A Step to Support Real-time in Virtual Machine

Seehwan Yoo, Miri Park, Chuck Yoo

College of Information and Communication
Korea University
Seoul, South Korea
Email: {shyoo, mrpark, hxy}@os.korea.ac.kr

***Abstract*—Real-time is one of the unique requirements in embedded systems. In this paper, we perform a feasibility study on how to support real-time in an embedded virtual machine system. Firstly, we argue that the I/O model of the current virtual machine monitor like Xen is not suitable to support real-time applications because it lacks in predictability and it does not guarantee a deterministic I/O processing. We provide an alternative I/O model for virtualized embedded systems. Devices are categorized into four groups: dedicated, active, running, dynamic. Dedicated devices make a virtual machine simple because they do not need to be virtualized for isolation. However, dedication does not mean the performance isolation. Our experimental results with dedicated device show that traditional dedication cannot guarantee the timely responsiveness in heavy interrupt cases. Specifically, responsiveness of real-time OS degrades as interrupt load increases. Therefore, a proper interrupt control mechanism is required at virtual machine monitor level in order to support timely responsiveness. In addition, our result supports that 1) short and prioritized interrupt processing helps responsiveness in a virtual machine system; 2) smaller time quantum results in better responsiveness also.**

I. Introduction

Embedded systems have been a target of virtualization in recent years. So-called digital conversions make customers need a variety of applications. However, real-time OS (RTOS) is not enough to support these diverse applications for it usually operates on limited environments. Thus general purpose OS (GPOS) is required to be used together with real time OS to support these growing needs. Virtualization is an enabling technology to support multiple operating systems-i.e. GPOS and RTOS- over one physical platform.

Recent trends make use of virtualization technology for the security purpose [1], [2]. As many digital devices become necessaries in daily life, security issues of their embedded software have been emphasized. Isolation of each virtual machine (VM), one of primary characteristics of virtualization technology, is another motivation that embedded systems want to employ virtualization technology.

Embedded software generally has timing constraints-real-time property. RTOS is required to guarantee temporal conditions for task execution. That is, OS manages all of the resources in the system to make the real-time tasks to be executed by a certain amount of time during each period. RTOS usually defines a task with execution-time, period, and deadline. It provides a guaranteed service with feasible scheduling algorithm - i.e. RM (Rate Monotonic) or EDF (Earliest Deadline First) [3]. The scheduler chooses a task to execute such that all tasks do not miss their deadlines. Thus, RTOS should maintain and predict the system to ensure the schedulability. The schedulability test analyzes the system in order to provide the guaranteed service for every real-time tasks in the system.

Questions arise whether RTOS is able to provide a guaranteed service even though it is implemented on a VM. Recent study [4] points out a problem of virtual machine monitor (VMM) scheduler that does not consider timing requirements of guest OS and proposes alternatives for real-time guest OS. Real-time guest OS is able to satisfy the temporal requirements if the VMM gives a high priority to RTOS when it schedules. However, in a virtual machine system, any guest OS cannot block the physical interrupts. Moreover, complex virtualization layer's overhead and VMM's scheduling algorithm can compromise the guaranteed execution of guest OS. Therefore, a novel I/O processing model which controls the interrupt is required to support a RTOS.

Virtual machine configuration is a difficult problem in embedded systems virtualization. Usually an embedded system with virtualization has a contraction that the whole system supports multiple different virtual ma-

[0]This study was partially supported by the Seoul Research and Business Development, Program, Seoul, Korea.

978-1-4244-2309-5/09/$25.00 ©2009 IEEE

chines, while each virtual machine has its unique target application. Thus, the virtualization layer should have an arbitration mechanism to solve the complex system requirements, especially related with devices. Tight resource limitation is also a big challenge. Therefore, recent virtual machine for embedded system divides a target system into several physical device groups, and dedicates the devices groups to a specific virtual machine. It makes not only the virtual machine system simple, but avoids the complex configuration problem.

We focus on a configuration that the RTOS running along with GPOS utilizes devices that are not used by RTOS. RTOS generally predicts the whole system's behavior including device drivers and interrupt processing. However, in a virtual machine environment, guest OS is unaware of devices attached to a GPOS and it may fail to provide the guaranteed service time because of unexpected interrupt preemption. Thus, interrupt processing should be properly controlled at VMM level to support a timely responsiveness. Moreover, the VMM should consider several side-effects when dedicated device model is applied.

