Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Container cascade fault detection based on spatial–temporal correlation in cloud environment

Ningjiang Chen[1,2,3], Qingwei Zhong[1], Yifei Liu[1*], Weitao Liu[1], Lin Bai[1,2,3] and Liangqing Hu[1,2]

Abstract

Containers are light, numerous, and interdependent, which are prone to cascading fault, increasing the probability of fault and the difficulty of detection. Existing detection methods are usually based on a cascade fault model with traditional association analysis. The tradition model lacks consideration of the fault cascade history dimension and container space correlation dimension which results in a lower detection effect. And the imbalance of fault data in the cloud environment to the detection method to bring interference. Instead, this paper proposes a cascade fault detection method based on spatial–temporal correlation in cloud environment. First, the container cascade fault relationship model is constructed by extracting the spatial–temporal correlation from historical container faults. Second, based on dynamic feedback data sampling combined with ensemble learning, a container fault model learning method is designed to solve the imbalance of fault data. Then, a real-time container cascade fault detection mechanism for container cascade failure is proposed. The experimental results show that compared with the existing fault detection methods, the proposed method can effectively improve the detection accuracy, recall rate, and F_1 value.

Keywords Containers, Cloud computing, Cascade fault, Fault detection, Association model, Ensemble learning

Introduction

With the popularization of cloud computing, cloud systems are applied in many fields. To improve the reliability of cloud systems, researchers contribute a lot in data dissemination [1], data management [2], and security [3]. Still, the fault greatly affected the reliability of cloud systems. In container-based cloud systems, faults generally occur on a large scale and in a time sequence, which may be the result of cascade faults [4]. Containers are light, numerous, and interdependent, which are prone to cascading fault, increasing the probability of fault and the difficulty of detection [5]. Fault detection methods used in traditional cloud platforms will result in degraded detection performance when applied to the containerized cloud platform. A single cascade fault, if not detected and processed in time, can affect most services in the cloud platform, resulting in huge losses. For example, the Google Cloud outage brought down Google's cloud services in multiple regions, including Dataflow, Big Query, Dialog Flow, Kubernetes Engine, Cloud Firestore, App Engine, and Cloud Console. The cascading fault model based on traditional correlation analysis does not well consider the spatial dimensional information such as application, service, node, and fault domain that

*Correspondence:
Yifei Liu
liu19990118a@163.com
[1] School of Computer and Electronic Information, Guangxi University, Nanning, Guangxi 530004, China
[2] Guangxi Intelligent Digital Services Research Center of Engineering Technology, Nanning, Guangxi 530004, China
[3] Key Laboratory of Parallel, Distributed and Intelligent Computing (Guangxi University), Education Department of Guangxi Zhuang Autonomous Region, Nanning, Guangxi 530004, China

containers are distributed in the container-based cloud platform, as well as the time dimensional information of the cascade fault spread over time, which makes it impossible to calculate the fault propagation probability of containers. Besides, fault detection methods are often based on historical data when they are running on cloud platforms, but few studies have focused on the problem of imbalanced data in a cloud environment, or have only used simple data preprocessing methods that cannot effectively address the effects of imbalanced data sample classification. Therefore, the reliability of containerized cloud platforms is facing the serious challenges.

To address these problems, we propose a Cascade Fault Detection method based on Spatial–Temporal Correlation named CFD-STC. The main contributions of this work are as follows:

(1) A cascade fault relation model based on the spatial–temporal correlation is proposed. The model can reasonably depict the cascade relations between containers and accurately calculate the probability of failure propagation.
(2) An ensemble learning method for imbalanced data is proposed. Dynamic feedback sampling and ensemble learning is adopted to improve the accuracy and recall rate of fault detection.
(3) Based on cascade fault relation model and ensemble learning, we propose a method for real-time detecting container cascade faults.

The rest of this paper is organized as follows. Background section introduces the background. The process of container cascade fault detection section describes the solution proposed in this paper. Experiments and evaluation section verifies the validity of the methods proposed in this paper through experiments. The conclusions and future work are presented in Conclusion section.

Background

Related work

Traditional fault detection methods are classified into those based on statistical models, mathematical models [6, 7], data mining [8], and machine learning [9, 10]. These methods depend on historical fault data and usually require prior knowledge of the fault and its performance to formulate a series of rules to define the fault of the system [11] or to analyze the association between log information before and after the fault to detect the fault [12]. LCS [13] is a typical fault cascade and propagation detection method based on historical fault data in the cloud environment. LCS with VMM [14] adopts a semisupervised method to improve the detection accuracy of LCS. Generally, the methods based on historical fault data analyze the occurrence of each fault singly and have high accuracy and timeliness for fault detection. However, it is difficult to accurately detect fault containers without considering the cascade and propagation relationships between faults.

The identification of fault cascades and propagation is an important step for effective fault recovery, and this topic is currently receiving extensive attention. Bui D et al. [15] constructed the propagation path through fault injection and verified the effectiveness of the method on the OpenStack cloud computing platform. Wang H et al. [16] proposed a cloud platform cascade fault recovery system for cascade faults in cloud data centers, but it is mainly for studying cascade faults among physical machines in the cloud platform. Wang T et al. [17] established a fault location model by analyzing the context of fault propagation in the cloud environment. Yu S et al. [18] obtained and analysed system logs to detect faults by injecting faults into the system, but it mainly aimed at detecting grey faults in the cloud storage system. Toka L [19] trained AI models for root-cause analysis of cascade faults in the cloud by time series clustering. In [19], it considered the time dimensional information of the cascade fault but ignored the spatial dimensional information. However, the existing works are usually limited to analyzing the paths of historical fault propagation and do not quantitatively evaluate the probability of fault propagation, while container faults are usually closely related to their locations in space and time, and the probability of fault propagation is different for different containers.

Since the fault time of the cloud platform is short compared to the normal operation time, the data generated by the cloud platform are imbalanced, and the percentage of fault data will be far less than that of normal data. This affects the detection method based on the operation data of the cloud platform. Singh P [20] points out that in the software fault dataset, the imbalance of the normal data and fault data adversely affects the fault detection algorithm. Attempts to tackle this imbalance are usually divided into two categories: oversampling [21] and undersampling [22]. The undersampling method solves the imbalance problem of samples by abandoning a part of the majority class samples [14]. However, as the data imbalance increases, too many majority class samples are lost, and the model cannot fit the sample features. This leads to a decrease in the recall rate of detection. The

oversampling method balances the original dataset by copying or generating sample features [16]. Because there are too many characteristics similar to the original sample, it is easy for the model to overfit the training sets due to the overlap of data samples. This results in lower accuracy of real-time fault detection. Ensemble learning trains multiple base models in parallel by constructing multiple balanced training sets and integrates the multiple base models into a final model. As a result, fault detection is generally better than any single base model and performs better on imbalanced fault data [23–25]. However, the traditional ensemble learning method adopts a random sampling method for fault data samples when constructing the training dataset and pays less attention to the sample distribution. This leads to a poor model training effect. At the same time, during model integration, only one round of base model training and integration is applied and this is not enough to obtain an optimal model.

In summary, the existing cascade fault detection methods face the following two challenges. First, the traditional cascade fault model has insufficient ability to characterize the fault cascade among containers. Usually, it only describes the propagation path of faults and fails to quantify the probability of fault propagation. Second, the existing methods do not focus on the imbalance of historical fault data and thus cannot guarantee both accuracy and recall rate as the imbalance rate increases.

The idea of solution

Our proposed detection method is shown in Fig. 1. It includes three key parts: cascade fault relation model construction, ensemble learning model training method, cascade fault detection method combined with cascade fault model and ensemble learning.

(1) **Cascade fault relation model construction.** Through the container fault cascade history, frequently associated fault container instances are mined to analyze the container fault propagation path qualitatively. Then based on the time series

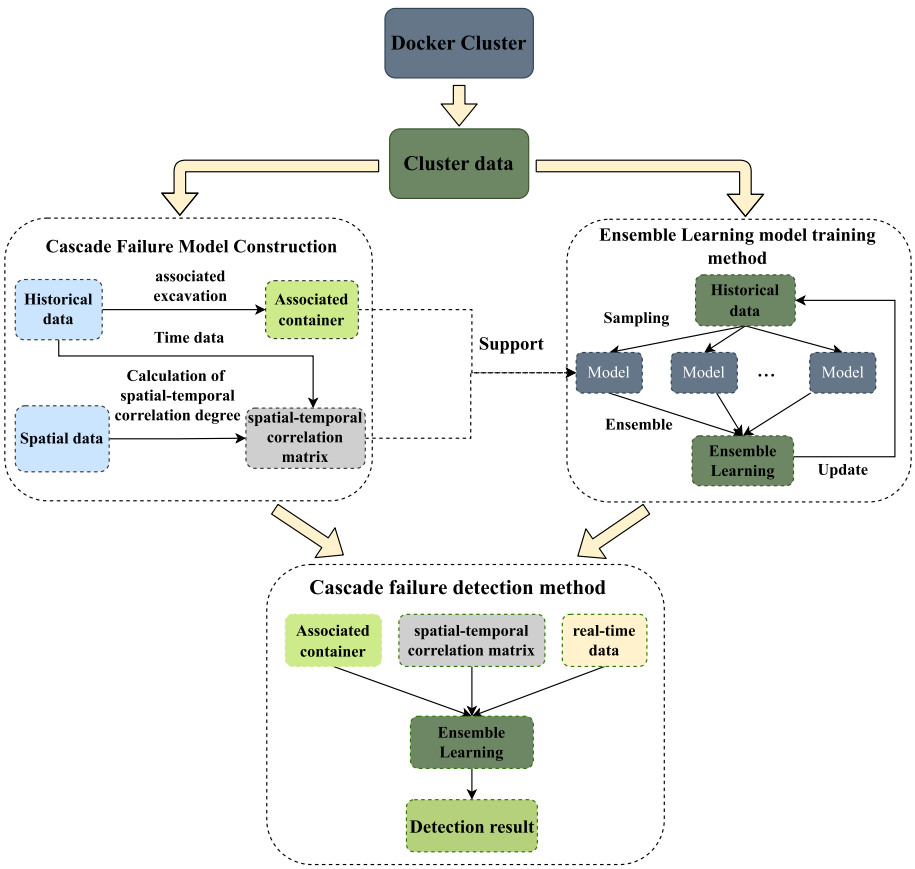

Fig. 1 The framework of cascade fault detection based on spatial–temporal correlation

dimension and spatial structure dimension of historical fault data, the time correlation and spatial correlation of container fault propagation are calculated to quantitatively characterize the association strength and fault propagation probability between the fault containers. Based on the combination of qualitative and quantitative methods, the cascade fault relation model based on container spatial–temporal correlation is established.

(2) **Ensemble learning model training method.** To avoid the error caused by the imbalanced historical dataset, multiple training sets with balanced categories are constructed by the sampling method, and multiple models are trained separately. Then, a weighted integration model is constructed by ensemble learning. By testing the effect of integrated model, the sample distribution of the imbalanced historical dataset is updated to iteratively improve the detection effect and model performance.

