The Journal of Systems and Software 000 (2016) 1–12

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

ReSeer: Efficient search-based replay for multiprocessor virtual machines

Tao Wang[b], Jiwei Xu[b], Wenbo Zhang[b,*], Jianhua Zhang[b], Jun Wei[a,b], Hua Zhong[b]

[a] State Key Laboratory of Computer Science, Beijing, 100190, China
[b] Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China

ARTICLE INFO

Article history:
Received 5 November 2015
Revised 20 July 2016
Accepted 21 July 2016
Available online xxx

Keywords:
Deterministic replay
Virtual machine
Genetic algorithm
Memory checkpoint
Xen

ABSTRACT

Efficient replay of virtual machines is important for software debugging, fault tolerance, and performance analysis. The current approaches of replaying virtual machines record the details of system execution at runtime. However, these approaches incur much overhead, which affects the system performance. Especially, in a multiprocessor system, recording the shared memory operations of multiple processors leads to a large amount of computing overhead and log files. To address the above issue, this paper proposes ReSeer—a search-based replay approach for multiprocessor virtual machines. ReSeer consists of three phases including record, search, and replay. In the record phase, we record only necessary non-deterministic events at runtime, and incrementally take memory checkpoints at a defined interval. In the search phase, we encode all the possible execution paths as binary strings, and use a genetic algorithm to search expected execution paths achieving the expected checkpoint. In the replay phase, we replay the system execution according to the searched execution paths and the logged non-deterministic events. Compared with current approaches, ReSeer significantly reduces performance overhead at runtime by searching expected execution paths instead of recording all the operations of accessing shared memory. We have implemented ReSeer, and then evaluated it with a series of typical benchmarks deployed on an open source virtual machine—Xen. The experimental results show that ReSeer can reduce the record overhead at runtime efficiently.

1. Introduction

Virtualization allows multiple operating systems to run on a single physical machine by partitioning physical resources into multiple virtual machines (VM). Each VM protected and isolated from the others performs as an individual physical machine. Users of each VM execute their own applications without fearing the system crashes caused by other VMs on the same physical machine. Furthermore, the virtualization consolidation increases the resource utilization of physical machines, and reduces the cost of system management. Thus, virtualization, a core technology of cloud computing, is prevalent in recent years (Barham et al., 2003). However, the virtualization increases the complexity of debugging software systems, because we ought to analyze both the VMs and software systems. Replay is important for software debugging (Chow et al., 2008), fault tolerance (Bressoud and Schneider, 1996), performance analysis (Attariyan et al., 2012), and system forensics (Dunlap et al., 2002). Because online program analysis brings significance performance overhead, system administrators often offline locate faults with the recorded logs, when some faults are triggered (Chow et al., 2008). Thus administrators can collect execution details in the replay phase to inspect the entire state of the system, and then locate the root causes of problems. Current replay approaches are mainly implemented in the process level for debugging parallel programs, whereas the replay in the virtual machine monitor (VMM) level is also necessary. First, as the virtualization has been widely adopted in cloud computing, the replay system can be easily deployed and applied with VMM. Second, because VMM can observe the system calls, signals, and interrupts of guest operating systems, the replay system for VMs can transparently record the entire execution information of an operating system in detail. For example, a deliberate hacker can intrude operating systems, and eliminate log files in the operating system. If the replay system records all the execution information in the VMM level, we can accurately analyze the suspicious operations of the hacker in the operation system (Rosenblum and Garfinkel, 2005). The replay for VMs is often effective in debugging faults related with operating systems and VMs in practice (Dunlap et al., 2008). For example, Dunlap et al. (2002) debug a non-deterministic attack caused by the ptrace race condition in Linux kernels before

* Corresponding author.
E-mail addresses: wangtao@otcaix.iscas.ac.cn (T. Wang), zhangwenbo@otcaix.iscas.ac.cn (W. Zhang).

http://dx.doi.org/10.1016/j.jss.2016.07.032
0164-1212/

version 2.2.19; King et al. (2005) debug the USB device driver, System call, Kernel race condition, and mremap bugs. Note that since the replay systems in the VMM level can be used to debug the low-level faults manifested in the operating systems or the physical resources. For example, if an application utilizes a USB device but cannot get a response, the system administrator can use the replay system to observe and analyze the operations on the device, and locate the root cause of the fault. However, the replay system cannot deal with the application-level faults related with the processing logic of an application, and the faults should be debugged with the replay techniques in the application level.

