Nagappan (2011)]. Then they analyze relevant logs based on the experience to diagnose and locate performance faults. However, most of application softwares in SaaS mode are in a distributed cluster environment, which cause application software to interact frequently among various layers. So various components in the system generate a large amount of log data, which not only increases the difficulty of performance fault diagnosis, but also makes the traditional fault diagnosis method difficult to perform real-time and comprehensive fault diagnosis [Duan and Babu (2008); Zou, Qin and Jin (2016)].

However, the performance-related log and fault log data in the current system are insufficient and considered to be sensitive [Schroeder and Gibson (2010)]. Therefore, it is of important significance to dig out the useful information for the automatic performance fault diagnosis from severely lack of performance-related log data.

The main contributions of this paper are as follows:

(1) By analyzing the KPI (Key Performance Indicator) of the software runtime resource layer, we establish the performance fault type set and extract the strongly correlated performance features in the performance log. We also leverage the analysis result of the warning log to perform the reasonable performance fault type annotation for the performance log.

(2) We deal with the incomplete performance log and the non-equalization problem of the type by using the mean filling for the same type and combination of SMOTE (Synthetic Minority Oversampling Technique) and undersampling methods and ensure the validity and equalization of the sample.

(3) We use the GBDT (Gradient Boosting Decision Tree) algorithm to construct a performance fault diagnosis model.

(4) We conducted an empirical case study of a disaster reduction system deployed on the cloud platform to verify that the proposed method has high efficiency and accuracy for the performance diagnosis of SaaS software systems.

The rest of this paper is organized as follows: Section 2 introduces the background and related work of this paper; Section 3 describes our methodology in detail; Section 4 describes the experimental setup; Sections 5 introduces experimental process and results analysis; Section 6 concludes this paper and outlines directions of future work.

2 Background & related work

This section summarizes the background and the related work on performance fault diagnosis methods and processing methods of performance log data.

2.1 The performance fault diagnosis method

In essence, the performance fault diagnosis is a problem of classification and identification. That is, the performance state of a system is divided into normal state and abnormal state. The anomaly state is specifically which kind of specific performance fault, and this is a recognition problem [Tsai, Bai and Huang (2014)]. Here we mainly introduce some performance fault diagnosis methods based on the log.

Lim et al. [Lim, Lou, Zhang et al. (2014)] proposed an automatic identification method based on Hidden Markov Random Field (HMRF) method for known and unknown performance problems or faults. Bezemer et al. [Bezemer and Zaidman (2014)] proposed a method for selecting performance improvement points (PIOs) based on log data, such as bottleneck components or performance fault points, to optimize performance bottlenecks or to process faults. Syer et al. [Syer, Jiang, Nagappan et al. (2013)] proposed a method to automatically diagnose different memory-related problems by combining the performance counter and the execution log.

Wang et al. [Wang, Zhang, Ye et al. (2016)] proposed an online incremental clustering method to recognize access behavior patterns. The method uses the correlation analysis to model the correlation between workloads and application performance/resource utilization metrics in a specific access behavior pattern. Wang et al. [Wang, Wei, Zhang et al. (2014)] proposed an incremental clustering algorithm for training workload patterns online. The method employs the local outlier factor (LOF) in the recognized workload pattern to detect anomalies, and locates the anomalous metrics with the student as t-test method.

2.2 The performance log data processing method

The collected performance log may have some redundant information or missing information or non-equalized cases in the overall data, and they cannot be directly used in performance fault diagnosis. Therefore, it is very important for how to process these performance logs. The following describes the current research methods.

(1) Performance Log Data Missing Processing

Gashler et al. [Gashler, Smith, Morris et al. (2016)] proposed an extended missing-value-

based substitution filling algorithm based on information obtained. He et al. [He and Garcia (2008)] introduced the multi-filling method, and its main idea is to construct m substitute values for each missing value. Rahman et al. [Rahman and Islam (2011)] proposed a technique DMI (A Decision Tree-based Missing Value Imputation Technique) that can effectively fill in missing values in datasets with numeric and classification attributes using a combination of C4.5 decision tree algorithm and EM (Expectation Maximization) algorithm.

(2) Performance Log Equalization Processing

The related performance log data often does not have the feature of class distribution equalization due to certain factors, which leads to the declination in the classification accuracy of traditional machine learning methods. At present, the problem of non-equalized performance log data classification is mainly based on data level research. Research based on the data level generally uses sampling mechanisms to improve the distribution of data. Chawla et al. [Chawla, Bowyer, Hall et al. (2011)] proposed an oversampling-based synthetic minority oversampling technique SMOTE. The algorithm makes use of the similarity between a few kinds of samples in the feature space to artificially generate data samples. This method improved the effect, but it introduced many subjective factors to the original data, which will cause excessive generalization problems. Liu et al. [Liu, Wu and Zhou (2006)] proposed a non-equalized data classification method based on the cascade model. This method combines the ensemble learning with the cascade model and uses the repeated undersampling method to perform data equalization.

3 Methodolody

Before introducing the method proposed in this paper, we first give some definitions:

Performance Log: It refers to recording the related performance information when the system runs, such as the CPU utilization, and it is recorded in the form of numerical value.

Warning Log: It means to record the abnormal state information when the system runs. It usually contains the recording time, the abnormal state, the abnormal level, etc., and it is recorded in the form of text.

Performance Fault: It refers to emerging system performance abnormal cases such as the excessive resource utilization rate, the long response time, and the declined throughput rate when the system runs.

This paper proposes a performance fault diagnosis method based on the GBDT algorithm for the performance log. The performance fault diagnose process is shown in Fig. 1. Firstly, we establish a performance fault type set by analyzing existing performance fault types, and perform feature determination on the collected performance log; Secondly, we analyze the collected warning log and determine the fault type of each warning log, and we give a reasonable performance fault type annotation for each performance log combining with the fault type of the warning log; Then we perform preprocess operations such as the missing value processing and the sample equalization processing for the performance log to form the training set; Finally, we use the GBDT algorithm to construct a performance fault diagnosis model and apply it to perform the performance fault diagnosis on SaaS software.

