Anomaly Detection and Diagnosis for Container-based Microservices with Performance Monitoring

Qingfeng Du, Tiandi Xie and Yu He

August 30, 2018

Anomaly Detection and Diagnosis for Container-based Microservices with Performance Monitoring

Qingfeng Du, Tiandi Xie, and Yu He

School of Software Engineering, Tongji University, Shanghai, China
{du_cloud, xietiandi, rainlf}@tongji.edu.cn

Abstract. With emerging container technologies, such as Docker, microservices -based applications can be developed and deployed in cloud environment much agiler. The dependability of these microservices becomes a major concern of application providers. Anomalous behaviors which may lead to unexpected failures can be detected with anomaly detection techniques. In this paper, an anomaly detection system(ADS) is designed to detect and diagnose the anomalies in microservices by monitoring and analyzing real-time performance data of them. The proposed ADS consists of a monitoring module that collects the performance data of containers, a data processing module based on machine learning models and a fault injection module integrated for training these models. The fault injection module is also used to assess the anomaly detection and diagnosis performance of our ADS. Clearwater, an open source virtual IP Multimedia Subsystem, is used for the validation of our ADS and experimental results show that the proposed ADS works well.

Keywords: Anomaly detection · Microservices · Performance monitoring · Machine learning.

1 Introduction

At present, more and more Web applications are developed in microservice design for better scalability, flexibility and reliability. An application in microservice approach consists of a collection of services which is isolated, scalable and resilient to failure. Each service can be seen as an application of its own and these services expose their endpoints for communicating with other services. With the adoption of a microservice architecture, a lot of benefits can be got. For example, software can be released faster, and teams can be smaller and focus on their own work.

To generate enough isolated resources for such a number of services, the following virtualization techniques are widely used. Virtual Machines(VMs) are traditional ways of achieving virtualization. Each created VM has its own operating system(OS). Container is another emerging technology for virtualization which is gaining popularity over VMs due to its lightweight, high performance, and higher scalability[1]. And the created containers share host OS together.

The development of virtualization technologies, especially container technology, has contributed to the wide adoption of microservice architecture in recent years. And the service providers start to put greater demands on the dependability of these microservices. Service Level Agreements (SLAs) are usually made between service providers and users for specifying the quality of the provided services. They may include various aspects such as performance requirements and dependability properties[2]. And severe consequences may be caused by a violation of such SLAs.

Anomaly detection can help us identify unusual patterns which do not conform to expected patterns and anomaly diagnosis can help us locate the root cause of an anomaly. As anomaly detection and diagnosis require large amount of historic data, service providers have to install lots of monitoring tools on their infrastructure to collect real-time performance data of their services.

At present, there are two main challenges faced by these microservice providers. Firstly, for container-based microservices, what metrics should be monitored. Secondly, even if all the metrics are collected, how to evaluate whether the behaviors of the application are anomalous or not.

In this paper, an anomaly detection system(ADS) is proposed and it can address these two main challenges efficiently. The proposed ADS gives a prototype for service providers to detect and diagnose anomalies for container-based microservices with performance monitoring.

The paper is organized as follows: Section II reviews the technical background and some widely used anomaly detection techniques. Section III first presents our ADS and its three main components. Section IV presents the implementation of the proposed ADS in detail. Section V provides validation results of the proposed ADS on the Clearwater case study. Section VI concludes the contribution and discusses the future work.

2 Background and Related Works

2.1 Backgroud

Microservice architecture is a cloud application design pattern which shifts the complexity away from the traditional monolithic application into the infrastructure[3]. In comparison with a monolithic system, microservices-based arhitecture creates a system from a collection of small services, each of which is isolated, scalable and resilient to failure. Services communicate over a network using language-agnostic application programming interfaces (API).

Containers are lightweight OS-level virtualizations that allow us to run an application and its dependencies in a resource-isolated process. Each component runs in an isolated environment and does not share memory, CPU, or the disk of the host operating system(OS)[4]. With more and more applications and services deployed on cloud hosted environments, microservice architecture depends heavily on the use of container technology.

