2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

LoRe: Supporting Non-deterministic Events Logging and Replay for KVM Virtual Machines

Jianxin Li[12], Shouyu Si[2], Bo Li[12], Lei Cui[2], Jingsheng Zheng[2]

[1]State Key Laboratory of Software Development Environment,
Beihang University, Beijing 100191, China
{lijx, libo}@buaa.edu.cn

[2]School of Computer Science and Engineering,
Beihang University, Beijing, 100191, China
{sishouyu, cuilei, zhengjs}@act.buaa.edu.cn

***Abstract*—Cloud computing brings a loose-coupled resources integration paradigm with virtualized, elastic and cost-efficient resource management capabilities. Virtualization-based logging and replay technologies give users the ability to record the executions of the whole virtual machines and recover them at any time in a peer to peer mode, and it has become an important approach to analyze the system vulnerability, debug the system execution, or recover a failed system. In this paper, we design a logging and replay system named LoRe in KVM (Kernel-based Virtual Machine) which is a widely-used full virtualization solution. In LoRe, the logging of non-deterministic events is achieved based on the Virtual Machine Control Structure (VMCS), and a *kernel notification chain* is designed to reduce the time consumption of the branches counter matching procedure in the replay process. Moreover, to use less cache and reduce the overhead of log transmission, a reusable *circular buffer queue* is designed and IOCTL is used for the data transmission. We implemented LoRe in kvm-kmod-2.6.32, and experimental study show that the overhead of LoRe is lower than 8%, and only a small storage space is used.**

***Keywords*-Cloud Computing; Virtualization; KVM; Logging and Replay; Non-deterministic Event**

I. INTRODUCTION

In recent years, Cloud computing [1] has become a popular computing paradigm over the Internet with virtualized, scalable and cost-efficient resource management approaches to integrate loose-coupled resources, and improve their utility. Many famous corporations such as Amazon, Google, Microsoft and Salesforce become cloud platform providers. Meanwhile, the service security and continuity are two critical issues in a public cloud computing environment. Although the cloud platform takes relatively advanced measures for the security and high-availability of virtual machine, it cannot completely prevent the users' sensitive information theft, system intrusion, and state monitoring for uncontrollable virtual machines. Execution logging and replay is an advanced ability to reconstruct the past execution of a system in conjunction with a checkpoint of the system state in a peer to peer mode [2]. It has been extensively used for system security analyses, fault tolerance, system diagnosis and debugging. For example, the replay of a process of privacy information theft can help administrator to analyze and solve the security intrusion or vulnerability, or granularly monitor and analyze the exceptions and errors occurred during the system's running.

There are various approaches for classic hypervisors, such as SMP-Revirt [4] based on para-virtualization platform Xen, Retracer [5] for VMware; ExecRecorder [6] based on emulator Bochs. KVM [3] is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. KVM is being used widely in virtualization infrastructures, but few solutions are provided in respect on executions logging and replay. The work [7] provides a logging and replay system for KVM, which is mostly close to ours, but it is incomplete or has several limitations based on following observations and analyses.

Based on the features of the hardware virtualization, a suitable transparent logging and replay mechanism should be designed firstly. The guest OS in a KVM virtual machine can't realize the existence of a virtual environment. However, a main feature of the hardware virtualization is that the host and the guest share the same set of CPU registers, among which we need performance counter registers to log the time location of an event, how to distinguish the user who modifies the value of these registers is a key problem.

In addition, reducing the time consumption in the replay process is a way to significantly improve the overall performance of logging and replay system. During the replay process, the first task is to match the time location of the event to be replayed. The previous system based on Xen employed the single step mechanism to accomplish this goal which brought a high time consumption and performance overhead. Therefore, the single step mechanism should be used as little as possible, and reducing the time consumption in replay process is an important approach to improve the overall performance of logging and replay system.

An effective data structure should be designed to reduce the use of kernel buffer. Data logged in logging process is saved in kernel cache first and temporarily, and then these data will be read out and storage in the disk. Therefore, an effective data structure should be designed to use less kernel cache and release the burden of frequent data transmission. Meantime, the integrity of the data should be kept.

To comply with the KVM architecture, how to ensure the data transmission and storage efficiently is a factor should be considered on implementation. An inappropriate interface design of data transmission may cause system vulnerabilities. Therefore, the corresponding data structure should be designed and the existing tools of KVM should be integrated

To address the problems mentioned above, we design and implement a KVM based system-level execution logging and replay system – LoRe. The contributions of LoRe mainly are as follows:

- A transparent hardware virtualization logging and replay mechanism is designed for KVM, and the

Virtual Machine Control Structure (VMCS) is employed for non-deterministic events recoding.