The rest of this paper is organized as follows. We first investigate the related work and analyze Xen's I/O model from the real-time perspective. Experiments in section V present the effect of interrupts on response time with the VMM model described in section IV. Section VI summarizes and gives future work to be dealt with.

II. Related Work

Embedded systems have become powerful enough to take in the virtualization technology. Recent research paper addresses the opportunities and challenges for pocket-size hypervisors [5]. Study on mobile hypervisor presents a reliable system design using virtual machine architecture [6]. In another research work [7], the role and limitations of virtualization in embedded systems were discussed.

Several researches have been performed with Xen hypervisor which is ported on ARM-based mobile phone development board. A paper presents porting efforts of early version of Xen to StrongARM [8], but it was an experimental system. It can only boot up the simple guest OS and perform a small number of tests. Recent paper [9] presents issues on ARM CPU and memory virtualization. Their implementation is based on QEMU processor simulator. More practical approach has been announced in secure-Xen [10] based on Xen 3.0.2 for ARM-based mobile phone platform.

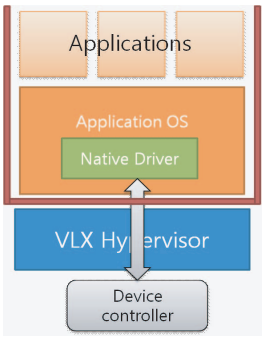

Fig. 1. VLX Dedicated driver model

Xen [11] is originally designed for incorporating hundreds of network servers but not for embedded real-time systems that have different requirements compared with server systems. Interrupt latency in Xen is not stable; thus it is hard to support the real-time requirement of digital appliances such as mobile phones. Furthermore, Xen's asynchronous I/O ring buffer is optimized for throughput, not latency. This asynchrony makes communication cost among the guest domains unpredictable. Recent study on Xen's I/O architecture and scheduler reveals problems that makes difficult for Xen to provide a real-time property [12]. In another research [4], the author discussed the timing requirements of VMs in real-time embedded systems and the possible approach to provide real-time support by a proper scheduler. The work is in progress and the performance analysis has not been done yet with implementation.

Trango [13] and VirtualLogix [14] are two commercial VMM implementations for ARM based processors. Recent paper from VirtualLogix presents several models of the device driver in VMM architecture [15]. However, comprehensive performance results are not presented. The VMM provides two device driver models in a virtual machine system. For a dedicated device, which is owned by only one virtual machine, they use native driver model. The device management functions are left to guest OS as presented in Figure 1. The authors insist that the performance is close to native one. For a shared device, the VMM provides bridge drivers inside a virtual machine monitor. Basically, their driver model is very similar to that of Xen. Therefore, fundamental problems that arise from the split driver model are unavoidable.

Traditional real-time operating systems have a solid theoretical background of performance analysis and management mechanisms [3]. Hierarchical scheduler [16]–[18] is regarded as an ultimate model of composing

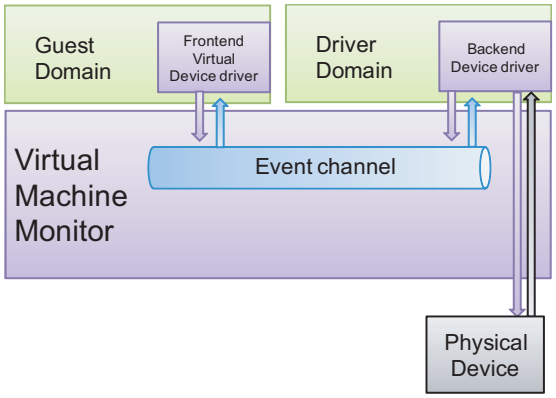

Fig. 2. Xen's split driver architecture

multiple real-time systems. There are several task models that consider the general periodic task scheduling model [17], explicit deadline period model [18], and bound-delay model [16]. Hierarchical scheduler performs resource scheduling or partitioning in the higher layer, and schedules tasks with the given resource partition in lower layer. Compositional analysis framework [18] provides a soundness of synthesis of multiple schedulers and a proof of preserving schedulability test. Bound-Delay model presents a more practical system model than previous real-time system studies. It uses time quantum and presents a soundness of bound-delay model in a partition scheduler. In addition, they provide a system application programming interface to easily deploy the model into a program.

