Accelerating Virtual Machine Storage I/O for Multicore Systems

Dongwoo Lee[†], Junghan Kim[†], Junghoon Kim[†], Changwoo Min[†*], Young Ik Eom[†]
[†]*Sungkyunkwan University, Korea*
[*]*Samsung Electronics, Korea*
{dwlee, junghan, myhuni20, multics69, yieom}@ece.skku.ac.kr

1 Introduction

In a virtualized system, virtual machine monitor (VMM) decouples a logical device in guest OS from its physical implementation in host OS. It provides many advantages: time- and space-multiplexing of I/O devices for higher utilization, seamless portability even across heterogeneous systems, and live migration of a running virtual machine (VM) between physical machines.

In classical *trap-and-emulate* virtualization, executing an I/O instruction in a guest device driver triggers an *exit* to the VMM since I/O instruction is a sensitive instruction that cannot be handled in guest mode. To emulate the trapped instruction, VMM passes an I/O request to a separated driver domain (dom0 in Xen) or a user process (QEMU in KVM) that actually issues a request to a physical device. When the physical device finishes the I/O request, a completion notification is delivered to the guest OS by traversing in the reverse order.

Previous work [1, 2, 4, 5, 6], however, shows that I/O intensive workload might suffer unacceptable performance degradation due to virtualization overhead. I/O virtualization overhead induced by exits can be classified into three: The first is *direct cost*, which is the cost of world switching between guest OS and VMM. The second is *indirect cost*, incurred by the slowdown due to cache pollution results from executing guest OS and VMM on a single CPU. While VMM deals with an exit, all processor in the guest OS become completely stopped. It introduces the third type of cost, *synchronous cost*. Landau et al. [4] showed that the direct cost on its own is high and the indirect and synchronous cost can be even an order of magnitude higher.

Many previous studies identified exits as a major source of overhead [2, 4, 5], and proposed several techniques to reduce the number of exits. However, they still have limitations: Virtualization polling engine (VPE) [5] requires a polling device driver in the host OS. The hardware extension proposed in SplitX [4] is not available on commodity processor. Exitless interrupts (ELI) [2] is efficient only for direct device assignment and focuses only on exits induced by interrupts.

We propose a novel I/O virtualization framework, accelerating I/O performance in a virtual machine towards bare-metal performance, and show experimental results on a storage device using our prototype implementation. Comparing to the previous work, our approach is a software-only approach that does not require any special hardware extension on CPU or I/O device supporting direct access from a guest OS.

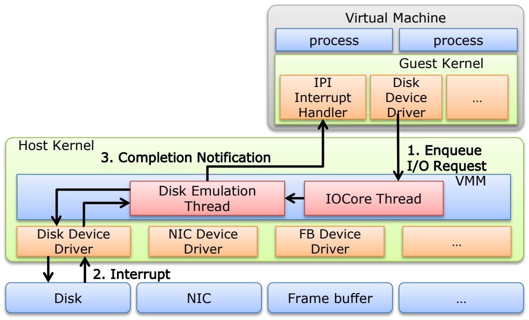

Figure 1: Architecture of the proposed I/O virtualization framework

2 I/O Virtualization Framework towards Bare-Metal Performance

Our I/O virtualization framework is designed for reducing the cost induced by exits. Figure 1 shows the overall architecture of the proposed framework. We propose three techniques to reduce the virtualization overhead by exploiting multicore architecture.

Exitless I/O Request: Instead of costly trap-and-emulate technique, a guest OS uses a para-virtualized device driver that communicates with VMM through shared request queue. Issuing an I/O request is just enqueuing a request in the shared queue. The polling thread, hereafter *IOCore thread*, in VMM checks if there are new requests in the queue and dequeues them for further processing. Since there is no exit to VMM, this process can be performed efficiently.