- To improve the performance of LoRe, a *kernel notification chain* is employed to reduce the time consumption of the matching procedure in the replay process together with the branch counter matching mechanism for KVM.
- To make use less cache and reduce the burden of performance, an efficient and reusable *circular buffer queue* is designed.
- To comply with the data transmission protocol of the KVM framework, IOCTL is also used to transfer data for LoRe data. And we implemented LoRe in kvm-kmod-2.6.32.27 as an extended service of CyberGurader of iVIC [8][20], and experimental results show that it is useful.

The remaining parts of this paper are organized as follows: in Section II, we analyze and compare current related works: the design and implementation of the LoRe are introduced in Section III and Section IV; some experiments were conducted and are presented in Section V. Finally, we draw some conclusions and discuss about the future work in the last Section.

II. RELATED WORK

The main virtualization based logging and replay systems include SMP-Revirt [4] based on para-virtualization platform Xen, Retracer [5] based on traditional full virtualization platform VMware, ExecRecorder [6] based on emulator Bochs and the work [7] based on KVM .

ReVirt [9][10] is the first paper introduced the logging and replay technique to the virtualization area. It used the UMLinux as the virtualization platform and conducted a logging and replay system for uniprocessor computer system. In 2008, the ReVirt team proposed a new, multi-core logging and replay system based on Xen. UMLinux and Xen are both para-virtualization platforms. The guest knows the existence of the virtual environment. Meanwhile, UMLinux and Xen carried out the system virtualization by intercepting the system calls, and the performance is worse than that of the KVM.

Retrace has a similar mechanism as the ReVirt. Retrace divided the logging process into two phases. Firstly, it recorded the minimum event sets and then expanded the sets to a complete log file with a low cost. Retrace is based on the VMware plat-form and the implementation details cannot be obtained.

ExecRecorder was a system-level failure recovery system based on Bochs. It forked a new process called parent process when checkpointing, then the parent process waited for the signal of recovery and child process continued to run. Bochs is an X86 PC emulator, software emulation used by Bochs makes it easier to log non-deterministic events because all instructions are known and handled by the VMM, but Bochs can't keep good performance because of the software emulation. Though ExecRecorder has very low overhead (<4%), the platform limits its wide usage.

The work [7] proposed a logging and replay system based on KVM, it analyzed the features of KVM, logged and replayed non-deterministic events by QEMU and KVM and transferred data through DebugFS. It did not consider the question of how to design efficient data structure and DebugFS made other modules be aware of the logging and replay system which brought low data security.

We design and implement a new logging and replay system - LoRe, which has better performance and security. LoRe has some natural implementation advantages over the other systems because of existing features of the KVM platform. Hardware virtualization makes logging and replay for KVM transparent to the guest and gives VM better performance. Copy-on-Write disk image capabilities require few platform modifications to accomplish logging and replay. At the same time, hardware virtualization has some special problems which we discuss in detail in Section III and IV.

III. DESIGN OF LORE

During the logging process, three kinds of non-deterministic events should be logged: external interrupts, device input and special instructions like RDTSC, and the logged information includes type, time and data. LoRe uses the triple <*BC, RIP, RCX*> to identify the time location when an event happens. *BC* (*Branches Counter*) is the value of branches retired since the launch of VCPU, *RIP* is the instruction pointer and *RCX* keeps the value of iterations remaining in case of string instructions. Figure 1 shows the architecture of LoRe.

Because the KVM module locates in the low level of the OS, so the event logger which is in the KVM module can get the value of registers easily, which makes it perfect to get the information of a non-deterministic event. And for the reason that the kernel cache is limited and temporary, the logged data should be read out and saved in the disk as soon as possible. Oppositely, the replay process sends the data into the kernel cache, and stops the guest at the right time location and injects the event into the guest for replay.

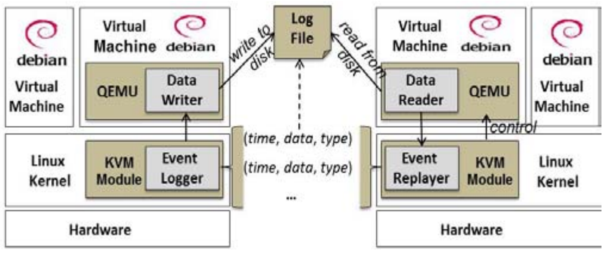
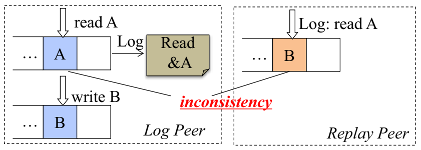


