FEAS: A full-time event aware scheduler for improving responsiveness of virtual machines

Denghui Liu, Jinli Cao
Department of Computer Science & Computer Engineering, Engineering & Mathematical Sciences
La Trobe University, Melbourne Victoria 3086, Australia
d8liu@students.latrobe.edu.au,
j.cao@latrobe.edu.au

Jie Cao
Jiangsu Provincial Key Laboratory of E-Business,
Nanjing University of Finance and Economics
Nanjing, China
caojie690929@163.com

Abstract

Due to the advances in software and hardware support for virtualisation, virtualisation technology has been adapted for server consolidation and desktop virtualisation to save on capital and operating costs. The basic abstraction layer of software that virtualises hardware resources and manages the execution of virtual machines is called virtual machine monitor (VMM). A critical part of VMM is the CPU scheduler which slices and dispatches physical CPU time to virtual machines. Xen's credit scheduler utilised blocked-to-boosted mechanism to achieve low latency on I/O intensive tasks. However, it suppresses event notifications for the guest domain that is not blocked. This may delays the response of a guest domain doing mixed workloads, as its virtual CPU is seldom blocked when processing CPU-intensive tasks. We enhance the credit scheduler by making it full-time aware of inter-domain events and physical interrupt request events. Our proposed scheduler not only improves the responsiveness of domains doing mixed workloads, but also minimises the possibly caused scheduling unfairness. The experimental evaluation demonstrates the benefits of our proposed scheduler.

Keywords: Virtual machine, Xen, Paravirtualization.

1 Introduction

Virtualisation technology involves the virtualisation of several critical parts of a computer, such as CPU, memory, network and storage. It partitions the underlying physical resources and makes them shared among multiple virtual machines (VMs) (or domains) either by assigning a portion of physical resources to each VM (e.g. hard disk) or by switching from one VM to another in a very short time frame to use the physical resources in turns (e.g. CPU). These VMs run in parallel on a single physical machine under the control of virtual machine monitor (VMM) and they can have different operating systems.

Virtualisation technology opens up the possibility of server consolidation which increases the efficient use of server resources by consolidating multiple servers running different operation systems onto a single physical server. Desktop virtualization is another major application of the virtualisation technology, in which case users only need a thin client to display the desktop interface locally while have all backend processing done in a dedicated VM that resides remotely in the Cloud. Both situations involve the execution of CPU intensive tasks and I/O intensive tasks, and often need to process a mix of both kinds at one time. The complexity of workloads makes it a great challenge for the VMM scheduler to maximise throughput and minimise latency while ensuring fairness.

This paper is based on the observation of Xen 4.0.1 (Xen 2011) platform. Its default scheduler, named credit scheduler, employs the BOOST mechanism to achieve low I/O response latency which works reasonably well when VMs have relatively monotonous workloads. The schedulable entities of a VM are the virtual CPUs (VCPUs) it has. The priority of an idle VCPU is boosted to get an immediate execution when it receives an event. This allows VMs performing I/O tasks to achieve lower response latency. However, the responsiveness of a VM diverges if it also does CPU intensive tasks at the same time. A VCPU waiting in the run queue does not get properly boosted when it receives an incoming event. The event notification is suppressed and thus has no effect on the scheduling. This might make the event sender wait unnecessarily and delay the following jobs.

An enhanced version of credit scheduler is presented in this paper to improve the responsiveness of busy VMs by taking advantage of Xen's split driver model and even channels. The device driver in Xen is split into two portions. Domain 0 or a dedicated driver domain hosts the front portion that directly interacts with the device, and the other portion resides in unprivileged guest domains. These two parts notify each other of waiting data using the Xen event channel mechanism and exchanged data via the I/O ring mechanism. The proposed scheduler monitors the events sent across VMM and boosts the runnable VCPUs receiving events originating from another domain or physical interrupt requests (PIRQs). To complement credit scheduler, the proposed scheduler prioritises not only blocked VCPUs but also runnable ones, and is called full-time event aware scheduler (FEAS). A VM processing mixed workloads can greatly benefits from prompt scheduling upon receiving an incoming event, particularly if it is an I/O related event.

The rest of this paper is organized as follows. Section 2 discusses previous research on VM scheduling and relates the virtual-machine monitor Xen. Section 3 presents the design of our proposed scheduler for Full-time event aware scheduling. Some experimental tests have been conducted to verify/demonstrate the performance

Copyright © 2012, Australian Computer Society, Inc. This paper appeared at the 35th Australasian Computer Science Conference (ACSC 2012), Melbourne, Australia, January-February 2012. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 122. M. Reynolds and B. Thomas, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

improvements and the analysis of results has also been included in Section 4. Finally, the conclusions and future work are presented in Section 5.

