Predictable Packet Latency in Xen-ARM*

Seehwan YOO†(a), Student Member, Kuenhwan KWAK(b), Jaeyun JO(c), Nonmembers, and Chuck YOO(d), Member

SUMMARY In this paper, we address latency issue in Xen-ARM virtual machines. Despite the advantages of virtualization in mobile systems, the current Xen-ARM is difficult to apply to mobile devices because it has unpredictable I/O latency. This paper analyzes the latency of incoming packet handling in Xen-ARM, and presents how virtualization affects the latency in detail. To make the latency predictable, firstly, we modify Xen-ARM scheduler so that the driver domain can be promptly scheduled by the hypervisor. Secondly, we introduce additional paravirtualization of guest OS that minimizes non-preemptible code path. With our enhancements, 99% of incoming packets are predictably handled within one millisecond at the destined guest OS, which is a feasible time bound for most soft real-time applications.

key words: virtual machine, I/O latency

1. Introduction

Virtualization has drawn attention in recent mobile systems because virtualization enhances reliability, enriches user experiences, and provides better software compatibility [2]–[5]. At first, virtualization enhances reliability of the software platform. Since the virtualization layer isolates accesses across different domains and monitors ‘sensitive’ information flows, a user domain can reliably perform its tasks even though another system component fails or is compromised. For example, recent study shows how virtualization secures a trusted domain from network attacks in mobile devices [6]. In addition to reliability, virtualization enables heterogeneous guest OSs to be incorporated within a physical hardware platform. Since the virtualization layer provides an independent logical hardware to each guest OS, it allows a user to run multiple guest OSs that have different user interfaces (UIs), libraries, applications side by side. Especially, multi-OS support is beneficial not only for users but also for developers since the software platform provides better software compatibility by allowing multiple guest OSs. Instead of redeveloping multiple UIs, libraries, applications over a target OS, they can be consolidated with much less effort on different guest OSs.

Xen-ARM is a version of Xen for ARM-based mobile devices [4], which presents small performance overhead [7]. For reliability, split drivers of Xen-ARM isolate faulty and error-prone device drivers from user domains by locating physical drivers to IDD (Isolated Driver Domain). Since a user domain is completely separated from IDD, a user domain can reliably perform its task even though IDD is compromised and fails.

Typical mobile virtual machines run a real-time guest OS and multiple general-purpose guest OSs, side by side. A real-time guest OS provides performance for real-time applications such as mobile communications or multimedia applications. In addition, general-purpose guest OSs provide rich abstractions for applications such as Web browsers and secure interfaces for banking applications.

To accommodate a real-time guest OS to Xen-ARM, I/O latency issue has to be addressed. Predictable I/O latency is particularly critical** because most real-time guest OSs demand timely handling for random I/O events. For example, a real-time guest OS should handle network interrupts within a predictable latency bound regardless of the CPU workload and other general-purpose guest OSs.

However, the I/O latency over Xen-ARM is very unpredictable because the I/O latency is affected by VCPU (Virtual CPU) scheduling*** within Xen-ARM. So, I/O latency in Xen-ARM can be largely presented although a guest OS schedules I/O task as soon as possible. At first, the Credit scheduler, the default inter-VM scheduler of the Xen-ARM hypervisor, is insufficient for supporting time-sensitive applications [8], [9]. Specifically, with the split driver model of Xen-ARM, packet handling can be considerably delayed because the split driver model requires additional scheduling of the driver domain. Secondly, the latency inside a guest OS can be negatively affected by paravirtualization. To timely handle incoming packets from network devices, a guest OS has to be carefully paravirtualized.

A goal of this paper is to make the packet handling latency predictable in Xen-ARM. So, we firstly present how I/O latency populates in the current Xen-ARM by analyzing an incoming packet’s handling path. Secondly, this paper proposes three enhancements to handle incoming packets

** In this paper, we use the term predictable, not for prediction purpose, but it stands for having stabilized and low deviation values.
*** In this paper, we use scheduling VCPUs, domains, guest OSs, virtual machines interchangeably when it is clear in the context.
within a predictable latency bound (one millisecond): 1) an improved scheduler for Xen-ARM with two additional priorities, 2) use of FIQ (Fast IRQ) interrupt with Xen-ARM and 3) enhanced idle paravirtualization of guest OS. Currently, we aim to support soft real-time applications because Linux is used as guest OS. Although Linux supports many commodity device drivers, its real-time performance has limitations.

This paper consists of seven sections. Section 2 provides related work on I/O latency in virtualization systems. Section 3 presents how packet reception latency populates in the current Xen-ARM VM. Section 4 proposes a new scheduler and paravirtualization to achieve under milliseconds network latency. Section 5 evaluates the proposed scheduler and paravirtualization through experimental results. Section 6 discusses further considerations such as multicore support. Finally, Sect. 7 provides some concluding remarks.