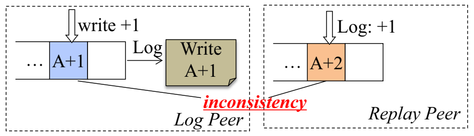
Fig. 1. Architecture of LoRe logging and replay system

A. Data Error and Inconsistency

A running state named checkpoint is necessary for logging and replay. Because of the re-execution in the replay process, devices like disk and memory will be written and read again, which may result in data error and inconsistency in the case of shared disks.



a) Data inconsistency



b) Data error

Fig. 2. Data inconsistency and error cases

Data inconsistency happens in such a case: we read data A at address &A of the disk, and then the address &A is logged. After logging, we write data B to the same address, so we get B not A when replaying this reading event.

Data error happens in this case: during the logging process, we plus 1 to the data A at address &A, then we will do the pulsing again by replaying, which actually plus 2 not 1 to the data at address &A.

Two disks and write redirection are the common methods to solve this problem, but the former one wastes store space and the latter one is very complex. In LoRe, we take advantage of the Copy-on-Write format of virtual disks, which stores the modifications of disk in temporary files instead of synchronizing the write back to the disk by using the snapshot flag. We save the operating system state by the function SaveVM of KVM, which tags the disk with the snapshot flag and ensures the disk unmodified during the logging process. The advantages of our method are no waste of store space and few modifications to the KVM system.

B. *Time Location based on VMCS*

One main feature of KVM is that the guest runs directly on the physical hardware, sharing the same registers with the host. In order to distinguish the different running environments, KVM provides the VMCS(Virtual Machine Control Structure) [11] to save and restore the running environment during the transition between the guest and the host, as is shown in Figure 3.In the process of recording the time location, the value of RIP and RCX can be obtained easily by VMCS, while the value of BC is difficult to get because of the sharing of registers between the guest and the host. At the same time, a virtual machine of KVM is just a common process in the host operating system, so the guest does not know the exact time point when it starts to run and setting the registers manually will make the courting larger than the right value. To solve this problem, we design a counting mechanism based on VMCS, which is shown in Figure 4.

We setup the registers in VMCS to enable the guest's counter[11] when the host is running. Because the guest running environment is invalid, the counter will not run until the guest starts running. The value of performance counter register is accumulated every time when the guest gives the control of CPU registers to the host.

C. *Mechanism of Branches Counter Matching*

In order to prevent the number of branches counter more than the correct value, the previous system based on Xen triggers the overflow interrupt (PMI) at a distance of N (N≥128) before the right branch, then the guest single-step runs to match the right value every branch in Figure 5, which brings high performance loss and time consumption.

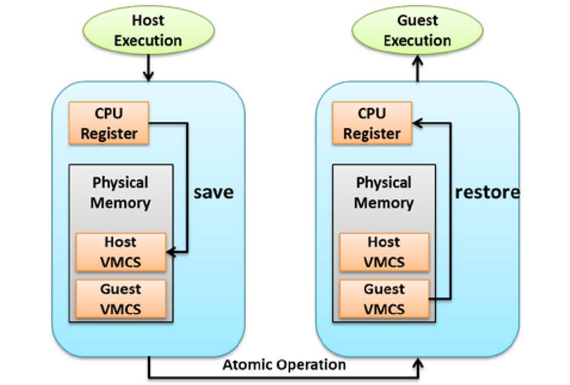


Fig. 3. Save and restore the running environment during the transition between the guest and the host

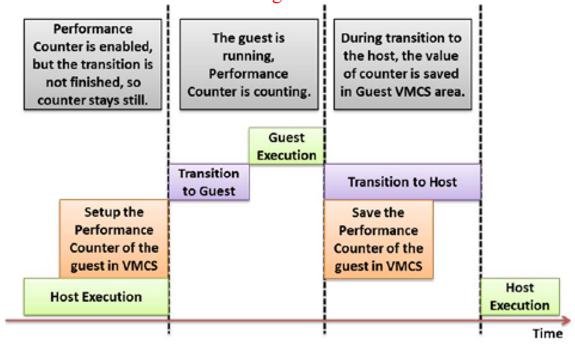
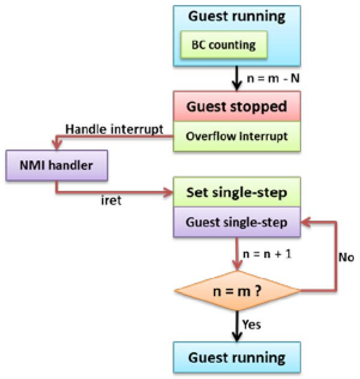
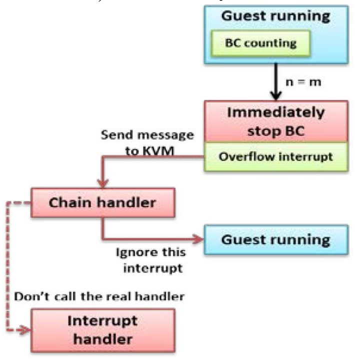

