A Semi-supervised Learning Based Method for Identifying Idle Virtual Machines in Managed Cloud: Application and Practice

Xian Yu[1,2], Kejiang Ye[1], Zihong Chen[2], Jia Yi[2], Xiaofan Chen[2], Bozhong Liu[2], and Chengzhong Xu[1(✉)]

[1] Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
kj.ye@siat.ac.cn, czxu@um.edu.mo
[2] Sangfor Technologies Inc, Shenzhen, China
{yuxian,yijia29285,chenxiaofan,liubozhong}@sangfor.com.cn

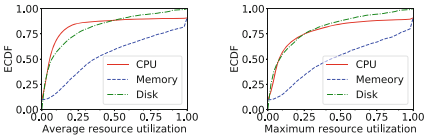
Abstract. Due to unreasonable virtual machine (VM) resource planning and complex load variation, the waste of VM resource has become a significant issue for many enterprises. Although existing technical solutions have proven to have certain ability to identify idle VMs, most of them are researched in private cloud or public cloud scenarios. And it lacks an effective method customized for managed clouds, where the previous work still suffers from the challenges of fewer labels, poor data quality and large scale of VMs in production environments. For this reason, we first investigate the resource usage data of thousands of VMs from a real managed cloud. Based on the analysis results, we propose an innovative and practical method to identify idle VMs. Through elaborate data processing, feature engineering, and model training, the proposed method enables to achieve excellent performance. Sufficient experiments based on real data from the managed cloud of Sangfor company also prove its practicality and effectiveness in the production environment. Up to now, this service has been deployed in Sangfor cloud for more than 5 months, continuously detecting over 10K VMs, and helping to save at least 1K vCPU cores, 2.5 TB memory and 100 TB disk space.

Keywords: idle virtual machine · machine learning · managed cloud data center · micro-service · random forest model · semi-supervised learning

1 Introduction

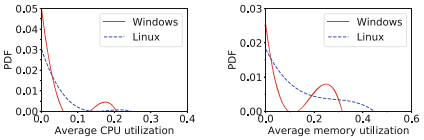
Managed cloud is an emerging cloud service mode [1]. It runs as a 2B service mode and serves a wide range of industries, such as education, government, commercial enterprises, etc. The operators of managed cloud provide an exclusive or shared resource pool for those enterprises to choose from. However, due to the

Y. Zhang and L.-J. Zhang (Eds.): ICWS 2023, LNCS 14209, pp. 65–82, 2023.
https://doi.org/10.1007/978-3-031-44836-2_5



(a) Average (b) Maximum

Fig. 1. Empirical cumulative distribution function (ECDF) of resource utilization of approximately 4K VMs in Sangfor cloud.



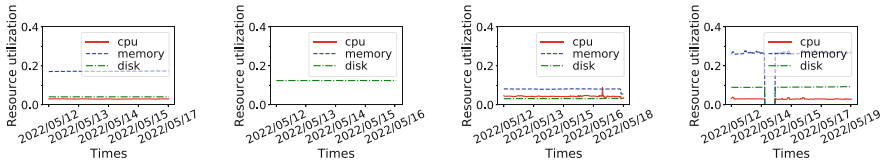
(a) CPU (b) Memory

Fig. 2. Probability density function (PDF) of average resource utilization of 200 idle VMs pre-determined by SREs.

unreasonable allocation and management of Virtual Machines (VMs) by tenants, a large amount of VM resources are wasted, greatly increasing the IT operating costs. For example, some tenants are used to allocate far more resources than they actually need when creating a new VM; software developers may temporarily open multiple VMs for testing purposes, but forget to release them afterwards. Figure 1 shows the resource utilization distribution of CPU, memory and disk of approximately 4K VMs from Sangfor cloud for a week, which owns more than 3000 physical servers and 10000 VMs. The analysis results reveal an extremely serious waste of VM resources in this cloud, especially CPU and disk resources.

To eliminate this problem and provide better cost management services, Site Reliability Engineers (SREs) will regularly clean up the infrequently used VMs or those with long-term low resource occupancy. Unfortunately, when it involves thousands of VMs, manual identifying the idle ones is quite time-consuming and costly. Moreover, the services or processes running on different VMs vary greatly, which makes it be impossible to accurately detect all idle VMs by setting a static threshold or simple rules [2–4]. To be more effective, a functional idle VM detection method is therefore fundamental for managed cloud. Although a large amount of existing work has been proposed to use dynamic thresholds [5], or bring in some machine learning (ML) models [6–9] to learn the variation pattern of the VM resource or performance metrics, these methods either produce too large overhead or require enough positive sample labels, or do not consider the impact of operating system (OS) type on the identification model. Consequently, these methods are hard to be applied to production environment.