III. Analysis of XEN I/O Model

Real-time system is supposed to be predictable and deterministic. Predictability means a responsiveness of a program execution within a given temporal expectation. Determinism is a behavioral limitation that the system can be analyzed before it is executed. Real-time systems support two major properties based on the system analysis.

Real-time schedulability test is the system analysis technique to support predictability and determinism. It provides 1) a test whether each real-time task can be scheduled to finish before its deadline, and 2) an admission control mechanism to maintain all the tasks in the system as schedulable at any time.

Here, we analyze Xen, one of the most widely used virtual machine systems. The major problem of real-time support in Xen architecture is the device driver model. Xen has the following split driver model as shown in Figure 2. Figure 3 presents latency overhead in traditional Xen VMM.

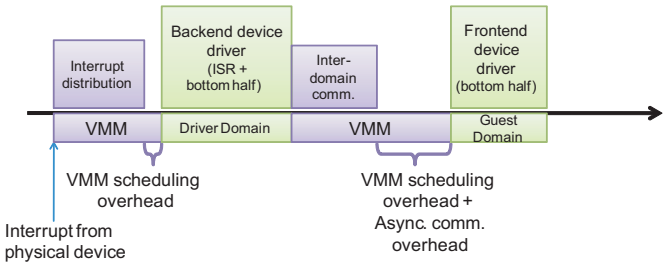

Fig. 3. Interrupt processing overhead in Xen

To handle I/O interrupts, Xen requires
1) A VMM scheduling to have the CPU at back-end driver,
2) An additional overhead for inter-domain communication, and
3) Another VMM scheduling to be received at the front-end driver.

Overhead incurred by the VMM scheduling is substantial. Because the device driver runs at a guest domain, the device driver cannot be processed if the driver domain used up its time quantum. This makes a problem when the I/O interrupt occurs very frequently. Frequent interrupt processing consumes CPU time very quickly and the interrupt processing at the back-end driver is stalled. The interrupt processing latency incurred by the scheduling has been addressed in recent paper.

Besides, there are fundamental problems in real-time perspectives as follows: Xen architecture lacks to provide a predictability and determinism.

- Predictability perspective: Xen does not have any special mechanism that controls the interrupt processing at VMM level. Processing time within an interrupt service routine cannot be bounded. Moreover, driver domain's low-level processing time consumes time quantum of the domain, and it is not bounded. In addition, Xen uses asynchronous ring buffer when guest domains communicate, which does not provide the bounded communication latency.
- Determinism perspective: Xen uses a credit scheduler which is not preemptive. Thus, it cannot preempt the processor when a task is required to be scheduled urgently. In addition, Xen does not have any protection mechanism of the processor overuse in per-VM basis. Therefore, real-time application that requires a guaranteed execution time would not meet its requirements. This makes system analysis impossible because the processing time cannot be guaranteed. Moreover, there is no performance anal-

ysis has been performed for RT-VM and non-RT VM configuration.

In summary, Xen is inappropriate for real-time applications for the following three reasons. First, Xen's split driver model has inherent processing overhead incurred from separated processing and scheduling. Second, Xen does not provide predictability because the interrupt processing time cannot be bounded in per-VM basis, and the asynchronous communication among the guest domains makes hard to expect the communication cost. Third, Xen cannot provide determinism because Xen's default credit scheduler is not a preemptive one, and it is difficult to deliver the control to an urgent domain. Besides, it does not guarantee an execution time to a task because of the unlimited interrupt handling, which makes hard to analyze the system. Moreover, fundamental theoretical basis for the schedulability analysis on RT and non-RT VM has not been provided up-to now.

IV. I/O Model for Real-time Support in Virtual Machine

We categorize devices into four groups based on usage model of devices. At first, *Dedicated devices*- are not shared among virtual machines, and dedicated to a target virtual machine. Once a device is dedicated to a VM, isolation is not required because only the guest who controls the virtual machine is responsible for the device. Other guest OSes and applications are unaware of the device, and cannot use it because the guest OS does not have any drivers and configurations.

For shared devices, we give other criteria for sharing model. There are shared devices, but explicitly changed by the user. Namely, the devices are active only when the user explicitly gives a control. Most of the human interactive devices are considered as this type. For example, keyboard, display are dedicated to a specific VM until the user explicitly switches the VM. It is similar to foreground task switching in a window-based interface. Only the active VM takes the control, and the other VM does not get interrupt or send event until it is activated. We call these devices as *Active devices*.

