SysOptic: a Fine-Grained Monitoring System for Virtual Machines Based on PMU

Pin Liu[12], Renyu Yang[32*], Jie Sun[12], Xudong Liu[12]
[1]*SKLSDE Lab, Beihang University, China*
[2]*Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), China*
[3]*School of Computing, University of Leeds, UK*
{*liupin, sunjie, liuxd*}*@act.buaa.edu.cn ; r.yang1@leeds.ac.uk*

Abstract—**Modern cloud datacenters indicate the frequent existence of complex failure manifestation. Failures have become the norm, and not the exception. This is a key challenge since assumptions that underpin designing reliable systems are monitoring systems status and detecting anomaly at runtime. Performance Monitoring Unit on CPU (PMU) can obtain fine-grained monitoring data by adopting interrupt sampling method based on hardware events. However, profilers in virtual machines fail to receive PMU relevant information directly due to the limited capacity of PMU virtualization. In this paper, we present a fine-grained monitoring system SysOptic based on PMU virtualization. First, we propose a method to expose PMU information PMU and ensure the visibility of such information at virtual machine level. Second, to maximize the PMU reusability, SysOptic supports the PMU sharing and simultaneous monitoring among multiple virtual machines. Furthermore, we also describe how to synchronize interrupts on physical machines to virtual machines by using injecting interrupts. Experimental results show that with the aid of SysOptic, profiler tools in virtual machines manage to perceive the existence of PMU and collect the monitoring data. The additional overhead incurred by SysOptic is at most 9.8%.**

Index Terms—**cloud computing, virtualizaiton, performance monitoring, PMU**

1. Introduction

Cloud data centers are multi-tenant environments where diverse workloads live together. Normally encapsulated in Virtual Machines (VMs) or Docker Containers, such workloads are co-located into the same servers sharing the underlying physical infrastructure to maximize the data center utilization. Accordingly, machine loads significantly increase with the growth of workload heterogeneity and data amount. Load fluctuation varies among different workloads and time periods and more resources are required to underpin the increasing workload demands. However, interference among co-located workloads may lead to performance degradation [1]. In resource-restricted cluster environment, workloads need to share and compete for resources with other co-located neighbors. Unsatisfied or insufficient resource allocation inevitably leads to severe performance problems [2] [3] [4]. In such a dynamic and error-prone environment, it is significantly important for fine-grained status monitoring and anomaly detection at runtime to underpin designing reliable systems [5].

Resource auto-scaling is one of solutions to mitigate the problem due to resource shortage [6] [7]. If load spike is detected in the system, additional resources will be binded to the specific workloads. However, purely adopting mechanism does not function efficiently when faults or/and heavily resource-consuming loops (buggy codes or other software deficiencies) manifest in the workloads. Alternatively, it is necessary to quickly detect and locate system anomaly and faults by using fine-grained monitoring, thereby reducing the negative impact of abnormal workload behavior on performance [8].

For this purpose, performance analysis and profiling tools can collect and extract fine-grained metrics from Performance Monitoring Unit on CPU (PMU) at physical machine level [9]. PMU is a set of registers for performance monitoring on CPU. It uses PMC (Performance Monitoring Counter on PMU) to count the number of hardware events generated during workloads' execution, such as the number of CPU cycles, instructions and cache misses, etc. Nevertheless, in the virtualized environment, QEMU-KVM can only acquire coarse-grained information such as CPU and memory, which is far insufficient for programs to exploit for fault detection [10] [11]. Literally, it is impossible to obtain the PMU information without PMU virtualization provided by VMM layer. Therefore, how to effectively bridge the gap between physical PMU and profiling within virtual machines is the key research challenge.

