Leveraging Static Probe Instrumentation for VM-based Anomaly Detection System

Ady Wahyudi Paundu(✉), Takeshi Okuda, Youki Kadobayashi, and Suguru Yamaguchi

Nara Institute of Science and Technology,
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{ady.paundu.ak9,okuda,youki-k,suguru}@is.naist.jp

Abstract. In this preliminary study, we introduce a framework to predict anomaly behavior from Virtual Machines (VMs) deployed in public IaaS cloud model. Within this framework we propose to use a static probe instrumentation technique inside hypervisor in order to collect monitoring data and a black-box signature based feature selection method using Linear Discriminant Analysis. As a proof of concept, we run several evaluation tests to measure the output quality and computation overhead of our Anomaly Detection System (ADS) using feature selection. The results show that our feature selection technique does not significantly reduce the anomaly prediction quality when compared with full featured ADS and gives a better accuracy when compared to ADS with system-call data. Furthermore, ADS with feature selection method creates lower computing overhead compared to the other two ADS.

Keywords: Anomaly detection system · Virtual Machine · Static probe instrumentation · Cloud security

1 Introduction

One main security threat in virtualization environments of cloud computing is the guest VMs [1]. A VM can be seen as a single point of failure inside a virtualization. To monitor each VM however is not an easy task. In public IaaS there is usually an agreement between a provider and a consumer about privacy that restricts any intervention from hypervisor administrator into the guest internal system. The lack of insider information from guest OS will decrease the quality of information collected from the VM. Hypervisor administrators need to find a method to collect information on the VM's internal operation as clear as possible without guest OS intervention. In this paper, we present a novel, the first to the best of our knowledge using static probe instrumentation inside hypervisor for host-based Virtual Machine monitoring process. This approach applies specifically to Anomaly Detection System (ADS). This technique employs embedded tracepoints inside Virtual Machine Monitor (VMM) to observe VM behavior. To reduce the big dimension of monitoring points inside the hypervisor, we apply

S. Qing et al. (Eds.): ICICS 2015, LNCS 9543, pp. 320–334, 2016.
DOI: 10.1007/978-3-319-29814-6_27

Linear Discriminant Analysis (LDA) technique using system performance aspect of VM. Another main contribution is an empirical evaluation for our framework where we answer questions regarding its feasibility, effectivity, efficiency, scalability, impact and adaptability. We also compare some of these evaluations to two other ADSes which are full-featured static instrumentation ADS and system-call based ADS.

The remainder of this paper is organized as follows. In Sect. 2 we discuss previous related work on anomaly detection in the cloud. We follow this in Sect. 3 by defining our architectural framework for using static-probe instrumentation technique to monitor VM behavior. Then we discuss our empirical evaluation in Sect. 4 to assess the performance of our approach. We use Sect. 5 to discuss some issues in detail, and finally conclude the paper in Sect. 6.

2 Related Work

There have been a lot of papers covering the domain of VM-based anomaly detection system to date. However, most of these researches are focusing on private cloud, where cloud administrator can enforce internal agents inside each VM they monitor. For instance, in some approaches, the author collected VM's internal resources usage information using distributed monitoring system like Ganglia [2,3]. Another example of this intrusive approach is using honeypot-VM kernel sensors to detect intrusions [4]. The most well known method in this subject is Virtual Machine Introspection (VMI) [5]. This approach does not need to employ agents inside the monitored VM, and instead use a wealth of information of VM from Virtual Machine Monitor, like CPU state, memory raw content, I/O state. However, VMI highly depends on certain information from the Operating System (OS) inside the VM to properly interpret the monitored objects that it collected. Such information for example is debugging symbols information file for Windows system or memory offset information file for Linux that have to be copied to the monitoring program residing at the host. All these approaches are not suitable in public cloud where customers are unlikely to allow any intervention within their system.

For monitoring public cloud systems, researchers need to use non-intrusive approaches. One of the options is network-based approach that monitors traffic from and to VM in order to create a model for anomaly detection [6,7]. This method is only able to detect anomaly related to network operation. Another approach is using the fact that from host's point of view, each VM can be seen as a normal user process. Therefore, to monitor the behavior of each VM, a host can collect user process related data, such as CPU usage, memory usage or I/O volume [8,9]. Others attempt to monitor system-call exchange between VM in user-space process and host's kernel-space [10,11]. Although non-intrusive approaches uphold clients privacy and have high attack resistance, they lacks visibility on internal VM operation. This problem is known as semantic gap problem.

