Guest-Aware Priority-Based Virtual Machine Scheduling for Highly Consolidated Server[⋆]

Dongsung Kim[1], Hwanju Kim[1], Myeongjae Jeon[1], Euiseong Seo[2], and Joonwon Lee[1]

[1] CS Department, Korea Advanced Institute of Science and Technology
[2] Pennsylvania State University
{dskim,hjukim,mjjeon}@camars.kaist.ac.kr, euiseong@cse.psu.edu, joon@cs.kaist.ac.kr

Abstract. The use of virtualization is rapidly expanding from server consolidation to various computing systems including PC, multimedia set-top box and gaming console. However, different from the server environment, timeliness response for the external input is an essential property for the user-interactive applications. To provide timeliness scheduling of virtual machine this paper presents a priority-based scheduling scheme for virtual machine monitors. The suggested scheduling scheme selects the next task to be scheduled based on the task priorities and the I/O usage stats of the virtual machines. The suggested algorithm was implemented and evaluated on Xen virtual machine monitor. The results showed that the average response time to I/O events is improved by 5∼22% for highly consolidated environment.

Keywords: Xen, Credit scheduler, Virtual Machine, Virtualization.

1 Introduction

The operating system virtualization is resurrected as a key technology for server consolidation, which reduces the cost for management and deployment. As blossoming in performance aspects [1,2], the use of virtualization has expanded gradually into various fields such as multimedia, game, and interactive applications [3,4,5]. Although diverse workloads have become feasible in the virtualized environment, contemporary virtual machine monitors (VMMs) generally focus on fairness of VMs and the improvement of I/O throughput. Therefore, latency-sensitive applications would preform poorly in the virtualized environment.

The scheduling turn-around time for each VM could be considerable under highly consolidated environment. Since VMM typically has lack of knowledge about guest-level tasks, the status and priorities of tasks, which run on each guest kernel, can not be considered when VMM chooses the next VM to be

[⋆] This work was supported by KOSEF grant funded by the Korea government(MOST) (No. R01-2006-000-10724-0) and also partially funded by the MIC, Korea, under the ITRC support program supervised by the IITA.

scheduled. Such CPU allocation mechanism can increase the response time of a latency-sensitive application with a high priority. As more VMs are consolidated on a physical machine, larger performance degradation occurs and consequently results in unsatisfactory quality of service [6].

This paper introduces a priority-based VM scheduling algorithm to reduce scheduling latency of a VM that requires timeliness response. In our scheme, VMM allocates CPU to each VM based on the guest-level task information, which is provided by each guest kernel. VMM prioritizes VMs by using the collected information about priorities and status of guest-level tasks in each VM. Our algorithm preferentially treats the VMs that run latency-sensitive applications in response to I/O by inspecting I/O pending status. The proposed algorithm is implemented and evaluated on Xen, which is a virtualization software widely used by many VM researchers.

The rest of this paper is organized as follows. Section 2 briefly describes related work and Xen architecture especially focusing on credit scheduler and I/O handling mechanism. Section 3 demonstrates the mechanism on how VMM uses information of guest-level tasks and proposes the priority-based VM scheduling algorithm. Section 4 evaluates our mechanism in terms of performance and fairness compared with the credit scheduler. Finally, section 5 summarizes the paper and presents the future work.

2 Related Work

2.1 Xen Virtual Machine Monitor

Xen is an open source virtual machine monitor based on para-virtualization, which makes a guest operating system aware of underlying virtualization layer through kernel modification [2]. The para-virtualization achieves large performance improvement by optimizing a guest operating system to virtualized architecture. In Xen architecture, a VM is referred to as a *domain* and the privileged VM, called *domain0*, controls other guest domains.

Credit scheduler, which is the default scheduler in Xen 3.0, manages CPU allocation for VMs based on *credit* value set by predefined weight of each VM [7]. The calculated credit is assigned to each VM every 30ms and is consumed proportional to the processing time of the VM; this consumption is conducted at the granularity of a tick interval (10ms).