In this paper , we design a novel monitoring mechanism underneath virtual machines with the same principle of PMU on physical machines. First, we propose an approach to expose PMU information to virtual machines. This method allows virtual machines to obtain PMU information through simulation instruction to make virtual machines to perceive the existence of PMU. Instead of directly exposing PMU information, our approach can also adjust the monitoring granularity by selectively submitting information of the PMC bits number to virtual machines. Second, we present

. This work is supported by the National Key Research and Development Program (2016YFB1000103) and NSFC(61421003). Renyu Yang is the corresponding author

a PMU reuse mechanism to assign a virtual PMC (VPMC) to the performance event, attach a free PMC to VPMC before monitoring, and recall the PMC after the procedure of monitoring finishes. Furthermore, we describe an approach to injecting PMC interruptions into virtual machines, which ensure the awareness of PMU interruption in virtual machines and guarantee the sample events can be timely processed. We integrated those techniques into a monitoring system which can be easily applied into realistic IaaS systems. Experimental results show that the virtual machine can obtain the information monitored by PMU through our system SysOptic. Furthermore, performance analysis tools can analysis this data to obtain functional level performance data of the load. Such as the CPU usage and cache miss of a function. In particular, the contributions are outlined as follows:

- We enable a visible pathway of PMU information to virtual machines . Virtual machine can perceive and leverage such information for fault detections.
- We propose a PMU reuse mechanism to make sure virtual machines can monitor more events with the limited number of PMC.
- We implement SysOptic to provide a full-link monitoring system for virtualized environments.

Organization. In Section 2, we discuss related work and the background. We present the system overview and key techniques in Section 3 and Section 4 respectively. Section 5 depicts the experimental results before we conclude this paper and discuss the future work in Section 6.

2. Background and Related Work

The combination of QEMU and KVM is the most common solution of Linux virtualization [12]. They provide secure and controllable resources in the form of virtual machines via virtualization of CPU, memory and network resources. This results in the equivalent performance of virtual machines and physical machines as long as virtualized resources can be isolated, monitored and controlled [13]. However, only dimensions such as CPU, memory, disk, and network usage are monitored inside virtual machines [13]. In comparison, at physical machine level more information can be collected and monitored. Numerous performance analysis and profiler tools can analyze fine-grained monitoring data through PMU hardware at process level or function level. With the prevalence of PMU technique among mainstream processor manufacturers [14], function-level performance monitoring has been changed from clock-interrupt-based cyclic sampling to PMU-events based sampling [15]. Obviously current defective PMU virtualization impede virtual machines from fine-grained profiler and anomaly detection.

Regarding virtual machine performance monitoring, Sa Wang et al. proposed Vmon to analyze the relationship between the LLC miss rates and VM performance degradation in order to predict the interference amone different resource-intensive VMs [16]. Mohamad Gebai et al.used kernel tracing to facilitate Cloud administrators to efficiently monitor VM usage [17]. However, these approaches can monitor coarse-grained information which is far from satisfied. In terms of PMU virtualization, Du J et al. proposed a virtualization method where PMU direct exposes information to virtual machine [18]. However, not all unscreened interruptions are caused by overflow of PMU counters, giving rise to the incomplete collection of event and sample amount. Therefore, an effective and complete event sampling and collection mechanism is urgently desired at virtualized level.

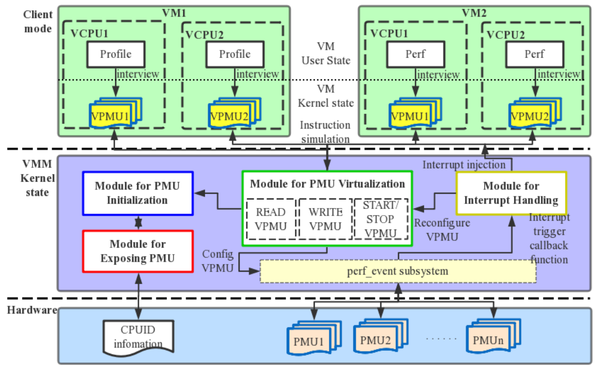

Figure 1. System architecture

3. System Overview

We adopt performance event sampling to realize the fine-grained monitoring. The basic principle of performance profiling is to program PMU relevant registers. Literally, PMU is a set of registers such as performance monitoring event selector (PMS), performance monitoring counters (PMC), status registers, and control registers. Fig. 1 describe the architecture of SysOptic. Both SysOptic and VMM are located in the midst of virtual machine layers and physical resources. VMM is mainly responsible for management of virtualized resources such as CPU, memory and disk whilst our SysOptic will primarily focus on the PMU virtualization. The two-fold functionality encompasses: 1) simulating instructions from virtual machines to operate PMU and delivering them to PMU; 2) feeding the information monitored by PMU into the performance profiler in virtual machines.

