Performance overhead analysis of virtualisation on ARM

Xiaoli Gong, Zhaoyang Shao and Qi Du
Department of Computer Science and Information Security,
Nankai University,
Tianjin, China
Email: gongxiaoli@nankai.edu.cn
Email: zhaoyangnk@gmail.com
Email: nkduqi@126.com

Aimin Yu
Institute of Information Engineering,
Chinese Academy of Sciences,
Beijing, China
Email: yuaimin@iie.ac.cn

Jin Zhang*
Department of Computer Science and Information Security,
Nankai University,
Tianjin, China
Email: nkzhangjin@nankai.edu.cn
*Corresponding author

Abstract: There is growing interest to replace traditional servers with low power systems. ARM has released new architecture with hardware virtualisation extension support. However, the performance of virtualisation on ARM-based low power platform is unclear. In this paper, a number of performance measurements on basic operations such as disk speed, CPU, memory and network throughput were done on a dual-core ARM Cortex-A7-based hardware platform with two popular open source hypervisor software, Xen and KVM. Basic benchmarks and three popular applications are measured to make a comparison between virtual machine and native machine. Our result shows that the average performance overhead of Xen and KVM virtual machine is between 3% and 4% when the host is lightly loaded, while the performance of three applications drops drastically with the workload stress increase in the system. The IO virtualisation should be optimised for the industrial usage of ARM-based virtualisation server.

Keywords: virtualisation; Xen; KVM; ARM; ARM Cortex-A7; performance analysis; performance measurement; performance overhead; benchmark; virtual machine; hypervisor; information communication technology.

1 Introduction

As one of the key technologies in the past five years, cloud computing has played an important role in changing the way companies consume and use information technology. The internet data centres (IDC) are the basic platforms for cloud computing. Nowadays most of the computers in IDCs are based on the X86 architectures, while virtualisation technology for X86 has made great strides in the past ten years. With the rapid development of ARM-based CPUs, it becomes a typical low power consumption solution for mobile system. At the same time, there is a growing demand for providing the benefits of power saving feature of ARM-based devices to the server system. In an attempt to strengthen the entry of ARM processors into the server market, ARM has proposed the Server Base System Architecture (SBSA), a definition of a standard platform for ARM-based servers. This specification solved the problem of compatibility in server market. In order to deploy virtualisation technology on ARM platform, we have to solve two problems: one is that which hypervisor should we choose and the other is how about the performance of the virtual machine on ARM platform. To date much of research and engineer effort about the porting to ARM platform has been focused on two main open source virtualisation solutions, which are Xen and KVM (Dall and Nieh, 2014; Hwang et al., 2008; Abels et al., 2005). However, these researches only focused on
the principle and implementation of ARM virtualisation, or on the security improvement for mobile devices (Chen et al., 2015).

Earlier papers on virtualisation performance analysis have focused on the X86 platforms (Langer and French, 2011; Graziano, 2011; Cherkasova and Gardner, 2005; Menon et al., 2005). The results showed that Xen is a quite capable hypervisor on X86, with an overhead of about 3.4% on average.

In order to find out whether ARM-based hardware is ready for the server market, the performance of the virtualisation system should be evaluated. The evaluation should include two aspects: first is the performance of ARM-based CPU and second is the performance downgrade when running in the virtual environment. However, most of the products based on ARM currently available are low cost chips for consumer electronics, whose performance is far below the tradition servers. Therefore, in this research we focus on the performance downgrade caused by the virtualisation hypervisor software. Thanks to Rasmusson and Corcoran’s (2014) work, we have got a paper about the performance overhead of KVM on ARM.

To make a fair comparison, we have run identical workload with Xen and KVM on ARM platforms, and we make the comparison between the native machine and the Xen (KVM) guest machines. The benchmark programs include computing, memory, IO and system calls. What is more, we also design the benchmark about concrete IDC services such as web server (Apache), database (MySQL), cached servers (Memcached). In this paper, we will give the results of measurements that were performed to make a comparison of the execution overhead between Xen and KVM hypervisor, on an ARM Cortex-A7-based hardware platform.