2 Related work

This section firstly discusses previous research on VM scheduling, then describes the architecture of Xen's split driver model and its credit scheduler.

2.1 VM scheduling

Three different VM schedulers have been introduced over the course of Xen's history, which are Borrowed Virtual Time (BVT) scheduler, Simple Earliest Deadline First (SEDF) scheduler, and Credit Scheduler. All these three are Proportional Share schedulers which allocate CPU in proportion to the VMs' weight. Cherkasova et al. (2007) comprehensively analysed and compared the impacts of schedulers and their respective scheduler parameters on the performance of I/O intensive applications running on virtual machines.

Ongaro et al. (2008) study the impact of the credit scheduler with various configurations on the performances of guest domains concurrently running a mixed workload of processor-intensive, bandwidth-intensive, and latency-sensitive applications. They suggest in their work that latency-sensitive applications should be placed in their own VMs to achieve the best performance. The purpose of our paper is to address this problem.

Govindan et al. (2009)'s communication-aware scheduler monitors the I/O ring and preferentially schedules the VMs that receive more data packets or are anticipated to send more data packets. However, the scheduler relies on accumulating the number of packets received or sent over a certain period of time and does not provide the immediate response to an incoming event.

Kim et al. (2009) made scheduler task aware by using the gray-box knowledge. The scheduler infers the guest-level I/O tasks by identifying the tasks using the CR3 register and then monitoring their time slices. A task is considered to be an I/O task based on two grey-box criteria: it immediately pre-empts the running task if the guest VM receives an event and its time slice is short. However, classifying the tasks just based on the CPU usage is not enough (Xia et al., 2009).

Xia et al. (2009) propose a pre-emption aware scheduling (PaS) interface. Same with our scheduler, PaS also improves the responsiveness of busy VMs by allowing the VCPU to pre-empt when an event is pending while it is waiting in the run queue. But in that approach the event channels on which the pre-empting condition is based need to be pre-known and registered to the guest kernel.

2.2 Xen and Split driver model

Xen 4.0.1 is used in our research. Xen adopts the paravirtualization approach and its guest operating systems require modifications to be able to run on the Xen platform. The Xen hypervisor sits between the hardware and the co-existing virtual machines. It has full control over hardware resources and dispatches them to different VMs according to a set of predefined rules.

Device drivers are the essential software for any operating system to communicate with physical hardware. Xen's split driver model divides the device driver into two portions, the front end and the back end (Figure 1). The back end handles the physical device and the front end acts as the proxy of the back end. The back end is typically in Domain 0 but sometimes in a dedicated driver domain. Unprivileged guest domains have the front end with which they can accomplish a network or disk request.

The two portions of device driver notify each other of critical events using event channels and pass messages using I/O ring buffers. Event channel is the primitive notification mechanism within Xen. When the remote domain is busy or yet to be scheduled, an event is asynchronously delivered from its source to it to indicate the relevant event on the source domain. The ring buffers are implemented in the shared memory pages shared by both driver ends. Front end and back end exchange data by sending over the memory addresses of data pages rather than doing a full copy. This zero-copy feature enables fast message passing and consequently fast I/O.

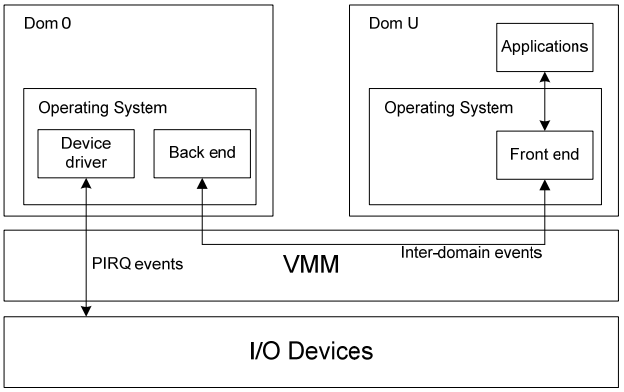

Figure 1: Xen split driver model

Based on the source of events, there are four types of events sent over the event channel. They are physical interrupt request (PIRQ) events, virtual IRQ (VIRQ) events, inter-domain events and intra-domain events. PIRQ events are mapped to the real IRQs of various physical devices. As an incoming IRQ generated by a device is likely not for the currently running domain, its corresponding PIRQ event is enqueued on the target domain and then processed when the domain is scheduled. Only privileged domains, such as domain 0 and driver domains, can handle PIRQ events. VIRQs are related to virtual devices created by Xen, like the timer virtual device. The main use of inter-domain events is for the front end and the back end of paravirtualised devices to notify each other of waiting data. Intra-domain events are a special case of inter-domain events where the events are delivered between the VCPUs of a single domain.