The event sampling based workflow consists of four important submodules – PMU exposure, VPMU (virtual PMU) initialization, PMU virtualization and interrupt handling. The main steps are:

a) The SysOptic queries the PMU information. Module for exposing PMU is responsible for obtaining PMU hardware information and providing part of it to module for VMPU initialization. For example, adjust the bit width of PMC before providing it.

b) SysOptic initializes VPMU related data structures. In combination with the information provided by the module for exposing PMU, module for VPMU initialization is responsible for the operation of VMPU related data structures when VMM operates VCPU information. For example,

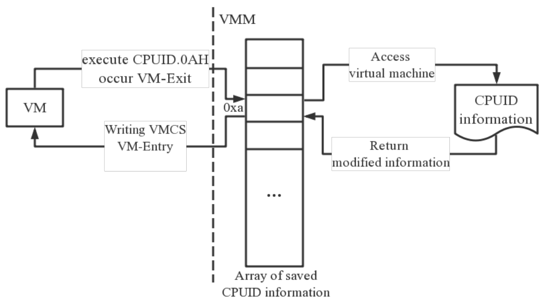
VMM
execute CPUID.0AH occur VM-Exit
VM
0xa
Writing VMCS VM-Entry
Access virtual machine
CPUID information
Return modified information
...
Array of saved CPUID information

Figure 2. Simulate CPUID.0AH instruction execution.

when VMM creates VCPU structure, it is responsible for initializing VPMU information that is not related to CPUID.

c) SysOptic assigns VPMU for monitoring events. After selecting the monitoring event, the VCPU binds a VPMC for the monitoring event. Meanwhile, module for PMU virtualization is responsible for assigning a free PMC to the VPMC.

d) Performance events are being monitored. PMC monitors performance events in a counting manner and records relevant data.

e) PMC overflows after the monitor event number reaches the set value. If the PMU chooses to interrupt, then proceed to step 6. If the PMC chooses not to interrupt, the module for VPMU virtualization is notified to reconfigure PMC and returns to step 4.

f) SysOptic injects the interrupt into the virtual machine. Module for interrupt handling is responsible for injecting the interrupt into the VCPU corresponding to the VPMC. Then module for PMU virtualization is called back to release PMC resources. The free PMC waits for the next allocation.

g) The performance analysis tool samples. After receiving the interrupt information, VCPU is responsible for notifying the interrupt handler in the performance analysis tool to sample at the interrupt site and analyze the performance results.

Particularly, step d) and g) are the most critical steps because they are mainly charge of fetching important process data during a monitoring period. SysOptic, as the middle layer, can realize the fine-grained monitoring over virtual machines.

4. Key Techniques

The above briefly introduces the basic workflow of SysOptic and explains how virtual machines monitoring with different functional modules. In this section, we will present the detailed design and implementation.

4.1. PMU Information Exposure

This module is used for being called by module for PMU virtualization. Its actual execution process is as follows. It simulates CPUID instructions to access PMU information on physical machines and returns the information to virtual machines. Fig. 2 presents the process of simulation instructions. First of all, VM-Exit occurs after the virtual machine executes CPUID.0AH instruction. Then the module returns the client mode after writing the appropriate PMU information on VMCS.

Exposing part of PMU information to virtual machines selectively mainly refers to adjusting the bit width of PMC according to actual monitoring ability and providing the adjusted bit width to virtual machines. The smaller the bit width value is, the more data are sampled at the same time, and the more accurate the analysis results are. However, high frequency sampling can seriously affect the performance of virtual machines. When the value of bit width is larger, the sample data is too small to analyze the effective performance results. Experiments show that: when the value of bit width is greater than 16 and less than 20, it can not only guarantee the accurate performance analysis results, but also guarantee the low performance consumption.

4.2. VPMU Initialization

This module is used for initializing VPMU data structure. Fig. 3 presents the VPMU data structure. Its actual execution process is as follows. It manipulates VPMU structure information when VMM manipulates VCPU. The entire process contains four phases: First, when VMM creates VCPU structure, this module is responsible for initializing VPMU information that is not related to CPUID. Second, when VMM resets VCPU structure, this module is responsible for initializing VPMU information that is related to CPUID. Third, when VMM updates VCPU structure, this module is responsible for updating VPMU information that is related to CPUID. Fourth, when VMM deletes VCPU structure, this module is responsible for deleting VPMU.