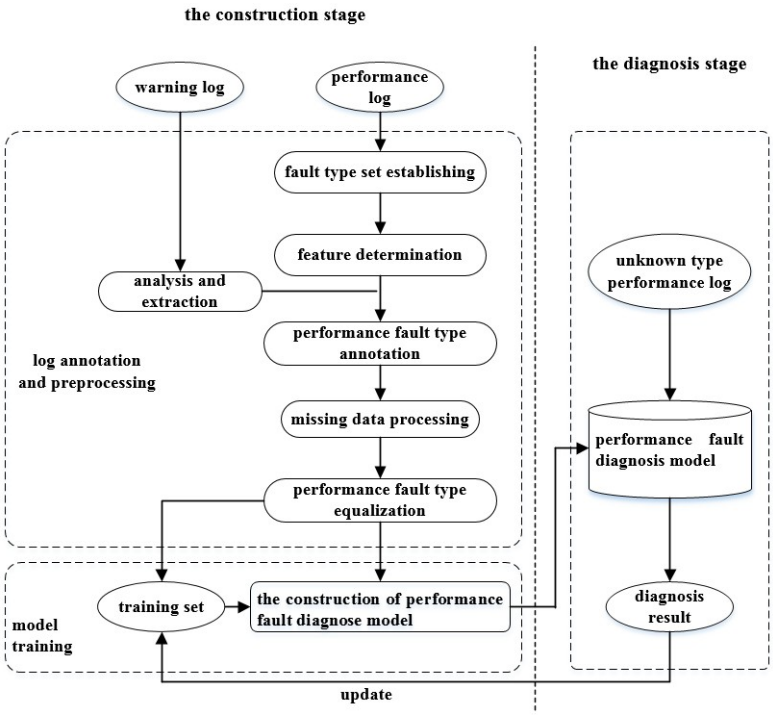

Figure 1: Overview of performance fault diagnose process

3.1 Establishing of the performance fault type set and determination of the performance log feature

Before performing the fault diagnosis based on the performance log, we need to establish the performance fault type set and determine the performance log feature.

3.1.1 Establishing of the performance fault type set

The performance of the application software system in SaaS mode can generally be reflected by indicators such as the response time, the throughput rate, and the resource utilization. The above three performance indicators are interconnected and interactive.

However, if the fault set is only defined as the change of these indicators, it is not enough for the maintenance staff to obtain the specific information directly such as the location of the performance fault, and cannot achieve the purpose of recovering the performance fault quickly. Therefore, this paper will describe the fine-grained performance fault type. The key performance indicators (KPI) at the resource level such as the CPU, the memory, the disk, and the network when the system runs are divided as shown in Tab. 1, and also include the normal status. Tab. 1 lists all performance fault types contained in the performance fault type set established in this paper. The diagnostic result is also based on the performance fault type set. We abstract the performance fault type set into the vector form of performance fault types, and represent them in the following form:

performance_fault_type = < *FTCU, FAPQL,*
FPI, FPO, FCR, FCW, FCTR,
FDU, FDR, FDW, FNI, FNO, Norm. >

Table 1: The set of performance fault types

Object	Performance Fault Type	Performance Fault Description
CPU	*FTCU*	CPU load occurs fault
	FAPQL	Processor queue length occurs fault
memory	*FPI*	Page in occurs fault
	FPO	Page out occurs fault
	FCR	Cache's reading rate occurs fault
	FCW	Cache's writing rate occurs fault
	FCHR	Cache occupancy occurs fault
disk	*FDU*	Disk occupancy occurs fault
	FDR	Disk's reading rate occurs fault
	FDW	Disk's writing rate occurs fault
the internet	*FNI*	The network's receiving data rate occurs fault
	FNO	The network's sending data rate occurs fault
normal	*Norm.*	No performance fault

3.1.2 Determination of the performance log feature

We need to perform the feature determination on the performance log to reflect the performance fault type contained in the performance fault type set. We still start from the resource layer KPIs such as the CPU, the memory, the disk, and the network when the system runs, and determine the feature of the five major category performance logs, which are based on the time feature, the CPU performance feature, the memory performance feature, and the disk performance feature and the network performance feature. The time feature is to prepare for the subsequent performance log annotation, and the rest are the performance fault types within the corresponding performance fault type set. Tab. 2 lists all features of the performance log and describes them briefly.

We can represent the performance log as follows:

performance_log = < *T, CPR_Util., CPU_Int., MEM_U, MEM_PI, MEM_PO,*
DISK_R, DISK_W, DISK_Q, DISK_IO, DISK_L, DISK_TT,
NET_PS, NET_PR, NET_KS, NET_KR>

Table 2: Performance features and description

Type	Feature	Description
time	*T*	Time of recording performance indicators
CPU	*CPU_Util.*	CPU utilization (%)
	CPU_Int.	The number of CPU interrupts per second (/s)
memory	*MEM_U*	Memory occupancy (%)
	MEM_PI	Page in rate (pages/s)
	MEM_PO	Page out rate (pages/s)
disk	*DISK_R*	Disk's reading operation rate (/s)
	DISK_W	Disk's writing operation rate (/s)
	DISK_Q	Disk queue length (queue length)
	DISK_IO	Disk IO rate (/s)
	DISK_L	Disk load rate (%)
	DISK_TT	Disk transmission time (ms)
the internet	*NET_PS*	Network's sending packets rate (packets/s)
	NET_PR	Network's receiving packets rate (packets/s)
	NET_KS	NIC's sending bytes rate (kb/s)
	NET_KR	NIC's receiving bytes rate (kb/s)

3.2 Log analysis and annotation

This paper combines performance logs with the supervised GBDT algorithm for perfor- mance fault diagnosis. Therefore, each performance log used to construct a model has one or more explicit learning objectives or annotations, which are the performance fault type. However, the performance log obtained in reality is often not labeled with the performance fault type, so we need to annotate the log type. Firstly, we need to analyze warning logs and confirm the fault type of each warning log. Then we use the time, the status, the component, and the fault type of the warning log to annotate the performance log properly.

Some monitoring tools record the status changes generated by the system to form some warning logs, which generally include the time or the timestamp of the warning, the status of the warning, and the detailed description information of the warning, the level of the warning, and the specific component of the warning, and so on. Tab. 3 lists the basic structure of warning logs generated by the system, and describes their attributes.

Table 3: Basic structure and description table of the warning log

Warning Log Property	Attribute Meaning	Attribute Format	Sample Example
time/time stamp	Specific time of the warning log recording	String type	12/23/2017 10:24:28
status	The occurrence or elimination of the	Enumeration type	Warning raised
description information	Describe in detail the cause of the warning occurring	String type	158.98 pages/second are being moved from virtual memory to physical RAM
warning level	The warning level	Enumeration type	Medium
component	The specific component of the	String type	Disk 1-Used MB (Used Disk 1)

The status attribute includes four major categories, namely raised (start), upgraded (gradation from one level to another), downgraded (downgradation from one level to another), and canceled (end), respectively. The warning level attribute includes *Normal*, *Low*, *Medium*, and *High*, respectively. Based on the above basic structure, the warning log can be repre- sented by the following form:

$< (Time), (Status), (Details), (Severity), (Component) >$

We can find that the warning log does not contain the clear performance fault type, however, to some extent, the *Component* attribute can reflect the warning log type. Therefore, we extract the *Component* attribute value and compare with the established performance fault type set. Finally, the performance fault type of the warning log is determined as the performance fault type in the performance fault type set. The general process is as follows:

1) Removing the *Time* attribute value of the warning log that does not meet the specified format.

