PaS: A Preemption-aware Scheduling Interface for Improving Interactive Performance in Consolidated Virtual Machine Environment

Conference Paper · January 2009

DOI: 10.1109/ICPADS.2009.51 · Source: DBLP

Yubin Xia
Shanghai Jiao Tong University
36 PUBLICATIONS 1,165 CITATIONS
SEE PROFILE

Xu Cheng
National Huaqiao University
83 PUBLICATIONS 615 CITATIONS
SEE PROFILE

PaS: A Preemption-aware Scheduling Interface for Improving Interactive Performance in Consolidated Virtual Machine Environment

Yubin Xia[1,2], Chun Yang[1], Xu Cheng[1]
[1] Microprocessor Research and Development Center, Peking University, China
[2] Shenzhen Graduate School of Peking University, China
{xiayubin,yangchun,chengxu}@mprc.pku.edu.cn

Abstract

As virtualization technology is used widely in cloud computing, there are more and more interactive workloads being deployed on virtual machine (VM) environment. Although improving interactive performance has been heavily studied in operating system area, in consolidated VM environment, the improvements of guest OS are usually offset by the more coarse-grained VM scheduler, which may cause poor interactive performance. The guest OS scheduler and VM scheduler are totally independent with each other, which leads to the so called 'semantic gap'.

To reduce this semantic gap, this paper presents PaS (Preemption-aware Scheduling) as an extension of VM scheduling interface. PaS introduces only two interfaces: one to register VM preemption conditions, the other to check if a VM is preempting. Thanks to the sophisticated techniques of interactive-process identification and optimization in traditional OS, it is trivial for guest OS to use the new interfaces: only 10 lines of code are added into Linux-2.6.18.8. The evaluation results show that PaS can significantly improve the interactive performance of consolidated VMs while keeping the fairness and performance isolation.

1. Introduction

As virtualization technique has developed rapidly in terms of reliability, performance, and administration, it has been used widely in data center as a key enabling technology for cloud computing. Besides traditional server workloads, there are more and more interactive workloads being deployed within VMs (Virtual Machine). For example, VMware introduces VDI (Virtual Desktop Infrastructure), which hosts desktop applications in VMs and users can access them with thin-client devices (e.g. private cloud) [22].

Since VMs may accommodate different types of workloads simultaneously, it is hard for the VMM (Virtual Machine Monitor) to allocate resources due to the so called *semantic gap* [4] with guest OS. The semantic gap is difficult to eliminate because different OSes have different abstractions of resource and the VMM cannot get such semantic information directly through its narrow interfaces. The complexity of workloads and the semantic gap make VM scheduling a great challenge.

Meanwhile, sophisticated technologies have been developed in traditional OS to improve interactive performance. However, they failed to benefit the VMM due to the semantic gap. Actually, most of their effects are offset by the more coarse-grained VM scheduler. Therefore, the responsiveness may degrade significantly in a consolidated VM environment. In our experiment, a text editor running in one VM may have to wait for a compilation task in another VM, which leads to high latencies of user operations.

This paper presents a cooperative VM scheduling mechanism to improve the performance of interactive applications running in VM. PaS, namely preemption-aware scheduling, is an extension of the VM scheduling interface, which provides preemption as a service to VMs. The extension introduces only two interfaces. One to register preemption conditions, and the other to check if a VM is preempting. A guest VM registers for preemption conditions to get preempting at appropriate moments, and yields CPU after finishing interactive jobs while preempting. We have implemented PaS on Xen-3.2 platform [1] with Linux-2.6.18 as guest OS kernel. Thanks to the sophisticated techniques of interactive-process identification and optimization in Linux, it is trivial to use the new interface: only 10 lines of code are added to the Linux kernel. The evaluation results show that PaS can improve the responsiveness of VMs remarkably while keeping the fairness and performance isolation.

The rest of the paper is organized as follows. We discuss related work in section 2. Section 3 describes the design of PaS. Section 4 shows the implementation of PaS. Section 5 evaluates the performance improvements of the new scheduling interface. Finally, we present concluding remarks and future work in section 6.

2. Related Work

This section first presents previous research on interactive performance and VM scheduling, then describes Xen's architecture and its Credit scheduler, on which our research is based.