2. Related Work

There are several commercial embedded hypervisors that specifically target on mobile and real-time systems. L4 microvisor is a mobile hypervisor that presents low latency by extensive architectural optimizations to run with mobile phones [2]. L4 microvisor claims that their small code base have deterministic execution time, and guest OSs are fully preemptible, so WCET (Worst-Case Execution Time) can be time-bounded [10]. For real-time schedulability analysis, Jungwoo et al. presented real-time scheduling with L4-based virtualization [11]. VirtualLogix and VMWare Inc. have another commercial mobile hypervisors VLX and Mobile Virtualization Platform (MVP) [5], respectively. VLX presents prioritized interrupt handling mechanism for RT (Real-time) guest OS, and shared driver model for RT guest OS and non-RT guest OS [3]. It also presents comparable performance with native Linux systems [12]. Although they focus on performance over hypervisor, latency analysis for shared devices under multiple guest OSs are not presented, so far. Although Xen is originally designed for enterprise systems, Xen-ARM [7] presents minimized code size, and is modified so that it can be used with mobile devices. It supports Xen’s split-driver model for shared devices and two schedulers: Credit for default fair-share and SEDF for real-time guarantee.

Several studies on the Credit scheduler of Xen focused on real-time and time-sensitive applications. Recently, Sisu et al. presents RT-Xen [13] that supports hierarchical scheduling with Xen. The authors proposed server-based scheduling model for a soft real-time guest OS. However, they do not address I/O performance and latency issues. Lee min et al. [9] tried to support soft real-time applications with Xen. Considering the urgency, their scheduler takes the laxity into account. By calculating the average execution time, the hypervisor maintains the laxity of each VCPU, and selects the most urgent one. However, laxity is highly dependent upon the expectation of average execution time that is inaccurate and introduces run-time overhead. Hwanju et al. [8] tackled the weakness of the Credit scheduler for I/O and CPU mixed workload. Since the Credit scheduler cannot distinguish the urgency among domains that have BOOST priority, the authors proposed partial boost. In the partial boost, the hypervisor estimates whether a VCPU is I/O intensive or CPU intensive. If a VCPU is I/O intensive, then it should obtain BOOST priority so that the guest OS can have good responsiveness; otherwise, the BOOST priority is negated. Despite its high accuracy, partial boost is also dependent upon inaccurate expectation of future execution. In mobile virtualization, real-time domain is usually statically assigned, so we can simply prioritize the real-time domain, instead of guessing the most urgent domain. Ongaro et al. [14] presented how the Credit scheduler affects the I/O performance in Xen virtual machines. The result reveals that the Credit scheduler shows bad performance for I/O and CPU mixed workload. The authors improved pending-event handling mechanism in order to achieve fair I/O performance. In addition, they introduced several optimization techniques such as run queue sorting, removing unnecessary tickling, etc. However, the current Credit scheduler has limitations for CPU-I/O mixed workloads. Our scheduler specifically focuses on predictable I/O latency under CPU-I/O mixed workloads.

For network performance, Menon et al. [15] presented optimizing methods for networking stack within Xen-based virtual machines. The authors investigated on several aspects in networking stack that virtualization makes differences from native systems such as IP checksum and segment offloading. Kaushik et al. [16] presented an approach for hardware-assisted virtualization using multi-queue NIC. Both focused on efficient network bandwidth utilization rather than latency. Kesavan et al. [17] presented that inter-VM scheduling of Xen affects adaptive resource management of a guest OS, which results in degraded network performance of latency-sensitive TCP variants. The authors proposed DVT (Differential Virtual Time) for proportional resource scheduling with gradual service rate change. Although those studies focus on network performance in virtualization, they mostly focus on achieved bandwidth, rather than latency.

High responsiveness and small I/O latency with high predictability are, in general, major goals of OS. For Linux, RT-PREEMP patch [18] has been proposed. The patch tries to make the Linux kernel fully preemptible, even within interrupt service routine. Open RTLinux and RTCore [19] modified Linux by introducing real-time subsystem inside the kernel. RTAI/xenomai [20] is another kernel extension to reuse existing real-time applications over Linux. They focused on real-time issues in native systems, not in virtual machines. Those studies assume only one OS. On the other hand, in virtualization, there are multiple guest OSs, and hypervisor performs inter-VM scheduling, which introduces

---

1FIQ is ARM-specific fast interrupt handling mechanism.
2We assume one millisecond as a feasible latency bound for soft real-time applications.
tens of milliseconds I/O latency. In addition, hypervisor is not aware of the status of a guest OS, so a guest OS can be scheduled out while handling interrupts.

3. Packet Latency in Xen-ARM

In this section, we will analyze the packet latency in Xen-ARM. Incoming packet’s handling path and corresponding latencies are presented in Fig. 1. To explain the latencies in Fig. 1, first, physical interrupt latency is time until Xen-ARM handles a physical interrupt. In Xen-ARM virtualization, all physical interrupts are handled by the Xen-ARM hypervisor. Xen-ARM directly accesses interrupt controller, and it does not expose any physical interrupt to guest OSs. Instead, Xen-ARM delivers device event to a guest OS in an asynchronous manner, called virtual interrupt. Virtual interrupt is implemented using event channel in Xen-ARM. Similar to physical interrupt, virtual interrupt triggers scheduling of guest OS. After Xen-ARM handles physical interrupt, Xen-ARM generates the corresponding virtual interrupt, and delivers the virtual interrupt to IDD (independent driver domain, or driver domain) so that IDD can handle the physical device event in the physical device driver routine.

Second, vcpu dispatch latency is the duration from the time when Xen-ARM notifies the driver domain (dom0) of the interrupt to the time when dom0 is dispatched by the Xen-ARM scheduler. When dom0 is dispatched, it begins I/O task from interrupt handler within the physical device driver. Note that dom0 is scheduled along with other domains.