In our work, we use static probe instrumentation data of hypervisor to monitor VM's behavior. We argue that this approach can give better visibility into the

VM compared to system-call data or computation resource usage data because it captures its data right inside the hypervisor. To our knowledge, our proposed approach of using this instrumentation technique for VM-based monitoring is a novel approach.

3 Architectural Framework

In this work, the object of our study is public IaaS clous system, so it renders out any monitoring mechanism that requires any interference to guest VM operation. Current external monitoring approach however still cannot give enough visibility because of semantic gap problem. Therefore, we have sought to identify a solution to increase monitoring visibility into the VM, transparently from VM user point of view. Our approach works by instrumenting the hypervisor, which gives us a better access into essential underlying operation of the VM without requiring any guest interruption. Furthermore, our observation points are logically closer to the object compared to previous approaches, and that will ensure a clearer perspective into the VM.

The framework of our approach consists of three main processes, namely data collection, feature extraction and anomaly prediction as depicted in Fig. 1.

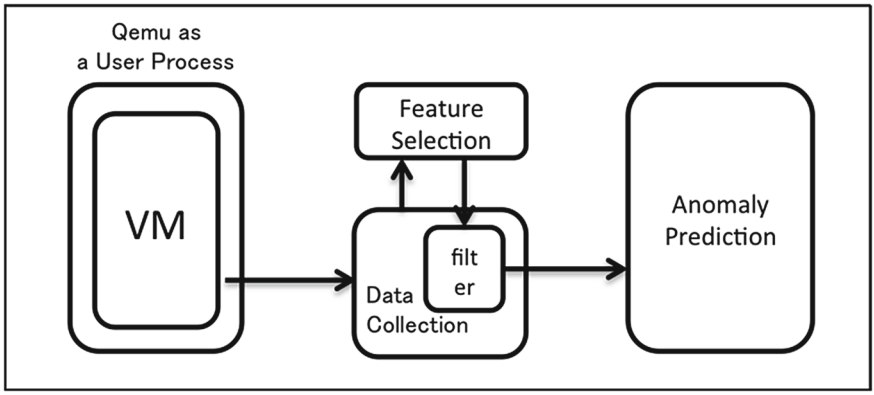

Fig. 1. High level abstraction of architectural framework

3.1 Data Collection Process

In this paper, we propose the collection of monitoring data by using static probe instrumentation inside the hypervisor. A probe is a small piece of code, usually in a form of function call that, when executed, records a certain information, such as its name which constitutes its location within the source code and its parameters value. This information can usually be used to find out the current

status and location of the program execution for debugging functions. We put our probe points inside certain VMM's functions and log them every time they are called. Three information could be extracted from these tracepoints. First is their occurrence frequency, second is their arguments value and third is their temporal relation between tracepoints. In this preliminary research on VM monitoring we only use their occurrence frequency (bag of tracepoints).

3.2 Feature Extraction Process

Generally speaking, debugging information from VMM can be huge enough to overwhelm monitoring process. For example, Qemu, one of the most widely used open source VMM, comes with around 1,200 trace points. For real-time anomaly detection, this amount of tracepoints is still very big to observe. Furthermore, from the machine learning point of view, not all of those tracepoints contain significant information. Some of the tracepoints can even act as noises. Therefore, we need to extract just small enough feature without compromising final anomaly prediction quality.

The choice of the proper tracepoints is not easy. The ideal of course is by having a complete understanding on how internal VMM works, which is usually called white-box approach. This clear picture of VMM inner works enables us to identify what functions are related to the processes we want to observe. In this preliminary research, we do not assume any knowledge of VMM internal operation. As a solution, we adopt black-box approach, where we compare each input to its output and tries to figure out their relation. In specific, we decided to use memory, I/O and network process signature as our basic profile. Many well-known and common malicious activities can be directly related to abnormal system performance [12], for instance Denial-of-Service, password dictionary attack and fuzz testing.

The purpose of feature extraction is to select subset of variables that can highly explain the change in memory usage, I/O read-write frequency or network send-receive volume. Hence, we need to use dimension reduction technique that considers class information. For that reason, we use Linear Discriminant Analysis (LDA) technique. LDA seeks to reduce data dimensionality while preserving as much of the class discriminatory information as possible, such that maximizing between-class to within-class covariance. For two class analysis, linear discriminant is defined as the linear function $y = w^T x$ that maximizes criterion function

$$J(w) = \frac{|\widetilde{\mu}_1 - \widetilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

where $\widetilde{\mu}_i$ is mean value for class i and $\tilde{s}_i^2$ is class i's scatter value along w projection. The output of LDA is linear combinations of its variables input.