2) Confirming the performance fault type for the warning log of the *Normal* level. We can judge the *Status* attribute value of the warning log directly. If the attribute value is *Normal*, we confirm that the warning log is the *Norm.* type.

3) Using the following algorithm 1 to confirm the performance fault type of the *non-Normal* level warning log.

After the above process, we can represent the warning log as follows:

$< (Time), (Status), (Severity), (Component), (classification) >$

Algorithm 1 The fault type comfirming algorithm of *non-Normal* level warning Log

Input:

L={*Time, Status, Details, Severity, Component*}, a non-Normal level warning log;
A: Performance fault type vector set, which is performance_fault_type;

Output:

T: The performance fault type corresponding to this warning log;

```
c ← Component attribute values in L
label ← extract c by using [ ^()]+
if label contains elements in A then
        T ← corresponding elements in A
else if label contains Total Percent then
T ← "total cpu utilization"
else if label contains Used Disk then
        T ← "disk usage"
end if
Delete Details attribute values in L
return T
```

Although the performance fault type in the performance fault type set can be reflected by extracting the *Component* attribute value of the warning log, it cannot fully reflect the performance fault type in the performance fault type set from the time, the CPU, the memory, the disk and the network performance characteristic. Therefore, we can only mark the performance log through the warning log, and reflect the performance fault type by determining the characteristics of the performance log.

After confirming the performance fault type corresponding to each warning log, we will annotate the performance log as follows.

The steps of the performance fault type annotation for performance log are as follows:

1) Finding the warning log pair based on the *Status*, *Level*, and *Component* of the warning log to record the start and the end of a certain performance status change.

2) Extracting the warning log pair, and confirming the start and the end time of the status change, and the performance fault type of the warning log pair. That is, the recording time of the front warning log in the warning log pair is taken as the start time of the performance status change, and the recording time of the later warning log is the end time of the performance status change, and the performance fault type is recorded as the performance fault type of the warning log pair.

3) Traversing through the performance log and annotating the performance fault type for the performance log corresponding to the warning log pair.

After processing, each performance log contains at least one performance fault type. In order to facilitate the construction of the diagnosis model later, the performance fault type

corresponding to each performance log is vectorized. In above constructed the performance fault type set, there are a total of 13 performance fault types, so we use a 13-dimensional 0,1 vector to represent each performance log. The type represented by each dimension is the type of performance fault type set from top to bottom. The value of each dimension is 0,1. For 1, it means that the performance log belongs to the type. For 0, it means that the performance log does not belong to the type. For example, if a performance log belongs to both the FCW and FDR performance types, its annotation is 0000010010000.

3.3 The performance log preprocessing method

This section introduces the preprocessing of the performance log, including processing for the missing value and the non-equalized data.

3.3.1 Missing value processing

This method uses the combination of deletion and substitution filling for the processing of missing value of the performance log. We use different methods for different types of missing values. The missing rate of the performance log can be filled by a median value method within a certain range, but when the missing rate exceeds a certain threshold or the missing data is important information, the performance log data is considered to have no value, and the value should be removed.

We consider that the performance fault type feature in the annotated performance log is the most important, followed by the missing rate of a performance log. First of all, GBDT is a supervised learning model, so it cannot be vacant for the performance fault type, otherwise it will not be effectively learned. Secondly, after the experimental analysis, when the missing rate of a performance log exceeds 26 percent, the diagnostic model has poor diagnostic performance, so if it exceeds this threshold, the performance log will be removed. Finally, if a certain performance log data meets the above two requirements, but it has the missing value, and we calculate the median value of all performance logs on the missing feature with the same performance fault type as the performance log to approximate filling. The algorithm 2 gives the specific algorithm for the substitution filling of missing value in this paper.

3.3.2 Data equalization processing

When the non-equalized performance log is used for GBDT training and the performance fault diagnosis model construction, the performance fault diagnosis model may be overfit- ting on the performance log of most performance fault types, the underfitting phenomenon occurs on the performance log of a small number of performance fault types, which greatly reduce the diagnostic capability of the performance fault diagnosis model and increases the recovery time of SaaS software performance fault. Therefore, we need to equalize the performance fault type before using the performance log.

In this paper, we use the combination of the SMOTE and the undersampling to equalize the performance log. The main idea of the method used in this paper is to select a performance fault type belonging to a few types as the partitioning criteria, and to subdivide the performance log of most performance fault types using the undersampling, so that per subset is the same number as the partitioning criteria. Moreover, performance

logs of a few performance fault types are synthesized in the SMOTE mode, so that the number of a few type set after the synthesis are the same as the partition criteria. Finally, these performance log sets or subsets are combined one by one and applied to construct the performance fault diagnosis model.

Algorithm 2 The substitution filling algorithm of missing value

Input:

D={d1, d2, d3, ... d*n*}, d*i* is the annotated performance log, and the structure is performance_log_labeled; D': copy of setD

Output:

```
    D: the processed performance logset
 1: for i←0 to the length of D do
 2:     number of vacant values Ncount←0
 3:     for j←1 to the number of features of D[i] do
 4:         if D[i][j]==null OR D[i][j]==NaN OR D[i][j]==nan then
 5:             Ncount ++
 6:             if the number of features of Ncount/ D[i]>26% then
 7:                 remove D [i] from setD
 8:                 break
 9:             end if
10:             the counting variablecount←0
11:             calculate the sum variabletotal←0
12:             for k←0 to the length of D' do
13:                 if D'[k] is the same fault type as D[i] then
14:                     if D'[k][j]==null OR D'[k][j]==NaN OR D'[k][j]==nan then
15:                         count++
16:                         total+=D'[k][j]
17:                     end if
18:                 end if
19:             end for
20:             D[i][j]←total/count
21:         end if
22:     end for
23: end for
24: return D
```

We analyze performance logs collected and find that the number of performance logs belonging to the *FDR*, *FDW*, *FCW*, *FPO*, and *FAPQL* performance fault types are relatively few, and the number of performance logs belonging to the *FCR*, *FCHR*, *FDU*, *FNT*, and *FNO* performance fault types are very few, there are more or even more performance logs belonging to the *FTCU*, *FPI*, and *Norm.* performance fault types. We consider that the total number of performance logs collected is not very large, if we choose one of a few types of performance faults as the partitioning criteria, this will not be conducive to improving the

diagnostic capability of the performance fault diagnosis model. However, if one of the three most performance fault types are selected as a partitioning criterion, it does not conform to our method thought. Therefore, we select the optimal performance fault type as the partitioning criteria among the relatively few performance fault types. Through the experimental comparison, we finally selected the *FAPQL* performance fault type as the partitioning criteria to equalize performance fault types. The performance fault type equalization process is shown in Fig. 2.

After the above process, a number of diagnostic models are generated in the end, and we use the voting method in the end. The result with the highest number of votes will be used as the diagnosis result. The specific process will be introduced in the next section.