Third, I/O completion latency is the time until dom0 finishes all I/O tasks (such as interrupt handling and corresponding bottom halves and softirqs within backend driver as well as physical device driver). I/O task within dom0 finishes at a netback routine (netbe_rx_action) in the backend driver.

After finishing the I/O task at dom0, a packet is ready for delivery to the destined guest OS (domU). Then, dom0 sends virtual interrupt to domU. Here, another vcpu dispatch latency is introduced by Xen-ARM. This latency is the time until the Xen-ARM scheduler dispatches domU. After domU is scheduled, another I/O completion latency is introduced until domU completes the I/O task.

As shown in Fig. 1, both vcpu dispatch latency and I/O completion latency within the split driver model, are presented two times. Since those latencies (vcpu dispatch latency and I/O completion latency) are two major reasons for unpredictable packet latency in Xen-ARM, we focus on these two latencies. In addition, we assume that a user guest OS (domU) schedules I/O tasks as soon as possible, and I/O completion latency within domU is not affected by task scheduling of the guest OS.

3.1 VCPU Dispatch Latency

This section analyzes vcpu dispatch latency, and it is mainly introduced by the Credit scheduler. So, we briefly explain how the scheduler works. The scheduler primarily focuses on fair scheduling among VCPUs. Each VCPU periodically gets the equal amount of credit from Xen-ARM, and the credit is debited as much as the VCPU executes. To equalize the utilization of among VCPUs, the scheduler selects VCPUs among ones that have remaining credit is larger than zero, of which priority is UNDER. When the VCPU consumes all its credit, and the credit value is less than zero, then the VCPU has OVER priority. In addition to UNDER and OVER priority, the scheduler has BOOST priority in order to mitigate the long latency of round robin algorithm. When a VCPU is boosted, then it can preempt the currently running VCPU, and resumes execution immediately.

Despite the BOOST priority, vcpu dispatch latency in Xen-ARM can be long because of non-boosted vcpu and multi-boost. Because the scheduler primarily focuses on fairness rather than guaranteed responsiveness, the scheduler conservatively applies BOOST priority in order not to break the fairness. So, if a VCPU is on the run queue, then the VCPU cannot obtain BOOST priority although it has pending I/O event. This leads to long vcpu dispatch latency when the VCPU is on the run queue due to its CPU-intensive workload. This is called non-boosted vcpu. Concisely, non-boosted vcpu leads to long vcpu dispatch latency, and it happens more often with CPU-intensive workload.

In addition to non-boosted vcpu, multi-boost occurs when multiple VCPUs have BOOST priority, and thus BOOST priority is negated by another boosted VCPU. Since the scheduler doesn’t distinguish urgency among boosted VCPUs, all boosted VCPUs compete for the CPU time against each other. Because dom0 has no CPU-intensive workload, non-boosted vcpu is mostly observed at domU. So, non-boosted vcpu affects domU’s vcpu dispatch latency. On the other hand, multi-boost affects vcpu dispatch latency to both dom0 and domU.

3.2 I/O Completion Latency

In Fig. 1, I/O completion latency is presented in both dom0 and domU. I/O completion latency is affected by inaccurate unboosting of VCPU. Once a VCPU is boosted, the VCPU is unboosted by next timer interrupt in order to preserve fairness. Namely, when a timer tick is triggered, the priority of the VCPU, that has BOOST priority, is changed to UNDER or OVER regardless of I/O completion. Because Xen-ARM doesn’t know when a guest OS completes I/O
task, the scheduler unboosts VCPU inaccurately.

Once a VCPU is unboosted inaccurately, then another VCPU that has BOOST or UNDER priority can preempt the running VCPU while the interrupt processing is not completed. This is called VCPU schedule out, and thereby long I/O completion latency is presented until Xen-ARM reschedules the VCPU and the corresponding guest OS completes the I/O task.

VCPU schedule out has two implications: 1) this leads to long I/O completion latency for the current packet and 2) if a guest OS is scheduled out while it disabled virtual interrupt, the schedule out increases the vcpu dispatch latency for subsequent interrupts. The first implication is quite natural because the VCPU needs to be rescheduled. For the second implication, dom0 can be scheduled out while its virtual interrupts are disabled. Then, subsequent interrupt handling at dom0 is seriously delayed because dom0 disables virtual interrupts for a long time. Namely, the vcpu dispatch latency of subsequent interrupts to dom0 is delayed until Xen-ARM re-schedules the VCPU, and dom0 re-enables interrupts. We call this VCPU schedule out with interrupt disabled.

This is a serious drawback for latency-sensitive applications because VCPU schedule out with interrupt disabled makes dom0, that performs all physical device operations on behalf of the user domains, stay long within non-preemptible code path. When dom0 is scheduled out with interrupt disabled, dom0 could miss subsequent interrupt because Xen-ARM can pend only one event for the same virtual interrupt. On the contrary, native Linux manages interrupt disabled code path as small as possible because Linux becomes non-preemptible when it disables interrupt.

3.3 Measured Packet Latency in the Current Xen-ARM

This section presents the packet latency in Xen-ARM by measuring the packet latency with ping tests. The measurement environment includes three guest OSs run over Xen-ARM: dom0 serves as the driver domain, rt_dom1 is the ping recipient domain that has soft real-time requirements in handling network packets, and dom2 has 100% CPU workloads to ensure work-conserving execution. An external server sends icmp request packets to rt_dom1 with random intervals (10–40 ms). Xen-ARM uses the Credit scheduler, and timer tick is set to 10 ms. We obtain the distribution graphs from one millions of ping samples. Hardware platform is equipped with cortex-A9 processor (running at 1 Gzh) and 100 Mbps network device (SMSC9415) attached via USB.