Another type is *Running device*. Running devices are under control of the currently running virtual machine. For example, timer is a representative running device. Timer interrupt is always delivered to the running VM. The VMM is aware of the running VM, and redirects the interrupt to the running VM. When a context switch occurs, the devices should not have state information; otherwise, the device context from the previous virtual machine can corrupt the present device context.

Finally, for a fully shared device is named *Dynamic device*. The VMM takes an event from a physical device and it determines the target VM at runtime. Besides, the guest OS can access to dynamic devices at any time. The virtualization layer should involve in proper handling of physical devices, and it should provide a virtual device interface to be safely isolated from the other virtual machines. The virtual machine monitor has a driver stub for each virtual device and redirect event to a designated virtual machine to be processed at guest OS context.

Categorized device driver processing simplifies the VMM structure, as well as performance isolation. Except for dynamic devices, interrupt processing is executed within the guest OS time quantum. That is, physical interrupt processing does not consume the execution of other virtual machines. Interrupt from a dedicated device is stored inside the virtual machine monitor, and delivered when the target guest becomes runnable. Similarly, interrupt from an active device is held in the VMM and handled when the designated guest is runnable. For a running device, device driver should be implemented in stateless manner as explained in the above. This makes VM-switch operation simple and light because device context does not need to be saved and restored.

For a dynamic device, VMM processes the basic interrupt handling and determines the target VM. To determine the target guest, sometimes the VMM should have a data to be processed. Thus, the VMM takes the data within it, and delivers the data without copy. To avoid the data copy between the VMM and guest OS, light-weight data communication mechanism such as page mapping is required. At the same time, VMM should avoid asynchronous communication because it makes the communication time unpredictable. To minimize the processing time at VMM, several optimization options can be applied.

V. Physical Interrupt Impact in Virtual Machine

A. Scheduler of MobiVMM

We implement a simple VMM (MobiVMM) on an industry-standard ARM platform (ARM11 mpcore board). Our MobiVMM [19] currently runs two VMs and each VM runs Linux as its guest operating system. Guest OS1 is regarded as RTOS while Guest OS2 is the GPOS. VMM schedules VM by the round robin algorithm with 20ms time quantum. Figure 4 depicts an example for two guest OS running over VMM.

