Optimizing System Call Latency of ARM Virtual Machines

Song-Woo Sok, Young-Woo Jung and Cheol-Hun Lee

1High-Performance Computing Dept., Electronics and Telecommunications Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon, 34129, Korea
2Department of Computer Science & Engineering, Chungnam National University, 99 Daehak-ro, Yuseong-gu, Daejeon 34134, Korea
3swsok@etri.re.kr, 4jungyw@etri.re.kr, 5clee@cnu.ac.kr

Abstract. This paper introduces ViMo-S, a type 1 hypervisor for ARMv7 and ARMv8-based ARM server systems. It supports full virtualization to run existing operating systems and applications unmodified. It uses ARM hardware virtualization extensions to optimize the performance of virtual machines, especially system call latency. Therefore, its virtual machines’ system call latency is near physical machine’s, while other hypervisors like Xen and KVM show relatively slower and unstable performances in benchmark tests.

1. Introduction
ARM-based processors are dominant in the mobile computing market because of low power consumption and high performance per watt ratio. As the performance of ARM processors keeps close to x86 processors, ARM servers are becoming popular for its low power and high-density characteristics.

These ARM-based servers give benefits of lower TCO with less power consumption, low heat emission, and smaller space needs. To utilize the server more efficiently, Supporting consolidation based on virtualization technologies is necessary [1].

Traditionally, ARM architectures had no support for virtualization technologies. Since ARMv7 architecture, hardware virtualization extensions are added to support virtualization software [2]. It helps virtualization software to be more lightweight, efficient and able to run existing operating systems and applications unmodified.

![Figure 1. ARM exception levels and software layers of non-virtualized and virtualized systems](image-url)
Figure 1 shows exception levels of ARM processors which include the virtualization extensions. ARM architectures also support EL3, most privileged exception level. However, it is specially designed to make an isolated, secure world and mostly used for mobile devices. Moreover, some ARM server SoCs do not support EL3 exception level. So, EL3 mode is not considered in this paper.

Without virtualization, EL1 is the highest exception level, and it is used to execute operating system kernel. While user programs run in EL0, the least privileged exception level. To support virtualization and run existing operating systems and applications unmodified, virtualization extensions add more privileged exception level, EL2. In Figure 1, non-virtualized, type 1 virtualized and type 2 virtualized software layers are shown.

Xen is a popular type 1 hypervisor that supports both x86 and ARM architectures. As a type 1 hypervisor, Xen ARM is executed before virtual machine (VM)'s OS boot. Then, it creates a special VM, domain0 which has responsible for controlling hardware devices and executing I/O operations on behalf of other VMs, called domainUs [3].

KVM is a type 2 hypervisor technology, and it runs as a part of host operating systems. It was developed to support full virtualization on x86 architectures. When it was ported to ARM architectures, KVM had to be redesigned to adopt virtualization extensions [4]. The result is called KVM/ARM, and its core component was separated into two parts. One part called as highvisor is executed as a part of the host’s operating system, the other part called as lowvisor runs in EL2 to use benefits of ARM virtualization extensions. KVM supports full virtualization, but full virtualized I/O operations have lower performance compared to para-virtualized ones. Therefore, KVM supports VirtIO para-virtual I/O device to increase performance and reduce overheads. VirtIO is a standard para-virtual I/O interface for virtualized computing environment.

This paper presents ViMo-S which is a type 1 hypervisor based on ARM virtualization extensions. It supports full virtualization for CPU and Memory, while it uses VirtIO interface for I/O operations of VMs. If an operating system has VirtIO drivers, it can be executed on ViMo-S' domainUs without modifications.

2. Design and implementation of ViMo-S

ViMo-S is the following project of ViMo (Virtualization for Mobile), a micro virtual machine monitor for ARM mobile systems. It supported full virtualization on ARM architectures that have no hardware virtualization support [5].

ViMo-S has been developed for ARM server applications based on ViMo architecture. Because ViMo-S employs ARM hardware virtualization technologies, it becomes simpler and more efficient compared to ViMo.

2.1. Design of ViMo-S

2.1.1. CPU Virtualization. Because ARM virtualization extensions are more suitable to type 1 hypervisors [1], ViMo-S was planned to be a type 1 hypervisor from the start. So, ViMo-S is executed right after the boot loader in EL2 level; then it initializes the first VM, domain0, and boots its operating system.

All VMs can have one to four virtual CPUs (VCPUs). When a VCPU is scheduled to a physical core, ViMo-S saves the current context of the core to the related VCPU then loads scheduled VCPU’s context to the core. Therefore, 2 or more VCPUs can share a core to run.

User applications execute many system calls when they are running. ViMo-S sets system calls to be routed to EL1, not to EL2. Because the VM's kernel directly handles all system calls and there is no intervention of the hypervisor, system call latency of the VMs on ViMo-S is near to a non-virtualized system.
2.1.2. Memory Virtualization. The ARM virtualization extensions support hardware 2-stage memory translation to support memory virtualization for VMs. The first stage’s page tables are configured by operating systems of VMs, and its translated address is not a real physical address (PA) but an intermediate physical address (IPA). If the second stage page tables are configured in EL2, then an IPA can be translated to a PA by hardware MMU.

ViMo-S allocates VM’s physical memory and configures second stage page tables for each VMs. ViMo-S allocates memory area exclusively to each VMs, so, a VM cannot access to other VM’s memory area. However, ViMo-S can create shared memory space by configuring each 2nd stage page tables of VMs to have same physical memory address entries.

2.1.3. I/O virtualization. ViMo-S supports VirtIO interface for para-virtualized I/O operations. If domainU’s operating system has VirtIO drivers, it can use a console, disk and network VirtIO virtual devices to execute I/O operations.