3.3 Anomaly Prediction Process

In our anomaly prediction process we implement a semi-supervised anomaly detection system. This system introduces only one class model, which is normal

profile. Any data input that does not conform to this model will be categorized into new class and considered as an anomaly. This approach is also known as one-class novelty detection system as all the learning data to create the prediction model come from one single class. This approach works better under assumption that a VM in the cloud works for one specific service, for example: web server, application server or mail server. Since this kind of servers create almost homogenous operation, it is easier to provide a sound training dataset.

We use one-class SVM as our prediction engine. This predictor uses several advantageous properties of Support Vector Machine (SVM) where it is more robusts to noise and can easily work with high dimensional data while allowing smooth and flexible nonlinear mappings. Data points that cannot be separated in their original space dimension are mapped to another higher dimension feature space where there is a straight hyperplane that separates one class to another. When the resulted hyperplane is projected back into the original input dimension, it would form some non-linear curve.

3.4 Threat Model and Limitations

In our threat model, we assume that the cloud provider and its infrastructures are trusted. We consider two threat scenarios, either attacks are coming from inside the VM we monitor or the attacks coming from outside, targeting a specific monitored VM.

The approach we present in this paper is a passive approach, since it can only detect anomalies after they happen. However, in the case of long anomaly process, administrator can react upon this information to minimize the impact of malicious anomalies.

As for anomaly detection system in general, our approach cannot distinguish between malicious and non-malicious anomalies. For that purpose, additional steps need to be taken which are beyond the scope of our paper. In our framework, the decision of process maliciousness is decided by manual inspection from the system administrator.

4 Evaluation

Within this section we performed several evaluations on the framework that we described in Sect. 3. There are four research questions that we want to answer in this segment:

1. Feasibility: Can the normal data collected for learning phase converge to a stable model with low false-positives values?
2. Effectiveness: How sensitive is the model created by the training data to distinguish normal to abnormal VM operation?
3. Efficiency and Scalability: What is the cost for a host to monitor the VMs, in term of CPU usage and execution time? Will they scale well over multiple observed VMs?
4. Impact: How much is the monitoring process affecting the performance of the monitored VM?

4.1 General Setup

Hardware Setup. Diagram of our setup is shown in Fig. 2. Host specification was a dual Intel Xeon CPU 1.86 GHz with 8 GB memory and 320 GB harddisk running Ubuntu 14.04. For the hypervisor we employed the combination of Qemu-KVM. Next, we deployed eight VMs inside the host. The VMs, for better measurements, were all identical with single Qemu Virtual CPU, 1 GB memory and 30 GB harddisk. All of the VMs were using Ubuntu 14.04 operating system and serve Apache-MySQL-PHP (AMP) services.

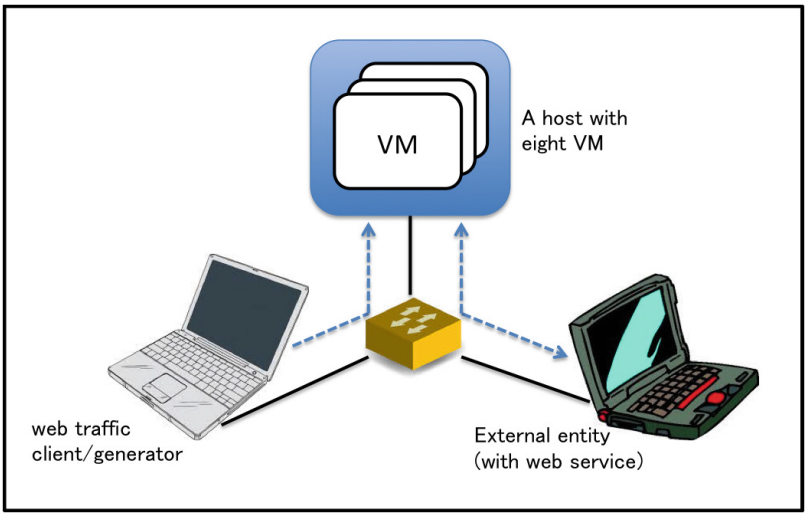

Fig. 2. Hardware setup

Normal Profile. For normal VM operation, we decided to implement a web service. Beside the fact that web service is among the most served function on the internet, it also allowed us to experience multiple normal profiles for our evaluation purpose. To emulate the operation of a web server, we used RUBiS (Rice University Bidding System) application. Using RUBiS, we can emulate a scalable dynamic web server operation.

Data Collection. There were two kinds of data collection in this evaluation process. The first data collection was static probe instrumentation as our suggested approach, while the second was system-call data that will be used as comparison methods to our approach. In the case of static probe instrumentation, we utilized "ust" (user space tracer) backend from LTTng user space tracing (LTTng-UST) library. For system-call collection we used "strace" tool. A single data unit was a collection of occurrence frequency of observed features within two seconds.