Performance Counter is enabled, but the transition is not finished, so counter stays still.
The guest is running, Performance Counter is counting.
During transition to the host, the value of counter is saved in Guest VMCS area.
Guest Execution
Transition to Guest
Transition to Host
Setup the Performance Counter of the guest in VMCS
Save the Performance Counter of the guest in VMCS
Host Execution
Host Execution
Time

Fig. 3. The BC counting mechanism based on VMCS



a) Xen-based system



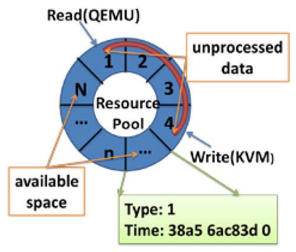
b) LoRe

Fig. 4. Branches Counter matching in Xen-based systems and LoRe

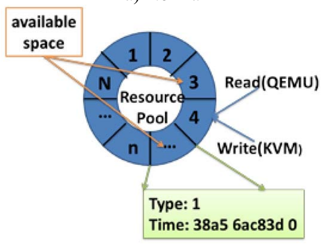
There exist two parts of time overhead in the mechanism above: the interrupt handler and the single-step procedure. LoRe employs the notification chain to make the overflow interrupt not happen until the right branch. Compared to common uses, LoRe does not need to do real interrupt handling after catching the overflow because our purpose is detecting the overflow not really handling it. It takes less time to match BC no single step, at the same time, no interrupt handler not only saves time but also simplifies the implementation. The mechanism above shortens the time to match BC and improves the performance of replay, as is shown in Figure 5(b).

D. Design of Kernel Cache Model

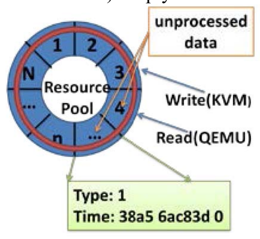
Most of the logging work is implemented in KVM module, but the kernel cache is limited and temporary, so logged data should be read out and written to the disk. We design a reusable cache model based on producer-consumer model, in which QEMU is the consumer and the KVM is the producer. The circular buffer queue model is shown in Figure 6.



a) Normal



b) Empty



c) Full

Fig. 5. The circular buffer queue model

Our queue has three control data: the write pointer, the read pointer and the available space, which are the same as common queues. The write pointer is used by KVM who puts data into the pool, the read pointer is changed by QEMU who gets the data out and the available space indicates the status of the pool.

Common queues have three states: normal, empty and full, but LoRe only allows the first two states, because the producer can't stop at any time. In the normal state, there exist both unprocessed data and available space, the producer and the consumer both work properly; In the empty state, all data has been read out, the consumer stops its work and waits; In the full state, there is no available space but the producer KVM continues putting data into the pool, so new data will overwrite the old ones , resulting in destroying the integrity of the logged data. To avoid the full state of the pool and reduce the pressure of the kernel cache, we let QEMU make data requests to the pool in every main loop of the vCPU, and LoRe transfer all the unprocessed data to QEMU at once.

E. IOCTL based data transmission mechanism

IOCTL is a way that QEMU communicates with KVM in kernel, so we combine IOCTL with the memory copy function to transfer data in LoRe. The main procedures are: In every main loop of the vCPU, QEMU asks whether there are unprocessed data in the pool through IOCTL; if there exist such data, KVM copies all unprocessed data to the memory address in the user space by the memory copy function and modifies the read pointer. The main advantage of the data transmission mechanism in LoRe is that all the process is handled in the KVM architecture, so other modules can't access the data information. By this way, the data can be conveniently transferred, which makes LoRe easier to be complied with the existing system.

IV. Implementation

According to the design principles above, we implemented LoRe in kvm-kmod-2.6.32.27 while the host operating system is Debian Squeeze. We make modifications to both QEMU and KVM, including some new console commands to control our logging and replay system and new *IOCTLs* related to data transmission. In KVM module, we insert our logging and replay module into some important functions like the interrupt handling function and the I/O handling function. The following sections show the details of our implementation.

A. Logging Non-deterministic Events

Taking the advantage of the low level of KVM, LoRe gets the information of non-deterministic events in the kernel module. The control flow of the logging run is shown in Figure 7.