(3) **Cascade fault detection method combined with cascade fault relation model and ensemble learning.** Based on the above construction of cascade fault relation model and model training method, we design a container cascade fault detection method, which collects the metric data produced by cloud containers in real time and inputs it into the integrated model. Through model prediction, it is possible to judge the probability of container fault.

The process of container cascade fault detection

Construction of container cascade fault relation model

Finding the associations among container faults is an important step in constructing the container cascade fault relational model. As shown in Fig. 2, the relational model for spatial–temporal association is constructed in two stages. The model is trained by historical fault data and spatial–temporal correlation

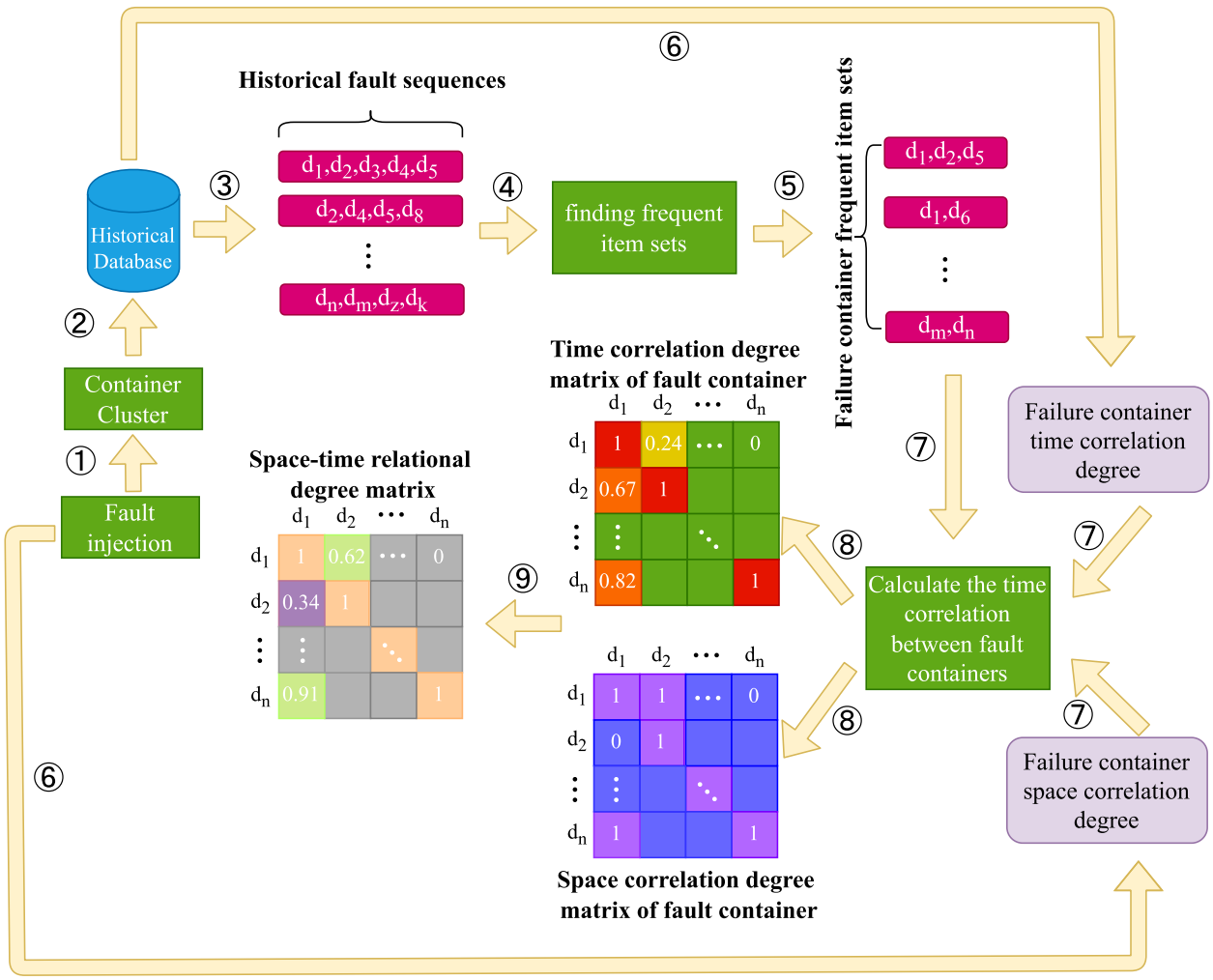

Fig. 2 Fault correlation model construction process

data to more accurately characterize the propagation path and propagation probability of container cascade faults.

Finding frequently associated fault container instance sequences

Historical data that are classified as fault data are extracted. Through correlation mining, the frequently associated fault container instance sequences in the historical fault data are found. This association of historical fault containers means that faults in the future operation stage will also be frequently associated, which represents the propagation path of container cascade faults. Moreover, all the introduced notations and their definitions that followed have been summarized in Table 1 for easy reference.

First, historical cascade fault data and the spatial structure (service, node and fault domain) of the container cluster must be obtained. The sequence of historical fault container instances and frequently associated fault container instances are obtained from fault data, and the set of fault container sequences is constructed. Then, the frequently associated fault container instances are found by association rules. This paper obtains four attributes for each data sample in the dataset through fault injection, and each data sample in the dataset acquired through fault injection has the four attributes of {*timestamp*, *ID*, *PerformanceData*, *State*}. Here *ID* is the unique identifier of the container instance, *PerformanceData* is the CPU usage, memory usage and other performance data in the current state of the container instance, *State* is the runtime state of containers (faulty or normal), and *Timestamp* is the interval of fault injection.

Table 1 Types and methods of fault injection

Notation	Definition
D	Historical Sequence
C	Association Sequence
L	Association Set
$S_T(i,j)$	The spatial correlation strength of the cascade fault
$S_S(i,j)$	The temporal correlation strength of the cascade fault
$S_O(i,j)$	The spatial–temporal correlation strength of the cascade fault
x_T, x_S	weights of temporal correlation and spatial correlation degree
$S = \{S_O(1,1), \ldots, S_O(n,n)\}$	Spatial–temporal correlation degree matrix
M	base model
δ	integrated model threshold
$F(x)$	Final integrated model
x_i	container historical performance data

Definition 1 Sequence of historical fault container instances (Historical Sequence). $D = \langle containerID_1, \cdots, containerID_n \rangle$,$containerID_i$ is the unique identifier of the container instance. It is considered to be sequential, i.e., the fault containers in Historical Sequence are ordered in sequence.

The expression of fault container association is like $D_i \rightarrow D_j$, and $D_i \cap D_j = \varnothing$. The strength of an association rule can be measured by its support and confidence. The support degree s and confidence degree c are calculated by Formulas (1) and (2), respectively.

$$s(D_i \rightarrow D_j) = \frac{\sigma(D_i \cup D_j)}{N} \tag{1}$$

$$c(D_i \rightarrow D_j) = \frac{\sigma(D_i \cup D_j)}{\sigma(D_i)} \tag{2}$$

$s(D_i \rightarrow D_j)$ is expressed as the proportion of the times that D_i and D_j break down at the same time. $c(D_i \rightarrow D_j)$ represents the proportion of occurrences of D_j in the sequence containing D_i when D_i appears, $\sigma(D_i)$ represents the number of occurrences of D_i in the Historical Sequence sets, and N is the total number of sequences in the Historical Sequence set.

Definition 2 Sequence of frequently failed associated container instances (Association Sequence). $C = \langle containerID_1, \cdots, containerID_n \rangle$, C equals is D or its subsequence with support and confidence greater than a custom minimum.

Definition 3 Sequence set of frequently fault associated container instances(Association Set). $L = \{C_1, C_2, \cdots, C_n\}$, C_i represents one of these related sequences C.

The mining process of the Association Sequence is shown in Fig. 3. The item with length 1 is obtained through the Historical Sequence set, and then the association set L_1 with no less than the minimum support is screened. Then, the Association Sequence L_1 is used as the basic fault container Association Sequence pattern and the known Association Set L_k is used to generate the candidate Association Sequence C_{k+1}. Next the association set L_{k+1} with length $K+1$ is calculated according to the minimum support. The above process is iterated until the next level candidate Association Sequence cannot be generated and it ends up with Association Set L. Based on the above mining process, the Association Set representing the fault propagation path can be obtained. However, more container fault cascade information is needed. Therefore, the probability of container cascade fault propagation will be calculated.

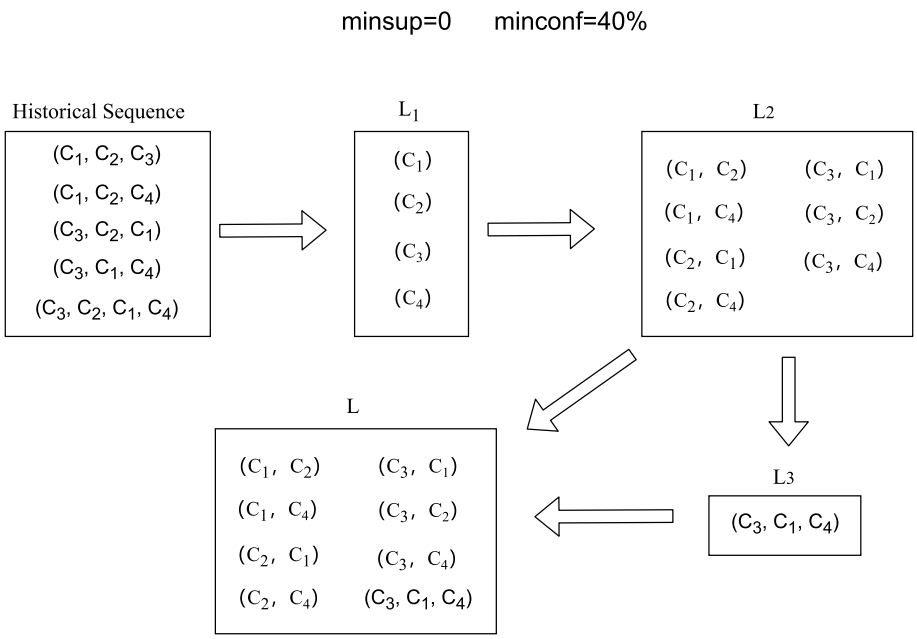

Fig. 3 Association Set mining process

Calculation of the spatial–temporal correlation degree of container faults

The propagation of container faults has a strong spatial–temporal correlation, in contrast to traditional software. It is the key to determining the cascade association of container faults and calculating the probability of fault propagation. Therefore, the spatial–temporal correlation strength of the cascade faults is described by the time and space data of the fault container. The relevant definitions are as follows.