Credit scheduler has three priorities: *BOOST(0)*, *UNDER(-1)*, and *OVER(-2)*. Two priorities (UNDER and OVER) are exclusively determined based on the remaining credit of a virtual CPU, which belongs to a VM. If the remaining credit of a virtual CPU is larger than zero, the virtual CPU has the priority of UNDER; otherwise, the virtual CPU has the priority of OVER. To guarantee fairness, a virtual CPU with UNDER priority is preferentially scheduled than those with OVER priority. When a virtual CPU with UNDER priority is woken by an event such as I/O completion, it acquires BOOST, which is the highest priority; this mechanism makes I/O-bound VMs be scheduled earlier than others. Although credit scheduler guarantees fairness, it results in a large response time of a latency-sensitive task with high priority since credit scheduler does not consider the priorities of individual tasks inside each VM.

Xen introduces *isolated driver domain*(IDD) to allow specific domains to access hardware directly. To do this, IDD includes native device drivers and conducts I/O operations on behalf of all guest domains [8]. A guest domain has virtual device driver, called a *frontend* driver, which communicates with a *backend* driver in an IDD. Xen uses shared I/O descriptor rings and event channels to communicate between an IDD and guest domains. I/O descriptor ring is a circular queue of I/O descriptor with producer/consumer pointers and is shared between an IDD and a guest domain. A virtual interrupt is delivered via an event channel for notification of I/O requests and completions.

2.2 Purpose-Specific Virtual Machine Scheduling

In addition to the fairness support of VM scheduling, there have been researches on VM scheduling optimization for specific workloads.

Govindan et al. [6] proposed a communication-aware scheduling algorithm to deal with the problem of VM scheduler on highly consolidated hosting platform. They showed that current VM schedulers do not consider communication behavior of VMs and thus result in degraded response time. Communication-aware scheduling algorithm takes into account network communication patterns when VMM chooses a VM to be scheduled. This algorithm preferentially schedules I/O-intensive VMs by using heuristic methods on the basis of the amount of I/O operations. This mechanism, however, could not improve the response time of the VM that has latency-sensitive tasks with high priority because it does not consider guest-level priority.

Cherkasova et al. [9,10] analyzed three CPU schedulers of Xen(BVT, sEDF, and credit) by measuring I/O throughput for different scheduling parameters. Their experiments demonstrate the performance impact of CPU allocation for domain0, which hosts I/O on behalf of guest domains. They show that frequent interventions of domain0 degrade I/O throughput because they incurs several domain switches and prevents guest domains from batching I/O requests. This work illustrates challenging issues related to VM scheduling mechanism for varied workloads.

Ongaro et al. [7] has explored the impact of VMM scheduling on I/O performance where different types of applications are run concurrently in multiple domains. Through various experiments, they improved I/O performance of Xen by optimizing the credit scheduler and alleviating the unfairness of event processing mechanism. The optimized credit scheduler sorts domains in run queue based on remaining credits so that short-running I/O domains are preferentially scheduled. It reduces the variance in the delivery of events by preventing the driver domain from tickling the scheduler when an virtual interrupt occurs. Their technique, however, cannot address a mixed domain, which include both I/O- and CPU-bound tasks. Our approach preferentially can schedule the latency-sensitive tasks, which is not even related to I/O or is in the mixed domain, by considering guest-level priority.

3 Guest-Aware Priority-Based Scheduling

3.1 Motivation

Since multiple guest domains share a single underlying hardware, a physical interrupt is not delivered to a corresponding domain immediately. A physical interrupt is received by VMM first and then is delivered to a destination domain. In the case of I/O interrupt, VMM forwards the received interrupt to domain0, which then is scheduled and notifies the target domain of the interrupt. Due to this procedure, the time when the target domain receives an I/O event depends on the status of the run queue of VMM. Domain0 is not guaranteed to be scheduled instantly after an interrupt, and furthermore several domains can be placed on the run queue between domain0 and the target domain. The target domain should wait until preceded domains finish their execution when it runs a latency-sensitive task with CPU-bound tasks simultaneously. Therefore, a latency-sensitive application in the target domain suffers low responsiveness, especially for highly consolidated server system.