2.3 Credit scheduler

The current default scheduler of Xen is the credit scheduler. It allocates fair shares of processor resources to guest domains. Each slice of physical CPU time is weighted by a certain number of credits. Thus, if domains receive the same number of credits, they should expect an equal amount of CPU time.

Each VCPU's state and credits are managed and scheduled separately. The VCPU's credit balance can be positive or negative, and correspondingly its state can be

under or *over*. VCPUs trade credits for CPU time. A tick interrupt is triggered every 10ms. At each tick event, the currently running domain is debited some credits for the period it has run. An accounting event occurs every 30ms. During the accounting process, VCPUs are recharged with credits proportional to their weights. Their states are adjusted accordingly. The *under* state is assigned for VCPUs with positive credit balance while the *over* state for the ones with negative credit balance. To fully utilise available CPU resources, the accounting process caps VCPU's credits at an amount that is worth one rotation's CPU time slice. If a VCPU's accumulating credits exceeds the cap, it is marked inactive and will not receive any more credits until it is active again. Its credits are forfeited and shared by other active VCPUs. A scheduling event occurs when a scheduling decision is needed, which triggers a function that firstly refreshes the current VCPU's credit balance based on how long it has run and then decides the next VCPU to be scheduled. The order of scheduling VCPUs is based on their priorities. VCPUs with the *under* priority are always run before those with the *over* priority. VCPUs with same priority are scheduled in a round robin manner. The scheduled VCPU is allowed to run for 30ms or until pre-empted by other VCPUs with higher priority whichever comes first. If a running VCPU runs out of credits during its scheduled interval, it will not spontaneously yield the CPU. Contrarily, it will continue running and its credit simply goes negative.

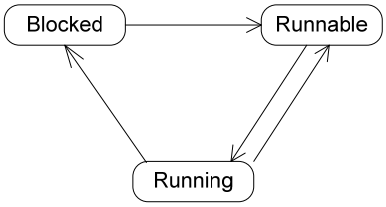

Figure 2: Run state transitions

A VCPU in Xen could be in one of the following four possible run states, *running*, *runnable*, *blocked* and *offline*. The VCPU that is currently running on a physical CPU is in the *running* state. Since multiple VCPUs share a limited number of CPUs, VCPUs might not be scheduled on any physical CPU as soon as they become runnable. These VCPUs in the *runnable* state are essentially waiting in a queue for their turn. In a conventional system an idle thread occupies the physical CPU when there are no jobs to do. A virtualised system avoids such a waste by letting other busy VCPUs have the unused CPU time when some are idle. Idle VCPUs are given a *blocked* state. An *offline* VCPU is neither runnable nor blocked. Typically it is paused by the administrator. Figure 2 depicts the transition of the states that are tightly related to scheduling.

When an event comes, credit scheduler wakes a blocked VCPU and puts it back to the run queue. Furthermore, if the waken VCPU is in *under* priority, its priority is promoted to *boost*. So it is high likely that the boosted VCPU pre-empts the running one and gets scheduled immediately. This function is carried out by the boost module which lowers the latency of I/O related tasks. However, this only benefits blocked VCPUs with positive credits. A runnable VCPU receiving an I/O related event cannot be promptly scheduled. This often happens with the domains doing mixed workloads of CPU intensive tasks and I/O intensive tasks.

3 Full-time event aware scheduling

This section firstly identifies the problems that cause long latency of I/O tasks. Then the full-time event aware scheduler is proposed and detailed.

3.1 Scheduling delays

Figure 3 shows the typical delays happening within a disk reading process. Delay D1 and D2 are associated with the scheduling of Domain 0. D1 is the duration between when the Domain U sends a reading request to Domain 0 and when Domain 0 is scheduled to actually send the reading IRQ to the hard disk. D2 is the duration between when the hard disk notifies Domain 0 the data to be retrieved is ready and when Domain 0 gets scheduled to set up the I/O ring and then notifies the Domain U. D3 happens on the Domain U side. It is the duration between when Domain 0 sends the notification and when Domain U is scheduled to finish the data reading process. A data writing process is similar.