The most important information on the VPMU structure is the information related to CPUID, which includes performance monitoring version, events that can be monitored, number of two types of PMC, bit width and so on. One type of PMC is the PMC for programmable performance monitoring events. It's numbered from bit 0 to 31, and it has a maximum of 32. The other is the PMC for monitoring fixed hardware events, starting at bit 32 and it has a maximum of 3. This paper mainly uses the first type of PMC which are freely allocated to monitored events.

4.3. PMU Virtualization

This module is used for the reuse of PMU. Since there are only seven or 11 PMC at each physical logical core, this module is responsible for reuse of PMC. As Fig. 4 shows, its process can be divided into three phases: 1-3 is the PMU non-overflow phase, 4 is performance event monitoring phase and 5-8 is the PMU overflow phase.

Fig. 5 illustrates PMU non-overflow phase. Each time the virtual machine monitors a performance event, The VCPU threads are scheduled to run. At the same time, the virtual machine allocates a VPMC for monitoring events

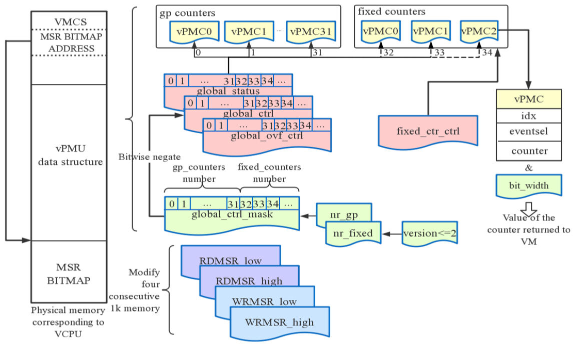

Figure 3. VPMU related data.

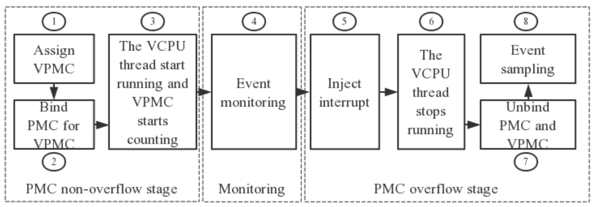

Figure 4. The process of using PMC in monitoring.

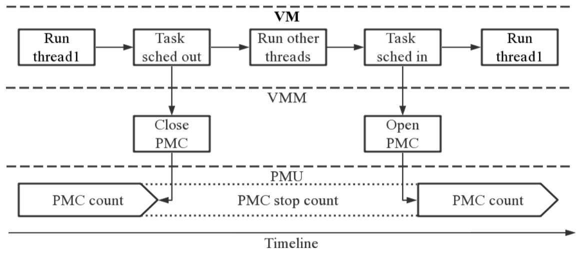

Figure 5. Module for PMU Virtualization (with PMU overflow).

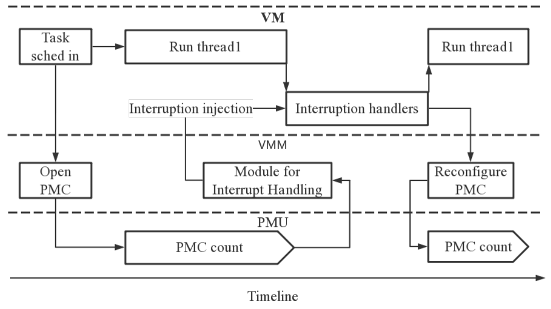

Figure 6. PMU virtualization (without PMU overflow).

to count. When VPMC starts to count, module for PMU virtualization allocates a free PMC to VPMC and writes VPMC values to PMC. Then the PMC begins to monitor the corresponding monitoring events.

Fig. 6 illustrates PMU overflow phase. When the PMU generates unscreened interrupts due to overflow, module for interrupt handling injects interrupts into the virtual machine, which is responsible for notifying the interrupt processing function to complete sampling. At the same time, module for PMU virtualization receives the callback information returned by module for interrupt handling , and then notifies the VPMC stop counting corresponding to this PMC. After PMC values are written to VPMC, the PMU resource is released and waits for the next allocation.