Anomaly detection is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset[5]. In a normal situation, the correlation between workloads and application performance should be stable and it fluctuates significantly when faults are triggered[6].

2.2 Related Work

With widely adoptions of microservice architecture and container technologies, performance monitoring and performance evaluation become a hot topic for the containers' researchers. In [7], the authors evaluated the performance of container-based microservices in two different models with the performance data of CPU and network. In [8][9], the authors provided a performance comparison among a native Linux environment, Docker containers and KVM(kernel-based virtual machine). They drew an conclusion that using docker could achieve performance improvement according to the performance metrics collected by their benchmarking tools.

In [2], the authors presented their anomaly detection approach for cloud services. They deployed a cloud application which consisted of several services on several VMs and each VM ran a specific service. The performance data of each VM was collected and then, processed for detecting possible anomalies based on machine learning techniques. In [6], the authors proposed an automatic fault diagnosis framework called FD4C. The framework was designed for cloud applications and in the state-of-the-art section, the authors presented four typical periods in their FD4C framework including system monitoring, status characterization, fault detection and fault localization. In [10][11][12], the authors paid attention to the system performance. To detect anomalies ,they built models with historical performance metrics and compared them with online monitored ones. However, these methods require domain knowledge (e.g. the system internal structure). Although these papers only focus on VM-level monitoring and fault detection, they give us much food for thought and methods can be used in container-based microservices similarly.

This paper is aimed at creating an ADS which can detect and diagnose anomalies for container-based microservices with performance monitoring. The proposed ADS consists of three modules: a monitoring module that collects the performance data of containers, a data processing module which detects and diagnoses anomalies, and a fault injection module which simulates service faults and gathers datasets of performance data representing normal and abnormal conditions.

3 Anomaly Detection System

This section overviews our anomaly detection system. There are three modules in our ADS. Firstly, the monitoring module collects the performance monitoring data from the target system. Then, the data processing module will analyze the collected data and detect anomalies. The fault injection module simulates service faults and gathers datasets of performance monitoring data representing normal and abnormal conditions. The datasets are used to train machine learning models, as well as to validate the anomaly detection performance of the proposed ADS.

For the validation of our ADS, a target system composed of several container-based microservices is deployed on our container cluster. The performance monitoring data of the target system are collected and processed for detecting possible anomalies.

Usually, a user can only visit the exposed APIs from upper application and can not access the specific service deployed on the docker engine or VM directly. Thus, our ADS is not given any a priori knowledge about the relevant features which may cause anomalous behaviors. The proposed ADS has to learn from the performance monitoring data with machine learning models itself.

3.1 Monitoring Agent

A container-based application can be deployed not only on a single host but also on multiple container clusters[13]. Each container cluster consists of several nodes(hosts) and each node holds several containers. For applications deployed in such container-based environments, performance monitoring data should be collected from various layers of an application(e.g., node layer, container layer and application layer). Our work is mainly focused on the container monitoring and microservice monitoring.

Container monitoring Different services can be added into a single container, but in practice, it's better to have many small containers than a large one. If each container has a tight focus, it's much easier to maintain your microservices and diagnose issues. In this paper, container is defined as a group of one or more containers constituting one complete microservice, it's same as the definition of pod in Kubernetes. By processing the performance data of a container, we can tell whether the container works well.

Microservice monitoring In this paper, a container contains only one specific microservice and a microservice can be deployed in several containers at the same time. By collecting the performance data of all the related containers, we can obtain the total performance data of a specific microservice. And we can also know whether a microservice is anomalous by processing these service performance data.

3.2 Data Processing

Data processing tasks Data processing helps us to detect and diagnose anomalies. Carla et al defined an anomaly as the part of the system state that may lead to an SLAV[2]. We use the same definition of anomaly as stated in Carla's work. An anomaly can be a CPU hog, memory leak or package loss of a container which runs a microservice because it may lead to an SLAV. In our work, there are two main tasks: classify whether a microservice is experiencing some specific anomaly and locate the anomalous container when an anomaly occurs.