Feature Selection. We collected five datasets, each with 100 unit data. Those datasets represented the following scenarios: idle, stress-memory, stress-I/O, stress-disk and stress-network. We used "stress" Linux application for generating memory, I/O and disk data while for network we use ping with flood option. Next we paired the idle dataset with each of stress dataset which resulted in four pair scenarios. For each pair we applied Linear Discriminant Analysis to extract tracepoints that best separate each scenario. Table 1 gives the best tracepoint with its associative scenario and its LDA score. All operations within this feature selection process were done using R [13,14].

Table 1. List of selected tracepoints with its associative scenario and LDA score

idle vs	TP name	LDA score
stress network	tap_send qemu_deliver_packet	0.9913
stress IO	bdrv_aio_flush	0.9869
stress memory	virtqueue_fill	0.5795
stress hd	memory_region_ops_write	0.9992

Anomaly Prediction. Results of one-class SVM are influenced by its ν and γ parameters. ν is a constant > 0 that determines the upper bound on the fraction of training errors and the lower bound of the fraction of support vectors. Over-minimizing ν will reduce training error, but increasing false-positives. On the other hand, not small enough ν will over-fit the model and increase false-negatives. γ is the coefficient (Lagrange multipliers) for our Radial Basis Function kernel. Therefore, as part of the setup, we try to find the best combination of these two paramaters to use throughout our evaluation. Using ν and γ as controlled variables, we try to optimize estimator output by performing grid search. We search within values range $0.01 - 0.1$ with 0.01 step for both ν and γ parameters. We used False Positive as target variable. That is, we search for a combination of ν and γ values that gave the smallest False Positive result. From our normal profile dataset, we randomly prepared five subsets data. The average result from grid search of each subset data gave $\nu = 0.01$ and $\gamma = 0.01$ as our optimal value. For this grid-search process and all anomaly prediction operaton in evaluation section, we use python scripts and scikit-learn [15] library.

4.2 Feasibility with Limited Data

One characteristic of public cloud service is to pay for what is used. Therefore, it is common to see short-lived VMs hosted in the cloud. An anomaly detection system that is able to generate a normal model from smaller input without sacrificing the output quality of the system is preferred. The model created must be able to produce low enough false-positives to be considered effective. For evaluation, we used five-fold cross-validation over 500 units data of normal scenario

from selected feature of static probe instrumentation approach. The 500 data were divided into 5 subsets of 100 data. Each subset was then used as evaluation input for anomaly detection model created from increased number of data from the other four subsets. Due to the fact that all the data used in this specific evaluation were normal, an anomaly status would be considered false-positive. Figure 3 shows the average result for false-positive value of 5 subsets, predicted by normal model that was created from different size of training data. It shows that using selected feature data from static probe instrumentation monitoring on normal scenario can create a stable learning model. Furthermore, to achieve at most 5 % false positives, we need at least 100 unit learning data.

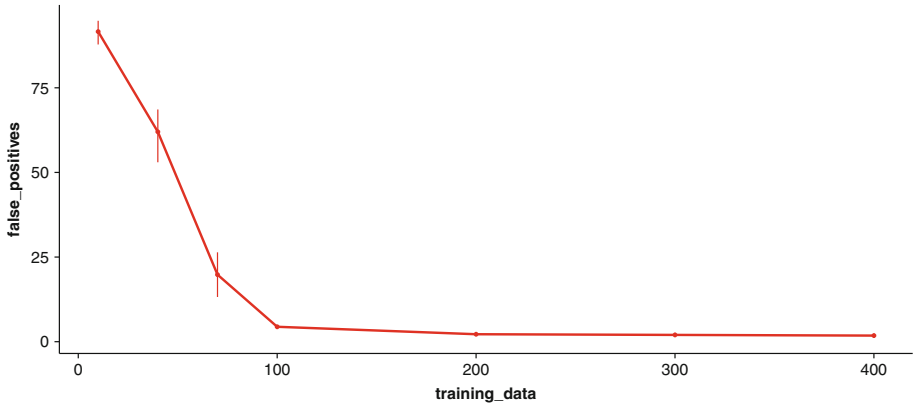

Fig. 3. Percentage of false positives as the number of training data increased

4.3 Effectiveness Against Diverse Attacks

For this evaluation part we emulated several attack scenarios as anomaly data and then observed if the predictor was able to identify them as anomaly or not. There were already many kinds of attacks that have happened in the cloud. However, one of our primary concern in this research is semantic gap information for non-intrusive VM monitoring. Therefore we try to imitate attacks that can represent either processes that extensively use computer resources and can be easily detected without semantic context or processes that in contrary happened in higher layer which makes it very hard to detect. For that reason we utilized these four attacks:

1. Synchronous-packet flood attack. This scenario was emulated using "hping3" tool and targeting port 80.
2. Password brute force attack. Using "ncrack" tool we try to crack an http basic authentication procedure.
3. Slow HTTP attack. We emulated this scenario using "httpslowtest" tool with attack in the body option.