The rest of the paper is structured as follows. Section 2 gives a simple introduction to virtualisation technology and hardware extension to support virtualisation. Section 3 introduces the benchmark program selection and the experiment environment is introduced in Section 4. In Section 5, we analysis the measurement result and Section 6 concludes the paper.

2 The virtualisation technology

In this section, we will do a simple introduction to virtualisation technology, including the software support and hardware support. The software support means the hypervisor and virtual machine management software. Here we focus on the two mainstream open-source hypervisor, KVM and Xen. The hardware support is the extension to make hypervisor available with acceptable performance, such as more privilege mode, memory management support, interrupt support and so on.

2.1 Software support

Kernel-based virtual machine (KVM) and Xen are two main open source virtual machine, and they are already ported to the ARM platform, which makes us to consider evaluate the performances overhead of them. The KVM is a full virtualisation solution for Linux on x86 hardware containing virtualisation extensions (Inter VT or AMD-V). It has been ported to the ARM platforms by Columbia University (Dall and Nieh, 2014). The architecture of the KVM has been showed in Figure 1.
KVM/ARM is the first hypervisor to leverage ARM hardware virtualisation support to run unmodified guest operating systems on ARM multi-core hardware. The KVM introduce the split-mode virtualisation, which makes the hypervisor splits into high visor and low visor. The high visor runs in normal privileged CPU integrate with the Linux kernel, which makes it easier to adapt to the existing device drivers and management tools without modification. The low visor works in hypervisor mode to leverage ARM virtualisation features. The administrators manage the virtual machines by QEMU or kvmtools command set from the user space of high visor. The hosted hypervisor makes KVM easy to maintain and gain wide adoption as the virtualisation platform of choice for ARM.

The Xen (Barham et al., 2003) is an open-source bare metal hypervisor, and it can support many instances of different operating systems in parallel on a single hardware. As shown in Figure 2, Xen runs directly on the hardware, everything else in the system is running as a virtual machine on top of Xen. Domain 0 is the first virtual machine created by Xen, which is privileged and provide the devices support on the platform.

For the device access in the virtual machines, the device driver is separated into fronted and backend. Domain 0 is the only domain which actually controls the hardware, and it also runs a set of drivers named as para-virtualised backends to give access to the real hardware to the other unprivileged guest domains. The guest domain gets access to a set of generic virtual devices by running the corresponding fronted drivers and the concurrent access of shared resource is taken care by the Xen hypervisor.

Xen runs in hypervisor mode, and the virtual machine kernels run in the original kernel mode and user mode is left for guest user space applications. In this way, Xen could significantly reduce the number of context switches comparing with KVM.

The Xen hypervisor supports two different types of guests: para virtualisation (PV) and hardware assisted virtualisation (HVM). PV does not require virtualisation extensions from the host CPU. However, PV guests require a PV-enabled kernel and PV drivers, so that the guest are aware of the hypervisor and can run efficiently without device emulation. In this way, Xen has been ported to ARM-based device even though without the hardware extension.
2.2 Hardware support

Before ARMv7 series, the ARM cannot support virtualisation due to the lack of privilege levels. Also, there are a number of sensitive instructions cannot be trapped by privileged mode. The Cortex-A7 is one of the ARMv7 architectures with hardware virtualisation support and it provides a low cost solution for low power computing. Here, a brief overview is proposed about the ARM virtualisation extensions and more detailed information is mentioned in Varanasi and Heiser’s (2011) paper.

- **CPU virtualisation**: The ARMv7 introduces a new processor mode, hypervisor (Hyp) mode. As shown in Figure 3, the hardware virtualisation extensions only apply to non-secure mode. The virtual machines will execute normally in user and kernel mode and the hypervisor call (HVC) instruction is used to enter the Hyp mode. The ARM architecture allows many exceptions to be configured to trap either into the hypervisor or the guest kernel. The system calls in guest system can be trapped and handled directly by the guest. It drastically reduces the frequency of hypervisor context switches.