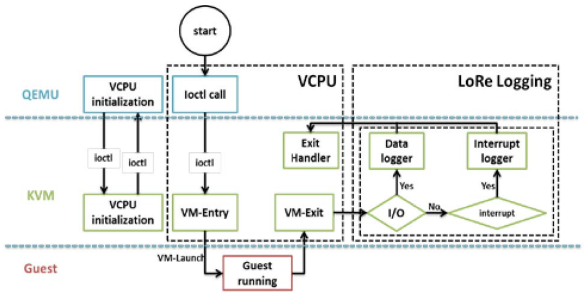


Fig. 6. Control flow of the logging process

When non-deterministic events happen, the guest will exit to root mode (VM-Exit). At this time, KVM sends the event to the event logger for information logging, and then handles this exit as normal and tells the guest to run again.

There are two kinds of events in logging process: interrupts and I/O events. Logging interrupts are relatively easy, because we just need to record the interrupt number and time location in the interrupt handling function. There exists two kinds of I/O: *Programming I/O* (*PIO*) and *Memory-mapped I/O* (*MMIO*). *PIO* data is one-byte. KVM interceptes the guest call to *IOREAD*, and emulates the instruction and stores the data in RAX register where we log. *MMIO* is also intercepted by KVM, and then KVM copies the data from the device memory to the guest memory space where we can get the data.

B. Performance Counters and Related Registers.

Throughout logging and replay, the performance counter provides a coarse, relative time at which events occur. Performance counter has a group of MSRs named IA32_PMC[0-3], all of them are controlled by the MSRs IA32_PERFEVTSEL[0-3]. During the logging process, we only use the IA32_PMCx and IA32_PERFEVTSELx registers. In the replay process, the most important work is matching the time location, during which we use overflow interrupt to detect the BC match.

In this process, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL and IA32_DEBUGCTL are used. IA32_PERF_GLOBAL _STATUS helps us to detect whether the performance counter is overflowed, IA32_PERF_GLOBAL_OVF_ CTRL is used to clear the corresponding bit of the status register and FREEZE_PERFMON_ON_PMI of the IA32_DEBUGCTL register can freeze the value of performance counter right after the PMI happens, preventing the interrupt handling process disturbs the value of the counter.

All registers mentioned above should be set up in the VM-Entry and VM-Exit fields of VMCS, and then the VMCS will save and restore the values of the registers automatically during the transition between the guest and the host.

C. Mechanism of Time Location Matching

In order to inject the event into the guest deterministically, we need to stop the guest at the right time location and return the control to KVM module.

We design an algorithm to match the time location, so some assumptions should be made first: *A* is the current event, *B* is the next event, so *B.BC* is the value of branches retired of *B, B.RIP* and *B.RCX* present the values of RIP and RCX registers.

First, we get the distance from *A* to *B*, $\Delta_{AB}=BC_A-BC_B$. Because the executions are completely the same during the logging process and the replay process, after executing Δ_{AB} branches, the execution should arrive at event *B*, and then we use hardware breakpoint to match RIP value and RCX match is achieved by single step.

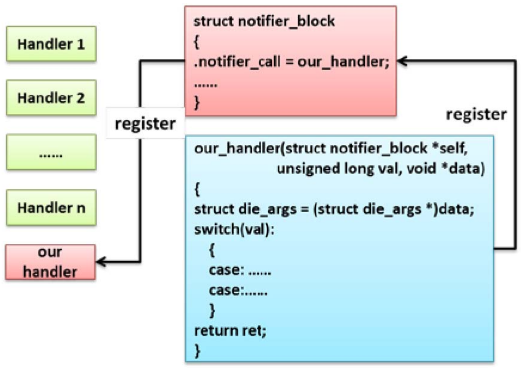
BC Matching. We can't know exactly the value of performance counter on the side of host when the guest operating system is running, so we use overflow mechanism to match the value. The P6 architecture defines a LAPIC interrupt for performance counter overflow (NMI). The performance counter is 40-unsigned integer, and its maximum value *max-pmc* can be described as 40 bits of 1. When the *max-pmc* is increased by 1, the performance counter will overflow. As such, if we set performance counter as *max-pmc*-Δ_{AB} +1, the overflow will happen after Δ_{AB} branches. Because of the features of KVM, this NMI will lead to a crash of the host operating system, in order to prevent this happening and we don't need a real interrupt handler but just a detector of the overflow at the same time, notification chain is the best choice.

Notification chain. It provides a way to get warnings when some events we are interested in happen, which is different from hard coding. The notification chain has registered some important notifiers in kernel, including *die notification chain*, *network device notification chain* and so on.

