Improving I/O Performance of All Peripheral Devices on Paravirtualized Platforms

Anonymous

Abstract

IOMMUs have been pervasively deployed on paravirtualized systems for the protection of the hypervisor and the security-critical data structures, e.g., the shared guest page tables. According to our observations, the updates of the guest page tables that are supposed to be orthogonal to the device I/O performance, would surprisingly lead to a large numbers of IOTLB misses. It implies that the I/O performances of all peripheral devices will be affected by the seemingly unrelated guest page table updates. Until now, researchers and developers are not aware of the existence of this dependence and do not consequently adjust the design of the paravirtualized hypervisor and the guest operating systems.

In this paper, we are the first one to deeply demonstrate the impact upon the device I/O performance due to the page table updates. To minimize the impact, we propose IOSUP (I/O Speed-UP), a novel software-only approach for decreasing the IOTLB misses, as well as retaining the security of hypervisor. We also implement an prototype on Xen and Linux kernel. We do small modifications of Xen (xxx SLoC) and Linux kernel version 3.2.0 (xxx SLoC). We also evaluate the I/O performance in both micro and macro ways. The micro experiment results indicate that the new algorithm is able to effectively reduce the miss rate of IOTLB with even less CPU usage, especially when the page tables are frequently updated. The macro benchmarks shows that the I/O devices always produce better (or the same) performance, especially when the system frequently generate many temporal processes.

1 Introduction

The para-virtualization technology [2, 5] is able to defend against the attacks from the software within guest Virtual Machines (VMs), but it is not armed with efficient protection approach to prevent DMA attacks [4]. To fix this gap, Intel and AMD propose the I/O virtualization (AMD-Vi [1] and Intel VT-d [3]) technology, which introduces a new Input/Output Memory Management Unit (IOMMU) to restrict DMA accesses on the specific physical memory addresses. Thus, the hypervisor could leverage IOMMU to protect itself by setting its occupied memory regions inaccessible for all DMA accesses. It is true for the hypervisor in the full virtualization (e.g., Hardware Virtual Machines (HVMs) and Xen [2]), since the memory boundary between the hypervisor and guest VMs are clear and statically fixed.

If the hypervisor and the guest VMs work in full virtualization (e.g., Hardware Virtual Machines (HVMs) and Xen [2]), the above solution is clean and secure to defend against both software and DMA attacks from guest VMs. However, in paravirtualized environment, there are many security-critical data structures, like Global Descriptor Table (GDT) and page tables, that are inevitably used by both the hypervisor and the guest VM [**?**, **?**]. In such situations, the adversary could launch DMA requests to illicitly modify those shared data structures to open the door for the malicious guest software to bypass the security restrictions set by the hypervisor.

I/O devices generate interrupts to asynchronously communicate to the CPU the completion of I/O operations. In virtualized settings, each device interrupt triggers a costly exit [2, 9, 26], causing the guest to be suspended and the host to be resumed, regardless of whether or not the device is assigned. Many previous studies that aim to improve device I/O performance While our work is different since it 1) aim to improve I/O performance for all I/O peripheral devices.

Our approach rests on the observation that the high interrupt rates experienced by a core running an I/O-intensive guest are mostly generated by devices assigned to the guest.

This revolutionary trend also urges us to significantly reshape modern operating systems to keep up the pace. Unfortunately, the current designs and implementations of modern operating systems lag behind the requirements.

In this paper, we focus on the improvement of I/O performance. Specifically, we aim to adopt guest OS kernel to further improve the I/O performance of all peripheral devices without sacrificing the security of the paravirtualized platforms. By deeply analyzing modern Xen hypervisor and Linux kernel, we surprisingly notice that the page table updates of guest OS could cause IOMMU to flush IOTLB. These flushes are necessary for the sake of the security of Xen hypervisor, but it inevitably increases the miss rate of IOTLB, and consequently reduces I/O performance, especially for the high-speed devices. Note that we are the first one to uncover this dependence between the security of paravirtualized (Xen) hypervisor and I/O performance. Based on this observation, we propose a novel algorithm that decreases the miss rate of IOTLB by carefully managing the guest page table updates, as well as retaining the security of paravirtualized hypervisor. We implement our algorithm with no modification of Xen and small customizations of Linux kernel version 3.2.0 by only adding xxx SLoC, and evaluate the I/O performance in micro and macro ways. The micro experiment results indicate that the new algorithm is able to effectively reduce the miss rate of IOTLB, especially when the page tables are frequently updated. The macro benchmarks shows that the I/O devices always produce better (or the same) performance, especially when the system frequently generate many temporal processes.

The rest of the paper is structured as follows: In Section **??** and Section 2, we briefly describe the background knowledge, and highlight our goal and the thread model. In Section **??** we discuss the design rationale. Then we describe the system overview and implementation in Section **??** and Section **??**. In Section **??**, we evaluate the security and performance of the system, and discuss several attacks and possible extension in Section **??**. At last, we discuss the related work in Section **??**, and conclude the whole paper in Section **??**.

2 Problem Definition

2.1. Page Table Update in Bare-Metal Environments

References

[1] AMD. Secure virtual machine architecture reference manual. December 2005.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[3] Intel. Intel 64 and IA-32 architectures software developer's manual combined volumes: 1, 2a, 2b, 2c, 3a, 3b and 3c. October 2011.

[4] Derek Gordon Murray, Grzegorz Milos, and Steven Hand. Improving xen security through disaggregation. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, pages 151–160, New York, NY, USA, 2008. ACM.

[5] Andrew Whitaker, Marianne Shaw, Steven D Gribble, et al. Denali: Lightweight virtual machines for distributed and networked applications. Technical report, Citeseer, 2002.