Since rt_dom1 is the major source of latency, we will look into the latency with rt_dom1, at first hand. Then, we will analyze latency with dom0 for further investigation.

3.3.1 Latency with rt_dom1

First, we measure packet latency with rt_dom1, and look into the effect by non-boosted vcpu. To present packet latency with rt_dom1, we measure the latency in the interval from I/O task completion at netback in dom0 to I/O task completion at icmp reply in rt_dom1. To test non-boosted vcpu, we give different CPU-intensive workload (20%, 40%, 80% and 100%) to rt_dom1. In addition, we assume that I/O completion latency within rt_dom1 is small enough to ignore because it is much smaller than the other latency. Thus, we intentionally omitted it from all our results. This small latency is partly due to icmp handling in Linux kernel, in which, icmp packets are handled within softirq context. So, little overhead is involved.

Figure 2 shows vcpu dispatch latency to rt_dom1. In the figure, when rt_dom1 has 20% workload, more than 88% of packets are handled within 1 ms. On the other hand, when rt_dom1 has 100% CPU load, only 46% of the packets are handled within 1 ms. The figure shows clear trend that less packets are handled within 1 ms time-bound as CPU load increases from 20% to 100%. This is mainly due to non-boosted vcpu, which is affected by CPU workload of itself.

Additionally note that, in the figure, most packets are handled within 30 ms that is the scheduling period of Xen-ARM. When a packet handling is delayed by non-boosted vcpu, the VCPU is dispatched at the next scheduling period. Consequently, 30 ms latency is presented.

3.3.2 Latency with dom0

Second, to analyze packet latency with dom0, we measure dom0’s vcpu dispatch latency and I/O completion latency at two intervals: 1) from the interrupt handler within Xen-ARM to the interrupt handler of dom0’s physical device driver (Xen-usb_irq@dom0) and 2) from the interrupt handler within Xen-ARM to the netback routine that is the end of I/O task (Xen-netback@dom0). Latency in 20%, 80% and 100% CPU load of rt_dom1 cases are presented in Fig. 3. In the graphs, solid line shows the latency distribution for Xen-usb_irq@dom0, and dotted line shows the latency distribution for Xen-netback@dom0. In the figure, dotted line is dom0’s vcpu dispatch latency distribution, and the gap between the dotted line and solid line is I/O completion latency.

In Fig. 3 (a), 98% of packets are handled within 1 ms,

Fig. 2
vcpu dispatch latency under various CPU load.
but 2% of packets are handled with delay due to dom0’s vcpu dispatch latency. This shows that Xen-ARM does not always schedule dom0 immediately. This is because multi-boost that makes dom0 contend with rt_dom1. When multi-boost happens, both dom0 and rt_dom1 have the same BOOST priority, and Xen-ARM selects either dom0 or rt_dom1.

In Fig. 3 (b), we can see a large gap between solid line and dotted line that presents long I/O completion latency within dom0. In Fig. 3 (b), 95.5% of packets are handled within 1 ms, and 4.5% of packets are handled with delay. Among 4.5%, 2% of them are delayed due to vcpu dispatch latency, and 2.5% of them are due to I/O completion latency.

In Fig. 3 (b), the gap between two lines (in X-axis) is as large as 40 ms, which means I/O completion latency is as large as 40 ms. This is partially due to inaccurate un-boosting. Namely, dom0 is occasionally scheduled out before completing its packet handling, and long latency is presented. Additionally observe that 80% CPU load case in Fig. 3 (b) shows longer I/O completion latency than 20% case in Fig. 3 (a). If rt_dom1 has higher CPU load, then rt_dom1 will execute longer to fully utilize its quantum, which leads to large dom0’s I/O completion latency.

When rt_dom1 has 100% CPU workload as shown in the graph 3 (c), dom0’s vcpu dispatch latency is minimized. The reason is that rt_dom1 is never boosted if the VCPU has 100% CPU workload (by non-boosted vcpu), and dom0’s VCPU is the only boosted VCPU. This proves that vcpu dispatch latency in Fig. 3 (a) and 3 (b) are due to multi-boost that is scheduling contention between dom0 and rt_dom1. Remember that dom0 has no CPU workload, so dom0 does not have long vcpu dispatch latency by non-boosted vcpu.

3.3.3 Overall Latency with dom0 and rt_dom1

To present overall latency characteristics, we measure the latency in the following two intervals: from interrupt handler at Xen to netback driver routine in dom0, (Xen-netback@dom0), and from interrupt handler at Xen to icmp handler in rt_dom1 (Xen-icmp@rt_dom1). In addition, rt_dom1’s CPU load is given 20%, 40%, 80%, and 100%.

Graphs in Fig. 4 present cumulative distribution of latency with dom0 and rt_dom1. Xen-netback@dom0 presents latency with dom0, and it is presented with dotted line. Xen-icmp@rt_dom1 presents overall packet latency perceived at rt_dom1, and it is presented with solid line. The gap between two lines presents latency with rt_dom1. Graph shows the percentile of handled packets within a given latency-bound. For example, in Fig. 4 (a), rt_dom1 can handle 86% of packets within 1 millisecond. In the graphs, dotted line is higher than solid line because network packets are processed at dom0, at first hand, and then, packets are handled at rt_dom1.