To this end, we investigate the application of ML methods in the identification of idle VMs, and present an idle VM identification algorithm based on the Random Forest model [10] and the corresponding micro-service deployment framework. The entire study is conducted on Sangfor cloud, and the service has been successfully deployed for a period of time. We first elaborately perform feature engineering to improve the accuracy and efficiency of idle VM detection by referring to the daily experience of SRE in identifying idle VMs. We then experimentally compare various prevalent ML models and select the best one as the final model to detect idle VMs. Interestingly, we train two detection models separately depending on the type of VM OS (Linux and windows). Consider-



(a) Memory usage dominating (b) Disk usage dominating (c) Fluctuation with little burst (d) Occurrence of system anomalies

Fig. 3. Typical resource usage patterns of idle VMs.

ing the labor-intensive labeling work, we also present a semi-supervised learning algorithm to help train the model, in which we particularly propose a sample augmentation algorithm to address the problem of insufficient positive samples. We conduct plenty of online data exception handling, so as to reduce the influence of data quality on identification results. Last but not least, the implementation of the entire system follows the principle of micro-service design, where the idle VM identification model runs stateless and retrain periodically, which facilitates rapid online expansion and update after deployment.

The contributions of this paper include:

- We analyze real VM resource occupancy data and design 15 kinds of key time-series features to detect idle VMs, and we also find that idle VMs installed with Linux generally have lower memory occupancy than idle VMs installed with windows.
- We investigate in depth how to identify idle VMs using resource metrics of VMs along with ML models, and propose a practical and lightweight idle VM detection method based on random forest-based model, as well as the corresponding micro-service detection framework.
- The proposed method has been applied to the production environment for several months, and the results of a large number of offline experiments and online service running data prove its effectiveness and efficiency.

The following paper is organized as follows. We first introduce the motivation and challenges of this work in Sect. 2, and propose a ML-based idle VM detection method in Sect. 3. We evaluate the proposed method in Sect. 4. We further analyze its performance and overhead after it is deployed to production environment in Sect. 5. Some related work is described in Sect. 6. We state the limitations of this work in Sect. 7 and draw a conclusion in Sect. 8.

2 Background

Idle VMs in this work refer to inactive VMs (or sprawled [11]), not just that it is not running any user programs. We expect to find idle VMs by analyzing their resource or performance indicators, which can be used as a basis for shrinking their capacity or closing them to save costs. In this section, we begin by presenting several findings from the analysis of VM resource usage data. The analysis

results serve as the main motivation for our method. Next, we summarize the main challenges of solving this problem in a production environment.

2.1 Observations

In order to better understand the characteristics of idle VMs, SREs randomly select several VMs in advance, and manually label 200 VMs that they think are idle based on their experience. Two thirds of these VMs have linux OS installed and the remaining have windows OS installed. We analyze the resource usage of all these idle VMs and make the following two findings.

- **Observation I:** The resource utilization of idle VMs running linux significantly differs from that of idle VMs running windows. We compare the average CPU and memory resource utilization distributions of idle VMs with windows and linux. The statistical probability density distribution (PDF) results are displayed in Fig. 2. According to the results, we can find that the average resource utilization of idle VMs with linux presents a long-tail distribution, while that of idle VMs with windows show an obvious crest change. This change may be caused by two types of windows services, Virtual Desktop Infrastructure (VDI) service [12] and traditional office service. The difference in resource distribution between VMs with windows and linux leads to the need for a differentiated model to identify idle VMs with different types of OS.
- **Observation II:** Idle VMs have similar patterns of resource usage and variation. Figure 3 shows four typical resource usage trends of idle VMs. Figure 3(a) and Fig. 3(b) respectively represent idle VMs dominated by memory occupancy and disk occupancy, and the curves are relatively stable. Figure 3(c) shows that the resource occupancy curve of idle VM fluctuates with little burst, and Fig. 3(d) indicates the idle VM is experiencing some kind of anomalies. Note that these idle VMs are manually marked by SREs, which means SREs believe that the resource occupancy of a idle VM should be similar to these patterns. We further analyze the commonalities between these patterns and find that all VMs labeled as idle by SREs generally have low resource utilization (CPU utilization, memory utilization, and disk utilization in Fig. 3 are all below 30%). Moreover, all these curves are not consistently volatile. These findings show the major differences between idle and non-idle VMs and become the primary basis for determining whether a VM is idle.

2.2 Challenges

Although a large amount of methods [2–5,13–15] have been proposed to detect inactive VMs, these methods are rarely applied to real production environments. For better application performance in the production environment, we summarize several key challenges that require to be broken through.

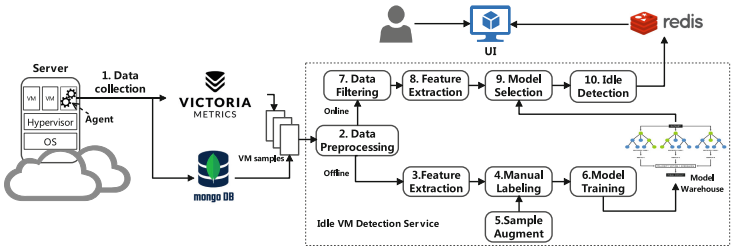


Fig. 4. The proposed idle VMs detection framework.

