An approach of collecting performance anomaly dataset for NFV Infrastructure

Qingfeng Du, Yu He and Tiandi Xie

September 8, 2018

An approach of collecting performance anomaly dataset for NFV Infrastructure

Qingfeng Du, Yu He, Tiandi Xie, Kanglin Yin, and Juan Qiu

School of Software Engineering, Tongji University
Software Engineering RD Centre, Jishi Building, Tongji University
Shanghai, China
https://github.com/XLab-Tongji
{du_cloud, rainlf, xietiandi, 14_ykl, Juan_qiu}@tongji.edu.cn

Abstract. Network Function Virtualization(NFV) technology is widely used in industry and academia. Meanwhile, it brings a lot of challenges to the NFV applications' reliability, such as anomaly detection, anomaly location, anomaly prediction and so on. All of these studies need a large number of anomaly data information. This paper designs a method for collecting anomaly data from Infrastructure as a Service(IaaS), and constructs an anomaly database for NFV applications. Three types of anomaly datasets are created for anomaly study, including datasets of workload with performance data, fault-load with performance data and violation of Service Level Agreement(SLA) with performance. In order to simulate an anomaly in a production environment better, we use Kubernetes to build a distributed environment, and to accelerate the occurrence of anomalies, a fault injection system is utilized. Our aim is to provide more valuable anomaly data for reliability research in NFV environments.

Keywords: Anomaly database· NFV· Kubernetes· IaaS· Clearwater· Performance monitoring· Fault injection.

1 Introduction

Network Function Virtualization(NFV) is becoming more and more popular. Many Communication Service Providers(CSP) have begun to migrate applications to Network Functions Virtualization(NFV) environment[1]. Detection of anomaly and anomaly location is very important for providing better network services. It is necessary to predict anomalies in some special circumstances. It needs to analyze the rules and connections in a large number of anomaly data. But in production environment, the cost of collecting these data is expensive. So it is meaningful to collect these anomaly data for research in the experimental environment.

At present, there are many databases for anomaly data, such as KDD CUP 99 dataset[1], NAB dataset[2], Yahoo Webscope S5 dataset[3], and so on. All of these could be a benchmark for evaluating algorithms for anomaly detection. But these datasets also exist some restrictions, like single label, data redundancy and so on. On this basis, we collect anomaly data from three different perspectives.

In NFV environment, the cause of the failure is not single. In order to describe different exceptions more accurately, the multiple types of fault tags are necessary. Our method uses fault injection system to specify fault types of anomaly data, making datasets more suitable to deal with the problem of multiple classification in machine learning[2].

In addition, the malfunction of system resources can also lead to system anomaly happen, the pressure of users on system workload will also lead to system anomaly behavior[3]. In production environment, increase of users may be an important factor leading to anomaly service compared to the occurrence of hardware anomaly events. Our method also collects anomaly data under different workload.

In NFV applications, the typical quality of service index is Service Level Agreement(SLA)[4]. When a violation of SLA occurs, it represents an anomaly service. Our method also collects performance data under different SLA level. It helps researcher to analyze the relationship between a occurrence of SLA violation and performance data of IaaS in a system.

At last, we propose several machine learning models based on supervised learning to detect SLAs of VNFs and anomaly in IaaS. And compare the experimental results of each model. The result of the comparison between the models show that our anomaly database has a certain reference value in the anomaly detection with VNFs Environment.

The paper is organized as follows: Section 2 introduces the technical background and our related work in the construction of the anomaly database. Section 3 introduces the architecture of the data collection. Section 4 shows the implementation of our experiment. Section 5 provides a classical case study of Clearwater project[5], gives a detailed description of the building of the anomaly database. And at last, we summarizes the contribution and discuss the future work in Section 6.

2 Background and Related Work

With the development of Internet applications and the maturity of hardware virtualization, The emergence of Infrastructure as a Service (IaaS) [4] provides the underlying hardware support for this architecture. It makes network providers

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[2]https://github.com/numenta/NAB
[3]https://webscope.sandbox.yahoo.com/catalog.php?datatype=s
[4]https://en.wikipedia.org/wiki/Service-level_agreement
[5]http://www.projectclearwater.org/

do not need care about the details of the underlying hardware devices, and concentrate on providing upper level services.In this context, Virtual Network Functions (VNFs) represent any virtual execution environment configured to provide a given network service. VNFs are often structured in several components each one hosted on single VMs.