This paper addresses the problem where current VM scheduler manages run queue without considering guest-level tasks. We propose the algorithm that assigns effective priority based on guest-level tasks. Our algorithm modifies the original run queue management that is simply sorted by credit-based priority. In our scheme, VMM dynamically re-assigns finer grain priorities than the original credit scheduler to guest domains based on the information about guest-level tasks in the run queue and wait queue. In addition, VMM infers which domain waits for I/O events by using the status of shared I/O descriptor rings in domain0.

3.2 Design

Our priority-based scheduling algorithm adopts an intrusive approach in that guest domains explicitly expose their local information to VMM. In this approach, a guest kernel is modified to inform VMM of the priority and status of its tasks using shared variables. Based on the information, the priority of a

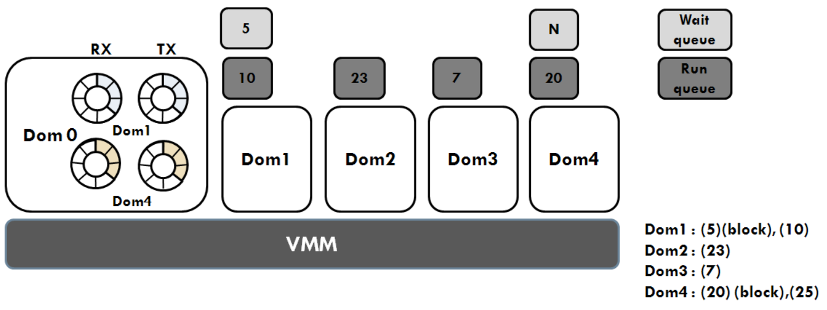

Fig. 1. Xen scheduling scheme

guest domain is simply taken as the highest priority of tasks in the run queue of the guest domain. Figure 1 shows the snapshot of guest domains and tasks of each guest domain in run queue and wait queue; the number in each task shows the priority of the task. First, we just use a simple approach by taking domain's priority as the priority of the highest active task in it. In this figure, dom1's priority is 10, which is the same as the highest priority of tasks in the run queue of dom1, and others are assigned similarly; smaller value is higher priority. Dom3 has the highest priority and will be scheduled first according to the simple approach.

This approach, however, has a limitation because it does not consider the tasks in wait queue. For example, since dom1 has a task with the highest priority, 5, in its wait queue, it should be scheduled earlier than dom3 if the event for which the task waits is pending at that time. To solve the above limitation, we take account of both run queue and wait queue of a guest domain to choose next VM to be scheduled. For this, a guest domain also has to expose the highest priority of the task that is on block state and is not included in run queue. In our approach, VMM inspects the status of shared descriptor rings to check whether I/O is pending with respect to the domain that has the blocked task with the highest priority. Hence, in the case of the above example, our mechanism makes dom1 scheduled in advance of others if corresponding I/O is pending at scheduling time.

The mechanism that inspects pending I/O and blocked tasks induces a problem if the pending I/O is unrelated with the blocked task of a guest domain scheduled by our algorithm. For this reason, it needs to correlate pending I/O with a blocked task exactly. The exact relation, however, can incur significant computational overhead due to the inspection of process-related clues from pending I/O. In case of TCP/IP networking, for example, VMM should examine almost all network packets including control packets such as ICMP or ARP to find a port number. Moreover, a guest domain should export port mapping of blocked tasks.