- **Challenge I: High performance.** A key point to accurately identify idle VMs through ML models is to select appropriate VM resource features. Although there are some public libraries (e.g. tsfresh [16]) that can extract thousands of temporal features , these features not only cause a large extraction overhead, but also requires sufficient samples to ensure the success of model training. More importantly, labeling the idle VMs is very time-consuming, some supervised classification models is thus unable to be applied directly, which further increases the difficulty of identifying idle VMs.
- **Challenge II: Lightweight design.** Unlike large-scale public cloud environments, the available resources in managed cloud environments are insufficient, which requires the designed detection method to be as lightweight as possible. Although the detection speed can be improved by multi-process or multi-threading techniques, these methods also bring about extra resource cost. As a result, the current outstanding large neural network models [17,18] are difficult to be practically applied to addressing this problem, which is also the main reason for this work focusing on the classical ML models.
- **Challenge III: Robust and extensible framework.** For one thing, the idle VM identification method needs to be able to proactively detect and eliminate online data quality problems caused by potential bugs, design defects and irregular updates and upgrades of the monitoring system, as well as the network congestion and device failure. For another thing, it also needs to automatically recognize some long-term holidays (e.g., China's National Day, Spring Festival) to avoid false positives. Furthermore, the design of this detection service must be stateless for a highly scalable capability to better adapt to large-scale VM scaling scenarios.

We set out how to solve above challenges in following Sect. 3.

3 Proposed Method

In this section, we propose a ML model based and micro-service framework to address the challenges mentioned above. Section 3.1 depicts an overview of the proposed idle VM detection framework. We introduce the metrics used to detect idle VMs in Sect. 3.2. We describe the extracted key features from these

metrics in Sect. 3.3. A probabilistic idle VM detection model and the associated semi-supervised training algorithm are respectively presented in Sect. 3.4 and Sect. 3.5. We state how to detect and deal with idle VMs in Sect. 3.6.

3.1 Overview

Figure 4 displays the proposed idle VM detection framework. The relevant data used to detect idle VMs will first be uploaded through the agent deployed in the VM. Static VM information will be stored in a configuration management database (CMDB), and other time series data (such as CPU utilization, memory utilization) will be stored in a time series database (TSDB). The idle VM detection service will pull these data to complete offline training and online detection.

The training phase is composed of step 2 to step 6. In this phase, the VM data will first undergo noise reduction and normalization, and then will be transformed into feature vectors. The feature vectors will be used as the input of the detection model. After that, SREs will manually mark few positive and negative samples. Based on these samples, this idle VM detection service will automatically generate more samples through the presented sample augmentation algorithm to improve the generalization capability. The trained model will be stored in the model warehouse and updated periodically. The phase of online detection mainly corresponds to step 7 to step 10, where the detection service regularly pulls the VM data and make identification. It will first preprocess the input data and further filter out those during a long-term holiday or in the agent upgrade stage. Next, it extracts the idle features and selects the appropriate model for detection based on the type of VM OS. The detection results will be updated to UI interface, and users can perform different treatments according to the detection results and mark the detected VMs as correct or incorrect. The feedback will be converted into training samples to participate in the next model update.

3.2 VM Data Collection

We mainly choose three VM resource indicators to detect idle VMs, including CPU utilization, memory utilization, and disk utilization. Unlike some previous researches, in addition to selecting these resource indicators, they also consider several more complex indicators, such as network connection, application-related system calls and key service processes. The main reason why we only select these indicators is to consider that the current user needs are mainly concerned with these types of VM resources, and the use of these indicators is more in line with the manual identification habits of SRE personnel. Moreover, too many features will not only cause high collection overhead, but also easily lead to degradation of detection efficiency. Note that our method is also suitable for rapid expansion and application to other metric data. Besides, we also collect the indicators of system uptime and the number of keyboard and mouse operations for those VMs with windows OS. These metrics are beneficial to compensate for the shortcomings of the ML-based detection model through several predefined rules.

To obtain these VM indicator data, we developed an agent that runs in VM in the form of a process and uploads the relevant data in real time. When collecting these data of tenant's VMs, we will strictly abide by the negotiation agreement with tenant, so there will be no data privacy issues.

3.3 Determining Idle Features

Although we have reduced the complexity of idle VM detection to a certain extent by controlling the number of participating detection indicators, due to the diversity of time series, each VM resource indicator can still extract up to thousands of features, which will inevitably lead to high performance loss and time overhead. Based on the observation 2 mentioned in Sect. 2.1, we summarize two kinds of key features for detecting idle VMs: statistical features and fluctuating features. In particular, we screen out 9 basic statistical features and 6 time-series fluctuation features from more than 1,000 features generated by tsfresh [16] through correlation test, variance test, and P-value test. The information description of the extracted features can be found in Table 1, and their detailed calculation process can refer to tsfresh [16].

Table 1. The extracted features from each kind of VM resource utilization sequence.