The existing anomaly databases collect a lot of anomaly data in different fields. KDD CUP 99 dataset is used for network attack diagnosis. Each of its data records whether or not it has been attacked at the moment. It means that there are only one label in dataset, normal or anomaly.

Even Mahbod Tavallaee and his collaborator further optimized KDD CUP 99 dataset called NSL-KDD, it still has the same limitations[5]. This paper provides a disturbance system to specify the type of fault load to analyze the influence of different fault types on the performance of the tested system.

Markus Thill present a comparative study where several online anomaly detection algorithms are compared on the large Yahoo Webscope S5 anomaly benchmark[6]. But the yahoo Webscope S5 dataset is more suitable for time series analysis. It continues to have some limitations for the classification of different faults. We present a new approach to collecting performance data that with fault label. It has more advantages in the classification problem of anomaly detection.

In this paper, we integrate common single fault time series analysis problems and multiple fault classification problems in complex systems, propose corresponding performance data collection system and disturbance system. Then establish varied dataset in our anomaly database, Provide reference for fault analysis in different scenes. The details is shown in our site[6].

3 Architecture of data collection

This section outlines the framework of our performance data collection. In order to accurately collect data that with a fault type label, the framework consists of three systems, target application system (target system), disturbance system and performance monitoring system (monitoring system), as shown in Figure 1.

3.1 Target system

Target system is a NFV application system, which is software implementations of network functions that can be deployed on a network functions virtualization infrastructure (NFVI). NFVI is the totality of all hardware and software components that build the environment where VNFs are deployed.

3.2 Disturbance system

The core function of the disturbance system is fault injection[7][8], it is used to accelerate the occurrence of anomaly events in the target system, such as

[6]https://github.com/XLab-Tongji

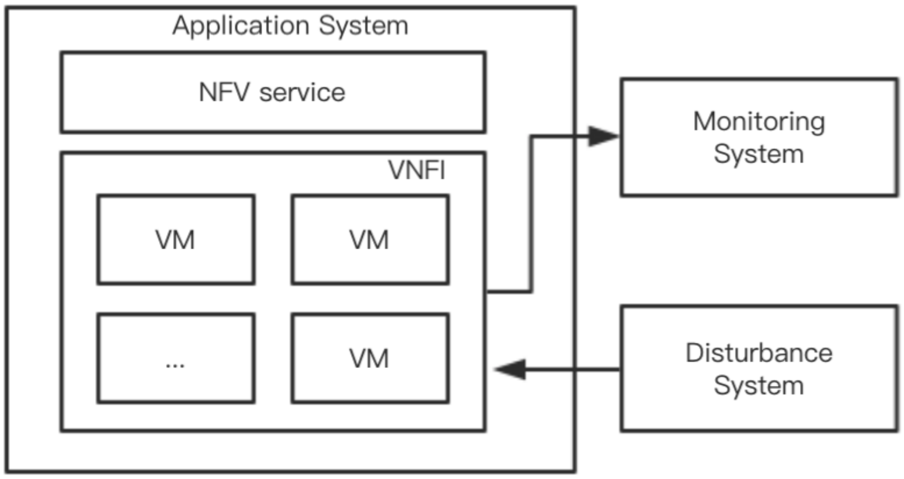

Fig. 1. Architecture of the performance data collection

hardware performance bottlenecks, SLA violation and so on. In this paper, we use linux system stress tool called stress-ng[9] to simulation system pressure to achieve fault injection function.

In order to produce different types of disturbance to the system, we use different types of fault injection in the target system:

- CPU stress fault
- MEMORY stress fault
- IO stress fault

Every type of fault injection will consume the system resources as much as possible to ensure the occurrence of anomaly events.

In most situations, anomaly diagnosis of platforms or systems is often directed against single point failure[10]. So we use a strategy to ensure that only one type of disturbance occurs on only one virtual machine at the same time.

When fault injection occurs, the disturbance system will record the log of the fault injection at the same time, including the start time, the duration, the type of fault and the target virtual machine. After the monitoring system collects performance data, the logs can be used to tag the performance data.

3.3 Monitoring system

There are many kinds of mature IaaS layer monitoring schemes at present, like Zabbix[7], Nagios[8], Cacti[9]. Considering our experimental environment and mon-

[7]https://www.zabbix.com/
[8]https://www.nagios.org/
[9]https://www.cacti.net/

itoring project items, we use Zabbix to monitor the system and collect performance data online.