Die notification Chain. We use the *die notification chain* in this paper to detect the overflow of performance counter. When a kernel panic happens, devices can get the information and handle it, we can change our device status or shut it down instead of shutting down the operating system. In order to use the chain, we need to register an event handler, which parses the information about the running state, does processing and tells the next operation needed. Figure 8 shows the procedure.

RIP matching. The mechanism used to match EIP is hardware breakpoint. The P6 architecture supports four hardware-assisted breakpoints for debugging purposes which can be set up to trigger on reads or writes from any address. The guest's use of these breakpoints is not allowed during logging and replay process. Hardware breakpoint in our system only involves several registers: Dr0 ~ Dr3 and Dr7. After the matching of *BC*, we set the linear address of RIP to one of the debug address registers DRx and configure necessary bits in register DR7. After the configuration, he guest is allowed to run normally until the debug exception is trapped. Before modifying the value of DRx, we need to save the original value and restore it after the match of EIP to ensure the normal run of the guest.

RCX Matching. If the value of the RCX register is not already the value of RCX in the triple, it must be set as RCX =RCX –RCX_n (RCX_n means the value of RCX in the triple) and the guest should be set to run in single step mode, then the guest continues to run until the right number of iterations is executed. At this point, time location match is over and event injection can be done. After the injection, the value of register RCX should be restored with RCX in the triple, in this way, the remaining repeat iterations will continue when the guest resumes.



```
struct notifier_block
{
.notifier_call = our_handler;
......
}
```

```
our_handler(struct notifier_block *self,
            unsigned long val, void *data)
{
struct die_args = (struct die_args *)data;
switch(val):
  {
   case: ......
   case:......
  }
return ret;
}
```

Fig. 7. The procedure of die notification chain

D. Replaying Non-deterministic Events

The event injection is done when the appropriate time location reaches, its main flow is as follows: data in the log file are sent into the kernel and the guest is monitored. When the time location reaches, KVM injects the event according to the event type. Timer interrupts can be replayed in the kernel, but the I/O events should be replayed with the help of QEMU. The control flow of replay is shown in Figure 9.

First of all, replay command is sent via console of QEMU and a checkpoint is restored, then KVM module masks all interrupts. Second, *Notification chain* is enabled in CPU. Then the time location match begins, and events are replayed at the right time. For timer interrupt, we need to write its interrupt number into `VM_ENTRY_INTR_INFO_FIELD`, while for I/O events, KVM return control to QEMU, where the event is replayed.

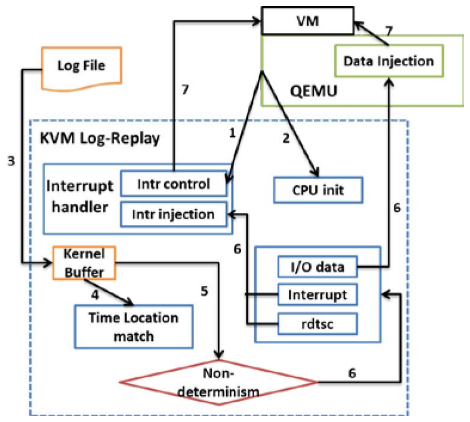


Fig. 8. Control flow of the replay process

V. Evaluation

A. Configuration of the Experiment Platform

The experiment computer is a 2.8 GHz Intel i7 processor with 4 GB of memory, which runs a uniprocessor guest virtual machine with 128 MB of memory. Both the host computer and the guest OS are install a version of Debian Squeeze with Linux 2.6.32 kernel.

B. Overhead Evaluation of Logging

In KVM, `VM-Exit` will bring in performance loss, but LoRe makes RDTSC trapped for deterministic replay, so we need to measure how much the performance loss is.

Logging Overhead of RDTSC. We run a program which invokes RDTSC for 100000 times iteratively, and then record the time cycles and real time in every cycle. The result is shown in Table 1. We can observe that emulating the execution of RDTSC has a high performance loss but logging overhead is relatively low, which is only 177µs and about 5.16%.

Table 1. logging Overhead of RDTSC

Type	*Time cycles*	*Time*(µs)
no trap	93	0.033
trap without log	9586	3.424
trap with log	10082	3.601

Logging Interrupts Evaluation. Logging interrupts is similar to RDTSC, we logged 5000 time interrupts and recorded the time overhead of logging. The result is shown in Figure 10. From the figure, we can conclude that logging interrupts takes about 318 time cycles (114µs), which is similar to the RDTSC experiment.