Definition 4 The spatial correlation strength of the cascade fault $S_T(i,j)$ is defined as follows:

$$S_T(i,j) = \frac{\sum_{k=0}^{N'-1}\left(1-\left(\frac{T_{k,j}-T_{k,i}}{\Delta T}\right)\right)}{N'}, 0 \le i, 0 \le j \tag{3}$$

where $T_{k,j}$ and $T_{k,i}$ represent the time when fault containers j and i are faulty in fault container item set k that contains a subset of fault container sequences (i,j). N' is the total number of containers in k. $S_T(i,j)$ represents the average propagation time from the historical fault container i to container j after normalization, $S_T(i,j) \in [0,1]$. The higher $S_T(i,j)$ is, the shorter the fault propagation time is and the stronger the time association is.

The multi-layer spatial layers in the cloud environment, such as clusters, fault domains, server nodes, services, and containers, are abstracted into the layers of the cluster spatial tree. The Euclidean distance of the service, node and fault domains of two containers is used to represent the spatial distance between two containers, and the reciprocal of the Euclidean distance is used to represent the spatial correlation strength between two containers. The cluster spatial tree is shown in Fig. 4.

Definition 5 The temporal correlation strength of the cascade fault $S_s(i,j)$ is defined as follows:

$$S_s(i,j) = \begin{cases} \frac{\sqrt{\frac{1}{(Ser_j - Ser_i)^2 + (Node_j - Node_i)^2 + (Fail_j - Fail_i)^2}}}{Z_t} \\ 1 Ser_j = Ser_i \text{ and } Node_j = Node_i \text{ and } Fail_j = Fail_i \end{cases} \tag{4}$$

where Ser_i represents the service to which container i belongs, $Node_i$ represents the node to which container i belongs, and $Fail_i$ represents the fault domain to which container i belongs. Z_t is the normalization factor that makes $S_s(i,j) \in [0,1)$. The larger $S_s(i,j)$ is, the greater the spatial correlation between container i and container j.

Definition 6 The spatial–temporal correlation strength of the cascade fault $S_O(i,j)$ is defined as follows:

$$S_O(i,j) = x_T S_T(i,j) + x_s S_s(i,j) x_T + x_s = 1 \tag{5}$$

The spatial–temporal correlation degree of faults among containers is the weighted addition of the temporal correlation degree and spatial correlation degree. x_T and x_S represent the weights of the custom temporal correlation degree and spatial correlation degree, respectively. Because $S_T(i,j) \in [0,1]$, $S_s(i,j) \in [0,1]$, and $x_T + x_s = 1$, the value of $S_O(i,j)$ is between 0 and 1.

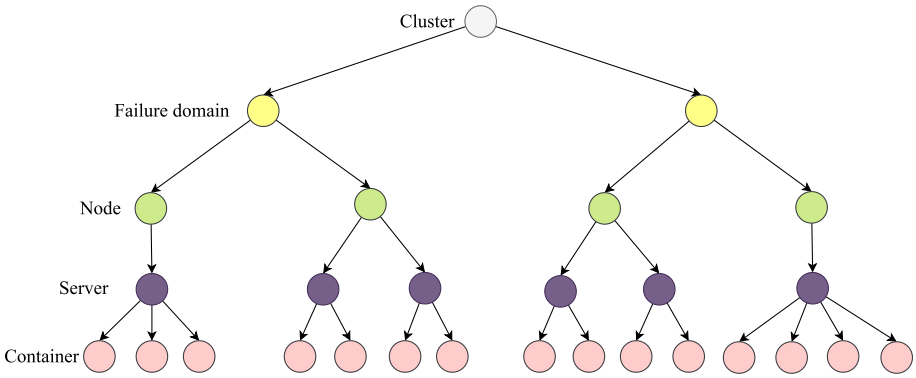

Fig. 4 cluster spatial tree

$S_O(i,j)$ represents the strength of the spatial–temporal correlation degree of faults among containers.

Definition 7 Spatial–temporal correlation degree matrix $S = \{S_O(1,1), S_O(0,1), \cdots, S_O(n,n)\}$, in which the value of each element is the fault spatial–temporal correlation degree $S_O(i,j)$ of container i and container j.

Formula (5) is used to calculate the spatial–temporal correlation strength of faults among containers in the correlation item set. This is because it is of practical significance to calculate the spatial–temporal association degree of faults only for containers with frequent fault association. The calculation of the spatial–temporal correlation degree of faults among containers is shown in Algorithm 1. The flowchart of Algorithm 1 is shown in Fig. 5.

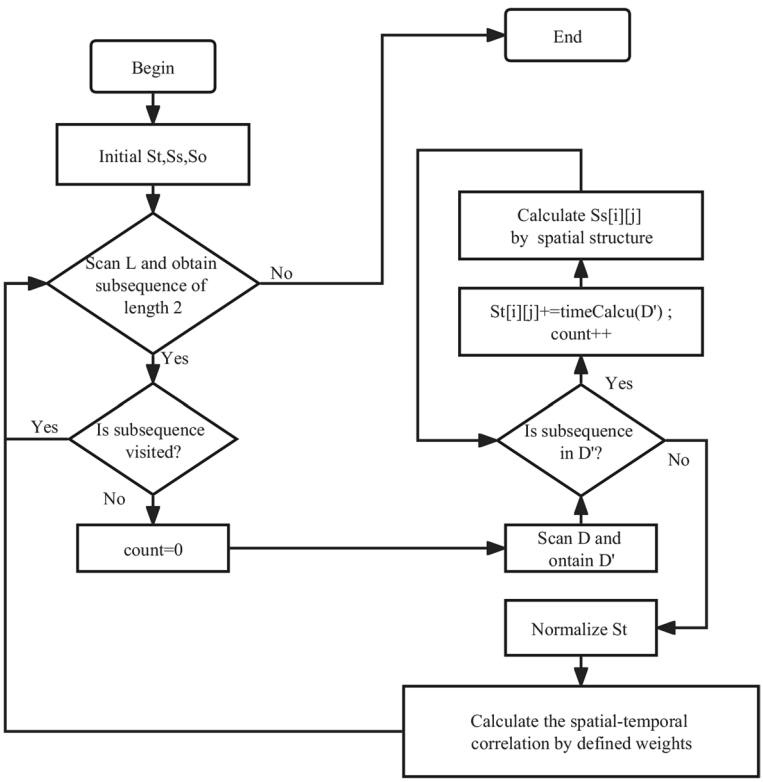

Fig. 5 The flowchart of Algorithm 1

Input: L; // Association Set;
D; // Historical Sequence,
ΔT ; //Fault timestamp interval;
x_s; // Weight of temporal correlation degree;
x_T; // Weight of spatial correlation degree;
Output: S; // the spatial-temporal correlation degree matrix

```
Begin
S_T = [], S_s = [], S_O = [];
for L' in L do
    if sequences (Container_i, Container_j) in L'
    and S_T(Container_i, Container_j) == 0 then
        count = 0;
        for D' in D do
            if sequences (Container_i, Container_j) in D' then
                S_T(Container_i, Container_j) += timeCalcu(D');
                count ++;
                if S_s(Container_i, Container_j) == 0 then
                    if Ser_i == Ser_j and Fail_i == Fail_j then
                        S_s(Container_i, Container_j) = 1;
                    else
                        S_s(Container_i, Container_j) = spaceCalcu(D');
                    end if
                end if
            end if
        end for
        S_T(Container_i, Container_j) = S_T(Container_i, Container_j)/count;
        S_O(Container_i, Container_j) = x_T * S_T(Container_i, Container_j +
          x_s * S_s(Container_i, Container_j);
    end if
end for
```

Algorithm 1. Calculate the Spatial-temporal Correlation Degree of Faults Among Containers

Association Set and Historical Sequence are traversed through two loop statements to calculate the spatial–temporal correlation strength, so the time complexity of the Algorithm 1 is $O(n^2)$, where n is the number of different container instances in the correlation item set. Compared with the number of containers in the cloud platform, the container instances left in the association mining stage account for a small proportion. Hence the time complexity is acceptable.

Model learning method for imbalanced historical fault data

The traditional data imbalance processing methods are coarse and cannot guarantee the accuracy and recall rate at the same time. Hence, a model learning optimization method is proposed based on dynamic feedback sampling combined with ensemble learning. The flow of the method is shown in Fig. 6. It consists of the following stages.

(1) **Data preparation**. Before training and optimization, the performance data and state data of containers are obtained by fault injection. The performance data, container fault relational model and container instance state, which are strongly associated with the container, are combined as the input vector of the training model, the current state of the container is used as the label.
(2) **Division.** The historical dataset is divided into a cascade fault dataset D_1 and a normal dataset D_2.
(3) **Sampling.** Normal data are sampled by random sampling only for the first time, and the subsequent sampling is performed according to the sample distribution feedback from the model. After several put-back extractions (depending on the number of basic models), normal data are combined with cascade fault data of containers to form a balanced training set of multiple normal classes and fault classes.

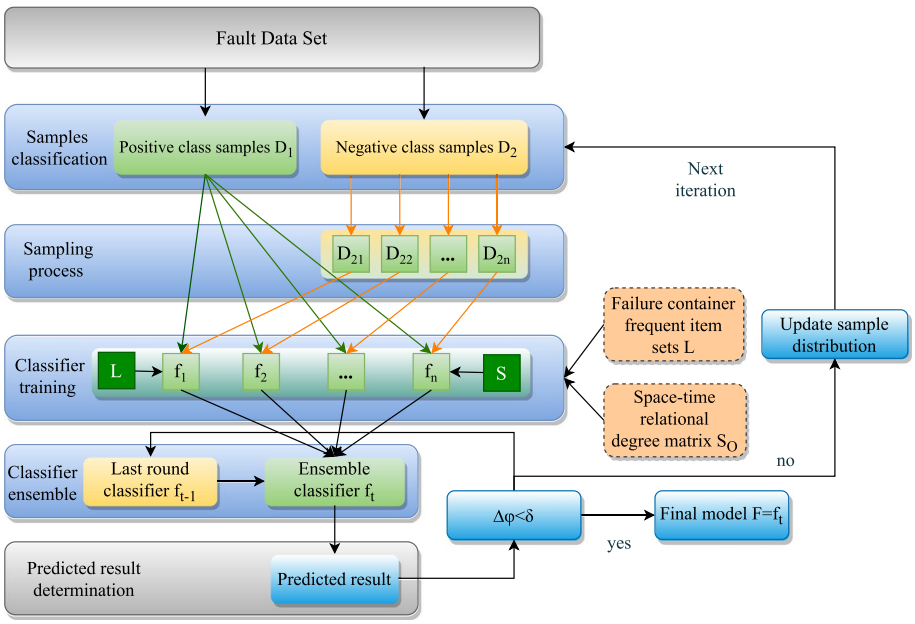

Fig. 6 Model learning method flow for imbalanced fault history data

(4) **Basic model learning.** The LSTM (long short-term memory) model [26] is used as the basic model because it can adapt well to the time sequence and backward dependence (fault propagation direction) of cluster historical datasets. Using a balanced training set to train the LSTM-based model, the basic model can accurately study the cascade relationship between fault containers and calculate the detection effect α_i and weight value w_i of each basic model in the historical dataset.