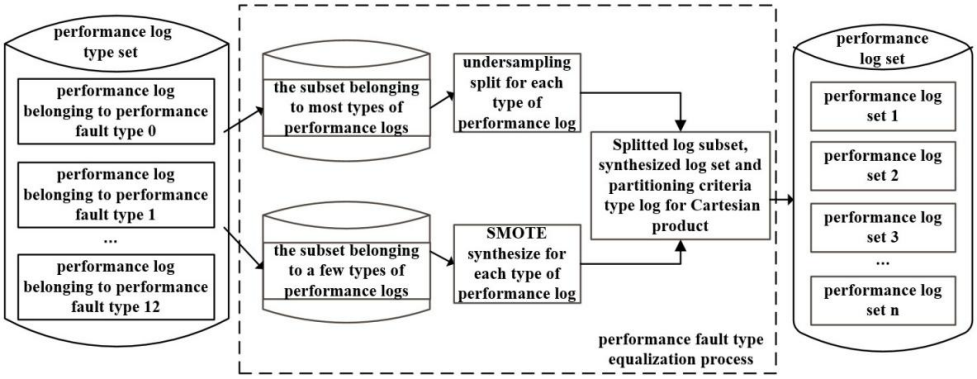

Figure 2: The performance fault type equalization process

3.4 Construction of the performance fault diagnosis model

We use the GBDT algorithm to construct the performance fault diagnostic model. First of all, GBDT is an integrated model. Through multiple iterations of the weak learning model, a strong learning model is formed to improve the diagnostic capability of the model, and its diagnostic capability can be improved by comparing to the single complex learning model. Secondly, GBDT adopts an enhanced learning model with residuals as the target in the negative gradient of the residuals in each iteration, which can reduce the complexity of construction time and enable the diagnostic model to be constructed quickly, and it also can be faster than constructing a single complex learning model. In summary, the GBDT is more in line with construction requirements of the SaaS software performance fault diagnosis model in terms of the diagnostic capability or the construction and the diagnostic speed.

Two key parts of GBDT, one is the choice of weak learning model and the other is the choice of loss function. In this paper, we use the Classification and Regression Tree (CART) as a weak learning model of GBDT. The leaf nodes of the finally established decision tree represent the regression prediction of the performance log. However, our performance log regards the performance fault type as the learning target, and it cannot apply to the regression prediction directly. Therefore, we need to logistically transform the type of performance fault (as shown in Eq. (1)), that is, we need to map the

performance fault type to the corresponding probability.

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{k=1}^{K} \exp(f_l(x))} \quad (1)$$

where K is the number of performance fault types, $f_k(x)$ is the estimated value of the diagnostic model.

In addition, since our goal of constructing the performance fault diagnosis model is to classify the performance log for multiple performance fault types rather than the regression, instead of choosing the common variances as the loss function, we choose the log likelihood function as the loss function, and we use the Eq. (2) to represent the loss function in GBDT.

$$L(F(x), y) = -\sum_{k=1}^{K} y_k \log p_k(x) \quad (2)$$

where K is the number of performance fault types (the value is 13 in this paper). If $y_k=1$, the performance log belongs to the k-th fault type.

Finally, we can form n performance fault diagnosis model. Then we will use the voting mechanism to output the final diagnosis result of the n diagnosis models, and 0, 1 strings will be converted according to the established set of performance fault types.

In addition, the method will save the obtained performance log and the diagnosis result using the performance fault diagnosis model, and periodically use the saved data to update the performance log set and the performance fault diagnosis model.

4 Experiment setup

This section first describes the source of the performance log in the experiment, and then introduces performance metrics used in this paper.

4.1 Log collection

Some large systems can generate a lot of performance logs and warning logs for us to use. In this paper, we use the Spotlight monitoring tool to remotely monitor the resource used by the comprehensive disaster reduction spatial information service application system in real time, mainly monitoring the master node system resource layer, and collecting the performance log. The application system includes a series of service components. These components visualize the risk and the loss of natural disasters in both the space and the time in a short time, and can provide the intuitive information for each disaster management work, so as to provide the SaaS service for the system. We deploy the system to the Spark cloud computing platform and deploy it on a master node and three slave nodes in a cluster deployment manner. The operating system of each node server is Ubuntu 14.04.2, and the JDK is 1.7.0, the Spark version deployed is Spark 1.3.1, Hadoop is 2.6.0, and Scala is 2.11.6. The monitoring tool can monitor the resources we need such as the CPU, the memory, the disk and the network usage.

We perform the continuous monitoring for the system about 72 hours and export monitoring results, which are warning log files and distributed performance log files for each performance indicator, respectively. Tab. 4 shows three consolidated performance

logs. Tab. 5 shows three warning log samples. Through statistics, we collected a total of 2448 performance logs and a total of 1838 warning logs, and they are stored in the csv format file.

Table 4: Performance log samples

12/23/2017 10:24:28, 67.2909, 71.1141, 76.2116, 67.7157, 13050.49, 11001.55, 10532.60, 8568.75, 613.2422, 0.8156, 0, 0, 0.8156, 0.0016, 0.8156, 0.1584, 1.9, 9.7866, 12.2333, 0.549278, 0.939533
12/23/2017 13:30:10, 45.6847, 50.4671, 49.4423, 51.492, 13180.54, 12325.07, 11138.17, 9481.67, 639.6404, 0.1574, 0, 0.1215, 4.1302, 0.0075, 4.2516, 0.7454, 1.8, 1.7315, 1.2593, 0.170327, 0.203224
12/23/2017 13:32:21, 44.2441, 50.7946, 53.3844, 49.5759, 17749.20, 16778.84, 16796.97, 11502.39, 617.6719, 4.6802, 74.8833, 0.6825, 9.8179, 0.0083, 10.5305, 0.8348, 0.8, 7.5078, 2.9251, 0.801554, 0.570077

Table 5: Warning log samples

12/23/2017 10:24:27, warning raised, Windows processor utilization is currently 70.58 percent., Low, Total CPU Usage (Average Total Percent)
12/23/2017 11:56:33, warning downgraded from Medium, 44.73 pages/second are being moved from virtual memory to physical RAM., Low, Page reads (Page ins)
12/23/2017 14:20:21, warning raised, 241.80 pages/second are being moved from virtual memory to physical RAM., High, Page reads (Page ins)

4.2 Evaluation measures

This method is used for the multi-classification task, and the performance metric of the traditional two-classification has certain limitations, because in the case of the multi-classification problem, we cannot determine that which class is a positive or negative example. Therefore, based on two-classification performance metrics, we can use Confusion Matrix, Overall Classification Accuracy (*Over_Acc*), Average Classification Accuracy (*Avg_Acc*), Average Classification Precision(*Avg_Prec*), Classification Accuracy of Each Category, Recall Rate of Each Category and F_b-score of Each Category are used as performance metrics to evaluate results of multi-classification.