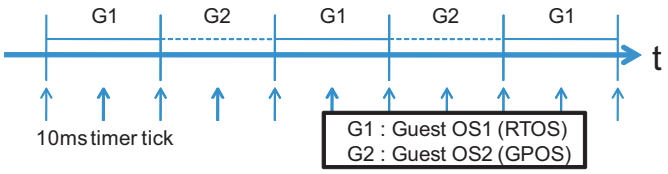

Fig. 4. MobiVMM scheduler

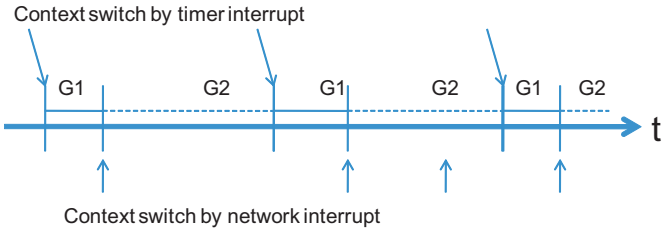

Fig. 5. MobiVMM's I/O handling

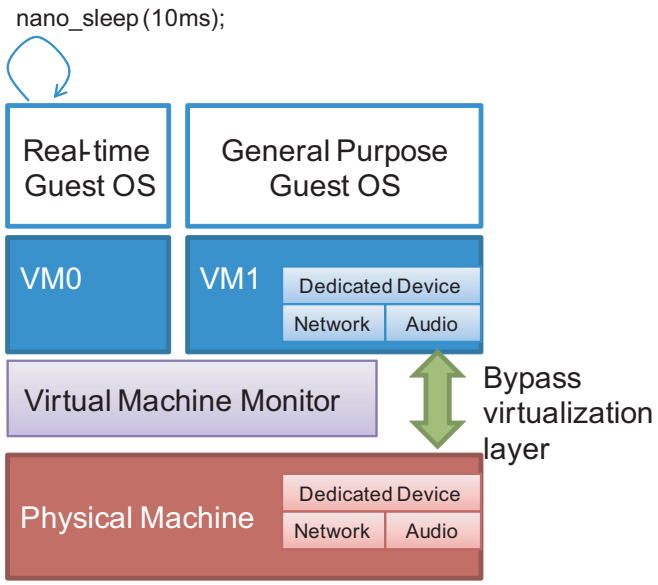

Fig. 6. MobiVMM system architecture

B. I/O Handing of Dedicated Device

Guest OS2 preempts whenever dedicated interrupts - audio and network - occurr. The VMM assumes that the I/O handling should be delivered as fast as possible, and the guest OS gets prioritized access when the interrupt is ready to be delivered as described in Figure 5. The rest of a quantum is consumed by guestOS2 and the VMM schedules at the next timer tick. That is, the remaining time of a quantum is dedicated to I/O processing in guestOS2 when preempted. GuestOS1 would be executed at the next time quantum. It lets guestOS2 process the I/O interrupts designated only to guestOS2 and makes VMM simple.

VI. Experiments

This section observes the response time of the real-time guest OS with varing physical interrupt load. Because the dedicated devices are not virtualized and serviced by the guest directly, the other guest OSes suffer from large response time. Scheduler has a strong impact because when an interrupt from dedicated device preempts the current VM's time quantum, the VMM scheduler should schedule more frequently to provide a timely response.

We measure response time at guestOS1(RTOS), and give network and audio interrupt load to guestOS2(GPOS). Response time of application on guestOS1 is measured as the elapsed time for 10ms nano sleep call. We assume guestOS1 is the RTOS and its response time reflects impact of other VMs. Overall system architecture is presented in Figure 6. Table I contains response time when no interrupts occur for both VM. 17.6ms is a response time when guest OS1 launched alone on VMM. 37.5ms shows delay due to the guest OS2.

TABLE I
Response time without interrupts

Configuration	Response time of app. on GuestOS1 (ms)
GuestOS1 only	17.6
GuestOS1,GuestOS2	37.5

Figure 7 explains the second case, two guest OSes running over VMM. We had the application slept and G2 receives two 10ms timers for the next time quantum. When G1 has two 10ms timers - four 10ms timers in view of VMM, G1 takes it only 10ms has passed since it could not receive the previous ones. Thus G1 wakes up the application after having the fourth 10ms timer, and total elapsed time for 10ms sleep call on G1 is around 37.5ms in the experiment.