A typical replay system usually has a record phase and a replay phase. In the record phase, all non-deterministic events are recorded in log files at runtime, such as uncertain instructions (e.g., RDTSC), uncertain functions (e.g., random function), system calls (i.e., getpid), interrupts (e.g., keyboard), signals (e.g., SIGKILL), traps, DMAs, and IO devices (e.g., printer). In the replay phase, the replay system deterministically executes the instructions and the non-deterministic events recorded in the log files. Uniprocessor replay approaches in the VMM level, such as Hypervisor (Bressoud and Schneider, 1996) and Revirt (Dunlap et al., 2002), record and replay the whole guest system. However, multiprocessor replay is much more complicated, because it deals with the frequent access races of shared memory. A typical approach for multiprocessor VMs is to record the orders of accessing shared memory (Dunlap et al., 2008; Laadan et al., 2010), but the recording operations incur significant performance overhead leading to a much slowdown and a large log size. To reduce the above overhead, current deterministic replay approaches can be classified as hardware assisted schemes and software-only schemes. The hardware assisted schemes use dedicated hardware supports to record shared memory operations with acceptable overhead and a limited log size (Basu et al., 2011; Tang et al., 2013; Honarmand and Torrellas, 2014). However, these schemes are limited in some specific modified processors. On the other hand, the software-only schemes record or use reasoning techniques to deduce the shared memory operations. These schemes can be widely used in kinds of processors and applications, but the current approaches often have impractical significant performance overhead (Dunlap et al., 2008; Lee et al., 2012; Yu et al., 2012; Chen and Chen, 2013).

This paper proposes ReSeer, a search-based VM replay approach for multiprocessor VMs, which includes three phases, i.e., record, search and replay. In the record phase, we record only necessary non-deterministic events at runtime, and take memory checkpoints at a defined interval. In the search phase, we encode all of the possible execution paths as binary strings, and use a genetic algorithm (GA) to search expected execution paths achieving the expected checkpoint. In the replay phase, we replay the system execution according to the searched paths and the non-deterministic events recorded in the log files. Compared with current approaches, ReSeer significantly reduces the record overhead at runtime by acquiring similar execution paths of the available checkpoint, rather than the exact same paths of the original execution. Furthermore, because GA uses a parallel population to search expected execution paths, we can improve the searching efficiency with parallel computing frameworks (e.g., MapReduce and Spark). The experimental results show that ReSeer can efficiently reduce the record overhead at runtime. Compare with SMP-Revirt, ReSeer reduces about 32.53% computation overhead in Xen, and improves performance by 21.43% in the compressing of Freebench, 8.69% in the decompressing of Freebench, 55.42% in the disk reading of Dbench, 57.14% in the disk writing of Dbench, 61.91% in the packet sending of Netperf, and 48.29% in the packet receiving of Netperf.

The contributions of this paper are summarized as follows:

- We proposed an efficient search-based replay approach – ReSeer, which searches execution paths instead of recording the access races of shared memory. That means it records much fewer non-deterministic events than current approaches do. Therefore, ReSeer can significantly reduce the record overhead and the size of log files.
- ReSeer searches the execution paths between any two adjacent checkpoints rather than the whole execution path from the start to the end of the system execution. As a result, we can search only in a suspicious period, and search the execution paths in parallel.
- ReSeer uses the page protection in Xen to search the execution paths related with the operations of writing shared memory. Compared with current search-based approaches searching the entire orders of accessing memory, ReSeer is efficient by limiting the search space.
- ReSeer searches the expected execution paths with a heuristic algorithm GA, which improves the search efficiency and makes it possible to use parallel computing frameworks.
- We have implemented ReSeer, and evaluated it with a series of benchmarks deployed on Xen. The experimental results show that ReSeer can effectively reduce the record overhead.

The rest of this paper is organized as follows. Section 2 presents the background of the virtualization technology. Section 3 presents ReSeer — our search-based replay approach for multiprocessor VMs. Section 4 presents the experimental results. Section 5 discusses the related work. Section 6 concludes this paper and directs the future work.

2. Background

2.1. Virtualization and Xen

Virtualization enables multiple VMs sharing physical resources to run on a single physical machine. VMM running VMs virtualizes computing, storage, and network resources to divide these physical resources into virtualized units. VMs, which are isolated from one another and imperceptible to the guest operating systems (OS), apply and use these virtualized units exclusively. VMM schedules the virtualized resources of these VMs on the whole. For example, VMM transfers the output of a virtualized network card to that of a physical network card. Virtualization is classified as full virtualization and para-virtualization. The full virtualization allows an unmodified guest OS to run in a VM just as in a physical machine. This approach enables various OSes to be deployed in the virtualized environment, but often incurs a serious performance penalty. In contrast, the para-virtualization requires modifications to the guest OSes cooperating with the VMM, but gains a near-native performance.

Xen is a widely used para-virtualization software, which optimizes the VM performance. Fig. 1 depicts the Xen virtualization architecture in which VMM abstracts the underlying physical resources and provides the management interfaces of VMs. VMM manages the physical resources of all VMs called domains, and launches the most privileged domain called Dom0, which is responsible for creating, migrating, and destroying the other unprivileged domains called DomU. VMM copies the input and output of DomU to Dom0, and then sent them to the hardware driver installed in Dom0.

2.2. Memory checkpoint

VMM, which manages the memory of a physical machine, intercepts and checks the memory operations of VMs. VMM ensures that every VM accesses the memory in its own memory space. Because all memory operations go through VMM, we can take the

Please cite this article as: T. Wang et al., ReSeer: Efficient search-based replay for multiprocessor virtual machines, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.07.032