In Fig. 4, overall latency is mainly affected by latency with rt_dom1. It is the major source of unpredictable and large packet latency. In all graphs, more than 96% of packets are handled within 1 ms at dom0. However, packet latency drastically increases when rt_dom1 is involved. In Fig. 4 (a), 88% of packets are handled within 1 ms when rt_dom1 has 20 % CPU load. In addition, only 46% of packets are handled within 1 ms when rt_dom1 has 100 % CPU load. The reason is non-boosted vcpu as shown in Sect. 3.3.1. Consequently, packets handled within 1 ms decreases from 86% to 73%, 61%, and 46% according to CPU load increases from 20%, to 40%, 80%, and 100%, respectively.

In Fig. 4, latency with dom0 presents different characteristics from latency with rt_dom1. Latency with dom0 is usually small. For example, in Fig. 4 (b), 96% of packets are handled within 1 ms. However, 2% of packets are handled with more than 30 ms delay. This implies that the scheduler works well in average cases, but long latency (scheduling period) is presented once dom0’s VCPU is scheduled out before dom0 completes the interrupt processing. Therefore,
latency with dom0 has significant impact on worst-case latency.

To present VCPU schedule out with interrupt disabled and schedule out of dom0, we count the schedule out of dom0 during another one millions of ping test, and categorize the schedule out count with the VCPU’s state. The rest of experimental configurations are the same with 20% CPU load case in the previous experiment. Table 1 shows the schedule out count for dom0 and rt_domain1 during the test.

In the table, VCPU schedule out with interrupt disabled is presented in the first row. The schedule out is observed 65 times. Rt_domain1 also experiences 134,161 number of VCPU schedule out with interrupt disabled. Dom0’s VCPU schedule out with interrupt disabled is more serious than rt_domain1 because dom0 handles all physical driver operations, that have to be processed in a timely manner, on behalf of the all other user domains. In the third row, schedule-out of dom0’s VCPU is observed 7,255 times. Most of them occur when the VCPU has UNDER and OVER priority, i.e. dom0’s schedule-out during I/O task execution. This is due to the de-prioritization of boosted VCPU by next tick timer, which results in long I/O completion latency of dom0.

To clarify, we provide explanation for the rest of the table. In the third row, rt_domain1 is scheduled out when it doesn’t have BOOST priority. In the table, 445,244 schedule out of rt_domain1 is observed when the VCPU is unblocked. The second row of the table shows that dom0 and rt_domain1 become idle, and scheduled out due to work-conserving policy of the Credit scheduler. When entering idle, the VCPU disables interrupt and blocks VCPU so that another VCPU can utilize it. So, there is no schedule out when interrupt is enabled (in the fourth row).

In summary, vcpu dispatch latency to rt_domain1 is mainly affected by non-boosted vcpu, but vcpu dispatch latency to dom0 is affected by multi-boost. I/O completion latency within dom0 is due to schedule out, and long I/O completion latency within dom0 is presented along with VCPU schedule out with interrupt disabled.

### 4. Enhancements for Predictable Latency

To handle packets with predictable latency bound, this paper proposes three enhancements: 1) improved scheduling in Xen-ARM, 2) use of FIQ interrupt, and 3) enhanced idle paravirtualization. First, we use additional priorities, DRIVER BOOST and RT BOOST, to the Xen-ARM scheduler. Two additional priorities are applied to the driver domain (dom0) and real-time domain (rt_domain1) whenever network interrupt is notified. To provide predictable latency-bound, dom0 is scheduled at the highest priority, and rt_domain1 is always scheduled when dom0 is idle.

#### Table 1: Schedule out count during one million ping test.

<table>
<thead>
<tr>
<th>Schedule out count</th>
<th>Dom0</th>
<th>rt_domain1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntDisabled &amp; Unblocked</td>
<td>2,936,727</td>
<td>876,784</td>
</tr>
<tr>
<td>IntDisabled &amp; Blocked</td>
<td>7,255</td>
<td>445,244</td>
</tr>
<tr>
<td>IntEnabled &amp; Unblocked</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
priority (DRIVER_BOOST), and rt_dom1 runs with the secondary priority (RT_BOOST) that is higher than normal BOOST priority.

DRIVER_BOOST largely enhances vcpu dispatch latency to dom0 by resolving multi-boost. Whenever Xen-ARM catches physical interrupts, dom0 always takes the highest priority although another VCPU has BOOST priority. Thus, dom0 can always handle interrupts in a timely manner. To minimize I/O completion latency within dom0, dom0 always runs with DRIVER_BOOST priority, and it is never unboosted.

RT_BOOST also largely enhances vcpu dispatch latency to rt_dom1 by resolving non-boosted vcpu. When rt_dom1 is notified of network interrupt, it runs with RT_BOOST regardless of the current CPU workload. In addition, rt_dom1 is distinguished from the other user domains because RT_BOOST is a higher priority than BOOST.