Response time of applications on Guest OS1 grows

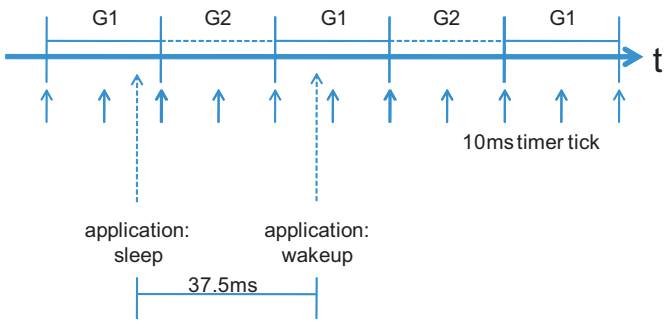

Fig. 7. 10ms response time for two guest OSes without interrupts

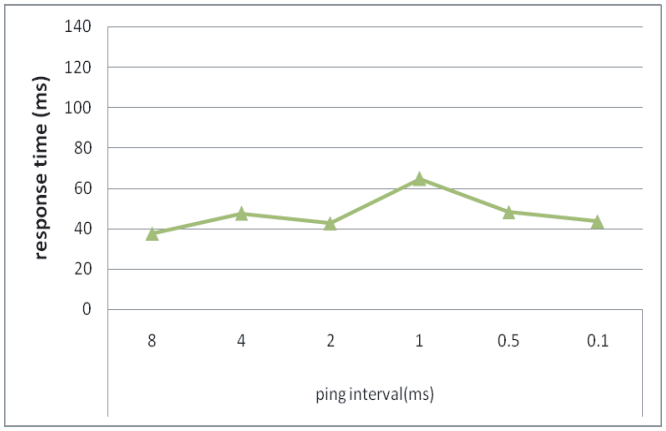

Fig. 8. Effect of network interrupt (another VM) on response time

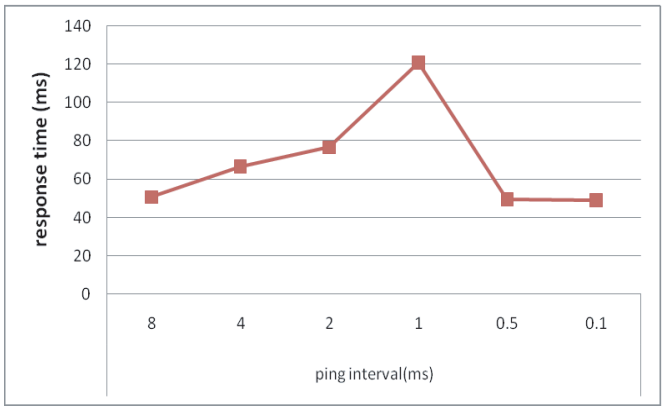

Fig. 9. Effect of network and audio interrupts

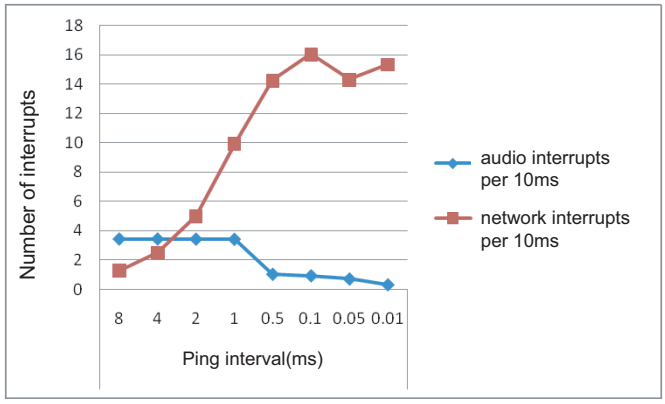

Fig. 10. Number of interrupts per 10ms

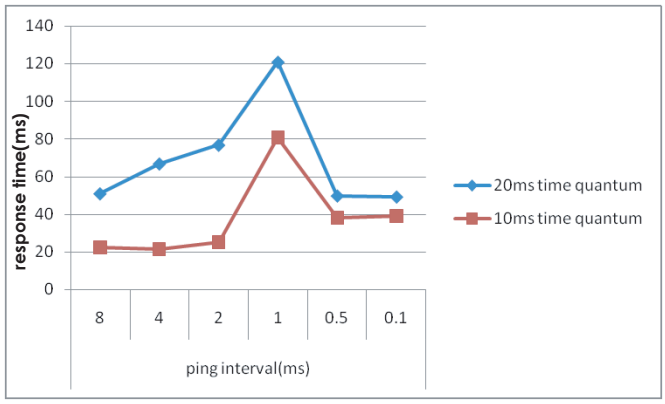

Fig. 11. Effect of time quantum of VMM scheduler on response time

as interrupts load dedicated to Guest OS2 increases. We sent ping from outside to Guest OS2 with increasing ping interval.

The response time ranges from 40ms to 60ms as ping interval reduces in Figure 8. Too many interrupts keep preempting and they prevent G1 from executing to wake up the application. 60ms response time at 1 ms ping interval explains this situation. However, it shows that response time reduces from 0.5ms ping interval. Guest OS2 now has too many interrupts to handle in a given time quantum. It blocks interrupts temporally and results in shorter response time of application on Guest OS1 because preemption does not happen for a certain time. Thus, average response time becomes better from 0.5ms.

We have generated another interrupts besides network interrupt to observe the effects of different type of interrupt. Now interrupts load is heavier than Figure8 along with audio device.

Overall response time is increased as shown in Figure 9 for interrupt load from audio device has been increased. At 1ms ping interval, it has the largest response time but suddenly falls down at 0.5ms. We observed the number of interrupts decreased at 0.5ms as shown in Figure 10. Especially, number of audio interrupts became 0 and audio glitches are heard. Too many network interrupts prevent audio interrupts because network device has higher priority than audio device.

The VMM basically schedules VMs with round robin algorithm. Time quantum is generally from 10ms to 100ms and we choose 20ms for our experiments. We change it to 10ms and compare them by testing the same configuration of Figure 9. Figure 11 shows the results.

Application experiences shorter response time for smaller time quantum. Frequent scheduling gives VMs short response time but context switching overhead should be considered.