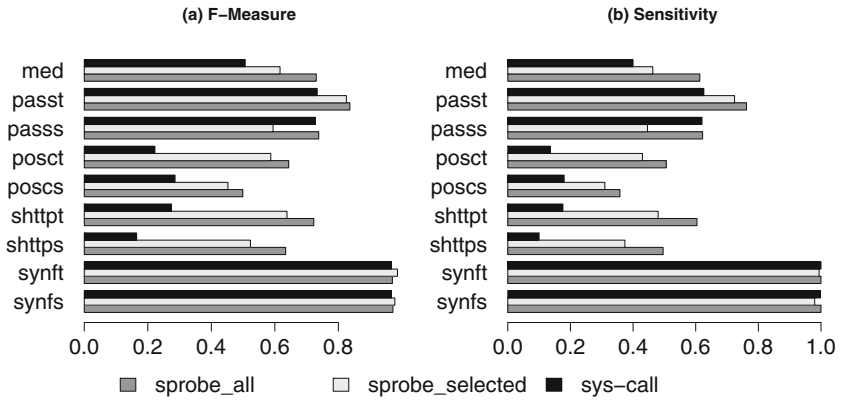

Fig. 4. Prediction results for three monitoring approaches and eight anomaly scenarios

4. Port scan attack. We emulated this attack using "nmap" tool and executed vertical port-scan attack.

For each type of attacks above, we performed both 'source attack' where the attack originates from the monitored VM and 'target attack' where monitored VM is the intended target of the attack. It resulted in total of eight anomaly type. We collected 500 unit data for each anomaly types. Just like normal data collection, one unit data was the list of variable occurrence within two-seconds of observation. We did the same amount of collection for the three data sources (static probe instrumentation using all feature, static probe instrumentation using selected feature and system-call) we want to compare. In total there were 24 scenarios that we evaluated.

To measure effectiveness, we used F-measure metric. F-measure defined as the harmonic-mean of sensitivity and precision.

$$Sensitivity = \frac{TP}{P}; Precision = \frac{TP}{TP + FP}; Fmeasure = \frac{2}{\frac{1}{Precision} + \frac{1}{Sensitivity}}$$

We divided 500 unit data for each scenario into five subsets. We then paired each subset of normal and anomaly, and feed them into one-class SVM outlier predictor. Over the five subsets, we calculated the averages of true-positives (TP), false-positives (FP), true-negatives (TN) and false-negatives (FN). Using these values, we calculated f-measure value of each scenario. We summarize the results of this evaluation in Fig. 4.a.

From Fig. 4.a. we note that, aside from synflood scenario, our static-probe instrumentation approach and system-call approach still give poor prediction values, which are below 90 %. The median of all scenario f-measure value for static-probe using all tracepoints, static-probe using selected tracepoints and system-call monitoring is 73 %, 62 % and 50 % respectively. It is also useful to look in detail to sensitivity value. This metric measure how accurate the system

is in detecting real positive data (anomaly data) only. The sensitivity results of this evaluation are given in Fig. 4.b. The median of all scenario sensitivity value for static-probe using all tracepoints, static-probe using selected tracepoints and system-call monitoring is 61 %, 46 % and 40 % respectively. Syn-flood scenario can be easily recognized because it directly affects the volume of send and receive data. However, since slowhttp scenario and portscan scenario do not change transfer rate, prediction is proved more difficult. Due to the fact that our approach works only by monitoring frequency of function-call, it can only detect changes in volume. It cannot however detect changes in sequence pattern for example, which might help increase accuracy. This Fig. 4.b also shows, that in all of scenario test, system-call prediction value was lower than static probe instrumentation prediction.