(5) **Model integration and testing.** Multiple basic models are weighted and combined with the previous round of integration model to obtain the integration model using the model weight w_i obtained in the learning stage. Then the detection effect enhancement value $\Delta\varphi$ of the integrated model is calculated. If $\Delta\varphi > \delta$, the sample distribution is updated so that the wrong samples detected by the integrated model receive more attention. The above steps are iteratively performed until $\Delta\varphi < \delta$, and the model is proved to have converged. The output of this round of integration model is the final integration model.

The training optimization process is shown in Algorithm 2. Since the outer while loop of the algorithm 2 cannot determine the number of loops, we only consider the time complexity of the algorithm inside the while loop. The time complexity of the Algorithm 2 is O(n), where n is the number of basic model. The flowchart of Algorithm 2 is shown in Fig. 7.

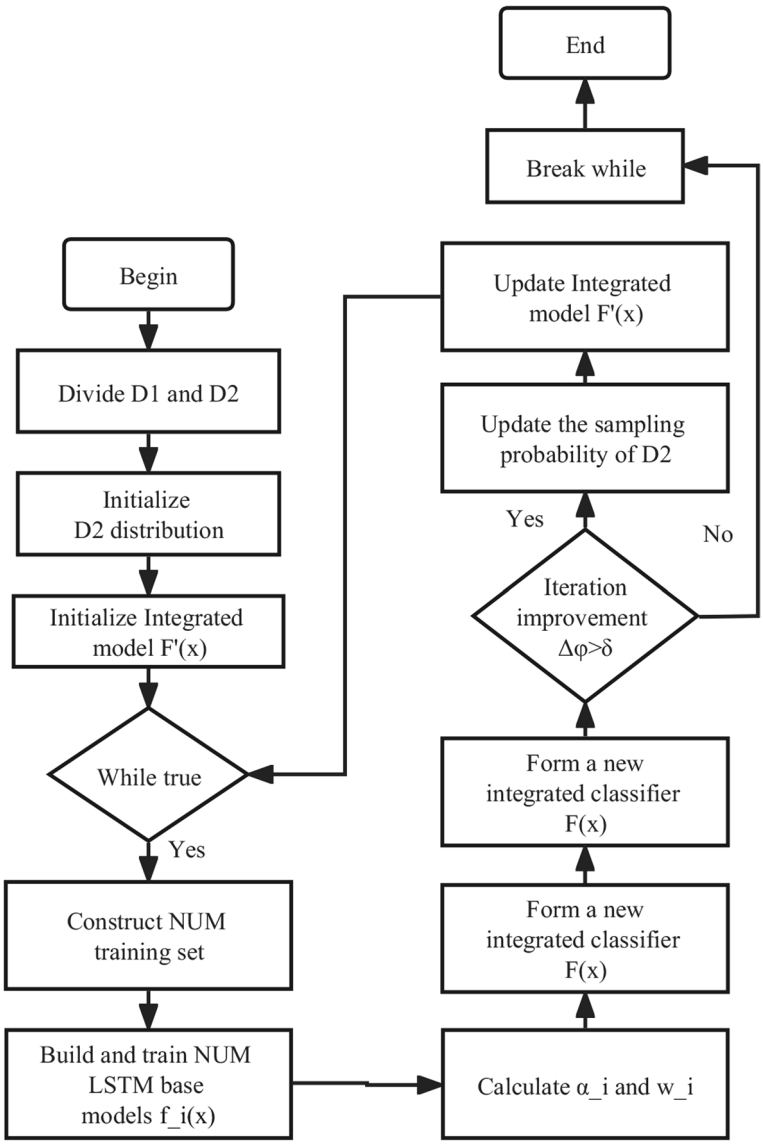

Fig. 7 The flowchart of Algorithm 2

Input: D; // Historical Sequence;
M; // base model;
δ; // integrated model threshold;
L; // Association Set;
S_O; // Average spatial-temporal correlation matrix of faulty containers
Output: $F(x)$; // Final integrated model

Begin
$D_1, D_2 = partDataSet(D)$
$D_2 = initSampleDistribution(D_2)$
$F'(x) = null$
while True
  for i *in* $M.size()$ **do**
    $D_i' = mergeDataSet(D_1, selectExample(D_2))$
  end for
  for i *in* $M.size()$ **do**
    $f_i(x) = modleTrain(m_i, D_i', L, S_O)$
  end for
  for ***i in M.size()*** **do**
    $\alpha_i = perdicet(f_i(x), D)$;
    $w_i = \left({}^{\alpha_i}/_{1-\alpha_i}\right)^2 / M.size()$;
  end for
  $F(x) = F'(x) + w_1 f_1(x) + w_2 f_2(x), \ldots, + w_{num} f_{num}(x)$;
  if $CalculateElevate\left(F(x), F'(x)\right) > \delta$ **then**
    $updateSampleProba(D_2)$
    $F'(x) = F(x)$
  else
    break
  end if
end while

Algorithm 2. Model learning based on dynamic feedback sampling and ensemble learning

Container cascade fault detection based on the cascade fault relational model

Next, real-time container cascade fault detection based on cascade fault relational model and ensemble learning will be introduced. The detection process is shown in Fig. 8. Considering the temporal sequence of cascade faults and the sensitivity of the LSTM model to time-dependent data, we adopt the sliding window mechanism. Continuous detection of container faults can be realized by collecting container data at the previous n time points in advance.

- **Data acquisition and input.** Obtain container historical performance data (including container CPU, memory, IO, etc.) at the previous n time. The state (the normal state is marked 0, and the fault state is marked 1) and the spatial–temporal correlation degree of the container are combined as input data, denoted as $\{x_1, x_2, \cdots, x_n\}$, and they are input into the integrated model F(x) trained above, where $x_i = \{ID, PerformanceData, Associatedmatrix, Othercontainerstates, State\}$.
- **Fault detection process.** The integrated model F(x) detects whether the container will fail at time $(n+1)$ based on the input vector$\{x_1, x_2, \cdots, x_n\}$. At the same time, it obtains the above data at time $(n+1)$ by the sliding window mechanism. Based on the sequence$\{x_2, x_3, \cdots, x_{n+1}\}$, the container state at time $(n+2)$ is detected. The detection window is continuously sliding to detect the possibility of container fault.

$$output_i = model.predict(\{x_i, x_{i+1}, \cdots, x_{n+i-1}\}) \tag{6}$$

- **Results output.** The output of the model is transformed into the failure probability of the current container state by the sigmoid activation function, and the value range is between 0 to1. The greater the value is, the greater the possibility of failure.

$$predict_i = sigmoid\left(output_i\right) \tag{7}$$

Experiments and evaluation

Set up

To simulate the cascade faults between containers in the cloud environment, experiments are carried out by establishing a Docker cluster. We deploy different container instances in the cluster to form different service instances, including data storage services, web services, distributed computing services, and monitoring services. Figure 9 shows the intercontainer dependencies and interservice dependencies in the container cluster. The Configuration setup table is shown in Table 2.

An experimental prototype system based on CAdvisor + InfluxDB + Grafana combined with the cascade fault detection method is constructed. The system framework is shown in Fig. 10. The software packages used by the system are Docker v19.03.8, cAdvisor v1.30.0, influxdb-1.7.6, grafana-6.0.0, Hadoop-3.1.0, Zookeeper-2.7.5, Ceph-9.1.0, docker container stress generation and testing tool docker-stress-ng v1.0. CAdvisor is used for data collection, and the collected data are used as the data source of InfluxDB in the form of time series data and then stored in the Database. Docker-stress-ng, a special stress test and fault injection tool for Docker containers, is used to inject CPU, memory, disk and network pressure on container instances so that their resource usage reaches the bottleneck of system-limited resources. The types of injected faults are shown in Table 3.

To find the key performance indicators reflecting the failure of the container, we analyze the container resource usage patterns in different states in the Docker cluster and compare the average CPU usage, average memory usage, and average disk IO time between faulty containers and normal containers [27, 28]. Finally, the number of container CPU requests, CPU usage, memory requests, memory usage, disk IO, and network traffic are used as performance indicators for container fault detection, and these data are collected in the data collection phase.

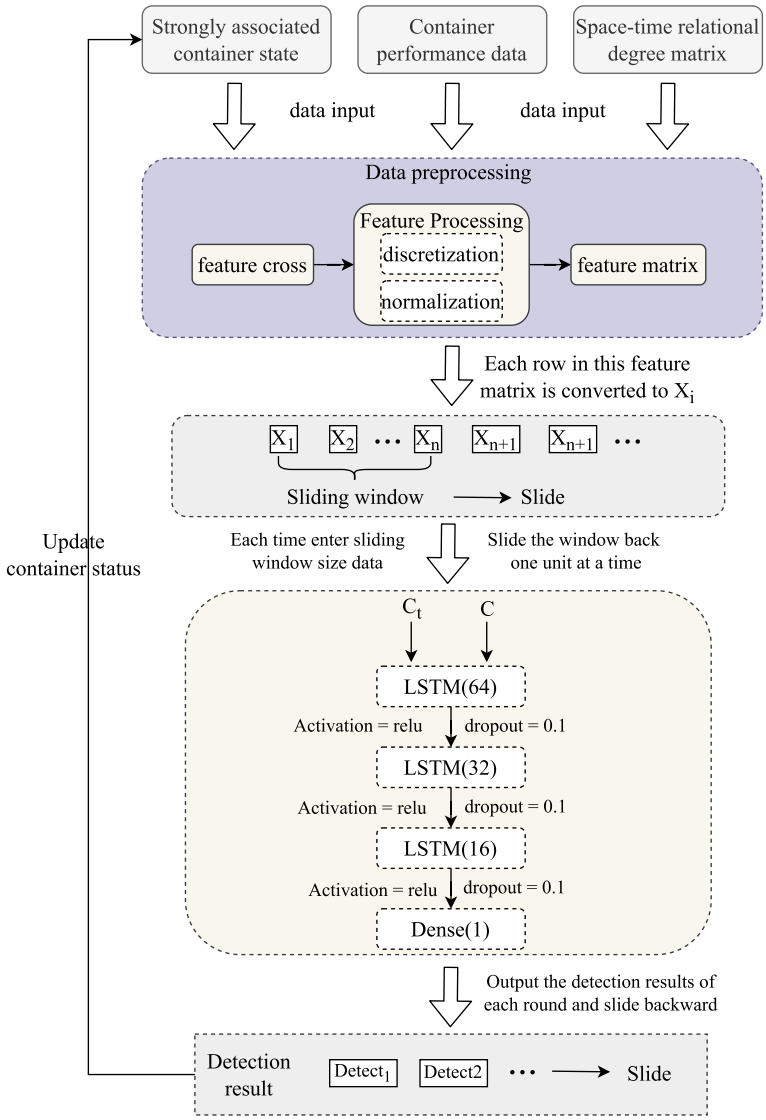

Fig. 8 Fault detection flow

Here are the metrics used for model evaluation.TP is the number of samples with positive observed value and positive detected value, FN is the number of samples with positive observed value and negative detected value, FP is the number of samples with negative observed value and positive detected value, TN is the number of samples with negative observed value and negative detected value. Accuracy, precision, recall rate and F_1 are calculated by Formulas (8), (9), (10) and (11).

$$accuracy = (TP + TN)/ALL \tag{8}$$

$$precision = TP/(TP + FP) \tag{9}$$

$$recall = TP/(TP + FN) \tag{10}$$

$$F_1 = 2/(1/Precision + 1/recall) \tag{11}$$

Evaluation of fault relational model

The model is built and trained with the dataset of the Docker cluster obtained by simulation experiment. The LSTM training model is configured to train the fault correlation model with different network structures ranging from 4 to 8 hidden layers and from 8 to 256 hidden neurons.