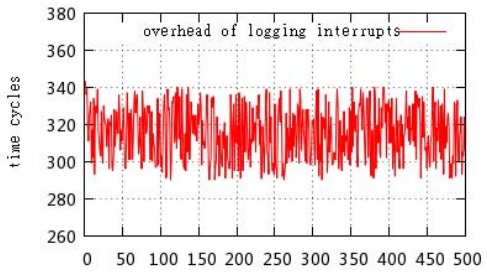


Fig. 9. Overhead of logging interrupts

Logging Performance Evaluation. We use `wget`, `kernel compile` [13] and `LU in SPLASH-2` [14] to show the comprehensive performance of logging. As the results shown in Figure 11, the performance loss is acceptable.

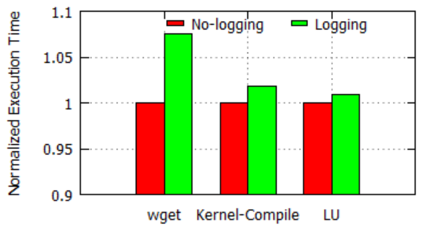


Fig. 10. Normalized execution time

Performance Comparison. To compare the overhead of logging of LoRe with current works, we have done some experiments based on kernel compile to show the performance of LoRe, ReVirt and EcecRecorder. Figure 12 is the results and the results show that the performance loss of LoRe is relatively lower than others.

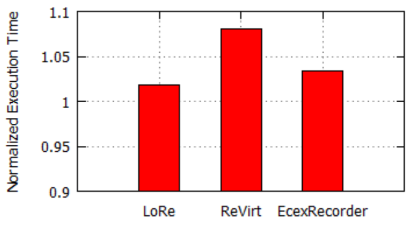


Fig. 11. Horizontal comparison of logging performance

C. Log Growth Evaluation

The size of log file increases with logging time. Every log entry is about 40 bytes uncompressed, and we record the size of log file after `gzip` compression in idle, iSCP and kernel compile cases. The result is as follows:

The log growth has relationship with the kind of workload. In idle state, the log growth is the slowest, 246.8KB/m; in kernel compile, the log growth is faster, 611.4KB/m; in iSCP, the log growth is the fastest, 6.61MB/m, 9.3G/day, the log growth is acceptable considering the high capacity of current storage.

Table 2. Data size in three benchmarks

Benchmark/ Time	0	1	2	3	4	5
Idle (KB)	0.02	3.76	7.41	12.17	16.31	20.59
iSCP (KB)	0.02	113.3	225.9	338.5	450.6	564.3
kernel compile (KB)	0.02	10.05	19.87	30.15	41.53	50.97

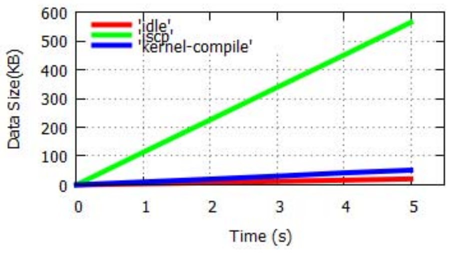


Fig. 12. Data growth of LoRe

D. *Overhead of Replay*

The overhead of replay depends on two factors: (1). At least two more `VM-Exit` to match time location. (2). No need for idle time which reduce the time overhead.

This experiment repeats those benchmarks mentioned in Section 5.2 and compares the result with normal and logging cases. The result is shown in Figure 14.

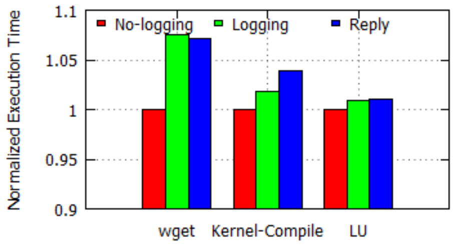


Fig. 13. Normalized execution time

According to the figure above, the performance of replay is better than that of logging when transferring files, about 7.2% performance loss; in the kernel compile benchmark, the performance of replay is worse than that of logging, about 3.9% performance loss; in the LU benchmark, the performance of replay is almost equal to that of logging, about 1.01% performance loss.

To compare the replay performance loss of LoRe with ReVirt and CASMotion [19], we recorded the replay time of kernel compile experiment and normalized the time by the logging time. Figure 15 shows the results. From the figure, the replay performance loss of LoRe is smaller than the others', showing that the notification chain mechanism improves the replay performance.

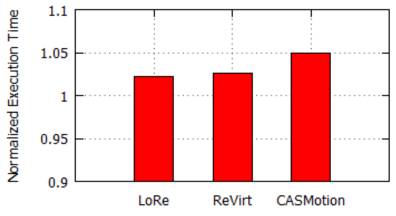


Fig. 14. Horizontal comparison of replay performance

Overall, logging and replay system based on KVM has low performance overhead and acceptable storage space.