2.1. Interactive Performance

Improving interactive performance has been heavily studied in OS area. Researchers mainly focused on two problems: interactive tasks identification and optimization. In early days, interactive applications had a lot in common with I/O-bound tasks. The identification was mostly based on tasks' CPU usage pattern, including CPU consumption, effective quantum lengths, types of context switches (voluntary vs. forced) and so on[11]. However, modern interactive workloads such as graphical user interface, web browser and multimedia player, are increasingly resource-intensive. It is no longer enough to infer the class of a task just by its CPU usage pattern[11].

A lot of other ways had been developed, most of which operated on a heuristic basis. The schedulers in Solaris[18] and Windows[21], raised the priority of processes associated with a window that had input focus. Etsion et al.[10, 11] proposed a scheme for identifying interactive applications by directly quantifying I/O activity (e.g. keyboard, mouse, and screen) between applications and end user. They instrumented X server to gather such information and then deliver it to the OS scheduler. Similarly, Zheng et al.[26] identified interactive processes by monitoring accesses to I/O channels and inferring when user interactions occurred. They could also determine what other processes the interactive processes may depend upon in processing the interaction. However, the information was based on high-level abstractions defined by OSes, which is invisible to the low-level VMM.

Rather than identifying automatically, some systems needed application developers to specify certain parameters for the OS scheduler (e.g. time constraints, classifications, etc)[24, 15, 7]. Again, these optimizations are offset by the VM scheduler which is at a much more coarse-grained scheduling level.

2.2. VM Scheduling

Cherkasova et al. studied three I/O-intensive benchmarks under three different schedulers of Xen[6, 5], which were BVT scheduler (Borrowed Virtual Time), SEDF scheduler (Simple Earliest Deadline First), and Credit Scheduler. They focused on how CPU resource was distributed between the driver domain and guest domains.

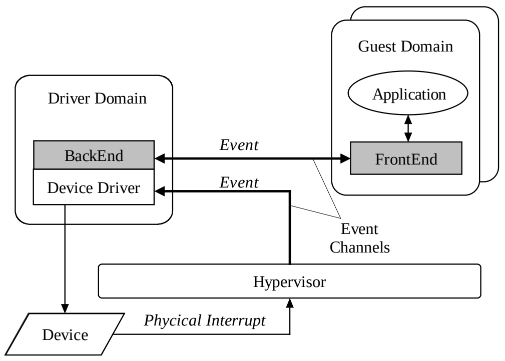


Figure 1. Architecture of Xen

Govindan et al.[13] made scheduler be aware of VMs' communication behavior and preferentially schedule I/O-intensive VMs. Their heuristic method, however, did not tackle the responsiveness of non-intensive interactive workloads. Ongaro et al.[19] studied the impact of VM scheduler on I/O performance by evaluating different configurations of schedulers. They concluded that a latency-sensitive workload has poor responsiveness if the workload is mixed with CPU-bound ones in the same domain.

Kim et al.[16] presented a way to exports priority in different OSes to VMM. However, as different OSes used different data structures and values to represent priority, it's non-trivial to do global scheduling across all the VMs. Their further research[17] presented a task-aware virtual machine scheduling mechanism based on inference techniques using gray-box knowledge. They identified I/O-bound tasks within VMs and gave them higher priority to achieve good I/O performance even in highly-consolidated environment. However, there are several limitations to get better interactive performance just by inferring. Details will be discussed in section3.

2.3. Xen and Credit Scheduler

We use Xen[1, 12] in our research and conduct the remaining discussion in its context. Figure 1 illustrates the organization of the Xen system. The hypervisor provides an abstraction of hardware to guest domains. It is responsible for CPU-scheduling and memory-allocating among guest domains. The driver domain is the only one that is privileged to access I/O devices directly with its device driver, and provides guest domains safe access to I/O devices. Each guest domain implements a front-end driver for I/O device, which cooperates with a back-end driver implemented in the driver domain. The two drivers communicate with each other through event channel and I/O-ring. The endpoints of event channel is represented by ports.