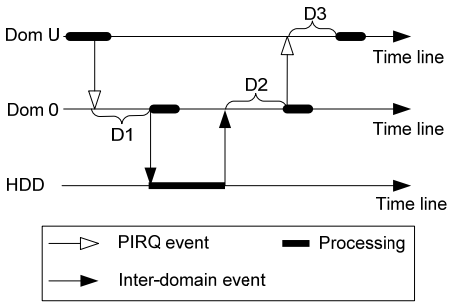

Figure 3: Scheduling delays within a disk reading process

All aforementioned scheduling delays can be reduced by scheduling the events' respective target domains soon after the events are received. The events of interest are either inter-domain events or PIRQ events (Chisnall, 2007). The current credit scheduler boosts the VCPU upon an incoming event call only when the VCPU is blocked and has not consumed more than its fair share of CPU time. In other cases, event calls get suppressed and have no effect on the scheduling. Our enhanced version of credit scheduler makes the scheduler always aware of event notifications, in other words, the scheduler is full-time event aware. Once detecting an eligible incoming event the scheduler changes the course of scheduling accordingly.

3.2 FEAS design

In FEAS every physical CPU has two additional queues: *immediate queue* and *postponed queue*, complementing the original run queue (Refer to Figure 4). VCPUs on the immediate queue are always preferentially scheduled over those on the run queue. The postponed queue is the temporary depository of the VCPUs that need to be scheduled as soon as possible but they have been scheduled more often than allowed. Once a designated trigger fires, postponed queue is swapped with immediate queue. In other words, postponed queue becomes immediate queue and the previous immediate queue

becomes the new postponed queue. Therefore, postponed VCPUs can get preferential scheduling after the trigger fires. The VCPU retains its position in run queue when joining or leaving immediate queue or postponed queue. VCPUs on the run queue are executed in round robin fashion. In such way the time slice VCPUs receive each rotation can be tightly controlled, and the side-effects caused on scheduling fairness by frequent pre-emption are kept minimal.

In general, similar with credit scheduler, FEAS also consists of three parts, namely *en-queuing*, *queue processing* and *de-queuing*. Figure 4 depicts the structure of FEAS. VCPUs that need to run are en-queued. Only runnable VCPUs can join queues and wait to be executed by CPU. Thus, *blocked* VCPUs that receive incoming events are set to be *runnable* beforehand. This process is called being *waken*. Credit scheduler boosts waken VCPUs by given them a *boost* priority to lower response latency because incoming events are often I/O related. FEAS prioritises VCPUs differently. It boosts VCPUs by assigning them to immediate queue or postponed queue (Explained in Sec. 3.2.2). Queue processing takes on heavy workloads off the de-queuing process since de-queuing process is in the critical path and meant to be fast. After queue processing, the de-queuing process usually de-queues the head VCPU in a queue straightaway and scheduled it to run next.

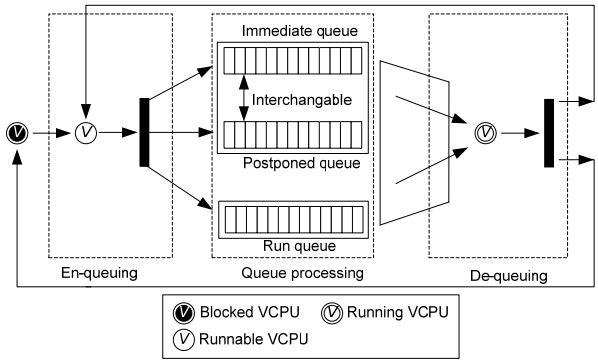

Figure 4: FEAS structure

The proposed scheduler involves three modules complementing the original credit scheduler.

3.2.1 New scheduling decision trigger

In credit scheduler a scheduling decision is made when a VCPU blocks, yields, completes its time slice, or is awaken (Xen wiki, 2007). A runnable VCPU with an incoming event call is not prioritised properly. It cannot process the event until it crawls to the head position of run queue. This may delay the processing of some events that are related to latency sensitive tasks. The proposed scheduler captures the event notifications for runnable VCPUs and then tickles the scheduler for a new scheduling decision. When scheduler is tickled, if the currently running VCPU is rather than prioritised, it is pre-empted and a new VCPU is selected to run next. Since the runnable VCPU receiving an event is prioritised, it is very likely for it to get an immediate execution.

FEAS is configured to react merely on inter-domain events and PIRQ events received merely by runnable VCPUs. As such an event may also wake and boost a blocked VCPU with *under* priority, a *waken* flag is set if the VCPU is woken. The scheduler only promotes runnable VCPUs whose *waken* flag is off.