- **Memory virtualisation**: To protect the hypervisor, the memory address translation cannot be handled directly in the guest kernel. The hypervisor will do the final step to map the address of guest system to the physical memory. The ARM virtualisation extensions provide hardware support for two level address translations. The hypervisor essentially sets the hardware to accept the address request generated by the guest OS and treated them as intermediate physical address (IPA). Another level of translation on these addresses is called to translate from IPA to PA. New page tables use ARM’s new LPAE page table format, with subtle differences from the page tables used by the kernel mode.
• **Interrupt virtualisation:** With virtualisations, the interrupt might need to be routed to the hypervisor or different guest OS. The physical interrupts are handled initially in the hypervisor. In order to avoid emulation of the interrupt controller, virtual interrupts are introduced, which means if there is an interrupt should go to one guest OS, the hypervisor maps a virtual interrupt for that guest OS. The virtual interrupts is supported by a new hardware component, the virtual CPU (VCPU) interface. This can be mapped into the guest to acknowledge and clear interrupts without trapping into the hypervisor. The sequence of interrupts events is shown in Figure 4. If interrupts are configured to be handled by the hypervisor, the hypervisor can explicitly forward the interrupt to the current guest by raising the appropriate virtual interrupt on the guest’s VCPU interface.

• **IO virtualisation with system MMU:** In virtualisation, IO performance is limited to the performance of hypervisor, because the hypervisor traps all of the operations in the full virtualisation mode. Pass-through I/O greatly improves performance by remapping the guest page tables to directly write to the physical device. This eliminates most of the overhead in trapping to the hypervisor for every operation. This technique brings the bulk of I/O processing to near-native speeds. Most platforms have hardware solutions for pass-through IO management. This mechanism is called IOMMU for IO memory management unit. While the technology is wildly used in industry such as VT-d from Intel and AMD-Vi from AMD, ARM has proposed their implementation as system MMU. However, system MMU is not implemented in the current released Cortex-A7 architecture and it can be seen that the IO performance drops in full virtualisation mode on ARM.

**Figure 3** ARM processor mode (see online version for colours)
3 Benchmark program design

In this research, we will evaluate the performance of the virtual machines on basic operations frequently used in the server system and classic applications in data centres. First, some basic operations including CPU, disk read, network throughput and system call are measured with workloads. Second, we measured real application performance using a variety of workloads. Both of the measurement are performed identically on native system, Xen-guest VM, KVM-guest VM. The tools we used to test the basic operations are very simple and effective, such as hdparm, geekbench2, lmbench and even dd command. These tools make the experiments simple to repeat.

3.1 CPU and memory

We use the Geekbench2 to measure the system performance for workloads that stressed the CPU and memory. Geekbench 2 provides a comprehensive set of benchmarks, such as ‘integer performance’, ‘floating point’, ‘memory’, ‘stream processing’, to quickly and accurately measure processor and memory performance. The binary program of the benchmark is online available and when all the test programs have completed, it will give a URL and we can get the result through this URL.

3.2 Disk read speed

The virtual in Xen (KVM) virtual machine, the virtual disk is xvda (vda). We design two experiments to test the performance of the virtual disk compared with the native disk, which is SD card. First, we measured the read speed on two different disk devices with command like: hdparm – Tt <dev>. We will test the disk and loopback device respectively.
After that, we measured the IO read and write rate on the disk device and loopback device of the native, Xen (KVM) system. We also assign the file size as 1 K or 1 M that the dd command read or write.

- **write:** `dd bs=1M count=100 if=/dev/zero of=test conv=fdatasync`
- **read:** `dd if=./text of=/tmp/test1 bs=1M count=100`.

For meaningful results, this operation should be repeated two to three times on an otherwise inactive system (no other active processes) with at least a couple of megabytes of free memory.

### 3.3 Network throughput

To observe the effect of network virtualisation from Xen (KVM), we used `nc` and `dd` to test the network throughput performance. We set up a listener process on one machine which listened on a port while we send 100 MB from `/dev/zero` of another machine to that port.

- **listener:** `nc-lk <port> >/dev/null`
- **sender:** `dd if=/dev/zero bs=1M count=100 | nc <listener> <port>`.

The transmission rate is printed out by `dd` when transmission is complete. The reported numbers below are the average over three runs, because this could get more accurate statistical data. There are errors ranging 0.4 MB/s up and down.

We have designed two experiments to test, respectively, the performance comparison between the native host and Xen (KVM) guest machines. We are also interested at the performance when two virtual machines concurrently send and receive messages.