Categories	Feature names	Description
Basic statistical features	Minimum	The minimum value of the sequence.
	Maximum	The maximum value of the sequence.
	Mean	The mean of the sequence.
	Median	The median of the sequence.
	Variance	The variance of the sequence.
	Standard deviation	The standard deviation of the sequence.
	Root mean square	The root mean square of the VM resource utilization sequence.
	Var larger than std	A boolean value indicating whether the variance of the sequence is greater than the standard deviation of it.
	Ratio larger than mean	The proportion of the objects that are greater than the mean value in the sequence.
Sequence fluctuation features	Variation coefficient	The discrete degree of the sequence calculated under a unified numerical dimension.
	Sample entropy	Measuring the probability of generating new patterns in the sequence.
	Permutation entropy	Quantitative assessment of random noise contained in the sequence.
	Binned entropy	The binned entropy of the power spectral density of the sequence.
	Complexity invariant distance	The degree of chaos in the sequence.
	Absolute sum of changes	The magnitude of the change in the sequence.

3.4 Detection Model Design

Intuitively, idle VM detection is a 0–1 binary classification problem. However, directly determining whether a VM is idle may confuse SREs, so that it is impossible to make the best disposal action. The main reason for this confusion is that the so-called idle VM also needs to consider issues such as idle time period and idle degree. Therefore, to eliminate this confusion, we turn this problem into a detection probability problem. Specifically, we define an idle probability P to indicate how likely a VM to be idle. In fact, we also predefined a set of idle probability thresholds, each of which corresponds to a different processing method for idle VMs (such as resource scaling, shutdown, etc.). The formal mathematical expression for this detection process is as follows:

$$P(V_k) = Func(\overrightarrow{f^1_{V_k}}, \overrightarrow{f^2_{V_k}}, \ldots, \overrightarrow{f^N_{V_k}})$$

where $f^i_{V_k}$ means the extracted feature vector of i-th indicator of VM V_k. Although many deep learning models have been widely used in image recognition [17], text classifications [18], they will also bring huge training and prediction costs. For this reason,our work concentrates on the classical ML models. After evaluating 10 kinds of ML models, and considering that the Random Forest (RF) model [10] naturally conforms to the characteristics of probability selection, we finally chose it to detect idle VMs. Moreover, RF model has a recognized good

Algorithm 1. An automated sample augmentation algorithm

Input: The mean distribution (λ^k_m) and the deviation distribution (λ^k_d) of k-th indicator of labeled idle VM samples, total number of generated samples (N), total number of indicators (M) and the length of generated indicator series (L)

Output: New generated idle VM sample set (Θ)

Set i, $k \leftarrow 0$;
Sampling function Φ;
while $i \leq N$ **do**
  Initiate a new null sample θ_i;
  while $k \leq M$ **do**
    Generate an equivalent sequence s^k_i with length of L, $s^k_i[0:L] \leftarrow \Phi_1(\lambda^k_m)$;
    Update $s^k_i \leftarrow s^k_i + \Phi_L(\lambda^k_d)$;
    Update $s^k_i \leftarrow s^k_i + \Phi_L(\mathcal{N}(0, \sigma^2))$;
    if $\Phi_1(\mathcal{U}(0,1)) \leq p$ **then**
      New variables $x_1, x_2 \leftarrow \lfloor(\Phi_2(\mathcal{U}(0, L)))\rfloor$;
      Set $s^k_i[min(x_1, x_2) : max(x_1, x_2)] \leftarrow 0$;
    end if
    Add s^k_i to sample θ_i;
    $k \leftarrow k + 1$;
  end while
  Add θ_i to sample set Θ;
  $i \leftarrow i + 1$;
end while

interpretability, which is very friendly for users. Detailed experimental comparisons between different models can be found in Sect. 4.

3.5 Semi-supervised Training

Labeling VM as idle is actually a very labor-intensive task, which leads to a lack of positive samples, that is, the samples marked as idle. Consequently, the trained detection model tends to identify VMs as active, resulting in a low precision. To address this issue, we propose an empirical sample augmentation algorithm to automatically generate idle VM samples. The core principle of this algorithm is to generate new samples after certain data perturbation on the basis of the original marked sample data distribution. The pseudo code of the algorithm is shown in Algorithm 1.

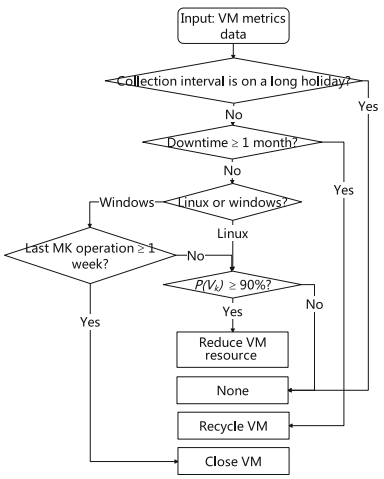


Fig. 5. Procedure of detecting and recommended dispose of idle VMs.