Data processing models Anomaly detection techniques are based on machine learning algorithms. There are mainly three types of machine learning algorithms: supervised, unsupervised and semi-supervised. All of these algorithms can be applied to classify the behaviors of the target system with performance monitoring data.

To detect different types of the anomalies which may lead to SLAVs, supervised learning algorithms are used. In our ADS, supervised learning algorithm consists of two phases, shown in Fig. 1 and Fig. 2.

Fig. 1 shows the training phase. It demonstrates how classification models are created. Firstly, samples of labelled performance data representing different service behaviors are collected and stored in a database. These samples are called training data. Then, data processing module trains the classification models with these training data. To simulate actual users requests, a workload generator is deployed. To collect more performance data in different types of errors, a fault injection module is deployed and it will inject different faults into containers. With more samples collected, the model will be more accurate.

The second phase is the detection phase. Once the model is trained, some real-time performance data can be collected and transferred to data processing module as inputs, and the data processing module can detect anomalies occurring in the system with the trained model. For the validation of the data processing module, some errors will be injected to the target system, and then the data processing module uses the real-time performance data to detect these errors.

The anomaly of a service is often caused by the anomalous behaviors of one or more containers belong to this service. To find out whether the anomaly is caused by some specific container, time series analysis is used. If several containers run a same microservice, they should provide equivalent services to the users. The workload and the performance of each container should be similar. For this reason, if an anomaly is detected in a microservice, the time series data of all the containers running this microservice will be analyzed. The similarity among the data will be measured and the anomalous container will be found.

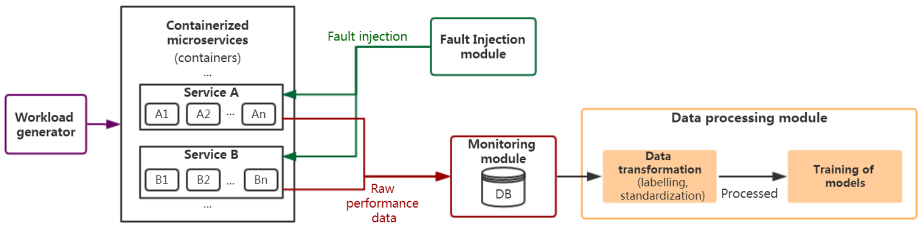

Fig. 1. Training phase, offline.

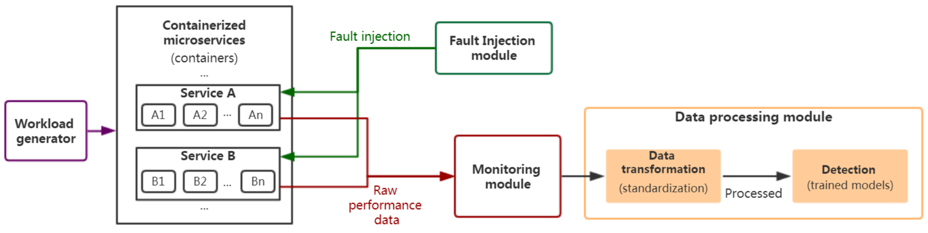

Fig. 2. Detection phase, online.

3.3 Fault Injection

Fault injection module is integrated for collecting the performance data in various system conditions and training the machine learning models. To simulate real anomalies of the system, we write scripts to inject different types of faults into the target system. Four types of faults are simulated based on the resources they impact: high CPU consumption, memory leak, network package loss and network latency increase.

This module is also used to assess the anomaly detection and diagnosis performance of our ADS. As shown in Fig. 2, after the classification models are trained, the fault injection module injects same faults to the target system, and real-time performance data are processed by the data processing module. The detection results are used for the validation.