### 3.4 System call

To observe the performance of system calls, a simple benchmark set Lmbench is used. Lmbench is a micro-benchmark suite designed to focus attention on the basic building blocks of many common system applications, such as databases, simulations, software development, and networking (McVoy et al., 1996). A set of basic system calls, such as page fault, tcp, pipe, and file operations, are provided and the program is written in ANSI C with the source code available online.

### 3.5 Application-based benchmark

Three applications workloads we used are showed in Table 1. We run the ApacheBench v2.3 to test the performance on the native system, Xen-guest VM, KVM-guest VM. We start the Apache Server in the machine which is tested and we use the `ab` command from the other machine out of the board:

```
ab -c 100 -n 50000 http://10.0.0.216/index.php
```
Table 1 Three applications workloads used in the benchmark

<table>
<thead>
<tr>
<th>Applications</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Apache v2.2 Web server running ApacheBench v2.3 on the local server, which measures number of handled requests per seconds serving the index file of GCC 4.4 manual using 100 concurrent requests.</td>
</tr>
<tr>
<td>Mysql</td>
<td>MySQL v14.14 (distrib 5.5.40) running the SysBench benchmark using the default configuration.</td>
</tr>
<tr>
<td>Memcached</td>
<td>Memcached using the memslap benchmark with a concurrency parameter of 1,000.</td>
</tr>
</tbody>
</table>

This tool completes 50000 requests and transfers about 2305000 bytes data. The cached systems are also be widely used in large systems. We use command below to measures performance between the Xen (KVM) virtual machines and native system.

```
memslap -flush -concurrency 30 -executor-number 10000 -servers 127.0.0.1
```

We chose to evaluate MySQL because it is a popular relational database, it is widely used in cloud, and it stresses memory, IPC, network and file system subsystems. We run the SysBench to measure the performance of MySQL. We set the innodb_flush_logs_at_trx_commit as 2 so that it will not flush the result to the disk per transaction. We make this set because the MySQL server process cannot start normally on the KVM’s virtual disk (VDA). The SysBench has executed about 210,000 operations which are made of 140,000 reads, 50,000 writes and 20,000 other operations.

4 Experiment environment

In this section, we will give an introduction to the running environment and software configuration of the experiment. All the software are based on the latest stable version from the official site and we do some slight modification when porting to the target hardware platform, and we believe it will not affect the performance measurement.

4.1 Hardware and deployment

The experiment target hardware is cubietruk board. Cubietruck is the third board of Cubieteam from Allwinner Corp. It is a nice mini-computer or a robotic platform running various Linux distributions or Android system. The Cubietruck consists of a motherboard, IO peripherals, microcontrollers system and one slot to plug the processor. The processor of the Cubietruck is A20 from Allwinner Corp, with dual-core Cortex-A7 working on 1.2 GHz. The Cubietruck has 2 GB of DDR3 RAM for memory and 4 GB of NAND Flash for storage (Cubietech Cubietruck, 2014).

The network interface of Cubietruck is RTL8211E 10M/100M/1G ethernet PHY. It uses state of the art DSP technology and an analogue front end (AFE) to enable high-speed data transmission and reception over UTP cable.
As shown in Figure 5, for the network performance test, we used two Dell machines with common ethernet interfaces and the throughput is limited to 100 M bit/s. Two Xen (KVM) virtual machines inside a Cubietruck with one Cortex-A7. Both guests are running Linux 3.13. The virtual machine use fronted and backend driver to communicate with the disk and network driver managed by xl (lkvm). Two Dell PC machines with Ubuntu 12.10 are attached to the network to send/receive traffic. The network traffic shaping does not observably affect the results of the experiments, as the network card of the Cubietruck was not able to reach up to 100 M bit/s. The experiments in this paper were conducted with the default RTL8211E NIC.