3.2.2 Interchangeable immediate queue and postponed queue

FEAS prioritises VCPUs doing I/O tasks no matter whether the VCPU is blocked and whether its priority is under or over. However, if there is no constraint, an I/O intensive domain can hold the PCPU for an unfair amount of time by taking advantage of this preference. So two new queues: immediate queue and postponed queue, are introduced to limit the frequency of VCPUs getting scheduled. A limit on the number of times a VCPU can get scheduled within a counting cycle is enforced on every VCPU. Let n_{limit} denote the upper limit of the number of scheduling times, c denote the cth counting cycle and $n_{i,c}$ denote the scheduling times of VCPU i during the cth counting cycle. A trigger which is usually a timer indicates the start of a new counting cycle. n_{limit} is a constant over all counting cycles and $n_{i,c}$ is initialised to 0 at the start of every counting cycle. VCPUs join immediate queue or postponed queue following the two rules below. Note that FEAS prioritises runnable VCPUs even if they have negative credits, so that, VCPUs can achieve optimal responsiveness.

$$\begin{cases} \text{If } n_{i,c} < n_{limit}\text{, then VCPU } i \text{ joins the immediate queue} \\ \text{If } n_{i,c} >= n_{limit}\text{, then VCPU } i \text{ joins the postponed queue} \end{cases}$$

On every scheduling decision, the scheduler always preferentially schedules the VCPUs in the immediate queue. If the immediate queue is empty, the VCPUs in the run queue are executed in the decreasing order of their priorities as usual.

Postponed queue is used as the temporary depository for those VCPUs that needs prompt execution but have already run more often than allowed. Upon the start of each counting cycle, the scheduler checks the status of immediate queue and postponed queue. If it finds the immediate queue empty while the postponed queue is not, it swaps these two queues and then sends a rescheduling request. As a result, postponed VCPUs can receive preferential and properly delayed scheduling.

The *boost* priority of a VCPU in credit scheduler is replaced by the action of joining the immediate queue. A waken and boosted VCPU can be recognised by checking the *waken* flag (refer to sub-section 3.2.1) and whether it is in the immediate queue.

Every VCPU also has an *is_immediate* flag. It is turned on when a VCPU scheduled from the immediate queue and turned off when it is de-scheduled or a tick event fired. Conditional pre-emption of the running VCPU is decided based on the *is_immediate* flag instead of by comparing priorities. A VCPU with the *is_immediate* flag set cannot be pre-empted.

3.2.3 Guaranteed VCPU's time slicing

Credit scheduler is a weighted round-robin (WRR) based fair scheduler. It achieves proportional fairness by adjusting VCPU's credits to control the frequency that the VCPU is selected to run and by running each VCPU for the same size of time quantum. Ideally, a VCPU is only

pre-empted when its time slicing expires or it spontaneously yields the CPU. However, credit scheduler pre-empts the running VCPU when a blocked VCPU is waken and boosted. Also, in the proposed scheduler the running VCPU is pre-empted when a runnable VCPU receives an inter-domain or PIRQ event. Both cases of pre-emption can happen anytime. So the pre-empted VCPU becomes the victim of pre-emption, because once being pre-empted it will lose its remaining time slice in this rotation and have to wait in the run queue until its next turn. Time slices allocated to VCPUs are loosely controlled in credit scheduler and usually the long term CPU time received by VCPUs is bound and balanced by the credits they receive. However, this does not work well with CPU affinity. Scheduling unfairness may be caused when VCPUs are pinned to some specific CPUs since they earn more credits than they could spend. Over-earned credits allow VCPUs that block and wake regularly to excessively pre-empt their competitors. Also, as FEAS allows temporary overdraft of future quantum and it prioritises VCPUs receiving events even if they have negative credit balance, this module limits the overdraft to one rotation's range.

In this module, every VCPU is allocated a quantum of 30ms each rotation and this quantum is guaranteed to be exhausted in this rotation. The quantum is deducted at the same time with debiting credits. After being pre-empted, the VCPU is inserted to the head position of the run queue if its quantum is still more than 1ms. Therefore, it can keep consuming its quantum later on. Otherwise, it is inserted into the run queue in the conventional way. For those de-scheduled VCPUs de-queued from the immediate queue, their position in the run queue is reserved. So the usage of their allocated quantum for each rotation can be accurately recorded.

FEAS maximises busy VCPUs' responsiveness by allowing temporary overdraft of future quantum. A runnable VCPU receiving many I/O related events may exhaust its quantum early by frequently joining immediate queue or postponed queue. If that is the case, the VCPU can no more be prioritised and have to wait in the run queue for its turn. When it gets its turn, it is given a minor slice of 500 microseconds to run. Therefore, the overdraft is limited within the quantum that is worth one rotation.