Xen uses Credit scheduler as its default scheduler. The

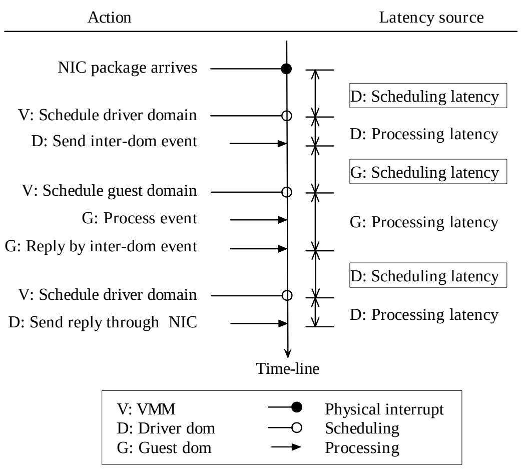


Figure 2. Request/reply latency components

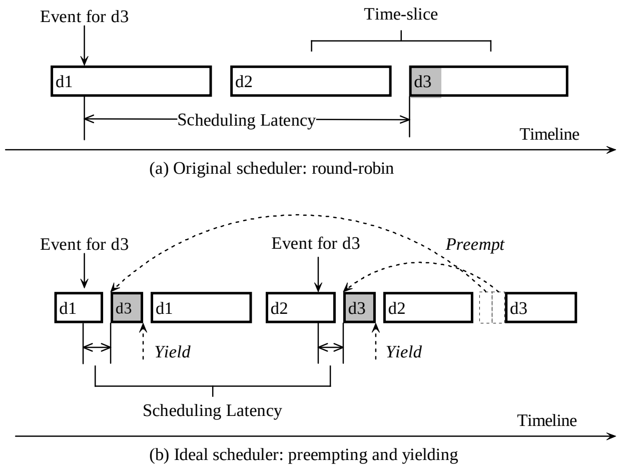


Figure 3. Scheduling behavior and latency

Credit scheduler regards a time quantum as credit, which is determined by the defined weight for each domain. Every 10 ms, the credit of a running VCPU (Virtual CPU) is debited by 100; every 30 ms, all active VCPUs are given the credit calculated by the weight of their domain. A VCPU's credit is used to determine its priority. If a VCPU's credit is equal to or larger than 0, its priority is UNDER (-1). Otherwise it priority is set to OVER (-2). All VCPUs with the higher priority are scheduled before those with lower priority. VCPUs are put in a run queue with priority decreasing. The scheduler picks the first one of the run queue as a next VCPU. A VCPU receives the time slice of 30 ms and runs consuming its credit once it is scheduled

The credit scheduler uses a *boosting* mechanism to improve the performance of I/O-bound domains by allowing a VCPU to preempt a running one. When an event arrives and the target VCPU is blocked, the VMM wakes up the VCPU and inserts it into the run queue. Since the VCPU has to waits all the preceding VCPUs, it may suffer long latency before the event is delivered. To improve responsiveness, the credit scheduler assigns a woken VCPU the highest priority BOOST (0) if its original priority is UNDER, and triggers rescheduling immediately. Therefore, the woken VCPU can preempt the current running VCPU and thus achieve low latency of event delivery.

3. PaS: Preemption-aware Scheduling

Figure 2 shows the components of latency within one typical network request/reply process. In a typical interactive process, an input of user is finally transformed into one or more events. The interval between event pending nd event delivering is defined as *scheduling latency*, which is decided predominantly by the CPU scheduler. In practice, as the time-slice of a VCPU is on the order of 10ms, the scheduling-latency usually dominates the total latency of event processing in consolidated VM environment.

Figure 3 shows scheduling latency of events under different schedulers. There are three domains in figure 3(a). An event for domain-3 arrives when domain-1 is running. When domain-3 is scheduled, it is able to make the event processing task (the shadow part) run immediately at the beginning of its time-slice. But before that, it has to wait for domain-1 and domain-2 to finish their time-slice, which can lead to long scheduling latency.

Figure 3(b) shows the ideal scheduling behavior in such condition. When an event arrivals, the domain-3 should get a chance to process the event as soon as possible, and yield CPU right after it finishes. We call it *preempt and yield*. Three factors are needed to be considered: the time to preempt, the time to yield, and fairness. There are many ways to achieve the ideal scheduling behavior, which can be classified to two categories: *implicit ones* which are usually based on inference, and *explicit ones* which depend on cooperation between the hypervisor and guest OS. Following subsections present the advantages and drawbacks of both categories.