Confusion matrix, also called error matrix, which is an n*n matrix. Each row represents the actual category of the sample, and each column represents the predicted value of the sample in the model.

The *Over_Acc* is the ratio of the total sample size to the total sample size of each category that is correctly classified. The *Avg_Acc* is the average of the classification accuracy for each category, and the *Avg_Prec* is the average of the precision of each category.

5 The experiment process and the result analysis

In this section, we introduce the experimental process and the experimental result in detail.

5.1 The experiment process

We use the GBDT algorithm to construct the diagnostic model. The important parameter in the interface include the step size *learning_rate*, the number of iterations *n_estimators*, the tree depth *max_depth*, the minimum number of performance logs required for internal nodes to continue to divide *min_samples_split*, and leaf nodes including the minimum performance log number *min_samples_leaf* and the self sampling ratio *subsample*. The selection of the above parameters will affect the diagnostic capability of the diagnostic model, so we use the grid search method to adjust the parameter.

Figs. 3-6 show the part important adjustment process, they mainly introduce the accuracy of different *n_estimators*, *subsample*, *min_samples_split*, *min_samples_leaf*. We finally define the parameters as follows: *learning_rate*=0.05, *n_estimators*=20, *max_depth*=3, *min_samples_split*=300, *min_samples_leaf* =95, *subsample*=0.7.

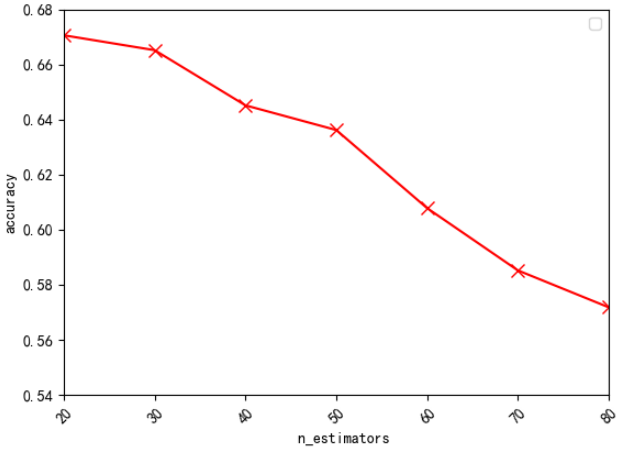

Figure 3: The accuracy of different *n_estimators*

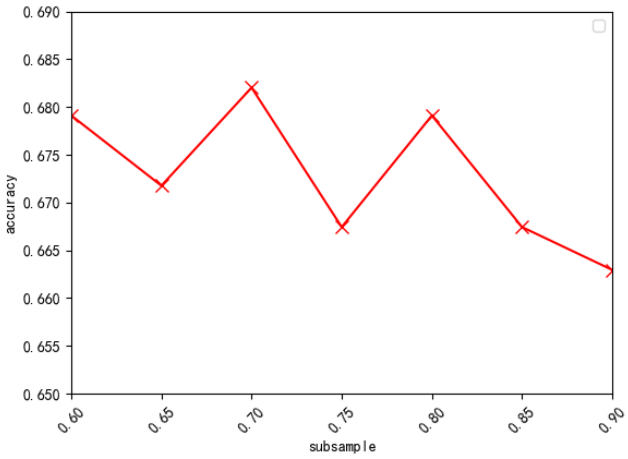

Figure 4: The accuracy of different *subsample*

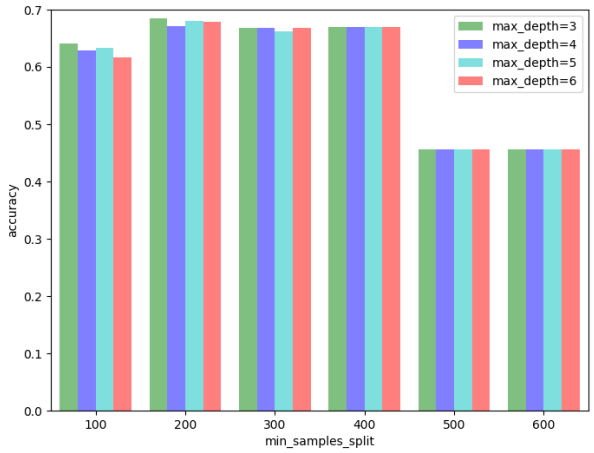

Figure 5: The accuracy of different *min_samples_split*

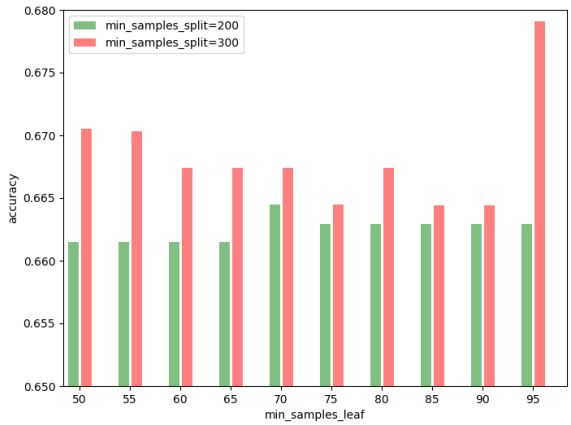

Figure 6: The accuracy of different *min_samples_leaf*

Table 6: The result of confusion matrix

Confusion Matrix		Prediction Value													Total Number of True Value
		0	1	2	3	4	5	6	7	8	9	10	11	12	
True Value	0	16	0	0	1	0	1	0	0	0	0	0	1	0	19
	1	1	15	0	0	0	0	0	1	0	1	0	0	1	19
	2	0	0	17	1	0	0	0	0	0	0	0	1	0	19
	3	0	0	0	16	0	0	1	0	0	0	0	0	1	18
	4	0	0	2	0	14	0	0	0	0	0	0	0	0	16
	5	0	0	0	0	1	15	0	0	1	0	0	0	1	18
	6	1	0	1	0	0	0	14	0	0	0	0	1	0	17
	7	0	0	0	1	0	0	0	17	0	1	0	0	1	20
	8	1	0	0	0	2	0	0	0	15	0	1	0	0	19
	9	1	0	0	0	0	0	1	0	0	16	2	0	0	20
	10	0	2	0	0	0	0	0	0	1	1	18	0	0	22
	11	0	1	0	0	0	1	0	0	0	0	0	16	0	18
	12	0	0	0	0	0	0	1	1	0	0	0	0	18	20
Total Number of Prediction Value		20	18	20	19	17	17	17	19	17	19	21	19	22	245

5.2 The experimental result analysis

We use the 10-fold cross-validation method to divide the performance log set after the missing value processing, and equalize the training set for the performance fault type, and generate 28 combined performance log sets, so we get finally constructed 28 performance fault diagnosis models for each set using the above method. Then we use the data in the test set as the input, and vote on the result of 28 performance fault diagnosis models to obtain the final diagnosis result. Moreover, we verify the diagnosis result and the learning target

in the the test set. Tab. 6 below is confusion matrix of the cross-validation result.