**Figure 5** Experimental setup

4.2 Software

The Xen hypervisor and tools we used is Xen-4.4.0-stable. The KVM we used is the version which is along with the Linux 3.13. All the guest machines in virtual environment are running Linux 3.13 with Ubuntu raring file system. For the SMP support is not ready for Cubietruck in virtualisation mode, so we disabled the SMP feature whether it is native system or the guest system. For the virtual machine management on Xen, there are xl tools go with the corresponding Xen source code. For KVM, kvmtools integrated with Linux kernel (Kvmtool Source Code, 2014). Two Dell PC machines with Ubuntu 12.10 are attached to the network to send/receive traffic. To enable Xen, the ARM processor must be booted in Hyp mode. This is not supported by the official bootloader from Allwinner, so we use a bootloader which has not been merged into the master branch. It sets up the CPUs in the right way before invoking the kernel. The bootloader is also supported device trees, so it can construct the device node by the fdt command. In the native system, the root file system is on the SD card. When try to boot the guest OS, the root file system is simulated by vda or xvda virtual block device in the host system, which is connected to a file on the SD card via loopback device.
5 Result and analysis

5.1 CPU and memory

Figure 6 shows the CPU and memory performance for running Geekbench2 in Xen-guest VM, KVM-guest VM and native system. The results of Geekbench2 tests are shown in Table 2.

We can see that the performance of Xen-guest VM and KVM-guest KVM on the ARM are almost the same. The Xen has a better performance than KVM, but the performance gap between them is not big. The performance of VM could reach 98.5% to 98.8% of the native machine on the ARM platform. From Figure 6 we can see that the performance guest machine of Xen on ARM does is almost the same as the native host. The result is satisfactory and it proves that the study about virtualisation for ARM is promising.

Figure 6 The scores about the CPU and memory intensive Geekbench2 test for Linux native, Xen-guest and KVM-guest

Table 2# CPU and memory test result of GeekBench2

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Xen-guest</th>
<th>KVM-guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream tests</td>
<td>593</td>
<td>586</td>
<td>577</td>
</tr>
<tr>
<td>Memory tests</td>
<td>1,099</td>
<td>1,080</td>
<td>1,081</td>
</tr>
<tr>
<td>Integer tests</td>
<td>437</td>
<td>431</td>
<td>431</td>
</tr>
<tr>
<td>Floating point</td>
<td>951</td>
<td>942</td>
<td>940</td>
</tr>
<tr>
<td>Geekbench score</td>
<td>764</td>
<td>755</td>
<td>753</td>
</tr>
</tbody>
</table>

5.2 Disk read speed

The averaged results of the disk read speed tests measure with hdparm command are shown in Table 3 and Figure 7.
Table 3  Cached reads and Buffered reads result of hdparm

<table>
<thead>
<tr>
<th>Platform</th>
<th>Device and backing store</th>
<th>Cached reads (MB/s)</th>
<th>Buffered reads (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>/dev/mmcblk0</td>
<td>457.24</td>
<td>20.18</td>
</tr>
<tr>
<td>Native</td>
<td>/dev/loop0 on mmcblk1</td>
<td>496.50</td>
<td>80.36</td>
</tr>
<tr>
<td>Xen-guest</td>
<td>/dev/xvda</td>
<td>445.82</td>
<td>85.07</td>
</tr>
<tr>
<td>Xen-guest</td>
<td>/dev/loop0 on xvda</td>
<td>438.95</td>
<td>71.85</td>
</tr>
<tr>
<td>KVM-guest</td>
<td>/dev/vda</td>
<td>423.29</td>
<td>111.64</td>
</tr>
<tr>
<td>KVM-guest</td>
<td>/dev/loop0</td>
<td>429.90</td>
<td>76.08</td>
</tr>
</tbody>
</table>

Figure 7  Disk read speeds to device, as measured by hdparm – Tt <dev>

The averaged results of the experiment measure with dd command are shown in Table 4 and Figure 8.

Cached reads displays the speed of reading directly from the Linux buffer cache without disk access. This measurement is essentially an indication of the processor, cache, and memory of the system under test. Buffered disk reads displays the speed of reading through the buffer to disk without any prior caching of data. This measurement is an indication of how fast the drive can sustain sequential data reads under Linux, without any file system overhead. First we measured the read speed on the native system. In theory, this should be the upper limit on the performance. The data showed that the SD disks were quite slow, around 20 MB/s, where a hard driver performance today usually is hundreds of MB/s. We also created a 100 MB file and mounted it as a loopback device: /dev/loop0. The result is that the speed of buffered read rate increases to 80.36 MB/s because the file’s content has been stored in the page cache. If the cache does not miss, the reading rate would reach 496.50 MB/s.