4 Implementation

This section presents the implementation of the three modules of the proposed anomaly detection system.

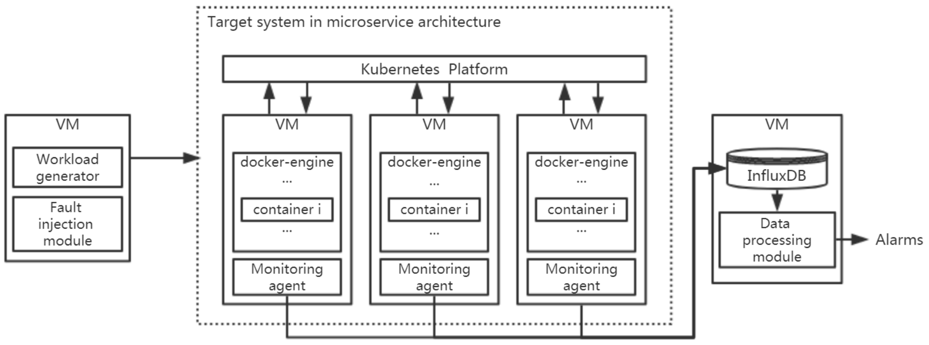

Fig. 3. Implementation of the ADS.

A prototype of the proposed ADS is deployed on a virtualized platform called Kubernetes. As shown in Fig. 3, the platform is composed of several VMs. VMs are connected through an internal netowrk. A target system in microservice architecture is deployed on the platform for the validation and the target system consists of several containers running on different VMs. The monitoring module installs a monitoring agent on each VM for collecting real-time performance data and stores the collected data in a time-series database called InfluxDB. The data processing module gets the data from the database and executes processing tasks with the data. The fault injection module and the workload generator work by executing bash scripts on another VM.

4.1 Monitoring module

As shown in Fig. 3, a monitoring agent is deployed on each of the VM. In a monitoring agent, several open-source monitoring tools are used for collecting and storing performance metrics of the target system such as cAdvisor and Heapster. CAdvisor collects resource usages and performance monitoring data of all the containers while Heapster groups these data and stores in a time series database called InfluxDB. The metrics in table 1 are collected for each service and container including CPU metrics, memory metrics and network metrics.

Table 1. Monitoring metrics

Metric name	Description
cpu/usage	Cumulative CPU usage on all cores.
cpu/request	CPU request (the guaranteed amount of resources) in millicores.
cpu/usage-rate	CPU usage on all cores in millicores.
cpu/limit	CPU hard limit in millicores.
memory/usage	Total memory usage.
memory/request	Memory request (the guaranteed amount of resources) in bytes.
memory/limit	Memory hard limit in bytes.
memory/working-set	Total working set usage. Working set is the memory being used and not easily dropped by the kernel.
memory/cache	Cache memory usage.
memory/rss	RSS memory usage.
memory/page-faults	Number of page faults.
memory/page-faults-rate	Number of page faults per second.
network/rx	Cumulative number of bytes received over the network.
network/rx-rate	Number of bytes received over the network per second.
network/rx-errors	Cumulative number of errors while receiving over the network.
network/rx-errors-rate	Number of errors while receiving over the network per second.
network/tx	Cumulative number of bytes sent over the network.
network/tx-rate	Number of bytes sent over the network per second.
network/tx-errors	Cumulative number of errors while sending over the network.
network/tx-errors-rate	Number of errors while sending over the network.

4.2 Data processing module

The data processing module executes the two tasks for each service as discussed in Section III. The classification models are trained with four algorithms included in library scikit-learn. The results are shown in Section V.

- Support Vector Machines (configured with kernel=linear)
- Random Forests (configured witih max_depth=5 and n_estimators=50)
- Naive Bayes
- Nearest Neighbors (configured with k=5)

After the detection phase, the anomalous service and the type of the anomaly can be got(e.g. CPU hog in Service A). Next, the anomalous containers should be diagnosed. If there is only one container running the anomalous service, it can be diagnosed as the anomalous container directly. However, if several containers are running the anomalous service, an algorithm is needed to diagnose the anomalous one. Clustering of time series data is a good solution and some algorithms can be used easily[14][15]. However, clustering needs a large amount of data, and people seldom deploy such a number of containers. In this case, we assume that there is only one anomalous container at the same time.