4.4. Interrupt Handling

This module is used to call back the module for PMU virtualization and interrupt injection. When PMC overflows, it can choose whether to interrupt by setting PMS. Therefore, when the PMU overflows, the process of this module can be divided into two situations: interrupt and non-interrupt. When PMC non-interrupt, this module reversely calls back the module for PMU virtualization to reassign free PMC to VPMC. When PMC interrupts, this module injects interrupt into the VCPU before the VM-Entry occurs to the virtual machine. Then the VCPU interrupts the VPMC corresponding to the PMC. After the VM-Entry occurs, the VCPU notifies the interrupt handler to sample at the interrupt site and analyze the performance results.

5. Evaluation

5.1. Setting-up

In this section, in order to verify the effectiveness of our method, we mainly design the experiments from two perspectives, i.e., the function and performance of methods.

The former are experiments designed according to three critical steps of PMU monitoring workflow under virtual machines, which involves the evaluation concerning the number of performance events, the number of sampling performance events, and the analysis of fine-grained performance under virtual machine. The latter is responsible for the evaluation of the performance of virtual machines with and without SysOptic respectively.

Objectives and Methodology. The purposes encompass a) Whether PMU can implement fine-grained monitoring of the performance events under virtual machines by using our system SysOptic; b) How much the performance consumption of virtual machines does SysOptic will bring. To demonstrate the superiority of system, we set the baseline as physical machine monitored by PMU. We implement our experiments in the following environment to verify the function and performance of the SysOptic system. We set both the physical and virtual machine environments as shown in Table1 and 2.

Metrics. According to the above experimental purposes, we set corresponding metrics to evaluate the experimental results. For the first purpose, we use the events count, samples count, percent of CPU usage/cache miss rate to evaluate the function of our system. These three metrics correspond to the data collected in three key steps of workflow of virtual machines for PMU-based fine-grained monitoring,

TABLE 1. PHYSICAL MACHINE CONFIGURATION

Processor	***Ram***	***OS version***
Intel(R) Core(TM) i5	8G	Ubuntu16.04.3
Kernel version	***QEMU version***	***Oprofile version***
4.10.0-28-generic	2.5.0	1.2.0

TABLE 2. VIRTUAL MACHINE CONFIGURATION

VM name	***OS version***	***Kernel version***	***CPUcores***	***RAM***
VM1	Ubuntu16.04.3	4.10.0-28-generic	1	1GB
VM2	Ubuntu14.04.1	3.13.0-32-generic	2	1GB
VM3	debian9	4.9.0-4-amd64	1	2GB

which are the statistics of performance events, sampling the performance events, and analyzing the result of sampling. For the second purpose, we set two metrics for evaluation. First, the performance score is used to evaluate the overall performance of the virtual machine under non-load. Second, the load running time is used to evaluate the performance of the virtual machine under load.

5.2. Comparison of events number

For the first purpose of our experiments, we use events count to evaluate the function of . This experiment used the stress-ng threads to run four loads of pi, matriprod, crc16 and ackermann respectively in physical machines and virtual machines. On physical machines, we use the tasket command to bind the stress-ng thread to CPU cores. On virtual machines, firstly we use the tasket command to bind the stress ng thread to VCPU and then use cgroup command to bind the VCPU to the corresponding physical logical core. Finally, we verify the accuracy of the number of performance events obtained in the virtual machine by comparing with the physical machine. Fig. 7 presents the statistical results of performance events number. The specific conditions are set as follows.

Native:1:t1 and native:1:t2 represent that two threads with the same load are running on a physical machine and are attached to the same logical core. vm1:1:t1 and vm1:1:t2 represent that two threads are simultaneously running on a single core virtual machine, and the VCPU is attached to a logical core. vm2:1:t1 and vm2:1:t2 represent that dual-core VMs simultaneously run two threads of the same load each of which is attached to one VCPU. And two VCPUs are attached to one logical core. native:2:t1 and native:2:t2 represent two threads under the same load are running on a physical machine and are attached to two logical cores. vm1:1:t represents that only one thread runs on a single-core virtual machine and VCPU is attached to a logical core. vm2:2:t1 and vm2:2:t2 represent that two threads under the same load are running on a virtual machine with dual-core and each of them is attached to a VCPU which is attached to a logical core respectively.