Prediction Comparison

There are several previous attempts to predict anomaly in IaaS environment without data collection from inside guest VM. Alarifi et al. [11] use bag of system-call data and implement heuristic approach by comparing the frequency table of each system-call to the testing data and calculate their difference. They register sequences of system calls using sliding window technique. For evaluation, they use unmalicious stress test in guest VM to emulate what they called "over-committed migration" attack. They reported that by choosing a window size of 10, they can achieve perfect prediction results, 100 % sensitivity and 0 % fall-out (False Positive Rate). Dean et al. [8] monitored guest VM's behavior through system-level metrics (e.g. CPU usage, memory allocation, network I/O, disk I/O) then applied unsupervised Self-Organizing Maps algorithm to predict anomaly. Injected faults to the guest's VM operation, such as memleak, CPUleak and nethog were used to evaluate the prediction system. As the result of their approach, their ADS can achieve up to 98 % sensitivity and 1.7 % fall-out. Wang et al. [9] proposed a method called EbAT to detect anomaly for utility cloud computing by analyzing metric distributions rather than individual metric threshold. Assesment were conducted by using application faulty and CPU exhaustion. Their results show that their proposed method gave 86 % sensitivity and 4 % fall-out, which is better than threshold-based methods. Doelitzscher et al. [16] tracked user behavior, such as time the VM created or destroyed and the number of running VMs and analyze them using supervised feed-forward neural network. To generate data for evaluation, they created a simulator of cloud environment. The reported result from their approach is 0.01375 % detection error rate. Sha et al. [10] applied Multi-order Markov chain to detect anomaly in cloud server systems. For evaluation purpose, they use system-call information from DARPA's 1998 Intrusion Detection Evaluation data set. In their paper however, there are no specific information on what is the quantified results on their anomaly detection scheme.

All the works mentioned above evaluate their work using anomaly scheme that significantly change the pattern of computing resources usage, such as CPU, memory or I/O. These approaches are indeed useful for detecting volume based attacks. In reality however, many attacks on computer system do not change

these resources data, hence are harder to detect. To detect those non-volume based attacks, higher semantic information is needed and for that researchers usually monitor internal operation of observed VMs. Our work were focused on trying to extract clearer semantic information without requiring to interrupt guest VM operation. That is why we choose to evaluate our framework using varied anomaly scenarios, from volume based attack scenarios such as Syncflood attack to non volume based attack such as Slow HTTP attack.

4.4 Efficiency and Scalability

We measured CPU usage and time needed to capture one unit data for certain numbers of VMs using "perf". For scalability evaluation purpose, we used one until eight VMs. The average results over five measurements for static-probe data source and system-call data source are given in Fig. 5.

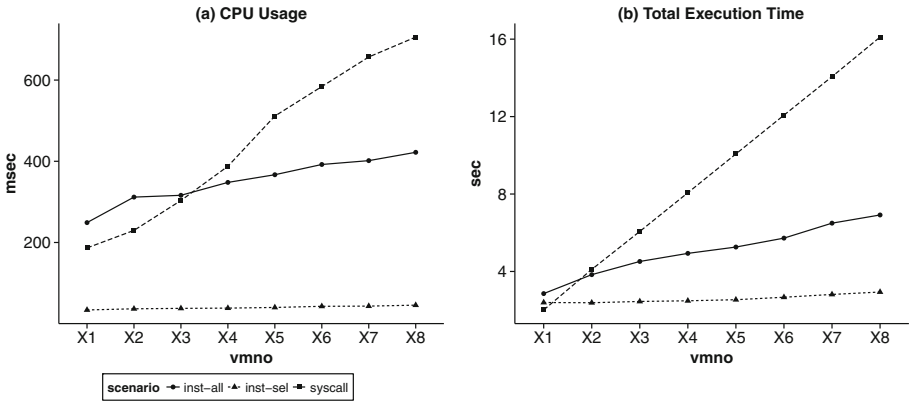

Fig. 5. "perf-stat" result to capture one unit data as the number of VM increased

The figure shows that as the number of VMs increased, the CPU time and the total execution time needed to capture one unit data for each VM increased as well. However, the increase rate of system-call approach is higher than both of static-probe instrumentation methods. From the increase rate we could also see that when we divided the cost, that is task-clock or execution-time, with the number of VMs, then the average cost to capture one unit data per VM using static-probe instrumentation would decrease as the number of VM increased, while relatively constant for system-call method. This is because for static-probe instrumentation methods, data units for all the VMs were collected collectively by one process, which is LTTng. On the other hand, data units for each VM in system-call approach were collected by different independent processes, therefore it created more overhead.

4.5 Performance Impact

We compared CPU and database (MySQL) performance of a VM when it is instrumented and when it is not using sysbench tool from inside one VM. In CPU benchmark, we took note on total execution time to calculate first 2000 prime numbers, while in database benchmark we counted how many transaction can be processed within one minute. Again, in this evaluation, we compared performance of both static instrumentation method, all-feature and selected-feature. Figure 6 presents the averages from ten benchmarking results.