4 Performance study

FEAS is implemented based on the credit scheduler of Xen 4.0.1 and tested on Linux-2.6.18.8. Since it works by monitoring the events sent between domains and between domain 0 (or driver domain) and physical devices, all source code modifications made are within the hypervisor and none is needed within the guest kernel. In our implementation, n_{limit} is set to 1 and the *tick* which fires every 10ms is reused as the trigger that indicates the start of a new counting cycle. The machine we are testing with has an Intel Core2 Duo 3.16 HZ CPU, 3.2 GB RAM and a Gigabit Ethernet network interface. A separate machine is used as the client for the network related experiments. The client machine is guaranteed of no bottleneck in any experiments. Two machines are connected via a 10/100M switch.

Four guest domains each of which has one VCPU are created. They all have the default weight of 256. Domain 0 is chosen as the driver domain. Its VCPU is pinned to CPU core 1 and VCPUs of four guests are pinned to CPU core 2. Dedicating a CPU core to Domain 0 can achieve better system performance since all I/O requests have to go through Domain 0 and this can reduce the number of CPU context switches required. This is also a common configuration in production servers. Guest 4 is connected to the client machine in all experiments. In the experiments guest domains, excluding Domain 0, are loaded in groups with CPU intensive tasks to simulate various production environments. Table 1 enumerates the different setups. The VCPUs are kept busy using cpuburn 1.4 (Softpedia 2011).

Domains	*Explanation*
All busy	VCPUs of all guest domains are running at 100%.
Others busy	VCPUs of all guest domains except Guest 4 are running at 100%.
All idle	VCPUs of all guest domains are idle most of the time.

Table 1: Experimental setup

4.1 Scheduling fairness

Fairness is one of the main goals of a scheduler, especially the scheduler of a VMM. Domains should not be starved and a malicious domain cannot take an unfair amount of CPU time slicing at any time. When CPU resources are in contention, each domain should receive CPU time proportional to their weights. Domains with the same weights are expected to receive the same amount of CPU time.

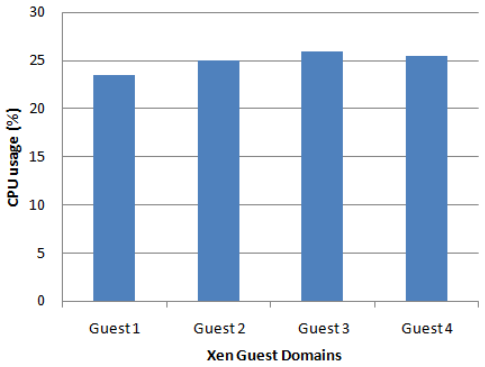

Figure 5: CPU time distribution

This experiment proves the scheduling fairness of FEAS by keeping all domains busy. Each domain runs cpuburn and eats as much CPU time as they are given. As can be seen from Figure 5, every domain can receive roughly equal amount of physical CPU time. This figure holds in all experiments where all guest domains are CPU hogging and guest 4 constantly receives ping or downloading requests. The CPU time slicing guarantor module described in section 3.2.3 keeps the influences of pre-emption on round robin scheduling to a minimum. A VCPU is marked inactive and its credits are forfeited when it accumulates too many credits. Ideally, excessive credit accumulation is due to the VCPU's little demand.

However, it may also be caused by undesirable starvation. Undesirable starvation is more frequent in FEAS than in credit scheduler, since the former boosts runnable VCPUs greedily to achieve high responsiveness even if its credit balance is negative. The CPU time slicing guarantor can efficiently ease this kind of starvation by guaranteeing that pre-empted VCPUs fully use their CPU slice in every rotation and that pre-empting VCPUs cannot overdraft their CPU time too much.

4.2 Latency sensitive processes

This experiment tests the performance of latency sensitive tasks in a domain doing mixed workloads. The client machine sends ping requests to the Guest 4 for 100 times and its response time is recorded.

All domains including Guest 4 are running CPU intensive tasks. As a result, they are hardly blocked. So under credit scheduler, even though Guest 4 constantly receives ping requests, it is not boosted. It cannot get scheduled and respond to the request until other VCPUs ahead of it in the run queue finish execution. So the latency under the credit scheduler is high and unpredictable.

On the other hand, FEAS can capture the incoming event notification all the way and scheduled the target domain immediately. It prioritises runnable VCPUs receiving an event call at most once per 10ms no matter whether its priority is under or over. Since the ping request is sent every second which is way longer than 10ms, the domain doing CPU intensive tasks can always respond to ping requests immediately.