The input of this algorithm mainly includes: The mean distribution (λ_m^k) and the deviation distribution (λ_d^k) of k-th indicator, which are derived from manually labeled idle VM sample data; Total number of generated samples (N), total number of indicators (M) for identifying idle VMs and the length of generated indicator series (L). This algorithm outputs newly generated idle VM samples. For each indicator in any newly generated sample, this algorithm first produces an equivalent sequence s_i^k with length of L based on the mean distribution λ_m^k (Line 1–6). The symbol of $\Phi_l(\lambda)$ in Algorithm 1 represents the random generation of l values based on the distribution λ. It then superimposes on the sequence s_i^k a deviation sequence randomly generated according to the distribution λ_d^k (Line 7). In order to enhance the generalization ability of trained model, we superimposes on the sequence s_i^k a noise sequence randomly generated according to a normal distribution of $\mathcal{N}(0, \sigma^2)$ (Line 8). Besides, we further select a continuous partial

sequence from the sequence s_i^k and set its value as 0, which is used to simulate the actual data loss caused by machine failure and system upgrade (Line 9–12). The generated data is then gradually added to the sample set Θ (Line 13–17). The parameters of σ and p are set to 0.002 and 0.01 respectively.

After generating enough training and testing samples, we adopt 10-fold cross validation method to train the model. It should be emphasized that the samples detected online and marked by users will also be regularly added to the training set to improve the accuracy of the model.

3.6 Online Detection

According to long-term observation, we find that changes such as system upgrade and network adjustment occasionally occur in the production environment. It will lead to the loss of a large amount of monitoring data, and misleads the model to mistakenly identify active VMs as idle, resulting in large false positives. To distinguish the data loss caused by such large-scale changes from the data loss caused by small-range equipment failures, we define a support degree q to represent the proportion of VMs that lose data in the same time period. If q is greater than a predetermined threshold (such as 60%), events such as batch system upgrade are considered to occur in the production environment with a high probability. In this case, the detection service will directly filter out these loss data, that is, they will not participate in the detection process.

To facilitate SRE to handle the identified idle VMs, we not only use the predicted results of the detection model, but also design some rules based on the on/off state of the VM and the operation of the mouse and keyboard (MK operation) to comprehensively decide the disposal method for idle VMs. The detailed design is shown in Fig. 5. We will first simply ignore to detect those VMs that are in the long holiday period, such as National Day, Labor Day. Next, we will continue to judge the online duration of the VM. If the VM has been shut down for more than one month, we will regard it as a zombie VM and recommend recycling it directly. Otherwise, we will use the trained model to further infer whether the VM is idle. If $P(V_k)$ is greater than or equal to 90%, we believe that this VM has a high probability of being inactive, and recommend reducing its allocated resources. Note that we have added an additional judgment rule for those VMs with windows OS, that is, if the VM has not been operated through the keyboard and mouse for more than a week, we consider this VM to be inactive, and recommend shutting it down. All threshold parameters mentioned in Fig. 5 can be adjusted according to actual requirements.

4 Experiment and Evaluation

For comprehensively evaluating the effectiveness of IdleVMDetector, we first conduct a large number of experiments in an offline environment. In these experiments, we not only compare the impact of selecting different ML models on performance and overhead, but also illustrate the benefits of our proposed feature engineering technique and sample enhancement algorithm.

4.1 Experimental Setup

We totally compare 10 different popular ML models: Decision Tree (DT), Bagging, AdaBoost, Gradient Boosting Decision Tree (GBDT), Random Forest (RF), Support Vector Classification (SVC) with rbf kernel, Logistic Regression (LR), and 3 kinds of Multilayer Perceptron (MLP) models. MLP(a, b) means the model has b layers of neural networks, and each layer has a neurons. All these models are implemented by python-3.8.4 and sk-learn library [19]. The training set consists of 45 manually labeled positive samples (idle VMs), 65 negative samples (busy VMs), and 500 samples generated by the proposed sample augmentation algorithm. We also generate 500 labeled VMs for validation. Considering the high labor costs, these idle and busy VMs are determined by IT personnel purely based on their historical resource usage. That is, these labels do not really reflect whether these VMs is being used. Therefore, we generate additional 500 labeled samples to participate in the validation set to improve the accuracy of the evaluation. The performance metrics used in this work include: precision, recall and F1-Score, which have been widely used in various classification problems. The overhead indicators include time consumption of model training and prediction, as well as the resource occupancy when running online.

All experiments are conducted in a VM by taking use of 2 processes. The VM is configured with 8 CPU cores and 16G memory. The configuration of each CPU core is Intel(R) Xeon(R) Gold 5220R CPU @ 2.20GHz. To eliminate the influence of experimental error, each experiment is repeated by 10 times.

Table 2. Performance and time overhead comparison of IdleVMDetector using different ML models (The model we finally adopt is highlighted in red).

Evaluation metrics	DT	Bagging	AdaBoost	BGDT	RF	SVC	LR	MLP-(50, 1)	MLP-(50, 2)	MLP-(50, 3)
Precision	0.929	0.931	0.935	0.938	0.946	0.784	0.900	0.897	0.870	0.855
Recall	0.928	0.932	0.941	0.936	0.941	0.934	0.971	0.897	0.903	0.912
F1-Score	0.928	0.932	0.938	0.937	0.944	0.852	0.934	0.896	0.883	0.882
Average training time (s)	0.011	0.066	0.199	0.677	0.094	0.021	0.066	0.628	0.700	0.994
Average prediction time (s)	0.003	0.009	0.028	0.004	0.008	0.052	0.003	0.003	0.003	0.011