To resolve inaccurate VCPU unboosting, we paravirtualize guest OS so that the hypervisor can be aware of the end of I/O task completion. In rt_dom1, when rt_dom1 has pending interrupts, rt_dom1 sets fiq context in the vcpu_info data structure. The flag is unset when rt_dom1 completes I/O task so that RT_BOOST is remained until the designated guest OS completes I/O task. In our implementation, RT_BOOST is remained until ping request is completed at icmp_reply(). In the routine, additional hypercall is invoked so that Xen-ARM can deprioritize the VCPU and triggers scheduling. Differently from dom0, Xen-ARM can change priority of rt_dom1 from RT_BOOST to UNDER or OVER after unboosting because rt_dom1 has its own CPU load, and RT_BOOST is applied only when the VCPU has pending network interrupt.

Concisely, the enhancements resolve scheduling latency introduced by the Credit scheduler. Because dom0 and rt_dom1 have different priorities, multi-boost is resolved for dom0 and rt_dom1. In addition, rt_dom1 obtains RT_BOOST priority regardless of its own CPU workload, non-boosted vcpu is resolved. Therefore, the vcpu dispatch latency can be managed at the scheduler level. In addition, by paravirtualizing guest OS, Xen-ARM can be aware of I/O task completion, and minimizes I/O completion latency by avoiding inaccurate VCPU unboost.

Second, we enhance interrupt handling for the network device using FIQ interrupts. FIQ interrupt is ARM processor-specific interrupt handling mechanism. Cortex-A9, a processor that Xen-ARM runs on, also has FIQ mode, which can preempt normal IRQ mode in order not to delay FIQ interrupt handling. However, current Xen-ARM and paravirtualized Linux (XenoLinux) does not support FIQ mode. So, all the devices interrupt handling is handled at the same priority.

Instead of using IRQ interrupt, we modified Xen-ARM to use FIQ interrupts for the network device. To use FIQ interrupt, Xen-ARM firstly needs to setup hardware context for FIQ mode, and then Xen-ARM configures the interrupt controller so that network device can generate FIQ interrupt instead of IRQ interrupt. We use the hardware feature of TrustZone’s secure interrupt, which enables a user to select generating FIQ interrupt instead of IRQ interrupts. Because FIQ interrupts have higher priority, VCPU can preempt IRQ interrupt handling, and IRQ cannot interfere with FIQ interrupt handling.

In addition to support FIQ at hardware, we modify Xen-ARM and guest OS to support virtual FIQ interrupts. When FIQ interrupt is generated, event channel’s pending fiq bit is set, and vPSR (Program Status Register of VCPU) also sets VPSR_F_BIT, accordingly. By applying this, a guest OS can enable/disable virtual FIQ interrupts.

Third, we enhance idle paravirtualization within a guest OS. With the original idle implementation, Xen-ARM has long packet latency, which is partly due to VCPU schedule-out with interrupt disabled. In the original idle implementation, when a guest OS enters idle, it firstly disables virtual interrupts, and blocks itself so that the physical CPU can be utilized by another guest OS (so-called work-conserving scheduler). By modifying idle function of dom0 not to disable interrupts when entering idle, dom0 can immediately handle interrupt.

In addition, we modify idle of rt_dom1 1) not to block itself (as non-work-conserving mode) and 2) not to disable interrupts in order to reduce the non-preemptible code path in Xen-ARM. Whenever rt_dom1 enters idle, the scheduler is invoked, which is non-preemptible. By applying the idle paravirtualization, non-preemptible code can be preemptively changed because the guest OS executes preemptible code path instead of invoking hypercall that involves the execution of non-preemptible code.

5. Evaluation

In this section, we evaluate our enhancements. At first, we present how three enhancements minimize packet latency under various CPU workloads. Secondly, we present latency distribution compared to native Linux in order to present how much Xen-ARM introduces virtualization overhead with respect to packet latency. Thirdly, we present CPU fairness and the feasibility for service differentiation for a real-time guest OS in a multi-guest OS environment.

5.1 Enhanced Latency with Predictable Latency-Bound

We firstly evaluate our two enhancements: our new scheduler with DRIVER_BOOST and RT_BOOST priority and FIQ support. Then, we will evaluate our enhancement regarding idle paravirtualization. To compare our enhancements, we perform the identical experiments presented in Sect.3.3.

Firstly, for the improved scheduling with additional priorities and FIQ support, we present the reduced overall latency. Table 2 presents the packet interrupts with and without our two enhancements, additional priorities and use of

†TrustZone is ARM-specific feature that supports enhanced security mode
Table 2  % of handled packets within 1 ms.

<table>
<thead>
<tr>
<th>rt_dom1's CPU load</th>
<th>20%</th>
<th>40%</th>
<th>80%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig. dom0</td>
<td>97</td>
<td>96</td>
<td>96</td>
<td>99</td>
</tr>
<tr>
<td>Credit/rt_dom1</td>
<td>86</td>
<td>73</td>
<td>61</td>
<td>46</td>
</tr>
<tr>
<td>New/rt_dom1</td>
<td>99</td>
<td>98</td>
<td>97</td>
<td>99</td>
</tr>
</tbody>
</table>

The table presents the percent of handled interrupts within 1 ms time-bound. To compare with our previous results in Sect. 3.3.2, rt_dom1's CPU load is given 20%, 40%, 80% and 100%. Note that dom2 has 100% CPU load. In the table, the first and the second row present the results with the original Credit scheduler; The third and the fourth row present the results when our new scheduler is used.