3.1. Implicit Inference

Some researchers tried to infer task-information by monitoring CPU usage pattern in VMM to improve I/O performance, e.g. Kim et,al.[17]. They identified tasks by CR3 register on x86 CPU and classified them to be I/O-bound or CPU-bound by their time-slice. Once a virtual interrupt arrives, they made the target domain preempt to process the event. After that, when the domain scheduled a task which was not considered to be I/O-bound, the hypervisor revoked CPU from the domain and scheduled others to run. Several parameters can be adjusted to keep fairness.

As all the three factors (preempt time, yield time and fairness) are all considered by the hypervisor itself, the guest OSes are not aware of such optimization and thus need no modifications. However, there are also several limitations of implicit inference:

- CPU usage is not enough. As mentioned in section2.1, modern interactive applications are no longer just I/O-bound. Other types of information are used in OS to infer interactive processes, such as focus window, user input and screen output (e.g. by instrumenting X server). Such information is highly OS-dependent and is not visible in VMM. Therefore, identifying interactive process just in VMM is much harder than in OS.
- Re-invent the wheel. As most operating systems have mechanism to recognize and optimize interactive processes, there will be a lot of duplicate work if VM scheduler does the same thing again. Instead, making full use of the existing sophisticated techniques is needed.
- Lack of thread information. As on x86, CR3 indicates the page directory of a virtual address space, and VMM can identify a process by monitoring CR3 changing. However, in Linux, thread switching doesn't need address space switching. Therefore, VMM is not aware of different threads.

These limitations make implicit inference only capable of improving performance of simple I/O-bound tasks, instead of modern interactive applications in consolidated VM environment.

3.2. Explicit Cooperation

Instead of doing all jobs in the hypervisor, explicit cooperation depends on both the hypervisor and the guest OS. New interface is needed for communication between the two. As different guest OSes have different build-in abstractions, it's ineffective to let the hypervisor make scheduling decisions of preemption. Instead, the decisions should be made majorally by the guest OS to make full use of the technologies of interactive performance optimization existing already. There are two principles that the new interface should follow:

- The interface should provide complete information. The guest OS should be able to use the information to optimize interactive tasks, just as it does natively.
- The interface should be minimal. Irrelevant information should not be provided through the interface. Meanwhile, the VMM should keep as simple as possible to ensure the security and efficiency.

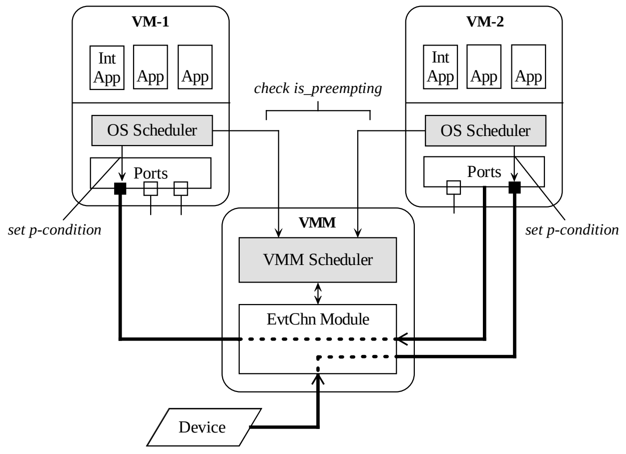


Figure 4. PaS interface overview

With these principles in mind, we have designed PaS, a preemption-aware scheduling interface.

3.3. PaS Interface

PaS provides preemption as a service to guest OS by introducing only two new interfaces:

- ***Interface 1: p-condition***. P-condition is short for 'preemption condition', which is used for VM scheduler to decide when to make a VM preempt. Guest OS can register p-conditions that the hypervisor provides. Once a registed p-condition is met, the coresponding VM will get a chance to preempt the current VM to do some process.
- ***Interface 2: is-preempting***. This interface is to check whether the current VM is in preempting-state. Guest OS can chose different tasks to run depending on its running state.

Figure 4 shows the overview of PaS architecture briefly. On Xen, event pending is used as p-conditions. First, a guest OS sets several ports of event channels as p-conditions. When an event is pending at such ports, either from Xen or from another VM, the VM scheduler makes the target VM preempt. The guest kernel scheduler only allows interactive processes to run when it is preempting, by yielding CPU after all the time-critical jobs are done.