Figure 2 shows the processing path of virtual I/O requests. DomainU’s VirtIO requests are trapped to ViMo-S and ViMo-S delivers I/O requests to the domain0’s VirtIO backend driver. Then, the VirtIO backend driver maps the requests to real hardware I/O operations and calls hardware drivers to process the requests.

![Figure 2. I/O Virtualization of ViMo-S](image)

2.2. Implementation
ViMo-S has been implemented on X-C1 Server Development Platform board developed by AppliedMicro.

The main SoC of the board is AppliedMicro’s X-Gene 1, an ARMv8 compliant processor. It has eight cores and runs at 2.4 GHz clock speed. The X-C1 has 16GB of DDR3 RAM, three ports of Gigabit Ethernet and two SATA ports. A Samsung 850 EVO 500GB SSD was used as storage device.

We used Linux 3.15-rc8 kernel for domain0 and Linux 3.18.0 for domainUs. Ubuntu 14.04 ARM64 version was used for both domain0 and domainU’s root file systems.

3. Benchmark tests and results

3.1. Benchmark test configuration
To evaluate the performance of ViMo-S, we used lmbench 3.0 to check system call latencies of virtualized and non-virtualized environments on the X-C1 board. The tests included ViMo-S, Xen ARM, and KVM/ARM configurations. The lmbench is a benchmark suite to measure system’s micro performance factors like system calls, memory accesses, network, and disk performances [6].

Table 1 shows configurations for each system.
First, we conducted lmbench tests of non-virtualized Linux, ViMo-S and Xen’s domain0 VMs. Table 2 shows the results of lmbench system call latency tests of raw Linux and domain0 VMs. For easy comparison, it includes latency ratios based on raw Linux results also.

### 3.2. Raw Linux and Domain0s

As shown in Table 2, raw Linux and ViMo-S have very similar results, while Xen Domain0 shows relatively slower performances. Xen’s domainU shows similar performance with Xen’s domain0’s. It has 1.8 to 3.3 times slower than raw Linux.

### 3.3. DomainUs and KVM/ARM guest

As shown in Table 2, raw Linux and ViMo-S have very similar results, while Xen Domain0 shows relatively slower performances. Xen’s system call latencies are 2.3 to 3 times slower than ViMo-S and raw Linux.

Table 3. lmbench test results for KVM/ARM guest and domainUs (unit=milliseconds)

|         | Simple syscall | Simple read | Simple write | Simple stat | Simple fstat | Simple open/close | Protection fault | Pipe latency |
|---------|----------------|-------------|--------------|-------------|--------------|--------------------|------------------|--------------|
| Raw Linux | 0.1318         | 0.1845      | 0.2312       | 0.8316      | 0.1936       | 2.4462             | 0.2057          | 4.7348       |
| ViMo-S   | 0.1316         | 0.1833      | 0.2308       | 0.8406      | 0.1951       | 2.1689             | 0.1993          | 4.6733       |
| Xen ARM  | 0.3834         | 0.5556      | 0.6008       | 2.5131      | 0.4516       | 5.9182             | 0.5072          | 13.6326      |
| ViMo-S/Linux | 99.8%        | 99.3%       | 99.8%        | 101%        | 100.8%       | 88.7%              | 96.9%           | 98.7%        |
| Xen/Linux | 291%           | 290%        | 260%         | 302%        | 233%         | 242%               | 247%            | 288%         |

### 4. Conclusion and Future Works

ViMo-S’s domainU has relatively slower performance than ViMo-S’s domain0. However, its latencies did not exceed 1.5 times of raw Linux’s.

Xen’s domainU shows similar performance with Xen’s domain0’s. It has 1.8 to 3.3 times slower than raw Linux.

KVM/ARM’s guest VM shows interesting results. For some system call tests, its latencies are slightly faster than ViMo-S but significantly slower for simple stat, simple open/close and pipe latency tests.
This paper proposed ViMo-S, a new type 1 hypervisor for ARM architecture. ViMo-S supports ARM virtualization extensions to optimize VM’s performance and minimize virtualization overheads. We showed that ViMo-S has minimized system call latency for VMs and provides similar performance with bare metal machines.

According to our experiment results, ViMo-S has better and more stable system call latency performances compared to Xen and KVM/ARM.

We are trying to optimize I/O performance of ViMo-S and doing more benchmark tests with more complex benchmark tools and real world applications.

Acknowledgments
This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) [R0101-16-237, Development of General-Purpose OS and Virtualization Technology to Reduce 30% of Energy for High-density Servers based on Low-power Processors]

References
[1] Dall C, Li S W, Lim J T, Nieh J and Koloventzos G. 2016 ARM Virtualization: Performance and Architectural Implications. In Proceedings of International Symposium on Computer Architecture ISCA 2016.
[2] Varanasi P and Heiser G 2011 Hardware-supported virtualization on ARM. In Proceedings of the Second Asia-Pacific Workshop on Systems p11. ACM.
[3] Stabellini S and Campbell I. 2012 Xen on arm cortex a15. Xen Summit North America, 2012.
[4] Dall C and Nieh J. 2014 KVM/ARM: the design and implementation of the Linux ARM hypervisor. In ACM SIGPLAN Notices Vol. 49, No. 4, pp. 333-348. ACM.
[5] Oh S C, Kim K, Koh K and Ahn C W 2010 ViMo (virtualization for mobile): a virtual machine monitor supporting full virtualization for ARM mobile systems. In Proceedings of Advanced Cognitive Technologies and Applications, COGNITIVE.
[6] McVoy L W and Staelin C 1996 lmbench: Portable Tools for Performance Analysis. In USENIX annual technical conference pp. 279-294.