In Table 2, the original Credit scheduler presents different number of interrupts at dom0 and rt_dom1. That is because rt_dom1's vcpu dispatch is delayed by non-boosted VCPU. In the third column of the table, when 80% CPU load is provided to rt_dom1, 96% of packets are arrived at dom0, but only 61% of packets are arrived at rt_dom1. Namely, 35% of packets are delayed by the rt_dom1's vcpu dispatch latency.

With our new scheduler (the third and the fourth row in Table 2), handled interrupts at dom0 and rt_dom1 are the same because it minimizes rt_dom1's vcpu dispatch latency. Due to the new priorities, RT_BOOST and DRIVER_BOOST, rt_dom1 is always scheduled after dom0's I/O completion. Namely, RT_BOOST priority resolves non-boosted vcpu, and rt_dom1's vcpu dispatch latency can be deterministically guaranteed at the scheduler level. From our experiments, rt_dom1's vcpu dispatch latency is 12µs in average, and 98% are handled within 200µs.

In the table, with our new scheduler, 99%, 98%, 97% and 99% of packets are handled within 1 ms with 20%, 40%, 80% and 100% of CPU loads, respectively. It means that rt_dom1 can handle interrupts within 1 ms regardless of its own CPU workload. The result implies that I/O latency could be predictably managed for real-time virtual machine. Yet, we still see 2~3% rooms for further optimization in latency with dom0.

Secondly, our new scheduler reduces latency with dom0 as well as rt_dom1. To present enhanced latency, Fig. 5 comparatively shows the latency with dom0 with and without our enhancements. In Fig. 5, enhanced dom0's vcpu dispatch latency is presented with dotted line (Xen-usb_irq - new), and that in the original scheduler is presented with solid line (Xen-usb_irq - orig). For brevity, we present only the case that rt_dom1 has 80% CPU workload, which presents the worst latency with dom0 in previous Fig. 3 (b).

In Fig. 5 (a), our new scheduler handles 100% interrupts without delay. That is because our DRIVER_BOOST removes the scheduling contention (between dom0's VCPU and rt_dom1's VCPU), and minimizes dom0 vcpu dispatch latency by resolving multi-boost. On the other hand, 98% interrupts are handled within 1 ms when the original Credit scheduler is used.

Figure 5 (b) shows that our enhancements mitigate dom0 I/O completion latency (from Xen-usb_irq to Xen-netback), compared with the original scheduler. In the figure, our new scheduler handles 97% interrupts within 1 ms, and the original scheduler handles less than 96% interrupts within 1 ms. Thus, 1.5% more interrupts are handled within 1 ms through our new scheduler. It is mainly because DRIVER_BOOST fixes inaccurate unboosting. Since dom0 is not preempted by any other VCPU during I/O task execution, no schedule out is observed due to inaccurate unboosting.

In Fig. 5 (b), we still see that 3% of ping requests are delayed by dom0, and handled after 1 ms. The latency is because of usb watchdog timeout for handling lost interrupt. We believe that this is partly due to non-preemptible execution at dom0 and Xen-ARM. Namely, dom0 couldn't receive virtual interrupt for a long time, and eventually missed some interrupts.

Thirdly, our idle enhancement further optimizes latency by making Xen and dom0 more preemptive. Figure 6 (a) shows enhanced packet latency of rt_dom1 under 80% CPU load. In the figure, more than 99.5% of packets are handled within 1 ms, which is about 97% without idle enhancement. The result supports that the enhancement makes dom0 more preemptive, and presents reduced latency. Remember that our idle paravirtualization makes the scheduler non-work-conserving, and works as if it has 100% CPU workload, as shown in Fig. 6 (b).
5.2 Comparison with Native System

In this section, we present the latency compared with native system in order to present Xen-ARM’s virtualization overhead. For comparison with native system, we compare response time by measuring RTT (round trip time) at external host. To evaluate the virtualization overhead, we compare baseline performances in all cases, and all guest OS have no CPU workload. The observed minimum, average and maximum latency values are shown in Table 3. Note that although we focused on the packet receiving path, ping performs both receive and transmit operations, and RTT includes all the latencies, and presents overall performance.

The table comparatively shows latency (unit is in microsecond) for ping responses under native, the original and new schedulers. With the original Credit scheduler, the average response time to \( r_t \text{dom1} \) is 912 \( \mu \)s, which is about 0.4 millisecond larger than native system, and standard deviation is about three times larger than that of native system. Concisely, with the original scheduler, latency to \( r_t \text{dom1} \) is long and very unpredictable.

5.3 CPU Fairness and Service Differentiation

For CPU fairness among domains, we perform another experiment that shows our scheduler’s I/O impact to CPU fairness. To present CPU fairness, at first, Xen-ARM runs two user domains \( r_t \text{dom1} \) and \( \text{dom2} \), where \( \text{dom2} \) consumes 100% of its CPU time, and \( r_t \text{dom1} \) varies its CPU load. Then, we measure the index utilization that provides baseline performance without any I/O. After measuring the index utilization, we measure CPU utilization under various I/O workloads. For I/O workload, another external server sends ping requests to \( r_t \text{dom1} \). In each case, CPU utilization and Jain’s fairness index [21] are presented in Table 4. Jain’s fairness index (JFI) indicates the degree of fairness by measuring how resource is fairly utilized among N users. The index value is given in a real number between 0 and 1, and the higher number means the better fairness. JFI is generally applicable to various resources such as CPU utilization and network bandwidth. For comparison, we present CPU utilization and the fairness index of the original scheduler as well. We apply two enhancements (additional priorities and FIQ support) except for idle modification since non-work-conserving provides absolute utilization regardless of CPU workload of the other guest OSs.