From the value on the diagonal of the 13*13 confusion matrix, it can be seen that the same number of predicted value and true value for each type of performance fault in the performance fault diagnosis model, that is, there are 207 performance logs in the 245 performance data test set that are completely correct for the performance fault type determination. Based on the value at each intersection point, the performance fault diagnosis model diagnoses one type of performance fault as another type of performance fault incorrectly. It can be seen intuitively that the correct or incorrect diagnosis result for each performance fault type. We can see that the model is stable for each type of prediction in this verification, and there is no underfiting or overfiting for a certain type of performance fault.

Fig. 7 shows the diagnostic capability of the model for each performance fault type from three aspects of accuracy, recall rate, and F_2 value after 10 cross-validation. At first, the method proposed in this paper is used to diagnose the performance fault when the system runs. Therefore, so we need to increase the accuracy of the model in priority, that is to say, performance faults belonging to the *Norm.* type are positive examples, *non-Norm.* types are negative examples, then the model requires higher false positive examples than false negative examples. Second, in the negative example, we need to pay attention to the recall rate of each performance fault type, that is, the sensitivity of the diagnostic model for diagnosing the *non-Norm.* performance fault type, and ensure that more type samples to be accurately diagnosed. Therefore, we need to pay attention to the accuracy of the *Norm.* type and recall rate of the *non-Norm.* type. We can adjust the b value of F_b, and b=2 will make recall rate more impactful, it is because recall rate occupies a greater weight when b>1.

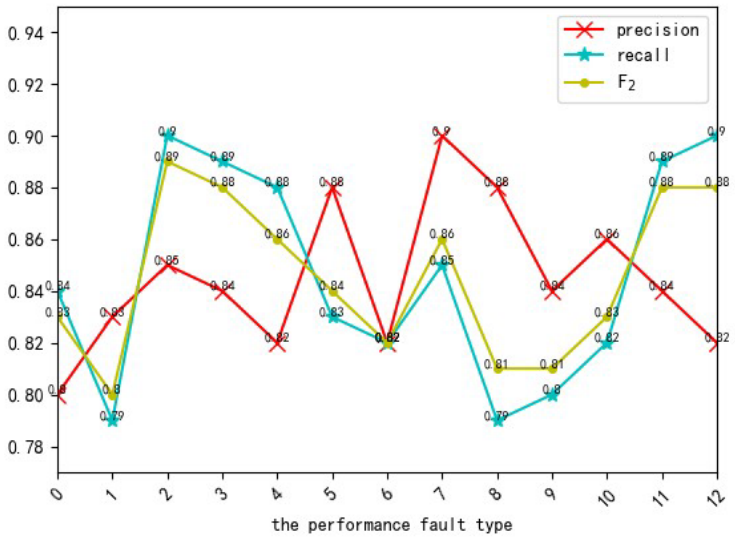

Figure 7: Results of various types of experiments

From the Fig. 7, we can see that the performance fault diagnosis model has a diagnostic capability of more than 0.75 for the performance log belonging to each performance fault type. By comparing with the performance fault type set, we know that the type 12 of the

performance fault is the *Norm.* type, in addition, the accuracy and recall rate are 0.818 and 0.9, respectively. Most other *non-Norm.* types have a recall rate of 0.83. The lowest is type 1, which is *FAPQL*, and its recall rate is 0.789. We can see that the diagnosis model has the little difference in the diagnostic capability of each performance fault type, and there is no obvious phenomenon of overfitting or underfitting for a certain type.

The following Fig. 8 shows the diagnose result that the diagnostic model is constructed without using the equalization process of the performance fault type. We can see that the diagnostic model has a large difference in diagnostic capability for each type. Performance logs belonging to most performance fault types have strong diagnostic capability, for example, type 0 (corresponding to *FTCU* type, accounting for 25 percent of the total number of performance logs) and type 12 (corresponding to Norm type, accounting for 44 percent of the total number of performance logs) are relatively high regardless of the accuracy or recall rate, basically all around 0.85 or even 0.9. However, the diagnostic performance of performance logs belonging to very few performance fault types is relatively weak, such as types 4, 6, 7, 10 and 11 (corresponding to *FCR*, *FCHR*, *FDU*, *FNI*, and *FNO*, all make up 1 percent of the performance log). Neither the accuracy nor recall rate is high, because we did not do the type equalization processing, and a few types and very a few types of performance logs used to construct diagnosis model are relatively few, which leads to the underfitting phenomenon of the diagnostic model on this type. By comparing Fig. 7 and Fig. 8, we can see that the importance and the necessity of the sample equalization in the entire performance fault diagnosis method.

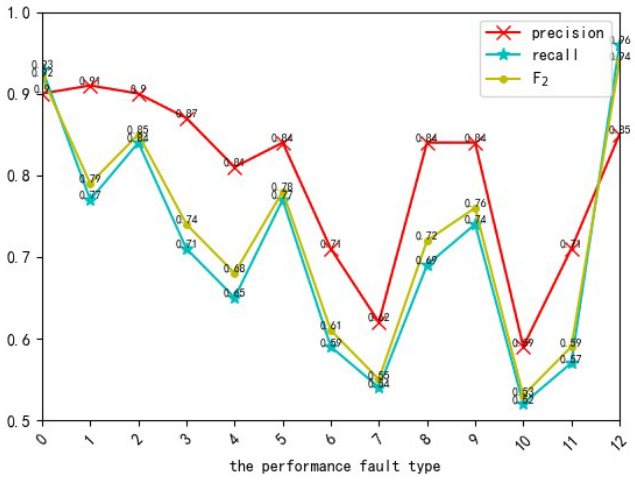

Figure 8: Results of various types of experiments

Tab. 7 shows *Avg_Acc*, *Over_Acc*, *Avg_Prec*, *Average* F_2, and *Training Time* and *Predicting Time* for the diagnosis model.

From the above table, it can be seen that *Avg_Acc*, *Over_Acc*, *Avg_Prec*, *Average* F_2 of the proposed method as a whole are all above 78 percent for the performance fault diagnosis, and it can perform the accurate and comprehensive diagnosis of various types in the case of non-equalization types, and it do not appear the underfitting phenomenon in a few types.

In terms of efficiency, the training of diagnostic model takes 2388 ms, and the predicting of diagnostic model takes 170 ms. Although the training time of the diagnostic model is longer, the training of the model will be completed before the real-time diagnosis in practical applications, so we only focus on the prediction time of the diagnostic model. Besides, we can also see that the prediction time of the diagnostic model is relatively short within the accepted range. Therefore, our performance fault diagnosis model has good results in terms of accuracy, precision, recall rate, and efficiency.