4.2 Comparison Between ML Models

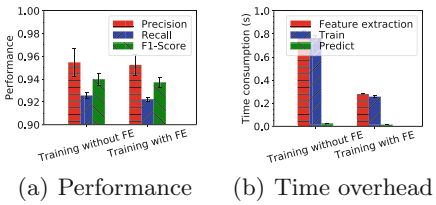
Table 4 shows the performance and overhead of the selected 10 kinds of ML models in identifying idle VMs. The experimental results show that the f1-score of the tree-based models (DT, Bagging, AdaBoost and GBT) is better than that of SVC and MLP models. The main reason for this phenomenon is that the features selected by our feature engineering are relatively independent, which is more suitable for tree-structured models to learn. That is also why LR model enables to achieve better F1-Score performance than SVC and MLP. On the contrary, SVC and MLP models will comprehensively consider the correlation between different features, leading to learning some irrelevant knowledge. Among these tree-based models, RF slightly outperforms other models ($1\% \sim 3\%$), and considering that the average time consumption of a single prediction of RF is extremely low (≤ 10 ms), we finally choose RF to detect idle VMs.

4.3 The Impact of Feature Engineering

According to the experimental results in Fig. 6, the efficiency of feature extraction, training, and prediction has been significantly improved after feature engineering. Among them, the time consumption of single feature extraction and prediction is reduced by nearly 65.2% compared to that before feature engineering. The reason for such progress is that a large number of useless features are eliminated through feature engineering, that is, only 15 of the original 2361 features are retained. More importantly, from Fig. 6(a), we can find that the removal of 99.4% features has little impact on the performance, and F1-Score drop rate is less than 1% after feature engineering. This is a good news for us and is within our expectations, because removing features also means that some valuable underlying information may be eliminated. It also proves that the feature engineering we have carried out not only improves the detection efficiency, but also preserves the valuable information in the original data as much as possible.

4.4 The Impact of Sample Augmentation Algorithm

Figure 7 shows the curve of the performance of IdleVMDetector changing with the number of constructed samples. When the number of newly generated samples is less than 1000, the detection performance improves significantly with the increase of the number of samples, the F1-Score of which is improved by about 17% compared to the model directly trained without generating any samples. The main reason for this improvement is that the newly generated samples not only solve the imbalance problem between positive and negative samples in the training process, but also expand the sample knowledge boundary that the model can learn, thus greatly improving the generalization ability of the model. This is also why the sample augmentation algorithm improves the performance of recall more significantly than the performance of accuracy. After the number of samples exceeds 1000, the marginal effect of the new knowledge that the model can learn decreases, so the detection performance gradually tends to be stable.



(a) Performance (b) Time overhead

Fig. 6. Comparison of effect on whether IdleVMDetector uses feature engineering (FE) techniques.

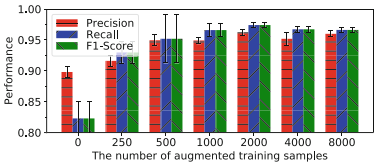


Fig. 7. Impact of the number of augmented training samples on detection performance.

5 Implementation and Deployment

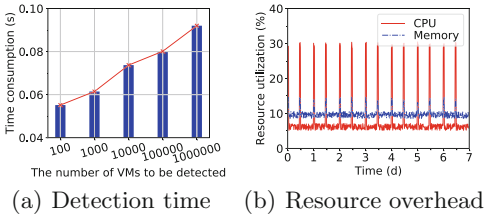
Considering that there are over 10K VMs in production environment, we have taken use of multi-process technology to improve the detection efficiency, and the entire service is packaged separately into a container image, managed by K8s [20], so as to ensure high scalability. In the process of interacting with the front-end, we also bring in the technique of asynchronous data transmission. The detection results will first be cached into Redis [21], and the front-end does not directly request the results from the detection service, but reads it from Redis. In order to assist SRE confirm the detection results, the front-end interface also provides some other statistical information of VM resources. The final interface interaction effect of our service is shown in Fig. 8.

We randomly record the results of one detection to introduce the effect of this service in production environment, where the number of configured process and the hardware configuration of the server are the same as the experimental configuration in Sect. 4.1. A total of nearly 3000 VMs are detected in this process, and among these detected VMs, only those VMs (133 VMs in total) with an identified idle probability greater than 95% are taken for disposal. All the detailed analysis results are presented in the following sections.

5.1 Performance Analysis

Table 3 shows 5 kinds of methods for handling the identified idle VMs. The results show that among all 133 identified idle VMs, 71.5% of the VMs have actually been dealt with, including 62.6% of VMs have been de-allocated, 2.2% have been shut down, 6.7% have been recycled. In addition, 16.5% of the VMs are not confirmed whether they really need to be handled due to user-related problems. The remaining 12% of the VMs have not been executed any action, because they all have special individual circumstances, for example, some VMs are newly created, so that their resources should remain unchanged in the short term. We summarize 6 typical scenarios where idle VMs should not be handled, as shown in Table 4. In fact, these idle VMs can be filtered out through the black

Fig. 8. An example of the effect of our idle VM detection service running online.