We propose a probabilistic method to relate pending I/O with the blocked task in the guest domain. This method makes a guest domain inform VMM of the list of recently woken tasks by I/O completion. As with LRU (Least Recently Used) mechanism, the VMM regards these tasks in this list as active tasks, which

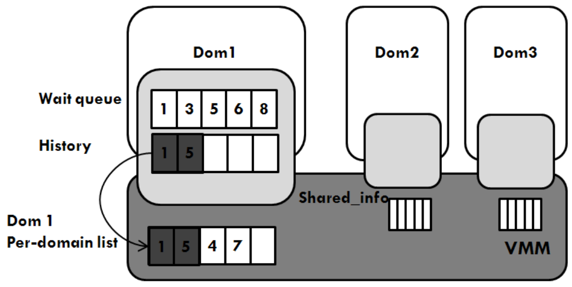

Fig. 2. Probabilistic method for relation between I/O and blocked tasks

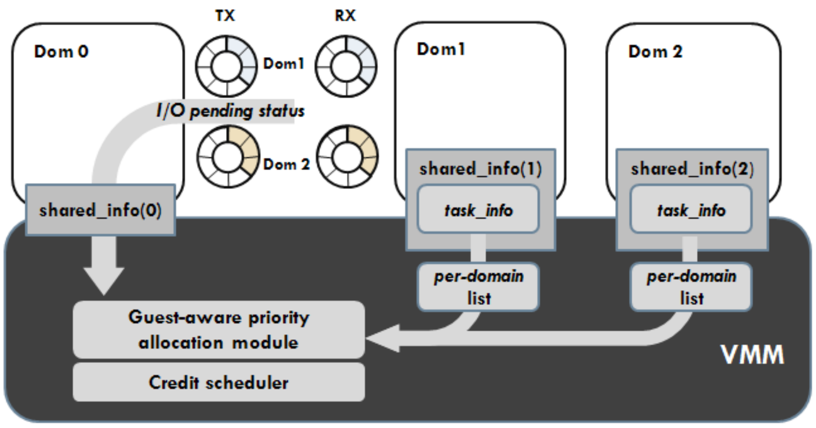

Fig. 3. System design

are likely to be involved with the I/O in near future. Figure 2 shows that two tasks,1 and 5, are recently woken by I/O while five tasks currently reside in wait queue of dom1. The per-domain list of dom1 maintains task 1 and 5 and can be referenced by VMM. When VMM prioritizes a guest domain, it checks both pending I/O and the per-domain list that maintains active I/O tasks. Only if pending I/O exists and the blocked task with high priority is within this list, the priority of blocked task is reflected to the effective priority of the guest domain.

3.3 Assumption

First, we assume that our VM system works in the trusted environment. In other words, there are no malicious users who intentionally raise the priority of its own task. The intentional priority boost can result in the performance degradation of other guest domains. Though VMM can detect and restrict malicious priority boosting, we left this as a policy issue.

Second, the exposed guest-level priorities should be coordinated by the unified scale since each operating system has its own priority system. In this paper, we only propose the way to scale priority systems of three prominent operating systems: Linux, FreeBSD, and Windows XP. The priority translation module have not yet implemented.

3.4 Implementation

Our implementation is based on Xen 3.0.4 and para-virtualized Linux 2.6.16. We modified a Linux kernel to share guest-level task information with VMM via the data structure shared between a guest kernel and VMM. As shown in figure 3, we introduce an additional structure which is called *task_info*. The shared structure contains information about runnable and blocked tasks with the highest five priorities. To relate pending I/O with blocked tasks, the shared structure also stores task IDs, which are recently woken by I/O. Since domain0 can access shared I/O descriptor rings for all guest domains, domain0 exposes pending I/O information to VMM via shared structure. The number of pending I/O is calculated by using producer/consumer pointers in the shared rings.

We use hierarchical priority scheme to support CPU fairness. In this scheme, the scheduler basically manages run queue in the original manner by three priorities of credit scheduler. When the scheduler chooses a guest domain to be scheduled next, an effective priority is decided on the basis of guest-level task information only if the selected guest domain has UNDER priority. Such mechanism prevents a guest domain with the highest priority task from monopolizing CPU resource. More importantly, this mechanism does not compromise CPU fairness of credit scheduler.