As a guest OS is not trusted, it may have malicious behavior such as preempting other guest OS for a long time and never yield CPU. PaS solves this problem by controlling the ability of guest VM preemption. The preemption period within one time-slice is fixed for one VM, determined by *p-ratio*, which can be set by the administrator.

$$p\text{-}ratio = \frac{preemption\ period\ within\ per\ time\text{-}slice}{one\ time\text{-}slice}$$

Once a VM has consumed all its preemption period, it has to wait for its next time-slice. As a result, the fairness is kept and the interference of malicious VM is bounded. Experiment results in section5.3 show the isolation of performance is guaranteed. From the guest OS's perspective, as its total preemption period is fixed, the best it can do is to make use of its preemption period appropriately to let as much as its time-critical process get CPU timely.

4. Implementation

This section describes the implementation of PaS interface based on Xen-3.3.1 and Linux-2.6.18.8. The hypervisor is responsible for domain preemption and accounting. The guest kernel schedules the interactive tasks when preempting, and yield CPU after they are done.

4.1. Full-time Preemption

In the original Credit scheduler, a VCPU can only preempt at wakeup time through boosting. When an event is pending and the target VCPU is waiting in run queue, no preemption will occur. This will lead to long scheduling latency especially when there are several VCPUs running concurrently. Based on event channel mechanism of Xen, we extend the original preemption mechanism to *full-time preemption* to improve responsiveness, so that a VCPU can preempt even when it is waiting in the run queue.

Besides the original run-queue, we add another queue, namely *p-queue*, short for *preemption-queue*, for those preempting VCPUs. There are two ways for a VCPU to enter the p-queue: wakeup boost, and wait-time preemption. Wakeup boost is as mentioned before, and wait-time preemption is triggered when a p-condition is met. The VCPUs in p-queue is scheduled in a FIFO mode. Once scheduling, the VM scheduler selects the first one in p-queue if it is not empty, otherwise it will select from the original run-queue. We add p-queue so that when a VCPU get preempting, its position in run-queue is reserved, in order to keep the preemption's side-effect minimal.

We also introduce *p-credit* to track the preempting period of VCPU. When a VCPU is preempting, it uses its p-credit. We refer to the original credit as *n-credit* (normal credit). The percent of p-credit in its total credit is decided by p-ratio. For example, if the p-ratio of one VCPU is 50%, then half of its incoming credit will be added to p-credit, and half to n-credit.

In order to track preemption time precisely, we make the VMM account actually consumed CPU time to each VCPU. The actual CPU time is acquired by time stamp counter (TSC) of IA-32. The relation of runtime and credit is as follows:

- If a VCPU was preempting, then the consumed credit will be deducted from its p-credit.
- Otherwise, the consumed credit will be deducted first from its n-credit, then from its b-credit. The remaining will be deducted from both p-credit and n-credit, by p-ratio and (1 - p-ratio) respectively.

Moreover, when a VCPU is going to run, its time-slice is calculated by its credit remained. When a VCPU is preempting, its time-slice depends on its p-credit, the more the longer; otherwise, its time-slice depends on the sum of its p-credit and n-credit. Specifically, if a VCPU is out of p-credit, it will get no chance to preempt.

4.2. Guest OS Modification

The PaS interface is implemented by adding two fields in *shared_info* structure which is shared by the hypervisor and guest OS, instead of adding hypercalls. We use Linux kernel-2.6.18.8 for both the driver domain and guest domains. The guest OS is responsible for two things: setting the p-conditions which is mapping to ports of event channel, and yielding when finishes interactive jobs. When the guest VM is in preempting state, it only allows interactive process to run. The interface is defined as follows:

```
/* PaS interface */
struct shared_info {
  ...
  uint32 evtchn_preempt[sizeof(uint32) * 8];
  uint32 is_preempting; /* up to 32 VCPUs */
}
```

The guest OS uses following code to register p-conditions:

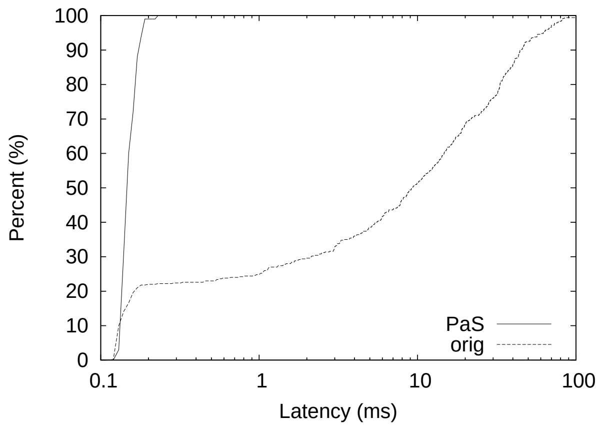
```
/* Register preemption-condition */
synch_set_bit(evtchn,
    HYPERVISOR_shared_info->evtchn_preempt);
```

Before returning to user space, we check whether the current process is considered to be interactive. If not, and the VCPU is preempting, the guest kernel will call the yield hypercall to give up CPU.

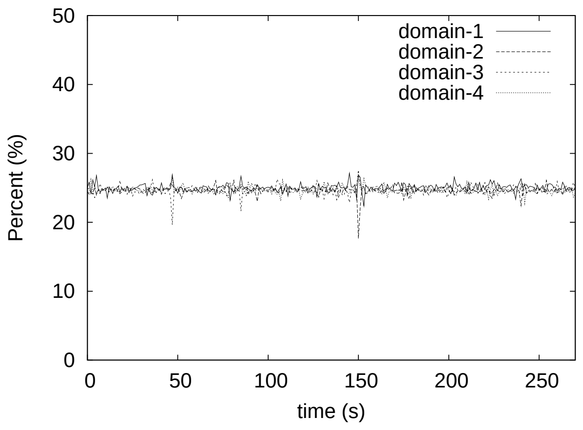
```
/* in kernel/sched.c */
void check_and_yield(void) {
  if ((HYPERVISOR_shared_info->is_preempting
        & (1 << smp_processor_id()))
      && !TASK_INTERACTIVE(current))
    HYPERVISOR_yield();
}

/* in arch/i386/kernel/entry-xen.S */
restore_all:
  call check_and_yield
```

The TASK_INTERACTIVE macro is used to check whether a process is interactive. It is part of the Linux



(a) CDF of udping latency



(b) CPU usage of different VMs

Figure 5. Micro-benchmark result

scheduler, which takes consideration of a process' static priority, CPU usage, sleep bonus, and using some experienced constants. The macro is effective in interactive process identification, which makes our implementation simple and useful.

5. Evaluation

Our prototype is installed on a 2.0 GHz AMD Opteron CPU, equipped with 4 GB RAM. We make our system run on a single physical core. A network client runs on a separate physical machine, an Intel Pentium 4 processor 2.60 GHz with 1 GB RAM; this machine is connected to the evaluated machine through an 100 Mbps Ethernet switch. Each guest VM is configure to 1 VCPU and 128MB memory, while the driver domain has the remaining. The default p-ratio is set to 50%.

5.1. Micro-benchmark Workload

We demonstrate the improvement of interactive performance on a consolidated machine with our mechanism in terms of responsiveness. To show the improvement for the worst case consolidation scenario, we concurrently run four CPU-bound domains with one domain to be evaluated. The evaluated domain contains both interactive and CPU-bound workloads so that the original scheduler does not identify the interactive task. We use domain0 as the driver domain.

Figure 5(a) shows the response time of a simple interactive workload. One domain runs a UDP echo server (namely udping) with a CPU-bound task, and a remote client repeatedly requests a small network packet (containing 'helloworld' string) to this server with random thinking time between 100 ms and 1000 ms. As shown in the CDF graph, our mechanism significantly improves the response time compared to the normal case. Figure 5(b) shows CPU usage for each domain during the experiment. This result demonstrates that our mechanism guarantees the CPU fairness as well.

We evaluate our system in case where multiple domains have different workloads, which consists of three mixed domains (CPU and I/O-bound) and three I/O-bound domains. Six clients conduct requests and responses with the think time between 10 ms and 1000 ms. Table5.1 shows that the domains including CPU- and I/O-bound tasks have much lower responsiveness than I/O-bound domains under the original Credit scheduler. Our mechanism improves the poor responsiveness of the mixed domains even better than the one of I/O-bound domains under the Credit scheduler.