(a) Detection time (b) Resource overhead

Fig. 9. Analysis of IdleVMDetector in production environment.

and white list mechanism before service detection. Overall, these results strongly demonstrate the effectiveness of our proposed idle VM detection method.

5.2 Efficiency and Overhead Analysis

Figure 9(a) and Fig. 9(b) respectively show the change of detection time of IdleVMDetector as the VM increases and the resource overhead of running for a period of time. The results in 9(a) prove that the detection time spent by our method shows a linear growth trend with the expansion of the scale of VMs, so that it can be effectively extended to larger cloud environments. As we expected, the resource utilization of the service during running process shows obvious periodicity, and the peak of the resource utilization curve corresponds to the interval for performing idle VM detection. As shown in Fig. 9(b), the CPU resources and memory resources occupied by the service when detecting tens of thousands of VMs per round are less than 20% and 30% respectively, which satisfy the resource occupation requirements we have set in advance.

6 Related Work

The idle VM identification methods can be classified into two categories: Rule-based method and ML-based method. The former set several rules (or annotations) to identify idle VMs, and the latter takes use of several machine learning algorithms to train a model based on the predefined indicators' features, and use this trained model to infer whether a VM is idle.

Rule-Based Method: Early work identifies idle VMs by intuitively setting a static threshold [2,4] or a simple rule [3]. PULSAR [4] only considers the CPU resource of a VM when identifying this VM as an idle VM, and as long as the CPU resource of this VM is lower than a predefined certain threshold. Snadpiper [2] determines idle VM by extending one kind of VM resource into a combination of three utilization factors (CPU, memory, network usage). Differing from PULSAR and Snadpiper, the work [3] directly recognize the earliest instance as inactive when the utilization of data center is low enough. Due to its nature of simplicity and intuition, rule-based methods are usually the first choice for SREs. This is also the main reason why some recent works still choose rule-based detection methods [5,13,15]. Nevertheless, these methods lack an accurate perception of the change of VM resource occupancy, leading to varying degrees of false negatives and false positives in the process of judging idle VMs.

Table 3. Disposition methods actually performed for identified idle VMs.

Disposition methods	Resource degradation	Shutdown	Resource recycling	Do nothing	Unknown
Proportions	62.6%	2.2%	6.7%	12.0%	16.5%

Table 4. Several typical scenarios of idle VMs that cannot be disposed of.

Categories	Explanation
New environment	Newly purchased and deployed VMs by users.
Resource redundancy	Some VMs need to ensure sufficient resources to meet sudden business loads.
Load balance	The VMs adjusted by the load balancing mechanism.
Authorization requirement	The authorization of some application software is affected by the configured resource size.
System configuration requirement	The VMs that deploy distributed applications, such as Redis or Message Queue.
Test environment	The User Acceptance Test environment needs to be consistent with the user environment.

ML-Based Method: Compared to rule-based methods, ML-based methods provide a more powerful ability to learn VM behavior and its resource variation patterns. The work in [14] leverage kinds of primitive information (e.g., running process, login history, network connections) and a linear support vector machine (SVM) to find the fingerprints of inactive VMs. To better identify cloud garbage instances, the authors in [22] argue that simply using resource metrics to make judgments may be misleading, and then bring in a a weighted reference model based on application information to capture dependency information between users and cloud instances. The iCSI system [6] combines rule-based approaches and ML based approaches to improve the accuracy of idle VM identification. It also chooses SVM as the basic learning model, and additionally adds several judging rules based on the analysis of VM functions and VM network affinity analysis. The work [7] detect underutilized VM through comparing the results of weighted ensemble resource predictions with a predefined threshold. The authors in [8] compare various statistical metrics of all performance data with repeatedly tuned threshold coming from a decision tree learning algorithm to identify idle VM by producing an idleness score. Other researchers in [9] believe that idle VMs should have similar resource change patterns, and they thus propose a clustering method to determine idle VMs based on the VM resource utilization metrics.

Although we also adopt a ML model to infer idle probability, the biggest difference from the existing work is that they do not consider the lack of idle VM labels in real situations, making them difficult to be widely used. Moreover, we train different detection models for different types of VM OS. This is also not considered in the previous work.

7 Limitation

This work starts from the perspective of VM resource occupancy and aims to take use of the relevant metrics to indirectly infer whether VMs are inactive. During online detection process, the proposed system also utilizes some VM running status metrics, such as VM startup duration and the number of key-and-mouse operations, to improve decision-making efficiency. The proposed method conforms to the manual detection idea of SREs. We do not consider the business behavior characteristics of VMs (e.g. VM access relationship, TCP or UDP connections, and process remote calls), which will lead to some network resource-intensive VMs being misreported. Besides, our study focuses on resource usage rather than real business load, and therefore does not consider metrics that is able to more accurately reflect VM service load, such as disk IOPS or business-related system calls. Despite the fact that this proposed method does not take these factors into account, they can be easily extended by drawing into relevant metric data, which is also one of the main goals of our future work.