4 Evaluation

4.1 Evaluation Environment

We used Xen hosted server with pentium 4 2.4GHz CPU and 1.5G RAM for experiments. A client machine, which is used for measurement of network response time, has Pentium 4 2.4GHz CPU and 1G RAM. We allocated 256MB memory to domain 0, and 128MB to each guest domain. The weights of all domains are equally fixed for fair sharing. In our experiment, the number of domain varies from four to eight. We evaluate our scheduling algorithm in two approaches: *GAPS-RO (Guest-Aware Priority-based Scheduling - Run queue Only)*, a simple method that reflects the highest priority of tasks in run queue only, and *GAPS(Guest-Aware Priority-based Scheduling)*, a method that considers tasks in both run queue and wait queue as well as pending I/O in domain0.

4.2 Scheduling Latency

A physical interrupt is delivered to the target domain through the VMM and domain0 as described in the above background section. Since network or disk I/O is batched and requires I/O processing, it is difficult to measure pure scheduling latency for repeated experiments; scheduling latency means the elapsed time until the target domain is scheduled just after a corresponding interrupt occurs. We implement *vlatdriver*, a simple split driver consisting of frontend and backend, to exactly measure scheduling latency using a virtual interrupt. After generating a virtual interrupt, vlatdriver records timestamps at VMM, the backend driver in domain0, and the frontend driver in the target domain.

We run five guest domains with CPU-intensive tasks; one's priority is higher than others for each experiment. Figure 4 shows the average latency and the lowest 10% latency as worst case for credit scheduler and *GAPS*. The result

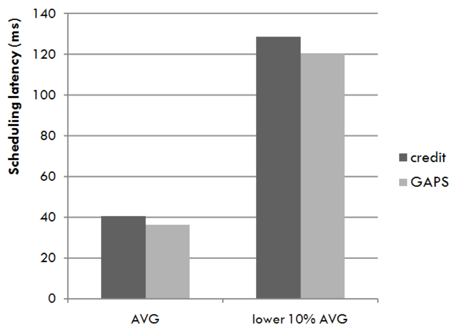

Fig. 4. Scheduling latency

demonstrates that scheduling latency is reduced for both the average and the worst case.

The scheduling latency of GAPS is more reduced compared with credit scheduler in the worst case than the average. Our algorithm preferentially schedules the domain with the highest effective priority when the domain is given UNDER priority; on the other hand, credit scheduler executes each domain in the round-robin manner in case where all domains are CPU-intensive. Therefore, GAPS decreases scheduling latency by reducing the waiting time until scheduling the domain with high priority.

4.3 I/O Response Time

We measure I/O response time through a ping-pong test in simple server/client environments; a client stressfully sends small requests to the server. Each guest domain runs a CPU-intensive task. Only dom1 also contains a server daemon with the highest priority. The priority of the CPU-intensive task in dom2 is higher than those of all CPU-intensive tasks in other domains, but is lower than that of the server daemon in Dom1. The experiment with this configuration evaluates the difference between *GAPS-RO* and *GAPS*. *GAPS-RO* preferentially schedules dom2 because the server daemon in dom1 is I/O-intensive and thus almost resides in wait queue waiting for a client's request. In *GAPS*, on the other hand, dom1 with the server daemon is likely to be scheduled earlier than dom2 by considering pending I/O and blocked tasks.

Figure 5 shows the response time of a client for credit scheduler, *GAPS-RO*, and *GAPS* as the number of guest domains increases. Both *GAPS-RO* and *GAPS* achieve lower response time than credit scheduler. In addition, *GAPS* reduces more response time than *GAPS-RO* since *GAPS* strives to schedule dom1 in advance of others. As the number of guest domains increases, the improvement in response time is larger. Consequently, our scheduler accomplishes low responsiveness of latency-sensitive applications on high consolidated environment.