Zabbix is an enterprise open source monitoring software for networks and applications with C/S model, the zabbix agent is installed in the VMs. The situation shows that agent monitoring is more accurate than agent-less monitoring, and can more accurately describe the performance model of a system[11].

The Table 1 shows the performance model in our approach. Zabbix agents will collect these metrics from VMs, and store them in it's MySQL database. We also offer a JAVA application to download these performance data throw RESTful API from Zabbix server.

Table 1. Zabbix monitoring metrics

Metric name	Description	Metric name	Description
net.if.in[]	Network interface discovery: Incoming network traffic	vfs.fs.inode[/var/lib/docker/aufs,pfree]	Free inodes on /var/lib/docker/aufs (percentage)
net.if.out[]	Network interface discovery: Outgoing network traffic	vfs.fs.inode[/var/lib/kubelet,pfree]	Free inodes on /var/lib/kubelet (percentage)
proc.num[,,run]	Number of running processes	vfs.fs.inode[/var/lib/rancher/volumes,pfree]	Free inodes on /var/lib/rancher/volumes (percentage)
proc.num[]	Number of processes	vfs.fs.size[/,free]	Free disk space on /
system.cpu.intr	Interrupts per second	vfs.fs.size[/,pfree]	Free disk space on / (percentage)
system.cpu.load[percpu,avg1]	Processor load (1 min average per core)	vfs.fs.size[/,total]	Total disk space on /
system.cpu.load[percpu,avg15]	Processor load (15 min average per core)	vfs.fs.size[/,used]	Used disk space on /
system.cpu.load[percpu,avg5]	Processor load (5 min average per core)	vfs.fs.size[/boot,free]	Free disk space on /boot
system.cpu.switches	Context switches per second	vfs.fs.size[/boot,pfree]	Free disk space on /boot (percentage)
system.cpu.util[,idle]	CPU idle time	vfs.fs.size[/boot,total]	Total disk space on /boot
system.cpu.util[,interrupt]	CPU interrupt time	vfs.fs.size[/boot,used]	Used disk space on /boot
system.cpu.util[,iowait]	CPU iowait time	vfs.fs.size[/var/lib/docker/aufs,free]	Free disk space on /var/lib/docker/aufs
system.cpu.util[,nice]	CPU nice time	vfs.fs.size[/var/lib/docker/aufs,pfree]	Free disk space on /var/lib/docker/aufs (percentage)
system.cpu.util[,softirq]	CPU softirq time	vfs.fs.size[/var/lib/docker/aufs,total]	Total disk space on /var/lib/docker/aufs
system.cpu.util[,steal]	CPU steal time	vfs.fs.size[/var/lib/docker/aufs,used]	Used disk space on /var/lib/docker/aufs
system.cpu.util[,system]	CPU system time	vfs.fs.size[/var/lib/kubelet,free]	Free disk space on /var/lib/kubelet
system.cpu.util[,user]	CPU user time	vfs.fs.size[/var/lib/kubelet,pfree]	Free disk space on /var/lib/kubelet (percentage)
system.swap.size[,free]	Free swap space	vfs.fs.size[/var/lib/kubelet,total]	Total disk space on /var/lib/kubelet
system.swap.size[,pfree]	Free swap space in %	vfs.fs.size[/var/lib/kubelet,used]	Used disk space on /var/lib/kubelet
system.swap.size[,total]	Total swap space	vfs.fs.size[/var/lib/rancher/volumes,free]	Free disk space on /var/lib/rancher/volumes
vfs.fs.inode[/,pfree]	Free inodes on / (percentage)	vfs.fs.size[/var/lib/rancher/volumes,pfree]	Free disk space on /var/lib/rancher/volumes (percentage)
vfs.fs.inode[/boot,pfree]	Free inodes on /boot (percentage)	vfs.fs.size[/var/lib/rancher/volumes,total]	Total disk space on /var/lib/rancher/volumes
vfs.fs.inode[/boot,pfree]	Free inodes on /boot (percentage)	vfs.fs.size[/var/lib/rancher/volumes,used]	Used disk space on /var/lib/rancher/volumes
vm.memory.size[available]	Available memory	vm.memory.size[total]	vm.memory.size[total]

4 Implementation

This section presents the implementation of our test bed environment. It includes infrastructure, kubernetes platform, monitoring system, attacker system and the clearwater-docker NFV application running in kubernetes platform, as shown in Figure 2.