VI. Conclusion

This paper analyzes the problems of service security and continuity in cloud computing platform, and compares implementation, performance loss and integrity of different logging and replay systems based on different virtualization platforms, then presents a new system level logging and replay system based on KVM: LoRe. Aiming at functional, performance, security and data size requests, LoRe carries out a Virtual Machine Control Structure (VMCS) based logging and replay mechanism for non-deterministic events. Meanwhile, it employs the notification chain of Linux kernel to improve the performance of replay process, it also designs an input/output control (IOCTL) based secured data transmission mechanism and an efficient and reusable circular buffer queue model. Experimental results show that LoRe achieves a lower performance loss and requires a smaller storage space.

Our future work focuses on how to replay multi-core virtual machines and how to introduce logging and replay technique into live migration and high availability of virtual machines, which will provide wider application fields for logging and replay technique.

Acknowledge

We thank the anonymous reviewers for their valuable comments and help in improving this paper. This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302602, National Nature Science Foundation of China under Grant No. 61272165 and No. 91118008, National High Technology Research 863 Program of China under Grant No. 2011AA01A202, and New Century Excellent Talents in University 2010 Beijing New-Star R&D Program under Grant No. 2010B010, and SKLSDE-2012ZX-21.

References

[1] Fox A, Griffith R, Joseph A, et al. Above the clouds: A Berkeley view of cloud computing[J]. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28.

[2] Cornelis F, Georges A, Christiaens M, et al. A taxonomy of execution replay systems[C]. Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet. 2003.

[3] Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]. Proceedings of the Linux Symposium. 2007, 1: 225-230.

[4] Dunlap G W, Lucchetti D G, Fetterman M A, et al. Execution replay of multiprocessor virtual machines[C]. Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2008: 121-130.

[5] Sheldon M X V M J, Weissman G V B. Retrace: Collecting execution trace with virtual machine deterministic replay[C]. Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2007). 2007.

[6] de Oliveira D A S, Crandall J R, Wassermann G, et al. ExecRecorder: VM-based full-system replay for attack analysis and system recovery[C]. Proceedings of the 1st workshop on Architectural and system support for improving software dependability. ACM, 2006: 66-71.

[7] Kiefer K E, Moser L E. Replay debugging of non - deterministic executions in the Kernel - based Virtual Machine[J]. Software: Practice and Experience, 2011.

[8] Li J, Li B, Wo T, et al. CyberGuarder: A virtualization security assurance architecture for green cloud computing[J]. Future Generation Computer Systems, 2012, 28(2): 379-390.

[9] Dunlap G W, King S T, Cinar S, et al. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 211-224.

[10] Liu H, Jin H, Liao X, et al. Live migration of virtual machine based on full system trace and replay[C]. Proceedings of the 18th ACM international symposium on High performance distributed computing. ACM, 2009: 101-110.

[11] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Vol. 1–5. Intel Corporation: Santa Clara, CA, December 2009. Intel Publication.

[12] Venkateswaran S. Essential Linux device drivers[M]. Prentice Hall Press, 2008.

[13] Deshane T, Shepherd Z, Matthews J, et al. Quantitative comparison of Xen and KVM[J]. Xen Summit, Boston, MA, USA, 2008: 1-2.

[14] Woo S C, Ohara M, Torrie E, et al. The SPLASH-2 programs: Characterization and methodological considerations[C]. ACM SIGARCH Computer Architecture News. ACM, 1995, 23(2): 24-36.

[15] Intel Open Source Technology Center. System Virtualization: principles and implemetation[M]. Tsinghua University Press. 2009.

[16] Kei Ohmura NTT Cyber Space Labs. Rapid VM Synchronization with I/O Emulation Logging-Replay. http://www.linux-kvm.org/page/KVM_Forum_2011.

[17] Ta-Shma P, Laden G, Ben-Yehuda M, et al. Virtual machine time travel using continuous data protection and checkpointing[J]. ACM SIGOPS Operating Systems Review, 2008, 42(1): 127-134.

[18] Vallee G, Naughton T, Ong H, et al. Checkpoint/restart of virtual machines based on Xen[C]. Proceedings of the High Availability and Performace Computing Workshop (HAPCW 2006), Santa Fe, New Mexico, USA. 2006.

[19] Bressoud T C, Schneider F B. Hypervisor-based fault tolerance[J]. ACM SIGOPS Operating Systems Review, 1995, 29(5): 1-11.

[20] Lei Cui, Jianxin Li, Bo Li, Jinpeng Huai, Chunming Hu, Tianyu Wo, Hussain Al-Aqrabi, Lu Liu: VMScatter: migrate virtual machines to many hosts. VEE 2013: 63-72