From the results, we could get the conclusion that the Xen-guest and KVM-guest will add about 3% to 8% performance overhead with the virtual disk. The Xen-guest got a better score because its para-virtualisation. When the IO request reaches, it will invoke the guest’s fronted driver, then backend driver in domain 0 and at last the disk driver will
Performance overhead analysis of virtualization on ARM

do the real work. The loopback device in the virtual environment has just reached the 40% to 50% performance compared with that in native system.

For cached reads the results look more stable. The cache reads, which do not invoke the backed driver very much, the guest machine achieves around 97% of performance of the native host.

We observe that the rate of buffered reads has increased from 20 MB/s (native) to 85 MB/s (Xen-guest) and 111.64 MB/s (KVM-guest). There are buffers to cache the data in the virtual machine environment, so the speed of reads from /dev/xvda or /dev/vda is higher than that in the native host. To measure the performance of direct IO access, we do data synchronisation after each data write and wait until all the data is written to the device. The result of IO read and write has been shown in Table 4 and Figure 8.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Device and backing store</th>
<th>100 M</th>
<th>1 K</th>
<th>100 M</th>
<th>1 K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>/dev/mmcblk0</td>
<td>1.0</td>
<td>13.5</td>
<td>0.071</td>
<td>4.6</td>
</tr>
<tr>
<td>Native</td>
<td>/dev/loop0 on mmcblk1</td>
<td>1.0</td>
<td>10.5</td>
<td>0.03</td>
<td>1.9</td>
</tr>
<tr>
<td>Xen-guest</td>
<td>/dev/xvda</td>
<td>0.9</td>
<td>90.4</td>
<td>0.017</td>
<td>1.5</td>
</tr>
<tr>
<td>Xen-guest</td>
<td>/dev/loop0 on xvda</td>
<td>0.95</td>
<td>68.1</td>
<td>0.001</td>
<td>3.3</td>
</tr>
<tr>
<td>KVM-guest</td>
<td>/dev/vda</td>
<td>0.52</td>
<td>75</td>
<td>0.022</td>
<td>2.1</td>
</tr>
<tr>
<td>KVM-guest</td>
<td>/dev/loop0</td>
<td>0.43</td>
<td>36</td>
<td>0.001</td>
<td>1.3</td>
</tr>
</tbody>
</table>

From Table 4, we can see that the virtualisation will add overhead performance on the disk read speed, but the write speed of the Xen-guest and KVM-guest is almost the same as the native system. When the file size is only 1 K, the speed of read and write will decrease largely compared with 100 M, this could explain why a lot of small file will cost
more time than a complete large file. If we set the block size as 1 K when we use the dd
cmd

5.3 Network throughput

Table 5 shows the results of network throughput in MB/s for sending to localhost, the
sending (receiving) data to (from) the Dell over the network, single machine
sending/receiving respectively for the native host and the guest machine in Xen (KVM).
The first column records the sender process and the second column records the listener
process. To observe if the file size will affect the performance, we perform experiments
for 1 K and 100 M respectively. We can reach the conclusion that the Xen-guest’s
network performance is almost the same as the native system. The KVM-guest’s network
performance is only the 54% of the native system. The KVM uses virtio and vhost to
minimise virtualisation overhead. Our results show that we have not solves the
KVM-guest’s network throughput problem. We should deal with this problem in
future work.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Listener</th>
<th>Rate (MB/s)</th>
<th>1 K</th>
<th>100 M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>Localhost</td>
<td>0.072</td>
<td>32.10</td>
<td></td>
</tr>
<tr>
<td>Xen-guest</td>
<td>Localhost</td>
<td>0.048</td>
<td>29.7</td>
<td></td>
</tr>
<tr>
<td>KVM-guest</td>
<td>Localhost</td>
<td>0.043</td>
<td>29.13</td>
<td></td>
</tr>
<tr>
<td>Native</td>
<td>Dell</td>
<td>0.056</td>
<td>11.80</td>
<td></td>
</tr>
<tr>
<td>Xen-guest</td>
<td>Dell</td>
<td>0.038</td>
<td>12.10</td>
<td></td>
</tr>
<tr>
<td>KVM-guest</td>
<td>Dell</td>
<td>0.038</td>
<td>6.40</td>
<td></td>
</tr>
<tr>
<td>Dell</td>
<td>Native</td>
<td>0.260</td>
<td>11.80</td>
<td></td>
</tr>
<tr>
<td>Dell</td>
<td>Xen-guest</td>
<td>0.283</td>
<td>11.80</td>
<td></td>
</tr>
<tr>
<td>Dell</td>
<td>KVM-guest</td>
<td>0.289</td>
<td>9.90</td>
<td></td>
</tr>
</tbody>
</table>