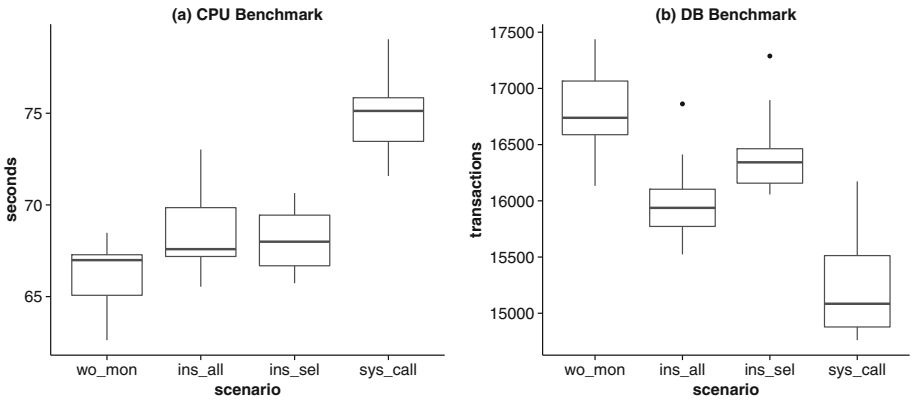

Fig. 6. The impact of monitoring for VM's performance

Both box-plot in Fig. 6 show that the monitoring process, either using static-probe instrumentation or system-call method affects the monitored VM. For CPU benchmark, static-probe instrumentation using all tracepoints, static-probe instrumentation using selected tracepoints and monitoring using system-call added 3.2 %, 2.8 % and 13 % benchmark execution time respectively, while for database benchmark they decrease 4 %, 2 % and 9 % number of database transaction per-minute, respectively. In general, static-probe instrumentation using selected tracepoints gave the least impact on monitored VM, while system-call monitoring on the other hand gave the biggest impact.

5 Discussion

5.1 Windowing to Increase Accuracy

The quality outputs of anomaly prediction as described in Sect. 4.3 are given by evaluating each unit data independently. We argue that by assessing the anomaly status in groups of sequenced data, the accuracy can be increased. This argument comes from an assumption that a change of status from normal to anomaly or reversed from anomaly to normal would not happen in a small space of time,

as our data unit describes a VM operation in 2 s. To illustrate this suggestion, consider a sequence of predictor results for an abnormal scenario [... -1 -1 1 -1 -1 -1 -1 1 1 -1 ...]. In our experiment, the sequence might have gave 30 % False Negative value. However, in a system where sequenced data are read collectively, say in a windows of five, the above sequence result will give 0 False Negative because all the windows are dominate by the value -1. The length of sequence to use for ADS had been proposed by several previous research, such as 6 in [17] and 10 in [11]. We argue that these values are unique for each case, so further studies could be taken to decide a proper window size to improve these anomaly prediction results.

5.2 Reducing False Positives

Even with the assumption that a normal scenario within a VM is rarely changed as they usually built with specific service in mind, in real life operation changes are unavoidable. One example for web operation is peak time when there is a significant increase on user using its service. Researchers in [6] show that migration process affects the quality of network anomaly detection in the cloud negatively and they still work on how to solve it. An unknown normal scenario will increase false positives which will decrease overall prediction quality.

We have conducted small side-experiment with peak web scenario and discovered that by using one-class SVM approach, we can simply capture dataset for the new normal scenario and incorporate it to our previous learning dataset. After re-learning process we have 1 % false positives where before re-learning we had 82 % false-positives. Further evaluation using anomaly scenario from Sect. 4.3 shows that the new normal-model after re-learning did not significantly reduce anomaly prediction quality.

5.3 Opportunity for Further Improvements

On Implementation. Our direct concern is on how to implement the feature-selection and collecting learning scenario data for new deployed VM in live operation. Several aspects to consider are many combinations of VM virtual hardware, VM operating system and VM service in real-life, not to mention their life span in the cloud. This opens many new research questions. For example how to find a generic set of initial selected-feature over multiple combination of virtual hardware and operating system? How effective it is to use unsupervised anomaly detection system within this framework? How to deploy an adaptive anomaly prediction method?

On VM's CPU Operation. Qemu+KVM VMM combo gave many advantages for this static-probe instrumentation monitoring approach. The fact that Qemu is an open-source product and it work as user process inside host OS makes instrumentation proses easier. Moreover, since CPU is directly translated

to native by KVM, the instrumentation on Qemu does not affect much performance of guest VMs. However it cost us visibility, because CPU operation can hardly be monitored. One simple solution is to concurently instrument KVM on kernel-space. This however will need another synchronization technique to combine both instrumentations.