Latency	CPU + I/O			I/O		
avg.(ms)	dom1	dom2	dom3	dom4	dom5	dom6
Credit	15.59	15.16	15.77	2.34	2.56	2.51
PaS	0.52	0.50	0.50	0.51	0.51	0.52

Table 1. Latency with different workloads

5.2. Realistic Workload

We evaluate the PaS interface over realistic workloads for a consolidated platform using VNC-IPA[23], namely VNC-based Interactive Performance Analyzer. Similar with VNCPlay[25], VNC-IPA is a record & replay tool based on VNC, which is used to evaluate the interactive performance by measuring the latency between user input and corresponding screen output.

As with other experiments, we concurrently run four CPU-bound domains with the p-ratio setting to 50%. One of the four domains uses Firefox to open a Google image-

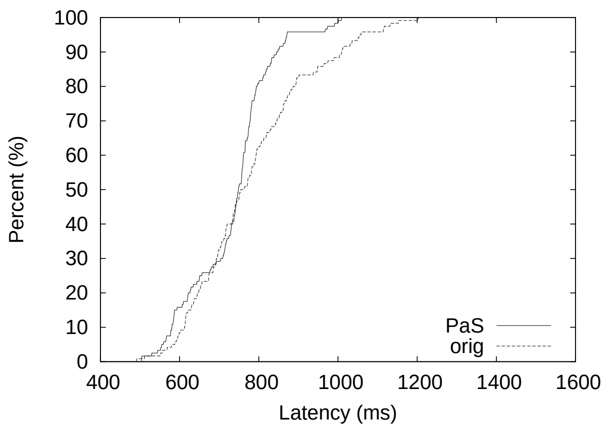


Figure 6. Web browsing latency

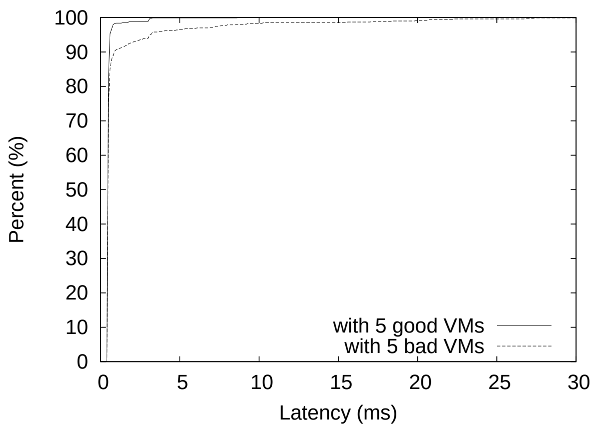


Figure 7. Performance isolation

search page which is full of images. The scenario is to page-up and page-down the browsing page, and the latencies between key pressing and screen updating are logged. We use VNC-IPA to replay this scenario for 200 times, and the distribution of latency is shown in figure 6. The max latency is reduced from 1.23s to 1.01s (by 17.9%), while the average top 5% latency is reduced from 1.13s to 0.97s (by 14.2%).

5.3. Performance Isolation

There exists possibility that malicious guest OSes may set all the preemption conditions and run for a long period each time it preempts others. Actually, in Pas, as the malicious behavior will cost p-credit of a VCPU quickly, it will not affect other VCPUs significantly. We present the latency of one VM running mixed-workload with 5 malicious VMs in figure 7. The p-ratio of each 'bad' VM is set to 100% to maximize its ability of preemption. 6 clients run udping concurrently with thinking time between 0.1 and 1ms. We also run the same test with 5 good behavior VMs, which will yield right after their interacitve tasks finish.

As the figure shows, the interactive performance of a VM with bad VMs is only slightly lower than the one with good VMs. It proves that the effects of malicious guest OS is limited. Moreover, the administrator can restrict the preemption capability of a VM by tuning its p-ratio.

6. Conclusion and Future Work

Although significant achievements have been made in traditional OS scheduler to improve interactive performance, most of their effects are offset by the VM scheduler on virtualization platform due to the semantic gap between the VM scheduler and guest kernel scheduler.