4.1 Infrastructure

The virtualized platform is a VMWare ESXI machine with 64 CPUs, 128GB memory and 2TB disk. It can provide multiple virtual machines on a physical machine. In this paper, we create 10 VMs on it. Every VM has 2 CPUs, 8GB memory and 20GB disk. VMs are connected through a 1000Mbps virtualized network. The VMs has the docker environment with version 17.03.2-ce that can deploy most docker container in it.

4.2 Kubernetes

Kubernetes is a powerful container management platform. We use it to deploy the Clearwater project as described below. Here we use the Rancher scheme[10] to deploy kubernetes platform on the VMs. The reason is it can easily deploy the kubernetes platform. The installation steps are described as following:

1. Confirm that the network between the virtual machines just created is working;
2. Select a host as the rancher server host and deploy the latest version of rancher docker image on it;
3. Waiting for the rancher server is running Correctly, access the rancher server page from the 80 port of the host;
4. Create a new environment for test bed based on kubernetes template;
5. Add all other VMs in this environment and wait rancher server add them to kubernetes platform automatically.

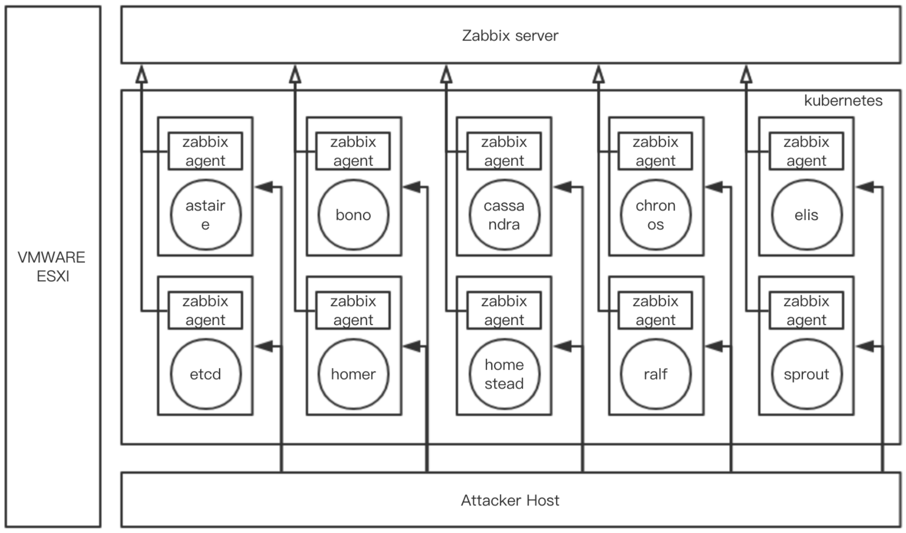

Fig. 2. Deployment of the test bed

4.3 Monitoring and attack system

The monitoring system consists of zabbix server host and zabbix agents. Zabbix agents were installed on each VM when they were created and connect to zabbix

[10]https://rancher.com/

server through the web page configurations. When the connection is set up, the agent will began to collect performance data and report them to the server at a set time interval.

Attacker host is also an independent host. It will execute the attack scripts which we provided to perform fault injection into VMs.

4.4 NFV application

The NFV application is a distributed computing system running NFV application. Here we utilise the Clearwater project. It is an open source implementation of an IMS for cloud platforms. It provides SIP-based (Session Initiation Protocol) voice and video calling, and messaging applications. It implements key standardized interfaces and functions of an IMS (except a core network) which enable industries to easily deploy, integrate and scale an IMS[3]. Clearwater project is consequently well suited for NFV related studies, it consists of about 10 components, every component plays its own unique functions in the system, and the relationship between components is shown as Figure 3. Due to the docker deployment scheme, every Clearwater docker container is configured to allow unlimited use of host resources.