The distance between two temporal sequences $\mathbf{x} = [x_1, x_2, ..., x_n]$ and $\mathbf{y} = [y_1, y_2, ..., y_n]$ can be computed via Euclidean distance very easily. However, the length of the two given temporal sequences must be the same. DTW algorithm is a better choice to measure the similarity between two temporal sequences. It finds an optimal alignment between two given sequences, warps the sequences based on the alignment, and then, calculates the distance between them. DTW algorithm has been successfully used in lots of fields such as speech recognition and information retrieval.

In this paper, DTW algorithm is used to measure the similarity between the time series performance data of the given containers. Once an anomalous metric in a service is detected, the time series data of all the containers running that service will be analyzed by the algorithm. And the most anomalous container which has the maximal distance from the others will be found.

4.3 Fault injection module

An injection agent is installed on each container of a service. Agents are run and stopped through an SSH connection and they simulate CPU faults, memory faults and network faults by some software implementations.

CPU and memory faults are simulated using a software called Stress. Network latency and package loss are simulated using a software called Pumba.

Injection procedures are designed after the implementation of the injection agents. To create a dataset with various types of anomalies in different containers, an algorithm is designed and shown in 1. After the injection procedure is finished, the collected data are used to create anomaly datasets.

Algorithm 1 Fault injection procedure.

```
Input: $container\_list, fault\_type\_list, injection\_duration, pause\_time, workload$
1: $GenerateWorkload(workload)$
2: for $container$ in $container\_list$ do
3:     for $fault\_type$ in $fault\_type\_list$ do
4:         $injection =$
           $Injection(fault\_type, injection\_duration)$
5:         $inject\_in\_container(container, injection)$
6:         $sleep(pause\_time)$
7:     end for
8: end for
```

5 Case Study

5.1 Environment description

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. The target system(Clearwater) runs on a kubernetes platform which consists of three VMs(which are rain-u2, rain-u3 and rain-u4). Each VM has 4 CPUs, a 8 GB memory and a 80 GB disk. VMs are connected through a 100 Mbps network. A monitoring agent is installed on each of the VM. The installed monitoring tools include cAdvisor, Heapster, InfluxDB and Grafana.

Clearwater is an open source implementation of IMS (the IP Multimedia Subsystem) designed from the ground up for massively scalable deployment in the Cloud to provide voice, video and messaging services to millions of users[16]. It contains six main components, namely Bono, Sprout, Vellum, Homer, Dime and Ellis. On our kubernetes platform, each container runs a specific service and can be easily scaled out. In this paper, our work is focused on Sprout, Cassandra and Homestead constituting the Call/Session Control Functions(CSCF) together, and we perform experimentations for these three services.

5.2 Clearwater experimentations

First of all, Clearwater is deployed on our kubernetes platform. All the services are running in containers and the number of the replica of component homestead is set to three. It means there will be three containers running the same service homestead. The performance data of a service is the sum of all the containers running this service.

Then, two datasets(dataset A and dataset B) are collected with the help of the fault injection module. The injection procedures are shown in Table 2. By combining the two datasets together, a third dataset can be obtained as dataset C. After being standardized and labelled, a dataset has a structure as shown in Table 3.

Since these three services constitute the CSCF function together, there will be some relationships among their performance data. And the question whether we can detect the anomalies with the performance data of only one service comes. To answer this question, each dataset is divided to three smaller datasets according to the service, and the classification algorithms are also executed on these datasets for the validation. The structure of the divided dataset is shown in Table 4.