According to the results, running a load thread yields twice as many events as running two threads on the premise

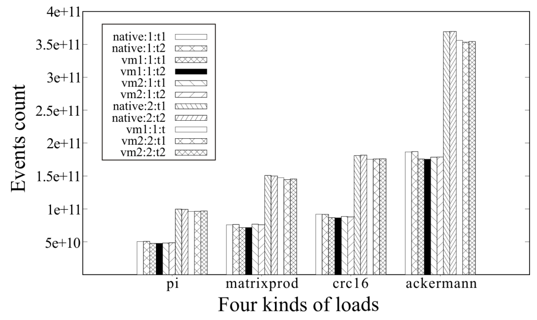

Figure 7. Comparison results of statistics monitoring performance events under different loads.

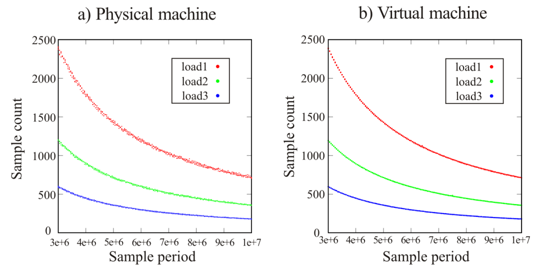

Figure 8. Comparison of the number of samples under different loads.

of a logical core. Furthermore, the performance events number in virtual environment is rather close to that of physical environment.

5.3. Comparison of the number of samples

On the basis of the previous experiment, for the first purpose of our experiments, we use samples count to evaluate the function of . This experiment runs under three different loads simultaneously on the physical machine and virtual machine. Then, we employ the performance analysis tool for sampling at different frequencies. Finally, we verify the accuracy of the number of samples obtained in the virtual machine by comparing with the physical machine.

Fig. 8 plots the number of event samples of different frequencies under three types of loads. In the figure, the red, green, and blue curves represent three types of loads respectively in which X axis represents the sampling frequency and Y axis represents the number of event samples. Overall, the number of event samples decreases as the sampling frequency increases. Moreover, the number of events samples under the physical machine is almost the same as the virtual machine under different conditions in this experiments. Consequently, the result can demonstrate the effectiveness of module for exposing PMU and module for interrupt handling.

5.4. Analyzing the performance monitoring data of virtual machine at function level

On the basis of the previous two experiments, for the first purpose of our experiments, we use percent of CPU usage/cache miss rate to evaluate the function of . In this experiment, the computational intensive load and memory intensive load were tested.

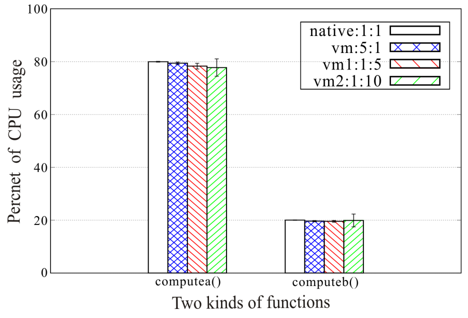

Figure 9. Comparison of performance analysis results for computational intensive workloads.

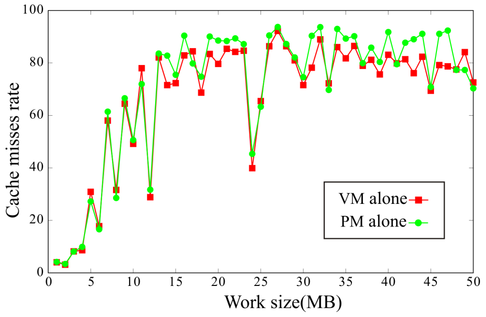

Figure 10. Diagram of memory-intensive load cache misses.

1. Concerning computational intensive loads, we construct two functions computea () and computeb () for floating point operations in an infinite loop on both the physical and virtual machines, and then we use perf report command to obtain the analysis results. Here, we will verify the accuracy of performance data at function level by comparing the percent of CPU usage of physical machine and virtual machine.