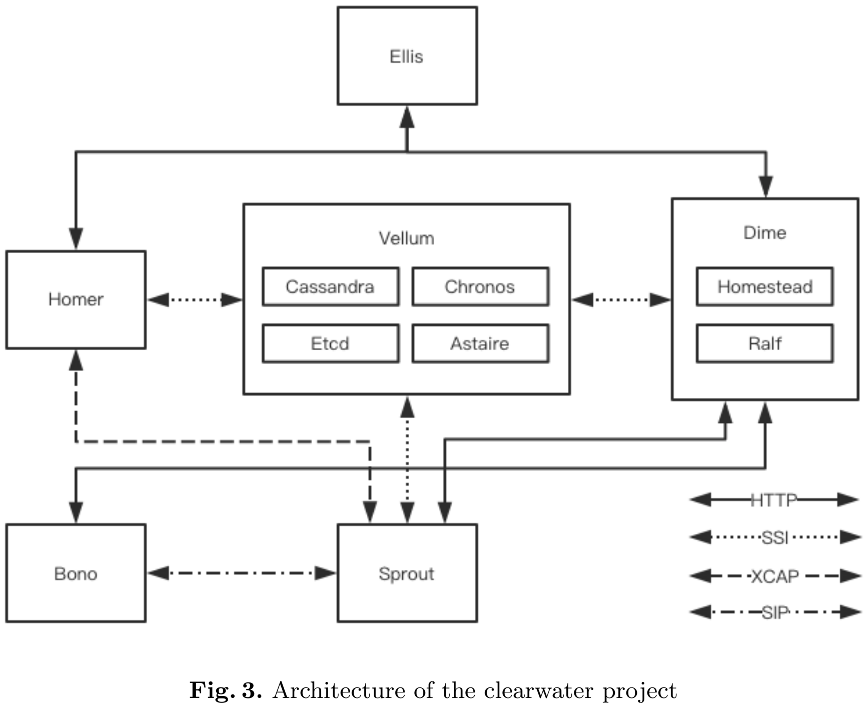

Fig. 3. Architecture of the clearwater project

Bono (Edge Proxy): The Bono nodes form a horizontally scalable SIP edge proxy providing both a SIP IMS Gm compliant interface and a WebRTC interface to clients. Client connections are load balanced across the nodes. The Bono node provides the anchor point for the clients connection to the Clearwater system, including support for various NAT traversal mechanisms. A client is therefore anchored to a particular Bono node for the duration of its registration, but can move to another Bono node if the connection or client fails.

Sprout (SIP Router): The Sprout nodes act as a horizontally scalable, combined SIP registrar and authoritative routing proxy, and handle client authentication and the ISC interface to application servers. The Sprout nodes also contain the in-built MMTEL application server.

Dime (Diameter gateway): Dime nodes run Clearwaters Homestead and Ralf components.Homestead (HSS Cache) provides a web services interface to Sprout for retrieving authentication credentials and user profile information. It can either master the data (in which case it exposes a web services provisioning interface) or can pull the data from an IMS compliant HSS over the Cx interface; Ralf provides an HTTP API that both Bono and Sprout can use to report billable events that should be passed to the CDF (Charging Data Function) over the Rf billing interface.

Vellum (State store): Vellum is used to maintain all long-lived state in the deployment. It does this by running a number of cloud optimized, distributed storage clusters including Cassandra,etcd,Chronos and Memcached.

Homer (XDMS): Homer is a standard XDMS used to store MMTEL service settings documents for each user of the system.

Ellis: Ellis is a sample provisioning portal providing self sign-up, password management, line management and control of MMTEL service settings.

As introduced before, the Bono, Sprout, and Homestead are the Core modules in the Clearwater project, they are working together to control sessions initiated by users. So our data collection work is mainly focused on these three modules.

When experiment begins, Clearwater is running normally to generate normal data, or running overloaded to generate anomaly data. When system is running normally, the attacker host can execute attack to disturb system to produce anomaly data and record the log. While the monitoring system is monitoring the VMs performance metrics and collect all normal and anomaly data on it to establish the database.

5 Case study

This section introduces a classic Clearwater case study. On the basis of the normal operation of system, disturbed the system by overload work stress and fault injection respectively to produce the anomaly dataset. And select the machine learning algorithm with better performance in anomaly detection[12][13][14][15] to verify the availability of datasets.

In order to produce a normal workload, use the official recommended tools clearwater-sip-stress-coreonly [11]. It can control the working stress of the system by specifying three parameters as:

– subscriber_count: the number of subscribers to emulate;
– duration: the number of minutes to run stress for;
– –multiplier: Optional parameters, multiplier for the VoLTE load profile (e.g. the default is 1 means 1.3 calls and 24 re-registers per sub per hour; passing 2 here will mean 2.6 calls and 4 re-registers per sub per hour).

We chose 500 subscribers, 60 minutes and 450 multiplier for experiment, At this point, the system can reach a 100% successful call rate. When the work stress continues to increase, the successful call rate began to decline. So we mark this point as a engineering level point x, it means the system has running in full workload under the current configuration.