This paper presents PaS (preemption-aware scheduling), which is an extension of VM scheduling interface. This extension introduces only two interfaces: one to register preemption conditions, and the other to check if the VM itself is preempting. The guest kernel uses these two interfaces to preempt at the right time, and yield CPU after finishing time-critical jobs when preempting. The preempting period is tracked and controlled by the VMM to minimize the impacts of malicious guest OSes. We have implemented PaS based on Xen-3.3.1 and Linux-2.6.18. Thanks to the sophisticated techniques of interactive-process identification and optimization in Linux, it is trivial for guest kernel to use the new interface and only 10 lines of code are added. We use micro-benchmark as well as realistic workload for evaluation. The results show that PaS can significantly improve the interactive performance of consolidated VM while keeping the fairness and performance isolation.

Currently, PaS is only implemented on uni-processor platform. We will extend it to multi-core/multi-processor platform in the future. Our current implementation just uses the existing TASK_INTERACTIVE macro as the only indication to decide whether to yield CPU when preempting. More accurate yielding policy is needed. We also plan to measure the performance of more CPU-intensive interactive tasks, such as MP3-decoding and complex web page browsing, to evaluate the benefit of PaS.

References

[1] 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 19th Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[2] L. Bin, A. D. Peter, and L. Dong. User-driven scheduling of interactive virtual machines. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, 2004.

[3] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel (3rd Edition)*. O'Reilly, 3 edition, 2005.

[4] P. M. Chen and B. D. Noble. When virtual is better than real. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS'01)*, 2001.

[5] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *ACM SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[6] L. Cherkasova, D. Gupta, and A. Vahdat. When virtual is harder than real: Resource allocation challenges in virtual machine based it environments. Technical report, February 8 2007.

[7] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (bvt) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. In *Proceedings of the 17th Symposium on Operating Systems Principles (SOSP'99)*, 1999.

[8] Y. Endo and M. I. Seltzer. Improving interactive performance using tipme. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS'00)*, 2000.

[9] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation (OSDI'96)*, 1996.

[10] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Human-centered scheduling of interactive and multimedia applications on a loaded desktop. Technical report, The Hebrew University, Mar 2003.

[11] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Desktop scheduling: How can we know what the user wants? In *Proceedings of the 14th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'04)*, 2004.

[12] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the xen virtual machine monitor. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OA-SIS'04)*, 2004.

[13] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: Communication-aware cpu scheduling for consolidated xen-based hosting platforms. In *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*, 2007.

[14] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06)*, 2006.

[15] M. S. L. Jason Nieh. The design, implementation and evaluation of smart: A scheduler for multimedia applications. In *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP'97)*, 1997.

[16] D. Kim, H. Kim, M. Jeon, E. Seo, and J. Lee. Guest-aware priority-based virtual machine scheduling for highly consolidated server. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing (EuroPar'08)*, pages 285–294, 2008.

[17] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee. Task-aware virtual machine scheduling for i/o performance. In *Proceedings of the 5th International Conference on Virtual Execution Environments (VEE'09)*, 2009.

[18] J. Mauro and R. McDougall. *Solaris Internals*. Prentice Hall, 2001.

[19] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *Proceedings of the 4th International Conference on Virtual Execution Environments (VEE'08)*, 2008.

[20] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of slim: A stateless, thin-client architecture. In *Proceedings of the 17th Symposium on Operating Systems Principles (SOSP'99)*, 1999.

[21] D. A. Solomon and M. E. Russinovich. *Inside Microsoft Windows 2000*. Microsoft Press, 2000.

[22] VMware. Vdi: A new desktop strategy. In *VMware White-Paper*, 2006.

[23] C. Yang, Y. Niu, Y. Xia, and X. Cheng. Performance analysis of interactive desktop applications in virtual machine environment. *The Chinese Journal of Electronics (CJE)*, pages 242–246, 2008.

[24] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, and J. E. B. Moss. Redline: First class support for interactivity in commodity operating systems. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI'08)*, 2008.

[25] N. Zeldovich and R. Chandra. Interactive performance measurement with vncplay. In *Proceedings of the 2005 USENIX Annual Technical Conference, USENIX'05*, 2005.

[26] H. Zheng and J. Nieh. Automatic user interaction detection and scheduling with rsio. Technical report, Department of Computer Science, Columbia University, May 2008.