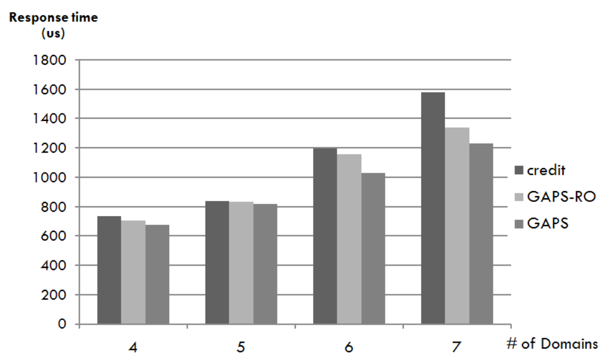

Fig. 5. Average I/O response time

Table 1. CPU fairness: The consumed CPU for each domain is normalized on the basis of dom1

	Dom1	Dom2	Dom3	Dom4
Credit scheduler	1	1	0.999	1
GAPS	1	1	0.998	0.994

4.4 Fairness Guarantee

We evaluate that our scheduling algorithm does not compromise CPU fairness supported by credit scheduler. Four guest domains run CPU-intensive tasks, and only dom1 additionally runs server daemon with the highest priority similar to the above experiment. The dom4's task has the lowest priority to show the possibility of starvation.

Table 1 shows the CPU allocation for each guest domain. The CPU allocation is calculated from the consumed credit of each guest domain during the experiment. All results are normalized by that of dom1. Although dom4 has slightly less CPU allocation than others for our scheduler, this difference is negligible. Our scheduling algorithm, therefore, still guarantees CPU fairness and incurs no starvation of the guest domain including the lowest priority task. However, we do not address I/O fairness, which is not considered in the original credit scheduler.

5 Conclusion and Future Work

Although virtualization technologies have advanced in terms of high degree of consolidation, the absence of support for latency-sensitive workload could be an obstacle to services that need good quality of responsiveness. To address this problem, we introduce a guest-aware priority-based scheduling, which runs on Xen-based system, to preferentially schedule high-priority and latency-sensitive tasks. Our mechanism guarantees CPU fairness because it is implemented over the credit scheduler of Xen.

In this paper, the proposed mechanism is achieved by the intrusive way for VMs to send information of guest-level information to VMM. The intrusive approach has some drawbacks. First, this approach requires guest kernel modifications. This requirement cannot be applied to closed-source operating systems such as Windows. Second, the guest-level information could be untrusted because the information is explicitly informed by guest domains; the intrusive way can impede VM isolation in the untrusted environment.

As future work, we are developing the non-intrusive mechanism for reducing the response time of latency-sensitive tasks. To support responsiveness without kernel modifications, we should determine a VM that has latency-sensitive tasks by inferring from I/O behavior and scheduling pattern. By this approach, we will achieve the responsiveness as well as fairness while preserving VM isolation.

References

1. Sugerman, J., Venkitachalam, G., Lim, B.H.: Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In: Proc. of the USENIX Annual Technical Conf., Berkeley, CA, USA, pp. 1–14. USENIX Association (2001)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM, New York (2003)
3. Neumann, D., Kulkarni, D., Kunze, A., Rogers, G., Verplanke, E.: Intel Virtualization Technology in embedded and communications infrastructure applications. 10(3) (August 2006)
4. VMware: http://www.vmware.com
5. Lin, B., Dinda, P.A.: Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In: SC 2005, p. 8. IEEE Computer Society, Los Alamitos (2005)
6. Govindan, S., Nath, A.R., Das, A., Urgaonkar, B., Sivasubramaniam, A.: Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In: VEE 2007: Proceedings of the 3rd international conference on Virtual execution environments, pp. 126–136. ACM, New York (2007)
7. Ongaro, D., Cox, A.L., Rixner, S.: Scheduling i/o in virtual machine monitors. In: VEE 2008: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 1–10. ACM, New York (2008)
8. Fraser, K., Hand, S., Neugebauer, R., Pratt, A.W.I., Williamson, M.: Safe hardware access with the xen virtual machine monitor. In: Proc. of Workshop on Operating System and Architectural Support for the on demand IT Infrastructure (2004)
9. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three cpu schedulers in xen. SIGMETRICS Perform. Eval. Rev. 35(2), 42–51 (2007)
10. Cherkasova, L., Gupta, D., Vahdat, A.: When virtual is harder than real: Resource allocation challenges in virtual machine based it environments. Technical Report HPL-2007-25 (February 2007)