Table 2. Fault injection procedures

Experiment	Injection procedures

dataset A	container_list = {sprout,cassandra,homestead1}
	fault_type = {CPU, memory,latency,package_loss}
	injection_duration = 50min
	pause_time = 10min
	workload = workloadA(5000 calls per second)

dataset B	container_list = {sprout,cassandra,homestead1}
	fault_type = {CPU, memory,latency,package_loss}
	injection_duration = 30min
	pause_time = 10min
	workload = workloadB(8000 calls per second)

Table 3. Dataset structure

Time	Cassandra CPU	Cassandra Mem	other metrics	Homestead metrics	Sprout metrics	label
2018-05-08T09:21:00Z	512	70142771	...	...	...	nomal
2018-05-08T09:21:30Z	350	120153267	...	...	...	cass_mem_leak
2018-05-08T09:22:00Z	322	70162617	...	...	...	sprout_cpu_hog

Table 4. Service dataset structure

Time	Cassandra CPU	Cassandra Mem	other metrics	label
2018-05-08T09:21:00Z	512	70142771	...	nomal
2018-05-08T09:21:30Z	350	120153267	...	cass_mem_leak
2018-05-08T09:22:00Z	322	70162617	...	sprout_cpu_hog

As we inject four different types of faults to three different services, there will be 12 different labels. We also collect the data in a normal condition and in a heavy workload, thus, there are 14 different labels totally in these datasets.

5.3 Validation results

Detection of anomalous service Four widely used algorithms are compared in this paper for training the classification models of our datasets, which are Support Vector Machine(SVM), Nearest Neighbors(kNN), Naive Bayes(NB) and Random Forest(RF). The purpose of these classifiers is to find out the anomalous service with the monitored performance data.

There are 757 records in dataset A, 555 records in dataset B, and 1312 records in dataset C. For each of the dataset, 80 percent of the records are used as training set to train the classification model and the rest 20 percent are used as test set to validate the model. The validation results are shown in Table 5 and 6.

Regarding the validation results in Table 5, the detection performance of the anomalous service is excellent for most of the classifiers with measure values above 0.9. For dataset A, all of the four classifier give excellent validation results. For dataset B, three of these classifiers give wonderful results except SVM. For dataset C, the performance of Random Forest and Nearest Neighbors still look excellent. These results shows that the dataset created by our ADS is meaningful, and both of the Random Forest and Nearest Neighbors classifiers have excellent detection performance.

To answer the question whether anomalies can be detected from the performance data of only one related service, we performed same experiments on the three divided datasets from dataset C. The classification results of the divided datasets(shown in Table 6) are not as good as the results using the entire dataset. However, Nearest Neighbors classifier still gives satisfying results on all of the three divided datasets. SVM seems to be the worst because it doesn't perform well on datasets with multiple classes. Consequently, Nearest Neighbors classifier is recommended if you have to use a dataset with only one service.

Table 5. Validation results of three datasets

Dataset	Measure	kNN	SVM	NB	RF
A	Precision	0.93	0.95	0.95	0.95
	Recall	0.93	0.92	0.93	0.92
	F1-score	0.93	0.93	0.93	0.92
B	Precision	0.98	0.75	0.98	0.99
	Recall	0.97	0.82	0.97	0.99
	F1-score	0.97	0.77	0.97	0.99
C	Precision	0.96	0.82	0.83	0.93
	Recall	0.96	0.80	0.79	0.91
	F1-score	0.96	0.78	0.78	0.91

Diagnosis of anomalous container The network latency anomaly of container homestead-1 is used for the validation. As discussed previously, there are

Table 6. Validation results of three services in dataset C

Service	Measure	kNN	SVM	NB	RF
Cassandra	Precision	0.91	0.48	0.61	0.89
	Recall	0.89	0.35	0.51	0.75
	F1-score	0.90	0.33	0.50	0.76
Homestead	Precision	0.92	0.27	0.56	0.71
	Recall	0.90	0.36	0.48	0.72
	F1-score	0.91	0.28	0.46	0.69
Sprout	Precision	0.88	0.31	0.47	0.85
	Recall	0.86	0.33	0.46	0.78
	F1-score	0.86	0.28	0.42	0.79

three containers running the service homestead. As a microservice application, the workload and the performance data of these three containers should be similar. Thus, the container with the furthest distance from others will be considered as the anomalous container. A python program is implemented to help us diagnose the anomalous container, and it gets the latest 20 performance data from the InfluxDB, calculates the distance and shows the result as shown in Fig. 4.