Fig. 9 presents the analysis results for both physical and virtual machines. The specific settings are as follows. Native:1:1 represents that a performance analysis process run on a single-core physical machine. vm:5:1 represents that five single-core virtual machines, each of which runs a performance analysis process simultaneously. vm1:1:5 represents that five performance analysis processes are running simultaneously on a single-core virtual machine. vm1:1:10 represents that a dual-core virtual machine run 10 performance analysis processes simultaneously.

We can observe that as the number of processes for performance analysis on a virtual machine increases, the CPU utilization decreases slightly since the process running affects the performance of the virtual machine. Moreover, the data captured by the virtual machine is almost identical to that of the physical machine. The experimental results also demonstrate that multiple virtual machines can run performance analysis processes for monitoring simultaneously.

2. Regarding memory intensive loads, we use the stress-ng command to access memory space of different sizes on both physical and virtual machines. Then we use perf report command to obtain the analysis results and average them. Here, we verify the accuracy of performance data at function level by comparing the cache miss rate of physical machine and virtual machine.

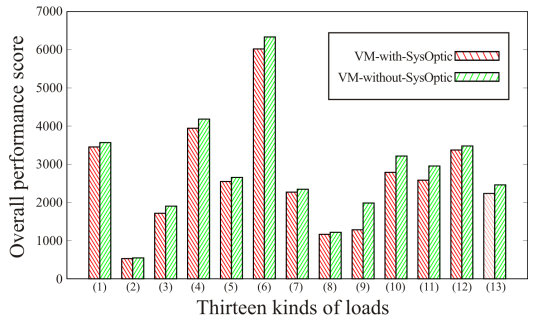

Figure 11. Results of overall performance test of virtual machine based on Unixbench.

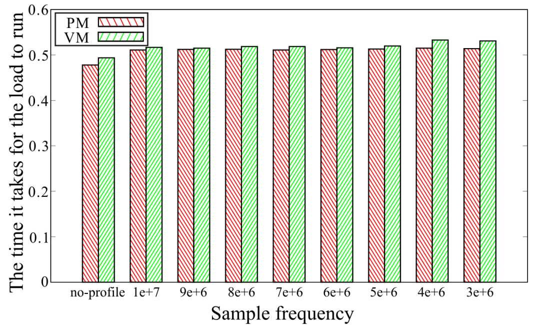

Figure 12. Results of program run time at different sampling periods.

Fig. 10 presents the analysis results for both physical and virtual machines. X-axis represents the size of the working set and Y-axis represents the cache miss. Results show that the cache miss rate of virtual machine and physical machine are basically same under the same working set, which can demonstrate that can be used for memory intensive loads.

5.5. Overall performance analysis of virtual environments

For the second purpose of our experiment, we use the overall performance score of the virtual machine to evaluate the impact of on virtual machine performance. This experiment runs 13 processes of UnixBench in virtual machines respectively and obtains each test item. Here, we compare each test item to the baseline of the ”George” workstation to get a score for each item. While running without running , the performance consumption of is verified by comparing the value of each test item in virtual machines.

Fig. 11 presents the overall performance analysis results of virtual machines. The 13 processes are Dhrystone2 respectively using register variables, Double-Precison Whetstone, Excel Throughput, File Copy 1024 bufsize 2000 maxblocks, File Copy 256 bufsize 500 maxblocks, File Copy 4096 bufsize 8000 maxblocks, Pipe Throughpu, Pipe-based Context Switching, Process Creation, Shell Scripts(1 concurrent), Shell Scripts(8 concurrent), System Call Overhead and Score.

The experimental results show that the overall performance of virtual machines decreases slightly after fine-grained monitoring with . Therefore, it also verifies that runtime has a low performance consumption.

5.6. Performance analysis for computational intensive loads

On the basis of the previous performance experiment, for the second purpose of our experiment, we use the time it takes for the load to run to evaluate the impact of on virtual machine performance. This experiment runs the same load as the blue curve in section 5.3 under the physical machine and the virtual machine. The number of interruptions increases as the sampling period increases. we verify the impact of interrupt processing on virtual machines performance by comparing the running time of the load under different sampling periods. Fig. 12 presents the running time of the load under different sampling periods.