In-VMM Device Emulation: Typically, I/O request is emulated in a separated driver domain or user process. Since it involves additional switching, it increases the synchronous cost. To reduce the synchronous cost and latency, we emulate the I/O request in VMM, i.e. host kernel. IOCore thread passes the dequeued requests to the corresponding *per-device emulation thread* that handles the I/O requests for a particular device. Since

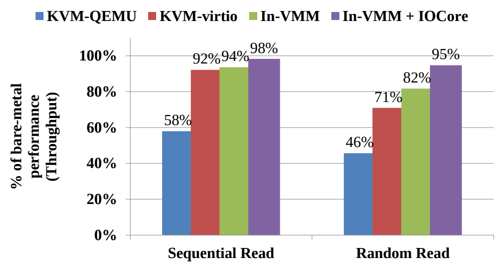

Figure 2: Performance comparison

the device emulation thread is in the host kernel, no additional mode switching is needed.

Exitless Completion Notification: After the device emulation thread completes the I/O requests, it sends back the completion notification to the CPU running the guest OS via an inter-processor interrupt (IPI). An IPI incurs two additional exits: one for injecting the interrupt to the guest OS and the other for signaling the completion of interrupt handling by letting the guest kernel write end-of-interrupt (EOI) register in local advanced programmable interrupt controller (LAPIC). To remove the first exit, interrupt descriptor table (IDT) in the guest OS is shadowed just in a similar way as page table shadowing. Since the *shadow IDT* is configured to directly deliver IPI to the guest OS, there is no exit. To remove the second exit, we use the newest LAPIC interface, *x2APIC* [3]. x2APIC can be configured to directly expose EOI register to the guest OS without any exits, when the guest OS writes the EOI register for signaling the completion of interrupt handling. These two techniques are proposed in Gordon et al. [2] in the I/O virtualization for direct device assignment. However, we adopt them in the context of IPI delivery to minimize the exit cost.

The proposed techniques can completely eliminate exits in I/O virtualization and significantly reduce the latency induced by mode switching. Therefore, we can significantly accelerate the I/O performance by reducing the three costs.

3 Experimental Results

To verify the proposed I/O framework, we implemented a prototype using KVM on Linux. Our prototype includes exitless I/O request and in-VMM device emulation for a block device. Exitless completion notification is still under development. We measured the performance of sequential reads and random reads on a high-end SSD in four different configurations: trap-and-emulate using KVM and QEMU (KVM-QEMU), para-virtualized device driver using `virtio` [7] (KVM-virtio), in-VMM device emulation (In-VMM), and in-VMM device emulation with the exitless I/O request technique (In-VMM + IOCore). As Figure 2 shows, in comparison with KVM-virtio, the in-VMM device emulation technique improves performance by 2-11%, and the exitless I/O request technique improves performance by 4-13%. As a result, our proposed approach closely reaches the bare-metal performance – 95% for sequential read and 98% for random read – without any special hardware support. Considering our prototype is not optimized and does not include the exitless completion notification technique, the performance result is quite encouraging.

Acknowledgements

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2011-0020520).

References

[1] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan. High performance network virtualization with SR-IOV. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –10, jan. 2010.

[2] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. ELI: Bare-Metal Performance for I/O Virtualization. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2012. To Appear, 2012.

[3] Intel Corporation. Intel 64 Architecture x2APIC Specification, 2008.

[4] A. Landau, M. Ben-Yehuda, and A. Gordon. SplitX: split guest/hypervisor execution on multi-core. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV'11, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

[5] J. Liu and B. Abali. Virtualization polling engine (VPE): using dedicated CPU cores to accelerate I/O virtualization. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 225–234, New York, NY, USA, 2009. ACM.

[6] H. Raj and K. Schwan. High performance and scalable I/O virtualization via self-virtualized devices. In *Proceedings of the 16th international symposium on High performance distributed computing*, HPDC '07, pages 179–188, New York, NY, USA, 2007. ACM.

[7] R. Russell. virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.*, 42:95–103, July 2008.