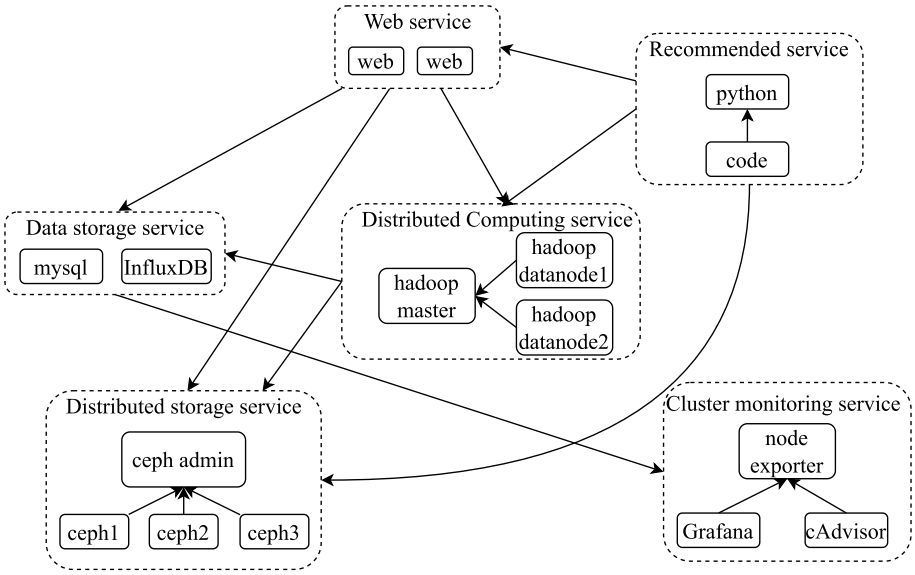

Fig. 9 The services configuration in the experiments

Table 2 Configuration setup table

Node	Hardware configuration	Container instances
1	2.4 GHz Intel Xeon E5-260 CPU,2G RAM	node exporter, Grafana, cAdvisor, ceph admin, hadoop master
2	2.4 GHz Intel Xeon E5-260 CPU,2G RAM	mysql, InfluxDB, web, web, ceph1, hadoop datanode1
3	2.4 GHz Intel Xeon E5-260 CPU,2G RAM	python, code, ceph2
4	2.4 GHz Intel Xeon E5-260 CPU,2G RAM	ceph3, hadoop datanode2

Performance comparison of cascade fault relational models

The formation of the cascade fault detection model includes two stages: construction of the relational model and training of the relational model. By comparing with the traditional correlation analysis method Apriori [29] and the fault history propagation path analysis method LCS [13], the effectiveness of the fault relational model proposed in this paper is evaluated for cascade faults. Table 4 shows the results based on 100,000 container cluster historical data.

Table 4 shows the number of fault propagation paths and times taken to construct the model. LCS calculates every historical fault propagation path that has occurred. Additionally, because it simply obtains the fault path, the model construction time is the shortest. When different faults occur in the fault propagation time window, Apriori first lists the occurring fault as a candidate fault propagation path and then judges whether the path is a fault propagation path after combining the minimum support and confidence calculation. CFD-STC is the same as Apriori in the calculation stage. After obtaining the final fault propagation path, CFD-STC also calculates the correlation degree, so the time consumption is greater than that of Apriori. However, this increased consumption in the construction will be compensated in the subsequent model training process by reducing the formation time of the fault detection model.

Figure 11 shows the accuracy changes in the LSTM model training process using three different methods after obtaining the container fault propagation path. It can be seen that CFD-STC has higher training efficiency in the model training stage, and the model is basically fitted after training approximately 400 times, while the other two methods are fitted after 600–800 times. This is because the method in this paper filters out some fault propagation paths that have low confidence through frequent itemsets calculation. In addition, it calculates the fault correlation between containers and takes it as a model training feature so that the LSTM model training process can better judge and learn the probability of different cascade fault propagation paths. Thus each round of training obtains more effective cascade fault information, which improves the training efficiency of the model. At the same time, the training model obtains more fine-grained fault propagation information, which better fits the fault propagation mode and obtains a higher detection model accuracy.

Comparison of fault detection effects

After the training of the fault detection model, the detection data are obtained in the same way as the training

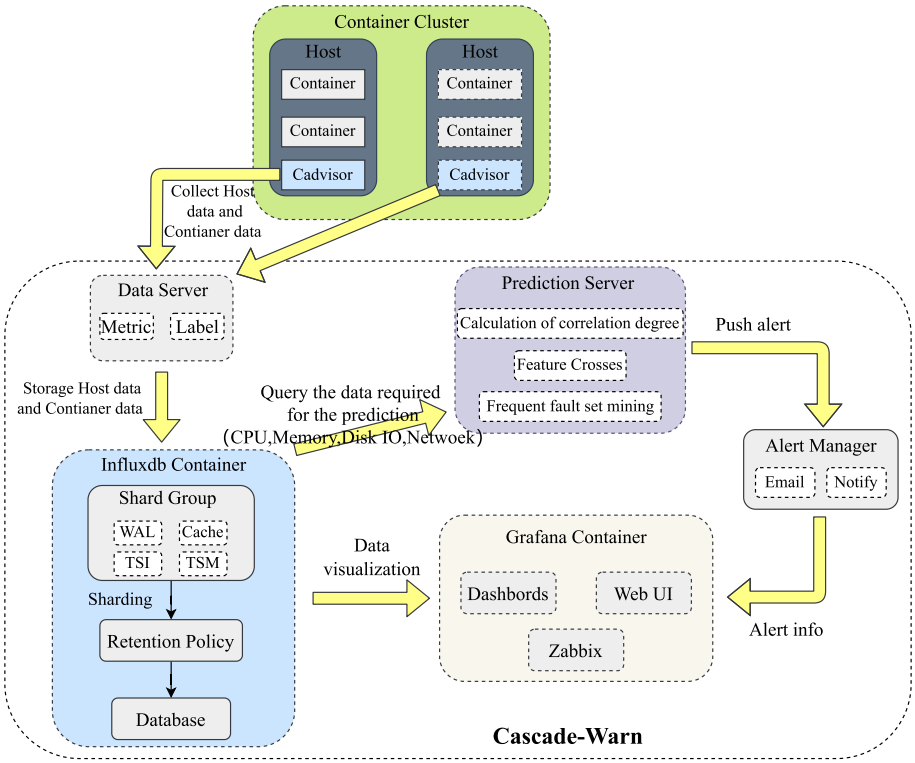

Fig. 10 Experimental prototype framework

Table 3 Types and methods of fault injection

Type of failure	Description	Action
CPU failure	Simulate the container to request CPU usage, make the system reach the limit, and cause the container to crash	Stress-ng generates CPU pressure in the specified container to occupy all allocated CPU
Memory failure	Simulate the container to request memory usage, make the system reach the limit, and cause the container to crash	Stress-ng simulates the creation of objects pointing to global static variables in the container, gradually filling up the memory of the container
Network failure	Simulate container network bandwidth is fully occupied	iPerf Continuously sends data packets to other container instances until the requested network bandwidth is occupied
Disk failure	Simulate container disk IO is fully occupied	Stress-ng simulates the disk write() function, continuously writes to the disk, occupying the container disk IO

Table 4 Model performance comparison

Model	The number of fault propagation path	Time to construct/sec
Apriori	376	13,608
LCS	1491	10,233
CFD-STC	376	13,834

data, and the detection data are extracted with the same features as the training data and then input into the three cascade fault detection models. The detection results are shown in Table 5.

Although the three methods are all cascade fault detection models trained by the same data and the same LSTM network structure, the detection effect and model performance of the LSTM fault detection model trained by the proposed approach are improved more, in terms of accuracy, recall rate and F1 value effect are improved by between 10 to 15% compared with the those of other models, and the model error is controlled to be a small value.

Effect evaluation of ensemble learning optimization method

The experimental method in this section is basically the same as that in Evaluation of fault relational model

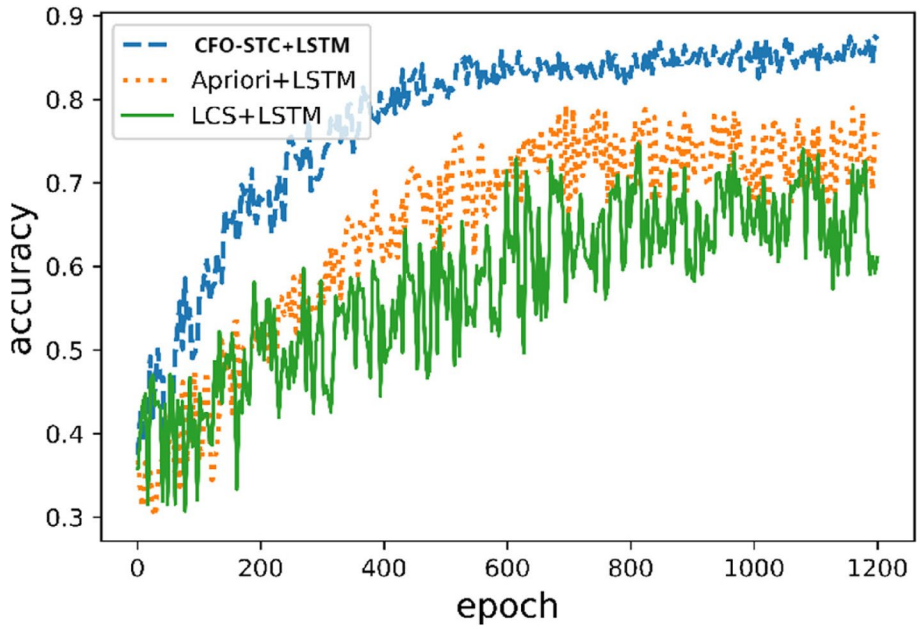

Fig. 11 Model training process

Table 5 Three methods to detect the effect of container cascade fault

Metrics	LCS + LSTM	Apriori + LSTM	CFD-STC + LSTM
RMSE	0.549	0.414	0.287
MAPE	0.607	0.511	0.315
R^2	0.622	0.761	0.908
Precision	0.689	0.736	0.823
Recall	0.431	0.707	0.808
*F*1	0.530	0.722	0.815

Table 6 The effectiveness of the verification method in the Docker cluster data set

Metrics	CFD-STC	CFD-STC-E
RMSE	0.287	0.213
MAPE	0.315	0.232
R^2	0.908	0.950
Precision	0.823	0.913
Recall	0.808	0.895
F_1	0.815	0.904

section. By applying the ensemble learning optimization method, the fault detection effect of the fault relational model in this paper is fully improved, and the performance is superior in the face of imbalanced cascade failure history data. While obtaining an efficient detection model, the resource utilization rate in the cloud environment is maintained at a low level.