5.1 Workload module

As described above, we use engineering level point x as a standard to produce workload. Test the performance data of the system under 0.8x, 1X, 1.5x, 2x and 2.5x pressure respectively. The structure of collected dataset is shown in the table 2.

5.2 Faultload module

In this paper, we forces on the single point fault, it means at the same time, there is only one type of fault be injected into one VM. 0.8x engineering level is chosen to be the normal system running workload to easily observe the anomaly representation generated by fault injection. The process of fault injection is shown in Figure 4.

Within a specified time period, the fault injecting program will select a Select random fault type, a random target virtual machine, and a random injection period to start a disturbance process. This process will continue until the total of time which fault injection consumed reaches the stipulated time period. As described in Algorithm 1.

The disturbance system also records the injected log while injecting the fault. The key information includes timestamp, fault type, target host and injection duration. As Algorithm 2 described, We use the fault injection log to indicate which fault injection stage each performance data record belongs to, like normal, cpu fault, memory fault or io fault. The result of data process is shown in Table 3.

In order to collect the anomaly SLA data, the workload module and faultload module work together to disturbance the system. We calculate the SLA level of the system from the percentage of successful requests(PSR). When $PSR \geq 90\%$, means the system is in good condition, marked as level 2. When $50\% \leq PSR \leq$

[11] https://clearwater.readthedocs.io/en/stable/Clearwater_stres_testing.html

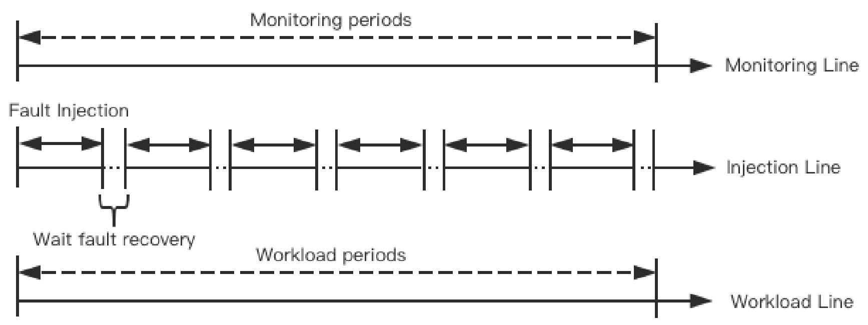

Fig. 4. Fault injection process

Algorithm 1 Fault Inject Controller

```
Input: $vm\_list, inject\_type\_list,$
    $duration\_list, duration$
1: $timer = 0$
2: while $timer < duration$ do
3:     $inject\_vm = random(vm\_list)$
4:     $inject\_type = random(inject\_type\_list)$
5:     $inject\_duration = random(duration\_list)$
6:     $timer+ = inject\_duration$
7:     $inject(vm, inject\_type, inject\_duration)$
8:     $sleep(pause)$
9: end while
```

90%, means the system is in unhealthy condition, marked as level 1. When $PSR \geq 50\%$, means the system is in bad condition, mark as level 0. The structure of dataset is shown in Table 4.

Table 2. Dataset A

Timestamp	Vm1-metric2	Vm1-metric1	...	Vm2-metric1	Vm2-metric2	...	Vm3-metric1	Vm3-metric2	...	Workload level
1521448560	70%	73%	...	69%	77%	...	66%	69%	...	1
1521448565	73%	73%	...	68%	75%	...	70%	74%	...	1
...										...
1521458230	98%	99%	...	97%	100%	...	95%	97%	...	2

5.3 Dataset verification

This part introduces four widely used machine learning algorithms, namely, support vector machine, nearest neighbor, naive Bayes and random forests. And use them to locate outliers in the system performance data.