Overall, the running time increases with the increase of the sampling period. Because the sampling operation has a slight impact on the performance of the virtual machine. Moreover, the running time of virtual machine is 1.65% larger than that of physical machine. Therefore, it also verifies that has low performance consumption.

6. Conclusions

In this paper, we reinforce PMU virtualization and enable the pathway of data fetching, collecting and extracting from physical machine to virtual machines. To improve the efficiency, we use the PMU reuse mechanism to ensure the simultaneous system monitoring among co-resident virtual machines. We integrated those techniques into *SysOptic* and experimental results show that the additional overhead brought by SysOptic is at most 9.8% but monitoring functions can realize fine-grained profiler beyond pure CPU, memory and disk usage. In our future work, we will continue to optimize the PMU virtualization and further reduce the performance overheads. We are also integrating the proposed fine-grained system monitoring mechanism into our resource management platforms [19] [20] to facilitate the fault detection, system failover and resource threshold control.

References

[1] R. Yang and J. Xu, "Computing at massive scale: Scalability and dependability challenges," in *Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on*. IEEE, 2016, pp. 386–397.

[2] M. Vanitha and P. Marikkannu, "Effective resource utilization in cloud environment through a dynamic well-organized load balancing algorithm for virtual machines," *Computers &amp; Electrical Engineering*, vol. 57, pp. 199–208, 2017.

[3] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4123, 2017.

[4] R. Yang, I. S. Moreno, J. Xu, and T. Wo, "An analysis of performance interference effects on energy-efficiency of virtualized cloud environments," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1. IEEE, 2013, pp. 112–119.

[5] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya, and R. Ranjan, "Emergent failures: Rethinking cloud reliability at scale," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 12–21, 2018.

[6] N. Rameshan, Y. Liu, L. Navarro, and V. Vlassov, "Elastic scaling in the cloud: A multi-tenant perspective," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2016, pp. 25–30.

[7] R. G. Martinez, A. Lopes, and L. Rodrigues, "Automated generation of policies to support elastic scaling in cloud environments," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 450–455.

[8] S. S. Alshamrani, D. R. Kowalski, and L. A. Gasieniec, "Efficient discovery of malicious symptoms in clouds via monitoring virtual machines," in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1703–1710.

[9] X. Wang and R. Karri, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 485–498, 2016.

[10] T. Kim, S. Choi, J. No, and S.-S. Park, "hypercache: A hypervisor-level virtualized i/o cache on kvm/qemu," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2018, pp. 846–850.

[11] J.-S. Ma, H.-Y. Kim, and W. Choi, "Kvm-qemu virtualization with arm64bit server system," in *International Conference on Cloud Computing*. Springer, 2015, pp. 334–343.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1. Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.

[14] G. M. Amdahl, G. A. Blaauw, and F. Brooks, "Architecture of the ibm system/360," *IBM Journal of Research and Development*, vol. 8, no. 2, pp. 87–101, 1964.

[15] E. W. L. Leng, M. Zwolinski, and B. Halak, "Hardware performance counters for system reliability monitoring," in *Verification and Security Workshop (IVSW), 2017 IEEE 2nd International*. IEEE, 2017, pp. 76–81.

[16] S. Wang, W. Zhang, T. Wang, C. Ye, and T. Huang, "Vmon: Monitoring and quantifying virtual machine interference via hardware performance counter," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2. IEEE, 2015, pp. 399–408.

[17] M. Gebai and M. R. Dagenais, "Virtual machines cpu monitoring with kernel tracing," in *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2014, pp. 1–6.

[18] J. Du, N. Sehrawat, and W. Zwaenepoel, "Performance profiling of virtual machines," *Acm Sigplan Notices*, vol. 46, no. 7, pp. 3–14, 2011.

[19] R. Yang, Y. Zhang, P. Garraghan, Y. Feng, J. Ouyang, J. Xu, Z. Zhang, and C. Li, "Reliable computing service in massive-scale systems through rapid low-cost failover," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 969–983, 2017.

[20] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "Rose: Cluster resource scheduling via speculative over-subscription," 2018.