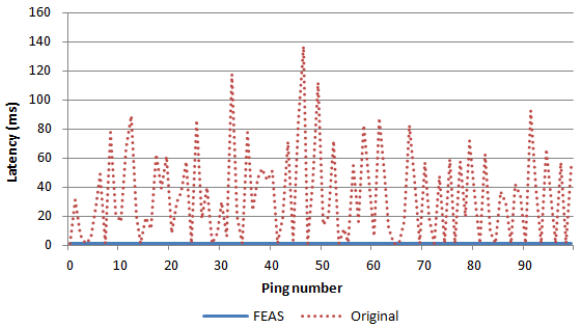

Figure 6: Ping Latency

4.3 Network intensive processes

The performance of a CPU-consuming domain on network intensive tasks is evaluated in this experiment. Guest 4 hosts a FTP server using vsftpd-2.3.4 (vsftpd, 2011) and the client machine downloads files of different sizes from it. The average transfer speed is recorded.

Figure 7 illustrates the results achieved in different situations. Both schedulers achieve similar results when Guest 4 is idle no matter whether other domains are busy. The *boost* mechanism from credit scheduler efficiently ensures the performance of idle domains on I/O related tasks when they co-exist on the same server with other domains that are doing CPU intensive tasks. However, when Guest 4 is also doing CPU intensive tasks, the download speed under FEAS doubles that under credit scheduler. Since Guest 4's VCPU is runnable most of the time when fully loaded, credit scheduler does not discriminate it from other co-currently running busy domains. Scheduling delays regarding handling I/O related events are thus much longer than when Guest 4 is idle. However, FEAS keeps I/O devices busy by promptly handling of incoming events and thus speeds up the transmission process.

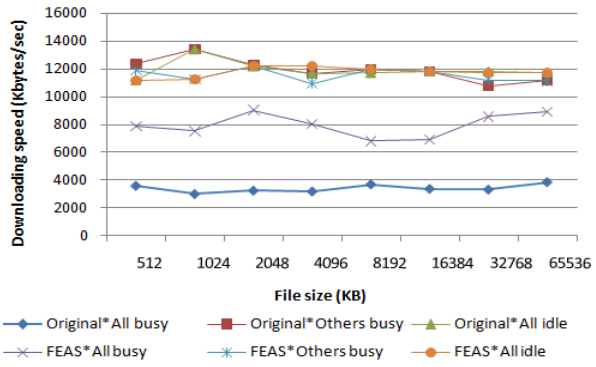

Figure 7: FTP downloading speed

4.4 Impact of co-working VMs

This experiment examines the I/O performances under FEAS when multiple co-located VMs concurrently process mixed workloads. Five duplicated guest domains are set up for this experiment on the same physical server used in previous experiments. Also, Domain 0 is pinned to CPU core 1 and Domain 1-5 are pinned to CPU core 2. All five guest domains are configured with a FTP server facilitated by vsftpd-2.3.4, and each hosts a 16384 KB file ready for download. To keep guest domains' VCPUs under constant pressure, they all run cpuburn. A multi-threaded C# program is designed to simultaneously download the hosted file from guest domains. Hence, during the period of concurrent network streaming dense inter-domain events and PIRQ events are fired across by all domains at the same time. All these events are caught by FEAS and its decision has a direct impact on the I/O performances and CPU fairness.

The multi-threaded program is modified to run with 1 to 5 threads respectively on the client machine and each thread downloads the test file from an exclusive VM. Suppose that n threads simultaneously stream files from n VMs (where $n = \{1, 2, \cdots, 5\}$), and that the ith streaming thread starts at t_i^s and finishes at t_i^f (where $i = \{1, \cdots, n\}$). Downloading speed s_n is evaluated to reflect the system performance as

$$s_n = \frac{n * size_f}{latest(t_1^f, \cdots, t_n^f) - earliest(t_1^s, \cdots, t_n^s)}$$

where $size_f = 16384KB$

The results are recorded and shown in Figure 8. The overall system throughput increases when downloading files from more VMs, and the I/O devices tend to operate at full speed if downloading files from all VMs. The reason is that the overall system performance depends on the CPU time and the scheduling latency of all streaming VMs. Thus, if all five VMs are delivering files, no matter which VM is scheduled, they all contribute to the overall system throughput. All in all, FEAS performs better than the original credit scheduler in terms of I/O throughput even when multiple VMs do mixed workloads at the same time.

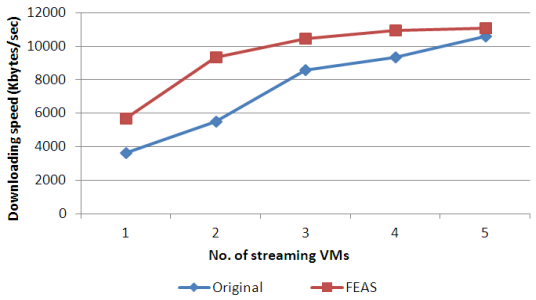

Figure 8: Download from multiple VMs

4.5 Scheduling overhead

Seconds	*Others busy*	*All idle*	*Domain 0*
Original	32.43754	8.78261	8.123744
FEAS	32.5931	8.80411	8.146923

Table 2: Duration for prime searching in seconds

To quantify the scheduling overhead FEAS causes over the original credit scheduler, we observe the running time of the prime searching function which is a lengthy and CPU-intensive process. This experiment finds all the 664,579 prime numbers less than 10^7 using the trial division algorithm (Wikipedia, 2011). The time required to complete the process in three cases is illustrated in Table 2. The percentage increase of running time introduced by FEAS in all three cases is less than 1%, which indicates that the overhead is negligible.