Comparison of different models

To verify the improvement of CFD-STC by combining the dynamic feedback sampling method and the ensemble learning method, we use the Docker cluster dataset and Alibaba [30] cluster dataset to compare the effect of the model. We compare the performance of CFD-STC and CFD-STC with the proposed model learning method, which is denoted CFD-STC-E. As shown in Tables 6 and 7, after training the model through the proposed sampling and learning methods, the fault detection effect on the two datasets has been improved, with the accuracy increased by 9.0 and 5.7%, and the recall increased by 8.7 and 7.9%, respectively.

Table 7 The effectiveness of the verification method in the Alibaba cluster data set

Metrics	CFD-STC	CFD-STC-E
RMSE	0.281	0.199
MAPE	0.302	0.203
R^2	0.917	0.961
Precision	0.863	0.920
Recall	0.824	0.903
F_1	0.843	0.911

The dataset of the Docker cluster obtained by fault injection and the dataset of the Alibaba cloud platform

are unbalanced. This causes interference with CFD-STC, which limits the accuracy and recall rate of cascade fault detection and increases the error rate. Through the probability distribution sampling method, multiple balanced training sets and corresponding basic LSTM models are constructed. Each basic model obtains a balanced training set that effectively shields the impact of data imbalance. Feedback sampling ensures that the data samples with detected classification errors in the overall dataset are repeatedly trained by the model. The integration mechanism makes the integrated model better than any single basic model in the same round. Therefore, CFD-STC-E has better cascade fault detection performance.

Comparison of different imbalance rates

To verify the superiority of the ensemble learning optimization method in the face of imbalanced cascade fault data, datasets with different imbalance rates (IRs) are extracted and constructed based on container cluster data and Alibaba cluster data (the calculation formula of IR is shown in Formula (12), in which N_2 and N_1 are the numbers of normal data and fault data samples, respectively), and 80% of them are used as the model training set and 20% are used as the model test set. The proposed fault relational model and LSTM are used to train the cascade fault detection model to verify the guarantee and improvement of the accuracy, recall and F1 (F-measure) of the container cascade fault detection in the face of imbalanced cascade fault data.

$$IR = \frac{N_2}{N_1} \tag{12}$$

In Fig. 12, we compare the accuracies and recall rates of the proposed method, the oversampling method, the undersampling method [31], the no data processing method, and the traditional ensemble learning method under different IR values. The accuracy rate and recall rate of the model without any data processing method decrease with increasing IR, especially when the IR is large, and there is a cliff-like decline. The undersampling

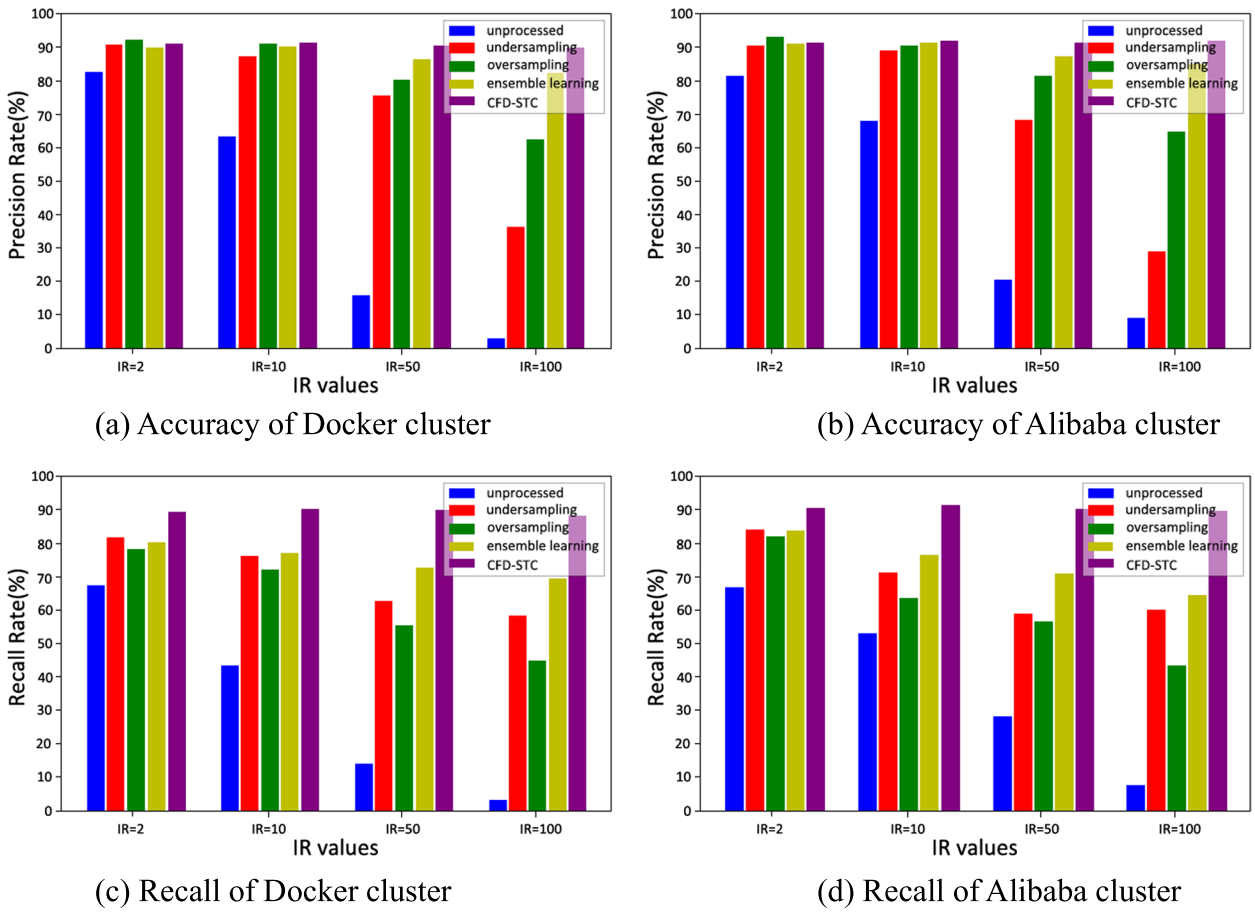

Fig. 12 Detection effect of cascade faults under different IR conditions

method constructs a balanced dataset by randomly selecting normal data samples. However, it loses the data information of many normal data samples. When the IR increases, although it can maintain the recall rate, its accuracy is reduced due to the decline in the accuracy of the normal data. The oversampling method keeps the balance of the training set by generating the fault data, and its accuracy can remain stable when the IR is small. However, with the increase in the IR, the new samples that need to be generated continue to increase. The model often overfits the training samples, resulting in a decrease in the accuracy and recall rate. Although the accuracy of the ensemble learning method is stable at a high level, with the increase in the IR, most class sample information of the detection model based on the ensemble learning method is discarded during the sample division or iteration process, resulting in a downward trend of accuracy. At the same time, the recall rate cannot be guaranteed well with the increase of the IR.

CFD-STC-E maintains a high level of accuracy and recall, and the accuracy and recall are basically stable when the IR is constantly changing. This is because the dynamic sampling method is used to construct multiple datasets in the training model. Through the weighted combination of multiple models and the dynamic adjustment of the sample distribution, the normal data features learned by the model are basically close to the overall normal class sample data features. Next, we compare the resource consumption of our method with other methods in the model training process. Figure 13 shows that the undersampling method has the least resource utilization in training under different IRs. The resource consumption of the undersampling method decreases with increasing IR when IR is greater than 1, while the opposite is true for the oversampling method. The resource consumption in the training process of the proposed method is similar to that of traditional ensemble learning. This is because we mainly improve the sample distribution update of traditional ensemble learning before training and the model integration method after training. It can be seen that the resource utilization rate of the proposed method is higher when the IR is low, and with the increase in the IR, the resource utilization rate decreases gradually and is only higher than that of the undersampling method. This is because the number of ensemble learning models is usually 10–20. When the IR is low, the data volume of the whole model is larger than that of the original data, so the resource utilization rate is higher than that of normal training. When the IR of the data increases, the data of the whole model gradually decrease compared to the initial data. In the real cloud environment, the resource usage of our method is better than most methods because the historical data imbalance rate of the cloud platform is between 10 and 50.

Fault detection effect evaluation

Based on the above system, we compare the detection effect of the proposed method with LCS and LCS with VMM in cascade failure propagation. Figure 14 shows the comparison of three methods for container cascade faults.

By comparing the model performance of the two cascade fault models on the Docker dataset, it can be seen from Fig. 14 that CFD-STC-E is better than the other two fault propagation detection models in terms of the accuracy and recall of the cascade fault detection effect. In addition, the error of the model is also less than that of the compared methods. The two methods in [32] only obtain the historical path of fault propagation through fault injection, which is stored in the database, and learn the characteristics of the normal path and fault path through deep learning. In the model detection stage, if the analysis of the process execution path does not conform to the normal path, it is identified as

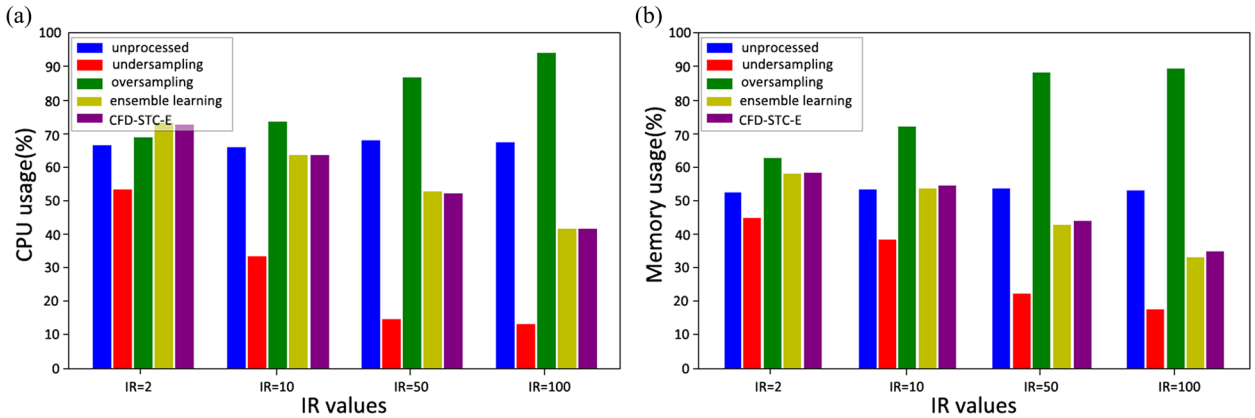

Fig. 13 Comparison of CPU and memory usage during training with different methods