Algorithm 2 Data Labeled Controller

```
Input: $performance\_data, injection\_log$
1: $labeled\_data = []$
2: while $performance\_data.has\_next()! = null$ do
3:     $data = performance\_data.next()$
4:     $data\_label = label(data, injection\_log)$
5:     $labeled\_data.append(data\_label)$
6: end while
```

Table 3. Dataset B

Timestamp	Vm1-metric2	Vm1-metric1	...	Vm2-metric1	Vm2-metric2	...	Vm3-metric1	Vm3-metric2	...	Normal	CPU	MEMORY	IO
152263940	70%	73%	...	69%	77%	...	66%	69%	...	1	0	0	0
152263945	73%	73%	...	68%	75%	...	70%	74%	...	1	0	0	0
152263950	73%	100%	...	69%	79%	...	72%	73%	...	0	1	0	0
...										...			
152267680	71%	74%	...	70%	75%	...	99%	72%	...	0	0	0	1

Table 4. Dataset C

Timestamp	Vm1-metric2	Vm1-metric1	...	Vm2-metric1	Vm2-metric2	...	Vm3-metric1	Vm3-metric2	...	SAL level
1521448560	90%	72%	...	92%	74%	...	85%	91%	...	2
1521448565	85%	77%	...	83%	75%	...	73%	88%	...	1
...										...
1521458230	66%	68%	...	92%	89%	...	87%	79%	...	0

Table 5. Validation results of anomaly dataset

Service	Measure	Nearest Neighbors	SVM	Naive Bayes	Random Forset
	Precision	0.98	0.89	0.95	0.97
Dataset A	Recall	0.97	0.88	0.93	0.96
	F1-score	0.97	0.87	0.93	0.98
	Precision	0.93	0.90	0.96	0.99
Dataset B	Recall	0.92	0.91	0.95	0.98
	F1-score	0.93	0.89	0.97	0.99
	Precision	0.94	0.87	0.89	0.98
Dataset C	Recall	0.97	0.93	0.91	0.96
	F1-score	0.96	0.92	0.94	0.97

There are 737 records in dataset A and dataset B, we employed the first 80% of them as the train set, having trained the learning methods, the rest 20 percent are used as test set to validate the algorithm model. The validation result are shown in Table 5.

The results show that the accuracy, recall rate and F1-sroce of each model reach a higher value. And because of the multi classification problem of the dataset, the random forest model achieves the best results.

6 Conclusion and Future work

In this paper, we describe an approach to deploy NFV application Clearwater projects through the Kubernetes platform. On this basis, we use disturbance application system and monitoring system to collect performance data of IaaS layer devices under NFV application scenario to build anomaly database. Three categories of anomaly datasets with specified label are collected, includes workload with performance data, faultload with performance data and SLA level with performance data. The details of the anomaly database can be accessed on our website[12].

Through some widely used machine learning algorithm, we verify these datasets and get high accuracy. This means these datasets have some reference value for anomaly detection. In the future, we will try more anomaly scenes and cause anomaly reasons, and build corresponding anomaly datasets to analyze them. We hope to be of certain guiding significance for the detection of anomaly in different scenes.

References

1. J. Liu, Z. Jiang, N. Kato, O. Akashi, and A. Takahara. Reliability evaluation for nfv deployment of future mobile broadband networks. *IEEE Wireless Communications*, 23(3):90–96, June 2016.
2. Mathijs Pieters and Marco Wiering. Comparison of machine learning techniques for multi-label genre classification. In *Benelux Conference on Artificial Intelligence*, pages 131–144, 2017.
3. Carla Sauvanaud, Kahina Lazri, Mohamed Kaâniche, and Karama Kanoun. Anomaly detection and root cause localization in virtual network functions. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 196–206. IEEE, 2016.
4. Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.
5. Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[12]https://github.com/XLab-Tongji/ADNFVI

6. M. Thill, W. Konen, and T. Bck. Online anomaly detection on the webscope s5 dataset: A comparative study. In *2017 Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–8, May 2017.
7. Roberto Natella, Domenico Cotroneo, and Henrique S Madeira. Assessing dependability with software fault injection: A survey. *ACM Computing Surveys (CSUR)*, 48(3):44, 2016.
8. Jeroen Delvaux and Ingrid Verbauwhede. Fault injection modeling attacks on 65 nm arbiter and ro sum pufs via environmental changes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(6):1701–1713, 2014.
9. Colin King. Stress-ng, 2018.
10. Yiping Wang and Xiaoyong Li. Achieve high availability about point-single failures in openstack. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 01, pages 45–48, Dec 2015.
11. R. Aversa, N. Panza, and L. Tasquier. An agent-based platform for cloud applications performance monitoring. In *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 535–540, July 2015.
12. A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.
13. Félix Iglesias and Tanja Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 101(1-3):59–84, 2015.
14. Amey Kulkarni, Youngok Pino, Matthew French, and Tinoosh Mohsenin. Real-time anomaly detection framework for many-core router through machine-learning techniques. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(1):10, 2016.
15. Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.