The Xen virtual machine’s receiving rate is up to the 99% of the native host. This is a
very satisfactory performance compared with the KVM virtual machine. Tables 6 and 7
show the network rate when two Xen (KVM) virtual machines communicate with two
Dell machines concurrently.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Listener</th>
<th>Rate (MB/s)</th>
<th>1 K</th>
<th>100 M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xen-guest1</td>
<td>Dell1</td>
<td>0.028</td>
<td>0.056</td>
<td>5.83</td>
</tr>
<tr>
<td>Xen-guest2</td>
<td>Dell2</td>
<td>0.028</td>
<td>0.056</td>
<td>5.60</td>
</tr>
<tr>
<td>Dell1</td>
<td>Xen-guest1</td>
<td>0.251</td>
<td>0.428</td>
<td>6.18</td>
</tr>
<tr>
<td>Dell2</td>
<td>Xen-guest2</td>
<td>0.177</td>
<td>0.056</td>
<td>6.00</td>
</tr>
</tbody>
</table>
Table 7  Concurrency sending and receiving with KVM-guests

<table>
<thead>
<tr>
<th>Sender</th>
<th>Listener</th>
<th>Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1K</td>
</tr>
<tr>
<td>KVM-guest1</td>
<td>Dell1</td>
<td>-</td>
</tr>
<tr>
<td>KVM-guest2</td>
<td>Dell2</td>
<td>-</td>
</tr>
<tr>
<td>Dell1</td>
<td>KVM-guest1</td>
<td>-</td>
</tr>
<tr>
<td>Dell2</td>
<td>KVM-guest2</td>
<td>-</td>
</tr>
</tbody>
</table>

We are interested at that if the network performance will be affected largely when there are more than one virtual machines transferring data concurrently.

We can see that the rate is almost half the single virtual machine. When two machines are running concurrently, one of them is 5.83 MB/s, and the send rate is 12.10 MB/s when the signal Xen-guest requests the network. The rate has a huge difference when the block size is 1 K or 1 M. This is because the cache performance could be used perfectly when the block size is large enough. We cannot get the accurate rate when the two KVM-guest machines are concurrently sending (receiving) data with 1 K block size. When we tried to run the test script 10,000 times, we found that one of the KVM virtual machine will be stuck.

From Figure 9 we found that when two KVM-guest machines are concurrently sending (receiving) data, the transfer rate is largely changing but the total rate is almost the same as that when signal virtual machine sends (receives) data. On the other hand, the two Xen-guest machines do not show this phenomenon. The rates in two Xen virtual machines are relatively stable. The Xen’s scheduling mechanism is implemented through the shared memory ring buffer.

Figure 9#  Transfer rate of two KVM-guest machines concurrent sending (receiving) data (see online version for colours)
5.4 System call

Figure 10 shows system call’s execution measurements between the Xen (KVM) VM and the native system. It could be found that the VM just add a little performance overhead.

KVM-guest has less overhead than Xen-guest about pipe, signal handler, openclose, and af_unix, but more for tcp. Both systems have the same overhead for the page_fault. The KVM relies on the Linux kernel to implement the system calls, and this makes it more effective than Xen when it comes to the system call. The tcp is about the IO virtualisation. The Xen-guest’s tcp performance is better than the KVM-guest because Xen uses para-virtualisation without the device emulation. This could make the VM gain better IO performance.

Figure 10  System call tests with Lmbench

5.5 Applications

The measurement result of Apache benchmark is shown in Table 8. We could reach the conclusion that there is a large performance overhead from the virtual machines whether it is Xen or KVM. Overall on ARM, the Xen (KVM) performs about 60% overhead than running directly on the hardware for Apache. This overhead is unsatisfactory for us. We think this is because the Cubietruck does not support the System MMU which could enhance the performance of the IO virtualisation. On the other hand, the web server should undertake a large of IO tasks. The Xen-guest could process the requests quicker than KVM-guest, it also has bigger transfer rate.