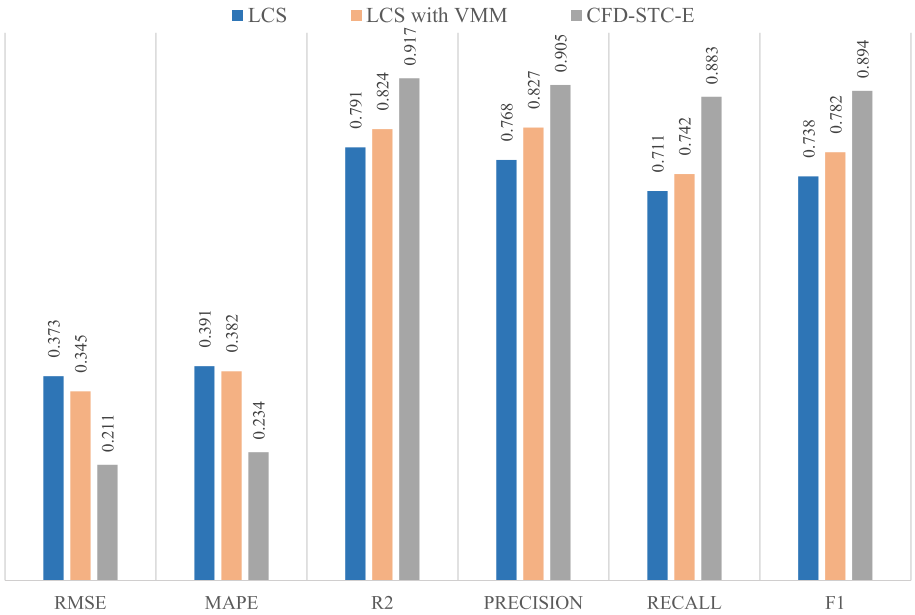

Fig. 14 Comparison of cascade fault detection effects of three methods

an abnormal path. In contrast, our method comprehensively considers the path of container fault propagation and the probability of cascade failure propagation between containers. This can better characterize the cascade failure propagation mode of containers, reduce the misjudgment of faults, enhance the accuracy, and reduce the error of the detection model. TPR and EPR are calculated by Formulas (13) and (14).

$$TPR = \frac{\text{number of cascade faults checked correct}}{\text{actual number of cascade faults}} \tag{13}$$

$$FPR = \frac{\text{number of cascade faults checked incorrect}}{\text{actual number of non}-\text{cascade faults}} \tag{14}$$

It is inappropriate to only consider accuracy to assess the prediction model against imbalanced datasets. The advantage of the ROC curve [29] is the fact that it depicts the trade-off between TPR and FPR. Its value ranges from 0 and 1. The closer it is to 1, the better the model predicts. The P-R curves [12] show the correlation between the recall and precision of the model. The receiver operating characteristic (ROC) curve and P-R curve of the three methods were compared, and the results are shown in Fig. 15. Under different FPR/TPR ratios, the proposed method is superior to the compared methods and has high accuracy. On the Docker historical dataset, the AUC (area under the curve) of the proposed method is 0.908, while the values of LCS and LCS with VMM are 0.802 and 0.769, respectively. This shows that CFD-STC-E is superior to the compared methods in correctly detecting cascade faults and correctly detecting non-cascade faults.

After 2, 4, 6, and 8 containers are randomly selected in the system to inject faults, the resulting alarm

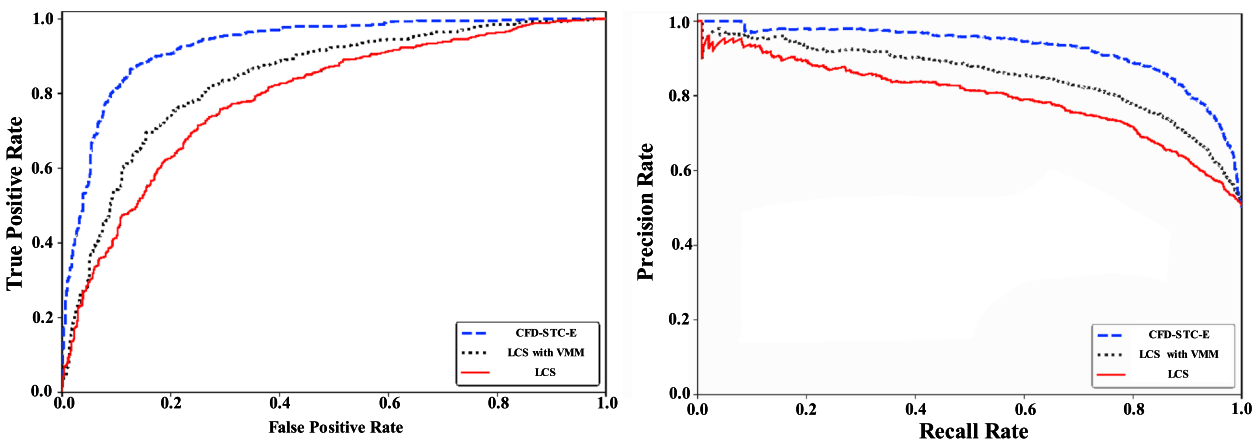

Fig. 15 Comparison of ROC and P-R curves of the two methods

situation of the fault model is shown in Fig. 16. The number of cascade fault warnings increases over time, which indicates that the fault cascade and propagation are caused by the injected fault container. The method in this paper can better capture the occurrence of container cascade fault and provide an early warning.

Figure 17 shows the change in the container failure rate (number of fault containers/total number of containers) to see whether the cluster enables our method, under different numbers of injected fault containers in the container cluster. Figure 17a–d represent the random selection of 2, 4, 6, and 8 container injection failures in the Docker container cluster, respectively. The x-axis represents the time after fault injection, and the y-axis represents the ratio of containers that fail at time t. It can be seen that the final cascade faults do not affect the entire container cluster, which is due to the degree of correlation between containers, as well as the fault isolation and fault tolerance measures in the cloud platform.

Conclusion

To address the problems of lacking fine-grained construction of targeted cascade fault models and the model degradation caused by imbalanced fault data, the cascading and propagation modes of inter-container faults in a containerized cloud platform are studied in depth. Firstly, the container fault spatial–temporal correlation is modeled from the fault cascade history dimension and container space correlation dimension. And the Cascade Fault Detection method based on Spatial–Temporal Correlation is designed. Secondly, aiming at the imbalance of