8 Conclusion

This paper systematically proposes a lightweight idle VM identification method based on a ML model. The proposed method has been successfully applied to a real cloud network environment. This method focuses on analyzing the relationship between the variation pattern of VM resource occupancy and idle status. Based on the observations from real VM data, we design 9 basic statistical features and 6 sequence fluctuation features, and bring in RF model to identify idle VMs. We present a sample augmentation algorithm to solve the problem of lack of positive samples, and introduce several online detection optimization strategies to improve the accuracy and robustness of identifying idle VMs. More importantly, we also summarize 6 scenarios where idle VMs unable to be processed, and further provide some thoughts on optimizing the accuracy of idle VM detection, which will also be the main point in our future work.

This system is deployed in the form of micro-services. Sufficient experiments demonstrate the effectiveness of our proposed idle VM detection method. The system has been running online for several months, showing excellent performance and has helped Sangfor cloud to save large VM resources by far.

Acknowledgment. This work was supported by Cloud Security Key Technology Research Key Laboratory of Shenzhen under grant No.ZDSY20200811143600002, the National Key R&D Program of China (No. 2021YFB3300200), National Natural Science Foundation of China (No. 62072451, 92267105), Guangdong Special Support Plan (No. 2021TQ06X990), and Shenzhen Basic Research Program (No. JCYJ20200109115418592, JCYJ20220818101610023). We would like to express thanks to the reviewers and editors for their constructive comments and suggestions. We also thank anyone who helped us improve this work.

References

1. Insight, Managed cloud services. https://www.insight.com/en_US/glossary/m/managed-cloud-services.html (2022)
2. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and gray-box strategies for virtual machine migration. In: NSDI, vol. 7, pp. 17–17 (2007)
3. Calheiros, R.N., Ranjan, R., Buyya, R.: Virtual machine provisioning based on analytical performance and QOS in cloud computing environments. In: 2011 International Conference on Parallel Processing. IEEE, 2011, pp. 295–304 (2011)
4. Breitgand, D., et al.: An adaptive utilization accelerator for virtualized environments. In: 2014 IEEE International Conference on Cloud Engineering. IEEE, 2014, pp. 165–174 (2014)
5. Chunlin, L., Hammad-Ur-Rehman, Q.: Adaptive threshold detection based on current demand for efficient utilization of cloud resources. In: 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS). IEEE, 2019, pp. 341–346 (2019)
6. IKim, I.K., Zeng, S., Young, C., Hwang, J., Humphrey, M.: iCSI: A cloud garbage VM collector for addressing inactive VMs with machine learning. In: 2017 IEEE International Conference on Cloud Engineering (IC2E) (2017)
7. Mazidi, A., Mahdavi, M., Roshanfar, F.: An autonomic decision tree-based and deadline-constraint resource provisioning in cloud applications. Concurr. Comput.: Pract. Exp. **33**(10), e6196 (2021)
8. Khandros, M., et al.: Machine learning computing model for virtual machine underutilization detection. Jul. 6 2021, uS Patent 11,055,126 (2021)
9. Gopisetti, A., Jha, C., George, J.R., Gaurav, K., Singh, J.: Usage pattern virtual machine idle detection. Feb. 10 2022, uS Patent App. 17/510,546
10. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
11. Lindner, M., McDonald, F., McLarnon, B., Robinson, P.: Towards automated business-driven indication and mitigation of vm sprawl in cloud supply chains. In: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM: and Workshops. IEEE 2011, pp. 1062–1065 (2011)
12. Microsoft Azure, What is virtual desktop infrastructure (vdi)? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-virtual-desktop-infrastructure-vdi/ (2022)
13. Fesl, J., Gokhale, V., Feslová, M.: Efficient virtual machine consolidation approach based on user inactivity detection. Cloud Comput. **2019**, 115 (2019)
14. Kim, I.K., Zeng, S., Young, C., Hwang, J., Humphrey, M.: A supervised learning model for identifying inactive vms in private cloud data centers. In: Proceedings of the Industrial Track of the 17th International Middleware Conference, 2016, pp. 1–7 (2016)
15. Zhang, B., Al Dhuraibi, Y., Rouvoy, R., Paraiso, F., Seinturier, L.: Cloudgc: Recycling idle virtual machines in the cloud. In: 2017 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2017, pp. 105–115 (2017)
16. Christ, M., Braun, N., Neuffer, J., Kempa-Liehr, A.W., et al.: Tsfresh. https://tsfresh.readthedocs.io/en/latest/text/introduction.html (2022)
17. Li, Y.: Research and application of deep learning in image recognition. In: 2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA). IEEE, 2022, pp. 994–999 (2022)
18. Li, Q., et al.: A survey on text classification: from traditional to deep learning. ACM Trans. Intell. Syst. Technol. (TIST) **13**(2), 1–41 (2022)

19. Scikit-learn.org: Scikit-learn. https://scikit-learn.org/stable/ (2022)
20. Kubernetes Enterprise: Kubernetes (k8s). https://kubernetes.io/ (2022)
21. Redis Enterprise: Reids. https://redis.io/ (2022)
22. Shen, Z., Young, C.C., Zeng, S., Murthy, K., Bai, K.: Identifying resources for cloud garbage collection. In: 2016 12th International Conference on Network and Service Management (CNSM) (2016)