On the other hand, our enhancements largely reduce maximum latency from 107 ms to 2 ms although our new scheduler introduces slight latency overhead: about 44 \( \mu \)s for \( \text{dom0} \) (44.02 = 576.54 – 532.52) and about 200 \( \mu \)s for \( r_t \text{dom1} \) in average (204.46 = 736.06 – 532.52). In addition, our enhancements make the latency more predictable by reducing standard deviation value from 1792.34 to 45.95.

Table 4

<table>
<thead>
<tr>
<th>Utilization</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0.148</td>
<td>0.261</td>
<td>0.358</td>
</tr>
<tr>
<td>( r_t \text{dom1} )</td>
<td>0.211</td>
<td>0.286</td>
<td>0.347</td>
</tr>
<tr>
<td>( \text{dom2} )</td>
<td>0.425</td>
<td>0.446</td>
<td>0.404</td>
</tr>
<tr>
<td>With our enhancements</td>
<td>0.209</td>
<td>0.254</td>
<td>0.288</td>
</tr>
<tr>
<td>( \text{dom0} )</td>
<td>0.369</td>
<td>0.536</td>
<td>0.511</td>
</tr>
</tbody>
</table>

Jain’s Fairness Index [21]

Original | 0.814 | 0.926 | 0.959
New scheduler | 0.888 | 0.981 | 1.0

On the other hand, our enhancements largely reduce maximum latency from 107 ms to 2 ms although our new scheduler introduces slight latency overhead: about 44 \( \mu \)s for \( \text{dom0} \) (44.02 = 576.54 – 532.52) and about 200 \( \mu \)s for \( r_t \text{dom1} \) in average (204.46 = 736.06 – 532.52). In addition, our enhancements make the latency more predictable by reducing standard deviation value from 1792.34 to 45.95.
achieves fairness in CPU utilization, it minimizes the impact to execution time regardless of the achieved latency results.

Finally, to assess the feasibility for service differentiation under a multi-guest OS environment, we run additional ping recipient domains (dom2 and dom3). Since we assume that rt_dom1 is the only real-time guest domain, rt_dom1 should handle interrupts within a predictable time bound regardless of the CPU load and the other domains. All three domains have 100% CPU load, and are ping receivers. Three external ping senders, S1, S2 and S3 send out ping requests to rt_dom1, dom2 and dom3, respectively.

Figure 7 shows ping latency (RTT) observed by external host with and without our enhancements. In Fig. 7 (a), although all three domains present almost identical RTT distribution with the original scheduler, the latency is badly presented with long response time. For 1 ms time-bound, all three domains responses only 40% requests. Figure 7 (b) shows RTT distribution with our new scheduler. In the figure, rt_dom1 responds to ping requests within 1 ms regardless of dom2 and dom3. So, it provides service differentiation in terms of packet latency. On the other hand, dom2 and dom3 present long responses. In the graph, dom2 and dom3 have almost the same latency distribution, which shows that our scheduler preserves fairness for I/O requests among the other non-real-time domains.

5.4 Applicability to the Other Devices

In this paper, we mainly focus on Ethernet device and packet latency. To test network latency, we choose ping as the target application because it is the most suitable benchmark for latency measurement. To confirm the effectiveness of our proposed enhancement schemes, MTD (Memory Technology Device) and WLAN (Wireless LAN) have been tested with more practical workload. Note that our enhancements (scheduling priorities i.e. RT.BOOST and DRIVER.BOOST, FIQ support, and idle paravirtualization) are generally applicable to different devices such as MTD and WLAN.

For MTD device, file-based database, Sqlite [22] is used. To test latency, INSERT queries are performed for 10,000 times, and completion times are measured. With the original Credit scheduler, 50% queries are handled within 20 ms, and the latency varies up to 120 ms for 90% percentile. With our new scheduler, more than 90% queries are handled within 20 ms.

In addition, ping has been tested with WLAN. With the original Credit scheduler, 45% packets are handled within 2 ms, but with our new scheduler, more than 90% packets are handled within 3 ms. The results show that our enhancements can efficiently reduce I/O latency not only for network devices, but also for other devices.

6. Discussion

Further consideration on I/O latency that has not been addressed in this paper is multi-core impact. Since our approach is not limited to single-core platforms, we believe that it is also helpful to network latency in multicore platforms. To have more advantages over multicore processor, the hypervisor has to schedule VCPUs in a way that pre-emptible VCPUs are maximized so that physical interrupt are always handled by those VCPUs.

7. Conclusion

In this paper, we present the packet latency in Xen-ARM. Due to the limitations of the current Credit scheduler and interrupt paravirtualization of guest OS, a user domain cannot handle network packets within a predictable time bound with Xen-ARM. To provide a predictable latency bound, this paper proposes an improved scheduler and additional enhancements by paravirtualizing a guest OS. Our DRIVER.BOOST and RT.BOOST priorities enhance vcpu dispatch latency and I/O completion latency in the driver domain as well as user domain. In addition, our FIQ-based network interrupt handling and idle paravirtualization minimizes packet latency. Through extensive ping tests, our new scheduler shows predictable response by presenting more than 99% network packets are handled at user domain within one millisecond.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0029848).
References