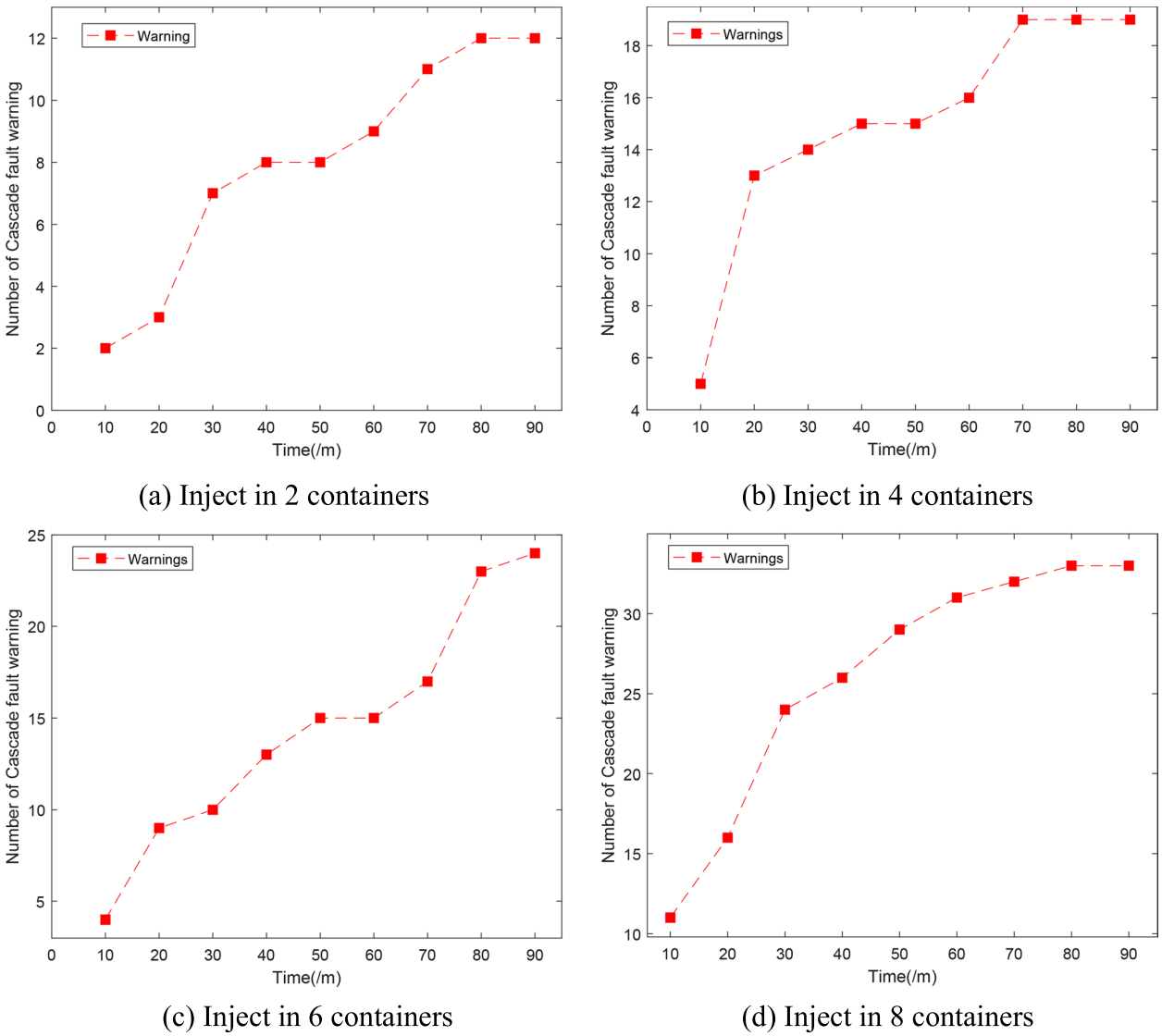

Fig. 16 Number of faults detected by the model

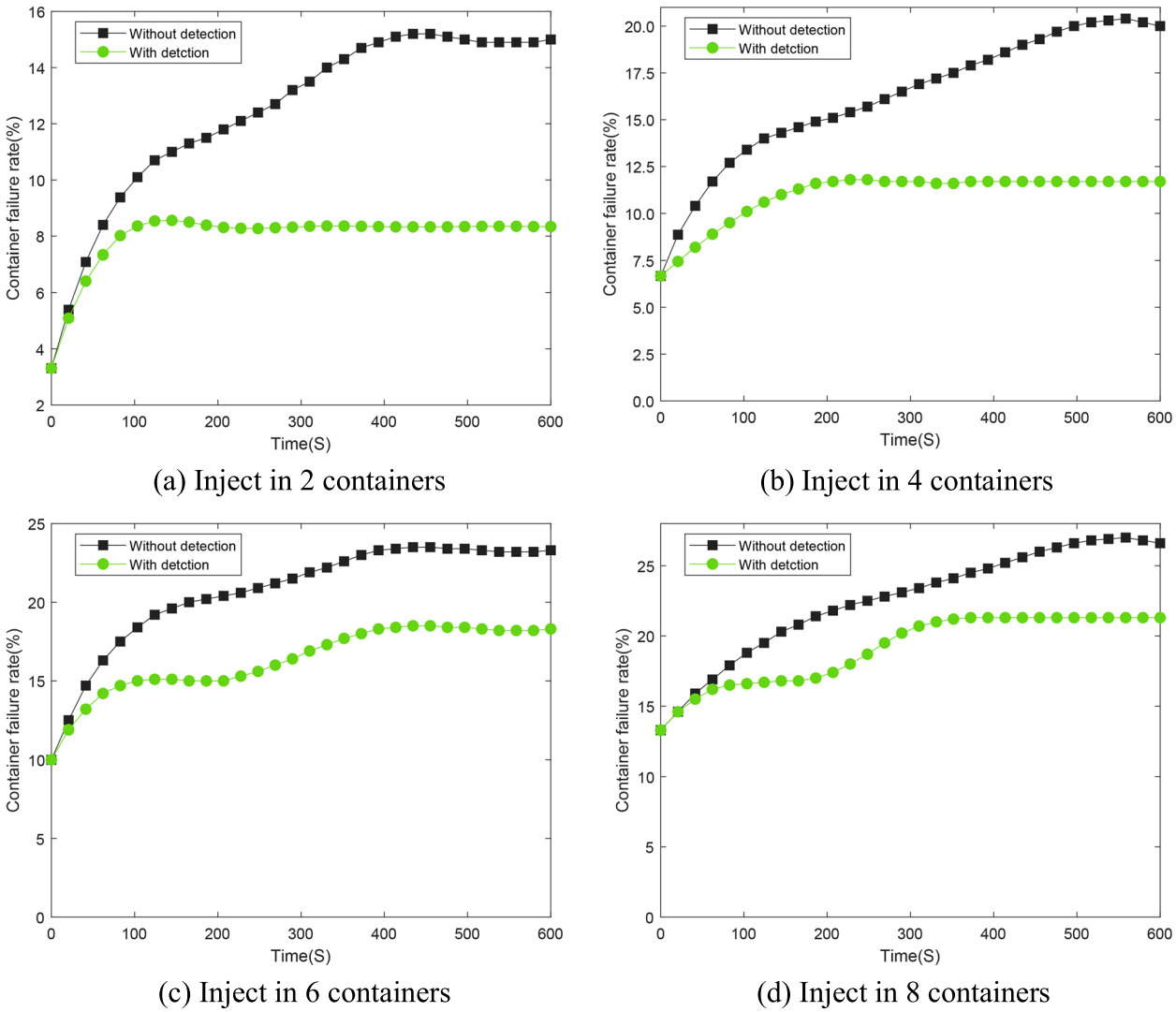

Fig. 17 Fault propagation rate comparison

data ignored by traditional detection methods, a model training method combining dynamic feedback sampling and ensemble learning is designed to further improve the detection effect of container cascade fault. The results show that the method is effective and accurate in terms of container fault detection. However, container fault cascade in cloud environment is focused on only. Considering the difference between container fault modes in other complex and dynamic environments and cloud environment, proposing a generic container cascade fault detection method will be the main research work in the future.

Authors' contributions
Ningjiang Chen (First Author): Conceptualization, Methodology, Resources, Supervision, Funding Acquisition; Qingwei Zhong: Validation, Formal Analysis, Software, Writing—Original Draft; Yifei Liu (Corresponding Author): Investigation, Resources, Supervision, Writing—Review & Editing; Weitao Liu: Resources, Supervision, Writing—Review & Editing; Lin Bai: Data collection and curation; Liangqing Hu: Data processing and analysis. The author(s) read and approved the final manuscript.

Funding
The authors disclosed receipt of the following financial support for the research, authorship: This work was supported by the Natural Science Foundation of China (No. 62162003, No. 61762008), and the National Key Research and Development Project of China (No. 2018YFB1404404).

Availability of data and materials
This declaration is not applicable.

Declarations

Ethics approval and consent to participate
This declaration is not applicable.

Competing interests
The authors declare no competing interests.

Received: 11 July 2022 Accepted: 4 April 2023
Published online: 17 April 2023

References

1. Bali RS, Kumar N (2016) Secure clustering for efficient data dissemination in vehicular cyber–physical systems. Futur Gener Comput Syst 56:476–492
2. Chaudhary R, Kumar N, Zeadally S (2017) Network service chaining in fog and cloud computing for the 5G environment: data management and security challenges. IEEE Commun Mag 55(11):114–122
3. Challa S, Das AK, Gope P et al (2020) Design and analysis of authenticated key agreement scheme in cloud-assisted cyber–physical systems. Futur Gener Comput Syst 108:1267–1286
4. The 10 Biggest Cloud Outages of 2020 (So Far) in 2020. pp 6–25. https://www.crn.com/slide-shows/cloud/the-10-biggest-cloud-outages-of-2020-so-far-/. Accessed 17 May 2022.
5. Manu AR, Patel JK, Akhtar S, et al (2016) A study, analysis and deep dive on cloud PAAS security in terms of Docker container security. In: 2016 international conference on circuit, power and computing technologies (ICCPCT). IEEE, Piscataway, pp 1–13
6. Birke R, Giurgiu I, Chen LY, et al (2014) Failure analysis of virtual and physical machines: patterns, causes and characteristics. In: 2014 44th annual IEEE/IFIP international conference on dependable systems and networks. IEEE, Piscataway, pp 1–12
7. Ye K, Liu Y, Xu G, et al (2018) Fault injection and detection for artificial intelligence applications in container-based clouds. In: International conference on cloud computing. Springer, Cham, pp 112–127
8. Yuan Z, Zhao P (2019) An improved ensemble learning for imbalanced data classification. In: 2019 IEEE 8th joint international information technology and artificial intelligence conference (ITAIC). IEEE, Piscataway, pp 408–411
9. Das A, Mueller F, Siegel C, et al (2018) Desh: deep learning for system health prediction of lead times to failure in hpc. In: Proceedings of the 27th international symposium on high-performance parallel and distributed computing. pp 40–51
10. Mohammed B, Awan I, Ugail H et al (2019) Failure prediction using machine learning in a virtualised HPC system and application. Clust Comput 22(2):471–485
11. Zhang PY, Shu S, Zhou MC (2018) An online fault detection model and strategies based on SVM-grid in clouds. IEEE/CAA J Autom Sin 5(2):445–456
12. Le VH, Zhang H (2021) Log-based anomaly detection without log parsing. In: 2021 36th IEEE/ACM international conference on automated software engineering (ASE). IEEE, Piscataway, pp 492–504
13. Bergroth L, Hakonen H, Raita T (2000) A survey of longest common subsequence algorithms. In: Proceedings seventh international symposium on string processing and information retrieval. SPIRE 2000. IEEE, Piscataway, pp 39–48
14. Cotroneo D, De Simone L, Liguori P, et al (2019) Enhancing failure propagation analysis in cloud computing systems. In: 2019 IEEE 30th international symposium on software reliability engineering (ISSRE). IEEE, Piscataway, pp 139–150
15. Bui DM, Huynh-The T, Lee S (2016) Fuzzy fault detection in IaaS cloud computing. In: Proceedings of the 10th international conference on ubiquitous information management and communication. pp 1–6
16. Wang H, Shen H, Li Z (2018) Approaches for resilience against cascading failures in cloud datacenters. In: 2018 IEEE 38th international conference on distributed computing systems (ICDCS). IEEE, Piscataway, pp 706–717
17. Wang T, Zhang W, Wei J, et al (2015) Fault detection for cloud computing systems with correlation analysis. In: 2015 IFIP/IEEE international symposium on integrated network management (IM). IEEE, Piscataway, pp 652–658
18. Yu S, Chen N, Liang B (2021) Predicting gray fault based on context graph in container-based cloud. In: 2021 IEEE international symposium on software reliability engineering workshops (ISSREW). IEEE, Piscataway, pp 224–234
19. Toka L, Dobreff G, Haja D, et al (2021) Predicting cloud-native application failures based on monitoring data of cloud infrastructure. In: 2021 IFIP/IEEE international symposium on integrated network management (IM). IEEE, Piscataway, pp 842–847
20. Singh P (2019) Learning from software defect datasets. In: 2019 5th international conference on signal processing, computing and control (ISPCC). IEEE, Piscataway, pp 58–63
21. Sara A, Wael M, Walid M (2018) Using SMOTE and heterogeneous stacking in ensemble learning for software defect prediction. In: Proceedings of the 7th international conference on software and information engineering. Association for Computing Machinery, New York, p 44–47
22. Johnson JM, Khoshgoftaar TM (2019) Survey on deep learning with class imbalance. J Big Data 6(1):1–54
23. Ribeiro VHA, Reynoso-Meza G (2020) Ensemble learning by means of a multi-objective optimization design approach for dealing with imbalanced data sets. Expert Syst Appl 147:113232
24. Kaur H, Pannu HS, Malhi AK (2019) A systematic review on imbalanced data challenges in machine learning: applications and solutions. ACM Comput Surv (CSUR) 52(4):1–36
25. Lu C, Ye K, Xu G, et al (2017) Imbalance in the cloud: an analysis on alibaba cluster trace. In: 2017 IEEE international conference on big data (big data). IEEE, Piscataway, pp 2884–2892
26. Lindemann B, Maschler B, Sahlab N et al (2021) A survey on anomaly detection for technical systems using LSTM networks. Comput Ind 131:103498
27. Heinrich R, van Hoorn A, Knoche H, et al (2017) Performance engineering for microservices: research challenges and directions. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion. pp 223–226
28. Jamshidi P, Pahl C, Mendonça NC et al (2018) Microservices: the journey so far and challenges ahead. IEEE Softw 35(3):24–35
29. Shao Y, Liu B, Wang S et al (2018) A novel software defect prediction based on atomic class-association rule mining. Expert Syst Appl 114:237–254
30. Hai D, Chang Z (2019) Alibaba Cluster Trace Program [EB/OL]. https://github.com/alibaba/ cluster data
31. Tahir MA, Kittler J, Yan F (2012) Inverse random under sampling for class imbalance problem and its application to multi-label classification. Pattern Recogn 45(10):3738–3750
32. Cotroneo D, De Simone L, Liguori P, et al (2020) Fault injection analytics: a novel approach to discover failure modes in cloud-computing systems. In: IEEE transactions on dependable and secure computing

Publisher's Note