Table 7: Comprehensive experimental results

Avg_Acc	*Over_Acc*	*Avg_Prec*	*Avg_F2*	*Training Time(ms)*	*Predicting Time(ms)*
0.788	0.855	0.837	0.859	2388	170

5.3 The comparison with other classification algorithms

We compare the method of constructing the performance fault diagnosis model with the GBDT algorithm used in this paper and Random Forest (RF), Decision Tree (DT) and AdaBoost algorithm, respectively. Because RF, AdaBoost, and GBDT are all integrated algorithms, and the weak learning model used in GBDT is the DT model, so we use these three algorithms to compare with algorithms used in this paper. Our comparison experiments are performed under default parameters of each algorithm to obtain the diagnostic capability and diagnostic efficiency of the performance fault diagnosis model constructed by each method in default.

The Figs. 9-12 mainly introduces the *Avg_Acc*,*Over_Acc*, *Avg_Prec*, *Avg_F2* comparison of four algorithms. From the comparison Figs. 9-12, we can see that the performance fault diagnosis model constructed using GBDT is superior to other three algorithms in terms of *Avg_Acc* and *Over_Acc*. It is slightly lower than the RF in terms of *Avg_F2*, but higher than other two algorithms. The GBDT is much higher than other three algorithms in *Avg_Prec*. Therefore, the diagnostic capability that use the GBDT algorithm to construct the performance fault model is relatively high compared to other three algorithms.

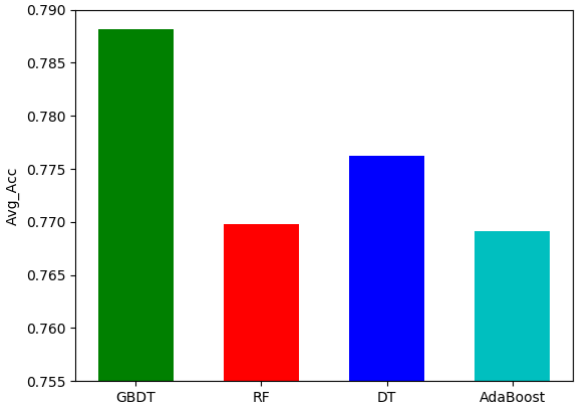

Figure 9: The *Avg_Acc* comparison of four algorithms

The Figs. 13-14 mainly introduces the *train_time*, *predict_time* comparison of four algorithms. From the comparison Figs. 13, 14, we can see that the time of constructing the performance fault diagnosis model using GBDT is higher than that of the other three, and much higher than that of DT. However, the time of that use the constructed performance fault diagnosis model to diagnose unknown type performance logs is still relatively low. Moreover, we can see that DT algorithm is the fastest, whether it is to construct a performance fault diagnosis model or to diagnose an unknown type performance log. However, in view of the Figs. 9-12, the diagnostic capability of DT is relatively low. In practical applications, we value the predicting time more than training time, because the construction of the model is a preparatory work, even if later updating to the model can also be processed in parallel without delaying the diagnosis of the system performance status. As long as the predicting time is within our tolerance time, it can meet SaaS software performance maintenance requirements. Therefore, compared with other methods, the proposed method is feasible in practical applications.