VII. Conclusion and Future Work

In this study, we focus on the problems of contemporary virtual machine monitor as a real-time platform. Xen virtual machine monitor is not appropriate

for real-time applications in terms of predictability and determinism. We propose an alternative I/O model to categorize the devices into smaller groups that simplifies the complex configuration of embedded virtual machine system. Dedication of device makes the virtual machine monitor simple; however, to support performance isolation, additional effort is required. Among the I/O models, we investigate that whether dedicated device model is feasible to support real-time services in terms of responsiveness.

Our investigation proves that the preemption occurred by physical interrupts causes serious performance degradation in terms of response time at another guest domain. As the interrupt load increases, responsiveness degrades seriously. It depends not only on the interrupt load, but also the interrupt type and priority. Network interrupt of which top half finishes vary quickly has less effect and prioritized interrupt processing would be helpful provided that the priority is given properly. In addition, our experiment shows that shorter time quantum gives better responsiveness.

This paper concentrates on the response time affected by devices which are not shared with other VMs. However, guest OS on each VM usually requires the same physical resources. We are proceeding with the VMM model which considers guaranteed I/O processing for shared devices. We plan to improve MobiVMM to support RTOS by a noble admission control mechanism of interrupt processing. Controlled interrupt load will help to support guaranteed service in terms of responsiveness as well as execution time.

REFERENCES

[1] A. Bose and K. G. Shin, "On mobile viruses exploiting messaging and bluetooth services," *Securecomm and Workshops, 2006*, pp. 1–10, September 2006.

[2] *Cyber-criminals target mobile banking*, Tower Group, 2007, http://www.vnunet.com/vnunet/news/2173161/cyber-criminals-targetmobile.

[3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[4] R. Kaiser, "Alternatives for scheduling virtual machines in real-time embedded systems," in *IIES08 - 1st EuroSys 2008 ACM SIGOPS Workshop on Isolation and Integration*, April 2008.

[5] L. Cox and P. Chen, "Pocket hypervisors: Opportunities and challenges," *Mobile Computing Systems and Applications, 2007. HotMobile 2007. Eighth IEEE Workshop on*, pp. 46–50, March 2007.

[6] J. Brakensiek, A. Droge, H. Hartig, A. Lackorzynski, and M. Botteck, "Virtualization as an enabler for security in mobile devices," in *Workshop on Isolation and Integration in Embedded Systems, IIES, 2008. Glasgow, UK. 1st ACM SIGOPS*, April 2008.

[7] G. Heiser, "The role of virtualization in embedded systems," in *Workshop on Isolation and Integration in Embedded Systems, IIES, 2008. Glasgow, UK. 1st ACM SIGOPS*, April 2008. [Online]. Available: http://ertos.nicta.com.au/publications/papers/Heiser_08.pdf

[8] D. R. Ferstay, "Fast secure virtualization for the arm platform," Master's thesis, University of British Columbia, 2006.

[9] R. Bhardwaj, P. Reames, R. Greenspan, V. S. Nori, and E. Ucan, *A Choices Hypervisor on the ARM architecture*, University of Illinois, Urbana-Champaign, 2006, cS523 Course Project Report.

[10] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim, "Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones," *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 257–261, January 2008.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*. New York, NY, USA: ACM, 2003, pp. 164–177.

[12] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling i/o in virtual machine monitors," in *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA: ACM, 2008, pp. 1–10.

[13] *Trango: secured virtualization on ARM*, Trango, http://www.trango-vp.com.

[14] *VirtualLogix Real-Time Virtualization and VLX*, VirtualLogix, http://www.osware.com.

[15] F. Armand, M. Gien, G. Maign?, and G. Mardinian, "Shared device driver model for virtualized mobile handsets," in *1st international conference on Mobile System Virtualization (MobiVirt)*, Breckenridge, CO, USA, 2008.

[16] X. A. Feng and A. K. Mok, "A model of hierarchical real-time virtual resources," in *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 26.

[17] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pp. 75–84, 2001.

[18] A. K. Mok and A. X. Feng, "Real-time virtual resource: A timely abstraction for embedded systems," in *EMSOFT '02: Proceedings of the Second International Conference on Embedded Software*. London, UK: Springer-Verlag, 2002, pp. 182–196.

[19] S. Yoo, Y. Liu, C.-H. Hong, C. Yoo, and Y. Zhang, "Mobivmm: a virtual machine monitor for mobile phones," in *1st international conference on Mobile System Virtualization (MobiVirt)*, Breckenridge, CO, USA, 2008.