5 Conclusion

VMs sharing the same hardware contend for limited resources. It is important to appropriately allocate shared resources among VMs that are running simultaneously. While fairness requires that each VM receives CPU time proportional to their weights, low latency is achieved by scheduling a VM as soon as it needs CPU especially if the VM has I/O tasks pending. Our scheduler is an enhanced version of credit scheduler that prioritises VCPUs doing I/O tasks by monitoring inter-domain events and PIRQ events sent between domains and physical devices. FEAS makes VMM full-time event aware and promptly schedules with best effort runnable VCPUs that receive I/O related events. The experiments show that VMs under FEAS performs better on I/O intensive tasks than those under credit scheduler if they also do CPU intensive tasks at the same time. The cost for the modifications needed to realise FEAS is proved to be negligible.

Currently, FEAS is only implemented and tested with the guest domains virtualised in para-virtualisation mode. Since hardware virtual machine (HVM) does not requiring the guest operating system to be modified, it can run proprietary operating systems like Windows as guest. Split drive model is implemented differently in PV-on-HVM kernels. Our future research will try to apply FEAS on fully virtualised virtual machines.

6 References

Barham, P., Dragovic, B., Fraser, K., & et al (2003): Xen and The Art of Virtualization, *ACM Symposium on Operating Systems Principles*.

Cherkasova, L., Gupta, D. & Vahdat, A. (2007): Comparison of the Three CPU Schedulers in Xen, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 35, Iss. 2, pp. 42–51.

Chisnall, D. (2007): *The Definitive Guide to the Xen Hypervisor*. Sydney, Prentice Hall.

Goldberg, R.P. (1974), Survey of Virtual Machine Research, *IEEE Computer*, Vol. 7, Iss. 6, pp. 34-45.

Govindan, S., Nath, A., Das, A., Urgaonkar, B. and Sivasubramaniam, A. (2007): Xen and co.: communication-aware CPU scheduling for consolidated xen-based hosting platforms, *Proceedings of the 3rd international conference on Virtual execution environments*, New York, USA.

Gupta, D., Cherkasova, L., Gardner, R. & Vahdat, A. (2006): Enforcing performance isolation across virtual machines in Xen, *In Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference*, Melbourne, Australia.

Iyer, R., Illikkal, R., Tickoo, O., Zhao, L., Apparao, P. & Newell, D. (2009): VM3: Measuring, modeling and managing VM shared resources, *Computer Networks*, Vol. 53, Iss. 17, pp. 2873-2887.

Kim, H., Lim, H., Jeong, J., Jo, H. & Lee, J. (2009): Task - Aware Virtual Machine Scheduling for I/O Performance, *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environment*. Washington, DC, USA.

Lin, B., Dinda, P. & Lu, D. (2004): User - Driven Scheduling of Interactive Virtual Machines. *5th IEEE/ACM International Workshop on Grid*. Washington, DC, USA.

Ongaro, D., Cox, A. and Rixner, S. (2008): Scheduling I/O in virtual machine monitors, *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, New York, USA.

Softpedia (2011), cpuburn 1.4, http://www.softpedia.com/get/System/Benchmarks/cpuburn.shtml, Accessed May, 2011

vsftpd: vsftpd - Secure, fast FTP server for UNIX-like systems. https://security.appspot.com/vsftpd.html. Accessed May, 2011.

Weng, C., Wang, Z., Li, M. & Lu, X. (2009): The Hybrid Scheduling Framework for Virtual Machine Systems, *ACM International Conference on Virtual Execution Environments*. Washington, DC, USA.

Wikipedia: Trial division. http://en.wikipedia.org/wiki/Trial_division. Accessed May, 2011.

Xia, Y.B., Yang, C. & Cheng, X. (2009): PaS: A Preemption-aware Scheduling Interface for Improving Interactive Performance in Consolidated Virtual Machine Environment. *International Conference on Parallel and Distributed Systems*. Shenzhen, China.

Xen: Home of the Xen Hypervisor. http://xen.org/. Accessed Jun, 2011.