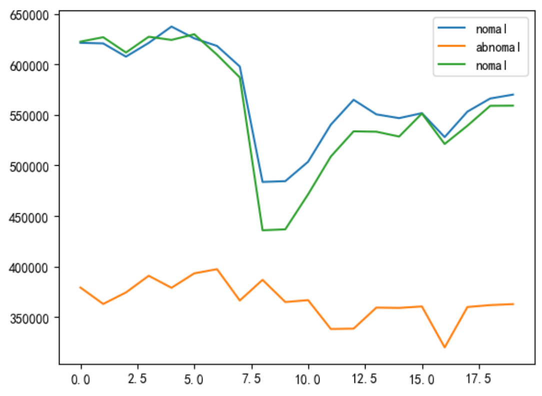

Fig. 4. Diagnosis of the anomalous container.

6 Conclusion and future work

In this paper, we analyzed the performance metrics for container-based microservices, introduced two phases for detecting anomalies with machine learning techniques, and then, proposed an anomaly detection system for container-based microserivces. Our ADS relies on the performance monitoring data of services and containers, machine learning algorithms for classifying anomalous and normal behaviors, and the fault injection module for collecting performance data in various system conditions.

In future, a more representative case study in microservice architecture will be studied. Currently, the fault injection module only focused on some specific hardware fault, and in future, some complicated injection scenarios can be added in this module.

References

1. Vindeep Singh and et al. Container-based microservice architecture for cloud applications. *Computing, Communication and Automation (ICCCA)*, 2017.
2. Carla Sauvanaud and et al. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, 2018.
3. Grzegorz Dwornicki Rusek, Marian and Arkadiusz Orowski. A decentralized system for load balancing of containerized microservices in the cloud. *International Conference on Systems Science. Springer, Cham, 2016.*, 2016.
4. Nane. Kratzke. About microservices, containers and their underestimated impact on network performance. *arXiv preprint arXiv:1710.04049(2017).*, 2017.
5. A.; Kumar V. Chandola, V.; Banerjee. Anomaly detection: A survey. *ACM Computing Surveys.*, 2009.
6. Tao Wang, Wenbo Zhang, Chunyang Ye, and et al. Fd4c: Automatic fault diagnosis framework for web applications in cloud computing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(1):61–75, 2016.
7. Marcelo Amaral, Jorda Polo, and et al. Performance evaluation of microservices architectures using containers. In *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, pages 27–34. IEEE, 2015.
8. A. Ferreira W. Felter and et al. An updated performance comparison of virtual machines and linux containers. *Technical Report RC25482(AUS1407-001), IBM*, 2014.
9. J. Kjallman R. Morabito and M. Komu. Hypervisors vs. lightweight virtualization: A performance comparison. *IEEE International Conference on Cloud Engineering,*, 2015.
10. Z. Zheng Y. Zhang and M.R. Lyu. An online performance prediction framework for service-oriented systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 2014.
11. Haibo Mi, Huaimin Wang, and et al. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1245–1255, 2013.
12. Shigang Zhang, Krishna R Pattipati, and et al. Dynamic coupled fault diagnosis with propagation and observation delays. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(6):1424–1439, 2013.
13. Claus. Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2015.
14. T Warren Liao. Clustering of time series dataa survey. *Pattern recognition*, 38(11):1857–1874, 2005.
15. Yanping Chen, Keogh, and et al. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.
16. Clearwater. Project clearwater. `http://www.projectclearwater.org/`.