Table 8  Apache performance tests

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Xen-guest</th>
<th>KVM-guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time taken for tests</td>
<td>87.57 s</td>
<td>159.08 s</td>
<td>144.36 s</td>
</tr>
<tr>
<td>Requests per second</td>
<td>580.43</td>
<td>314.31</td>
<td>346.37</td>
</tr>
<tr>
<td>Transfer rate</td>
<td>261.31 kbs</td>
<td>141.56 kbs</td>
<td>155.93 kbs</td>
</tr>
</tbody>
</table>
On the other hand, the machines we used are single CPU, the apache is a classical multi process web server. There are lots of locks which are used to avoid resource contention. All of above may decrease the apache’s performance on the virtual machine.

Table 9 shows the result of the three tests. We find that the VM’s add about 16% to 17% of that running directly on the native system. The performance is better than the Apache above. The performance of Memcached is closely related to the performance of the system memory. We have known that the memory performance of Xen (KVM) could be 98% to 99% compared with the bare mental by the Geekbench2 tests above. For this reason the Memcached’s performance running on the virtual machine is satisfactory.

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Xen-guest</th>
<th>KVM-guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load data time</td>
<td>86.22 s</td>
<td>102.128 s</td>
<td>103.621 s</td>
</tr>
</tbody>
</table>

The Mysql benchmark measurement result is shown in Table 10. In the experiment, 140,000 read operations, 50,000 write operations and 20,000 other operations are performed. We find that the Xen virtual machines perform better than KVM virtual machines. By analysing current data, we conclude that the phenomenon is caused by the performance problems of KVM’s IO performance.

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Xen-guest</th>
<th>KVM-guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions (per sec.)</td>
<td>40.47</td>
<td>37.68</td>
<td>22.10</td>
</tr>
<tr>
<td>Readwrite requests (persec.)</td>
<td>768.88</td>
<td>715.99</td>
<td>419.81</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>247.113</td>
<td>265.365</td>
<td>452.589</td>
</tr>
</tbody>
</table>

6 Conclusions and future work

We have finished a number of directive system level tests to measure the performance of many base operations and three popular applications for Xen-based (KVM) virtual machine running Linux 3.13 on a Cortex-A7-based platform. The performances haven been compared to a native Linux 3.13 on the same platform. We can get the following conclusions.

First, the research about virtual technology on ARM is promising. The Xen and KVM virtual machines will add just 2% to 3% overhead on some basic parameters, such as CPU, memory, and disk rate. We can see that the Xen’s performance is a little better than KVM, but the gap is very little. The KVM-guest’s disk read performance benefits from the virtio infrastructure. When it comes to network throughput, the performance of KVM-guest in our environment is disappointing. We have tried to enable the host’s vhost module, but the performance has not been improved effectively. On the other hand, the Xen has got a good score because of its split driver-mode.

Second, whether the specific applications are suitable form running on the Xen (KVM) virtual machines is related with to the application’s features. The apache is based on the multi process and has high requirement of IO performance. For this reason, if the virtual machine’s IO performance problem cannot be solved, it cannot apply the apache web server. Because the Memcached has high performance requirements of memory,
so the Memcached has been suitable for running on the virtual machines. The MySQL running on Xen virtual machine gets a better performance than that on KVM virtual machine. Third, there are a lot of works need to be done before the ARM platform could be applied the virtualisation technology massively. For example, the IO performance of Xen and KVM is needed to be improved through hardware support such as System MMU. The Xen (KVM) also needs to optimise their software architecture.

For the future work, we will continue our study from following several aspects. First, we need to solve the KVM virtual machine’s network throughput problem. Secondly, we will conduct further analysis on several typical cloud applications. The principle behind the data is needed to be discovered.

Acknowledgements

The project is funded in part by a grant from the Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20130031120028) and Research Plan in Application Foundation and Advanced Technologies in Tianjin (Grant No. 14JCQNJC00700).

References