6 Conclusion

In this preliminary work, we introduced a framework for anomaly detection system in public IaaS Cloud that use Qemu-KVM hypervisor. For VM monitoring we proposed a novel approach using static probe instrumentation method. In addition, to streamline the features used, we also introduced black-box based feature selection method using Linear Discriminant Analysis. Finally we successfully conducted several empirical evaluations to answer questions regarding its feasibility, effectivity, efficiency, scalability, and impact and also compared some of those evaluations to two other ADSes which are full-featured static probe instrumentation ADS and system-call based ADS.

From our evaluation we found out that our anomaly prediction system still did not provide satisfying prediction quality with just 62 % f-measure value as median for the eight anomaly scenarios that we tested. However, further comparison with static probe instrumentation using all tracepoints and system-call monitoring shows that using static probe instrumentation with selected tracepoints gave overall best solution.

We are very encouraged by these preliminary results and are considering to extend this research in the following ways:

- We would like to investigate further how to increase prediction quality using this static probe instrumentation approach. Due to the logical distance between observation points, which are within hypervisor internal operation, and VMs as the monitored objects are close, we believe there are generic guest VM's operation information that can be recognized using static-probe data within hypervisor. One forward approach that we want to investigate is by utilizing tracepoint's parameters and their temporal relations (sequence pattern).
- Moreover, for future research in this topic, additional study on Qemu-KVM internal operation should be considered. A deeper understanding on how Qemu/KVM works will help to improve anomaly prediction system, for instance to enable researcher to implement more heuristic approaches or correctly choose the right tracepoints.

References

1. Chandramouli, R.: Security recomendations for hypervisor deployment. Draft NIST Special Publication 800–125-A, NIST - National Institute of Standards and Technology (2014)

2. Bhaduri, K., Das, K., Matthews, B.L.: Detecting abnormal machine characteristics in cloud infrastructures. In: ICDMW 2011 Proceedings of the IEEE 11th International Conference on Data Mining Workshops (2011)
3. Vallis, O., Hochenbaum, J., Kejariwal, A.: A novel technique for long term anomaly detection in the cloud. In: 6th USENIX Conference on Hot Topics in Cloud Computing (2014)
4. Asrigo, K., Litty, L., Lie, D.: Using vmm-based sensors to monitor honeypots. In: Proceedings of the 2nd International Conference on Virtual Execution Environments, VEE 2006, pp. 13–23 (2006)
5. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: The 10th Annual Network and Distributed System Security Symposium (2003)
6. Adamova, K., Schatzmann, D., Plattner, B., Smith, P.: Network anomaly detection in the cloud: the challenge of virtual service migration. In: 2014 IEEE International Conference on Communications (ICC), Proceedings, pp. 3770–3775 (2014)
7. Huang, T., Zhu, Y., Zhang, Q., Zhu, Y., Wang, D., Qiu, M., Liu, L.: An lof-based adaptive anomaly detection scheme for cloud computing. In: IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW) (2013)
8. Dean, D.J., Nguyen, H., Xiaohui, G.: Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In: ICAC 2012 Proceedings of the 9th International Conference on Autonomic Computing (2012)
9. Wang, C., Viswanathan, K., Choudur, L., Talwar, V., Satterfield, W., Schwann, K.: Statistical techniques for online anomaly detection in data centers. In: IFIP/IEEE International Symposium on Integrated Network Management (2011)
10. Sha, W., Zhu, Y., Chen, M., Huang, T.: Statistical learning for anomaly detection in cloud server systems: a multi-order markov chain framework. IEEE Trans. Cloud Comput. PrePrinted (99) (2015). Doi:10.1109/TCC.2015.2415813
11. Alarifi, S.S., Wolthusen, S.D.: Detecting anomalies in iaas environment through virtual machine host system call analysis. In: The 7th International Conference for Internet Technology and Secured Transactions (ICITST), 2012 (2012)
12. Avritzer, A., Tanikella, R., James, K., Cole, R.G., Weyuker, E.J.: Monitoring for security intrusion using performance signatures. In: WOSP/SIPEW, pp. 93–104 (2010)
13. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2014)
14. Cerdeira, J.O., Silva, P.D., Cadima, J., Minhoto, M.: Subselect: selecting variable subsets, R package version 0.12-4 (2014)
15. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
16. Doelitzscher, F., Knahl, M., Reich, C., Clarke, N.: Anomaly detection in iaas clouds. In: IEEE International Conference on Cloud Computing Technology and Science (2013)
17. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system call. J. Comput. Secur. **6**(3), 151–180 (1998)