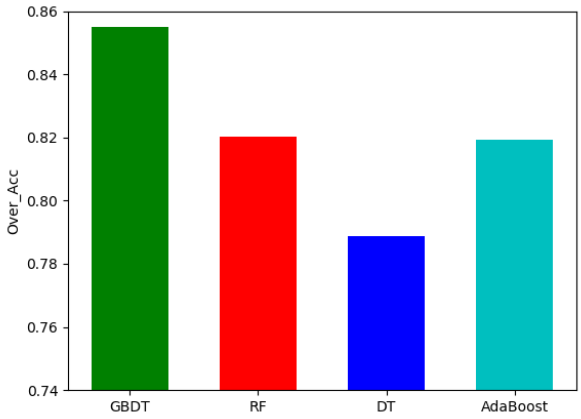

Figure 10: The *Over_Acc* comparison of four algorithms

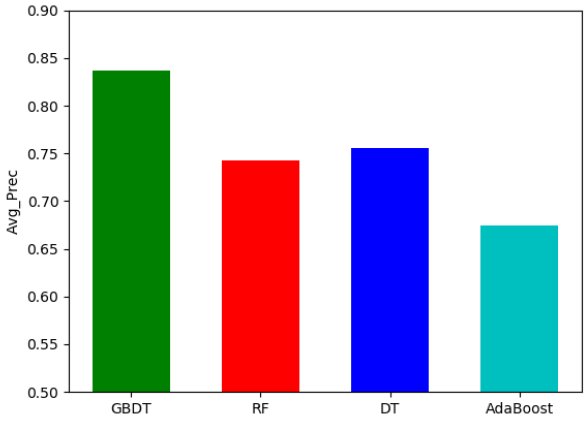

Figure 11: The *Avg_Prec* comparison of four algorithms

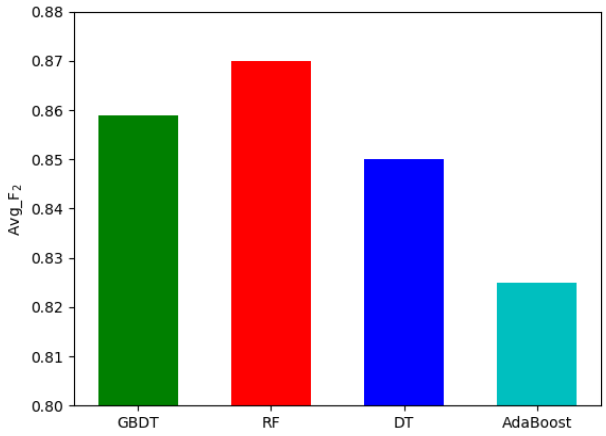

Figure 12: The Avg_F_2 comparison of four algorithms

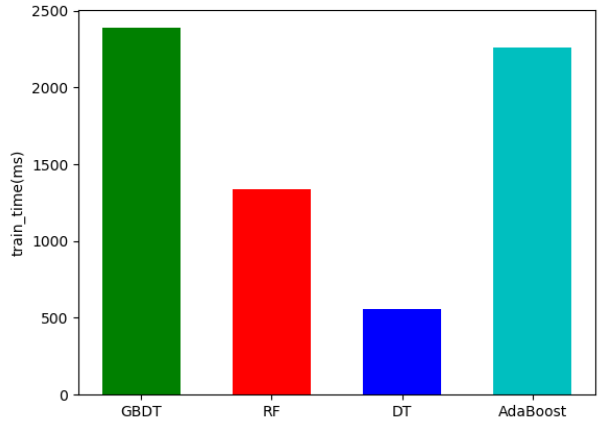

Figure 13: The *train_time* comparison of four algorithms

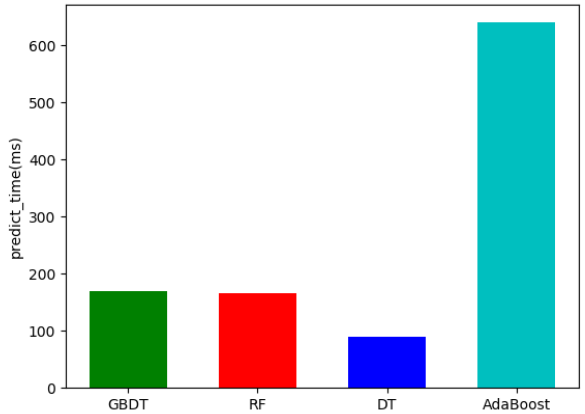

Figure 14: The *predict_time* comparison of four algorithms

Combining the above comparison Figs. 13, 14, we can see that even if GBDT is

relatively low for DT in efficiency, but from the perspective of the comprehensive diagnostic performance of the model, and we can think that choosing a GBDT for the performance model construction is more reasonable than other algorithms.

6 Conclusion and future work

In this paper, we study the SaaS software performance fault diagnosis method through the requirement of the SaaS software performance maintenance. First, by analyzing the resource layer KPI when the SaaS software runs, the performance log features are determined. Based on the analysis result of the warning log, the performance log fault type is annotated. Then we deal with the incomplete performance log and the non-equalization problem of the type by using the mean filling for the same type and the combination of the SMOTE and undersampling methods. We apply the GBDT machine learning method to the performance fault diagnosis. Finally, the efficiency and accuracy of the proposed method are verified with a disaster reduction system deployed on the cloud platform.

We will consider future research work from the following three aspects: Firstly, we will apply the performance fault diagnosis method to other SaaS software systems. Secondly, we will verify that whether the tool chain can help the SaaS operators to diagnose the performance fault better and further adjust the tool chain by collecting feedback from operators. Thirdly, we will expand the scope of these performance features, which can fully satisfy the requirement of performance diagnosis for new hardwares, e.g., GPU, SDN, smart net card, and meet the specific requirement of other emerging applications, e.g., graph computing, deep learning.

Acknowledgement: This work is supported in part by the National Science Foundation of China (61672392, 61373038), and in part by the National Key Research and Development Program of China (No. 2016YFC1202204).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Bezemer, C. P.; Zaidman, A. (2014): Performance optimization of deployed software-as-a-service applications. *Journal of Systems and Software*, vol. 87, no. 1, pp. 87-103.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P. (2011): Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321-357.

Ding, R.; Fu, Q.; Lou, J. G.; Lin, Q.; Zhang, D. et al. (2012): Healing online service systems via mining historical issue repositories. *IEEE/ACM International Conference on Automated Software Engineering*, pp. 318-321.

Duan, S.; Babu, S. (2008): Guided problem diagnosis through active learning. *IEEE International Conference on Autonomic Computing*, pp. 45-54.

Gashler, M. S.; Smith, M. R.; Morris, R.; Martinez, T. (2016): Missing value imputation with unsupervised backpropagation. *Computational Intelligence*, vol. 32, no. 2,

pp. 196-215.

Gill, P.; Jain, N.; Nagappan, N. (2011): Understanding network failures in data centers: measurement, analysis, and implications. *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350-361.

He, H.; Garcia, E. A. (2008): Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, vol. 21, no. 9, pp. 1263-1284.

Jin, W.; Liu, T.; Zheng, Q.; Cui, D.; Cai, Y. (2018): Functionality-oriented microservice extraction based on execution trace clustering. *IEEE International Conference on Web Services*, pp. 211-218.

Ju, J.; Wang, Y.; Fu, J.; Wu, J.; Lin, Z. (2010): Research on key technology in SaaS. *International Conference on Intelligent Computing and Cognitive Informatics*, pp. 384-387.

Lim, M. H.; Lou, J. G.; Zhang, H.; Fu, Q.; Teoh, A. B. J. et al. (2014): Identifying recurrent and unknown performance issues. *IEEE International Conference on Data Mining*, pp. 320-329.

Liu, X.; Wu, J.; Zhou, Z. (2006): A cascade-based classification method for class-imbalanced data. *Journal of Nanjing University (Natural Sciences Edition)*, vol. 42, no. 2, pp. 148-155.

Malik, H.; Hemmati, H.; Hassan, A. E. (2013): Automatic detection of performance deviations in the load testing of large scale systems. *International Conference on Software Engineering*, pp. 1012-1021.

Mavridis, I.; Karatza, H. (2017): Performance evaluation of cloud-based log file analysis with apache Hadoop and apache spark. *Journal of Systems and Software*, vol. 125, no. 3, pp. 133-151.

Nagaraj, K.; Killian, C.; Neville, J. (2012): Structured comparative analysis of systems logs to diagnose performance problems. *USENIX Conference on Networked Systems Design and Implementation*, pp. 26-39.

Rahman, G.; Islam, Z. (2011): A decision tree-based missing value imputation technique for data pre-processing. *Australasian Data Mining Conference*, vol. 121, pp. 41-50.

Schroeder, B.; Gibson, G. (2010): A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337-350.

Syer, M. D.; Jiang, Z. M.; Nagappan, M.; Hassan, A. E.; Nasser, M. et al. (2013): Leveraging performance counters and execution logs to diagnose memory-related performance issues. *IEEE International Conference on Software Maintenance*, pp. 110-119.

Tsai, W.; Bai, X.; Huang, Y. (2014): Software-as-a-service (SaaS): perspectives and challenges. *Science China Information Sciences*, vol. 57, no. 5, pp. 1-15.

Wang, T.; Wei, J.; Zhang, W.; Zhong, H.; Huang, T. (2014): Workload-aware anomaly detection for web applications. *Journal of Systems and Software*, vol. 89, no. 3, pp. 19-32.

Wang, T.; Zhang, W.; Ye, C.; Wei, J.; Zhong, H. et al. (2016): Fd4c: automatic fault diagnosis framework for web applications in cloud computing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 1, pp. 61-75.

Wen, X.; Zhang, X.; Zhu, Y. (2015): Design of fault detection observer based on hyper basis function. *Tsinghua Science and Technology*, vol. 20, no. 2, pp. 200-204.

Wu, L.; Garg, S. K.; Versteeg, S.; Buyya, R. (2014): Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments. *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 465-485.

Zawoad, S.; Dutta, A. K.; Hasan, R. (2013): Seclaas: secure logging-as-a-service for cloud forensics. *ACM SIGSAC Symposium on Information, Computer and Communications Security*, pp. 219-230.

Zou, D. Q.; Qin, H.; Jin, H. (2016): Uilog: improving log-based fault diagnosis by log analysis. *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 1038-1052.