Scheduler Support for Video-oriented Multimedia on Client-side Virtualization

Hwanju Kim[†], Jinkyu Jeong[†], Jaeho Hwang[†], Joonwon Lee[§], Seungryoul Maeng[†]

[†]Computer Science Department, Korea Advanced Institute of Science and Technology, Daejeon, South Korea
[§]School of Information and Communication Engineering, SungKyunKwan University, Suwon, South Korea
{hjukim, jinkyu, jhhwang}@calab.kaist.ac.kr, joonwon@skku.edu, maeng@kaist.ac.kr

ABSTRACT

Virtualization has recently been adopted for client devices to provide strong isolation between services and efficient manageability. Even though multimedia service is not rare for the devices, the virtual machine hosting this service is not guaranteed to receive proper scheduling support from the underlying hypervisor. The quality of multimedia service is often compromised when several virtual machines compete for computing power. This paper presents a new scheduling scheme for the hypervisor to transparently identify if the workload handles multimedia and to provide proper scheduling supports. An implementation of our scheme has shown that the virtual machine hosting a video-oriented application receives propoer CPU scheduling even when other virtual machines host CPU intensive workloads.

Categories and Subject Descriptors

D.4.1 [**OPERATING SYSTEMS**]: Process Management—*Scheduling*

General Terms

Experimentation, Performance

Keywords

Virtualization, Multimedia, Scheduling, Client-side virtualization, Xen

1. INTRODUCTION

Many computing environments have been increasingly virtualized at a low-level so that multiple operating systems (OSes) can run on a single physical machine simultaneously and securely. This type of computing is based on machine virtualization, which enables an individual computing environment to be encapsulated into a Virtual Machine (VM) as software. Machine virtualization usually needs a thin software layer, called a *hypervisor*, which presents a completely isolated domain for each VM. Virtualized environments improve manageability, trustworthiness, efficiency, and flexibility, despite their performance overheads. In server-side virtualization, the most compelling advantage is the enhanced resource utilization by consolidating several underutilized servers on a few physical machines.

Client-side virtualization has also drawn attraction as a means of presenting multiple computing domains on a local device such as laptops/desktops and consumer electronics devices. Client-side virtualization allows a user to divide applications into multiple VMs according to their security demands and functionality. Those applications that need trusted environment could be securely isolated into a separate VM [13]. Moreover, virtualization enables a client device to run different OSes according to the application characteristics. For example, a smartphone application could run on a real-time OS while a downloaded game application could run on a general purpose OS that provides rich graphical user interfaces. These attractive trends can be seen in converged consumer electronics devices [3, 16] and corporate-owned laptop PCs [27].

For client-side virtualization, however, commodity hypervisors fail to ensure satisfiable multimedia quality, especially when multiple VMs compete with each other for CPU resources. Current VM schedulers simply provide a knob for a user to adjust the CPU share or time constraints for each VM [7]. By using this interface, a user can specify the adequate resource requirements for a VM that hosts a multimedia application in order to achieve the expected quality. This scenario requires a priori knowledge about the CPU capability and computational demands of the application, which is impractical due to the diversity and rapid change in both of them. More importantly, a virtualized OS redistributes the CPU share given by the hypervisor among runnable tasks according to its local scheduling policy. If the underlying hypervisor fails to allocate enough CPU share, many OS-level optimizations for multimedia will not be effective on a virtualized client. Several proposals for enhancing VM schedulers are aimed at server virtualization [14, 31] or depend on user-specified parameters [25, 23] for a specific VM that hosts a soft real-time workload.

This paper presents a VM scheduling scheme for client-side hypervisor to enhance the service quality of a video-oriented application. The proposed solution satisfies the following two goals: Firstly, our solution does not rely on user- or application-specified configurations. A general-purpose VM could embrace any workloads including multimedia, while the hypervisor regards the VM as a resource allocation unit

MMSys'12 , February 22-24, 2012, Chapel Hill, North Carolina, USA.

without any knowledge about the tasks running on the VM. Therefore, it is hard for the hypervisor to decide how much CPU is allocated to a VM even when a user knows the exact demands for the desired multimedia quality. Secondly, our scheme is aimed at a hypervisor-based solution without OS cooperation. Since a hypervisor can host diverse OSes, OS involvement makes deployment cumbersome, because it requires modifications to each hosted OS.

Our proposed solution transparently estimates multimedia quality and dynamically adjusts CPU allocation based on the estimates as feedback. To this end, we consider a display rate (i.e., frame rate) as a metric for evaluating multimedia quality that is sensitive to human vision [11]. For hypervisor-level estimation, a display rate is derived from low-level events that can be transparently monitored by the hypervisor without OS involvement. The estimated quality is used for the hypervisor to adjust CPU allocation. The feedback-driven CPU allocator, named *multimedia manager*, cooperates with a VM scheduler to meet desired multimedia quality by adapting CPU share on the fly.

In order to evaluate the proposed scheme, we implement a Xen-based prototype that incorporates the multimedia manager and the Credit scheduler, which is a proportional share scheduler included in Xen by default. The actual adjustment of the CPU share and a refined preemption scheme for achieving responsiveness are implemented in the Credit scheduler, while the rest of our scheme is scheduler-independent to ensure compatibility with other VM schedulers. Video playback and 3D gaming are selected as CPU-intensive multimedia workloads with which various synthetic and real workloads compete for CPU resources. The experimental results show that our scheme effectively supports multimedia workloads in dealing with heavy CPU contention.

The rest of this paper is organized as follows: As background information, Section 2 describes usage scenarios of client-side virtualization and I/O virtualization mechanisms. Section 3 describes related work and our motivation. In Section 4, we present the preliminary analysis of the Xen Credit scheduler for multimedia workloads. Section 5 introduces our proposed scheme and Section 6 presents the evaluation results. Finally, Section 7 draws conclusions and suggests future direction for our research.

2. CLIENT-SIDE VIRTUALIZATION

Desktop virtualization is introduced for flexible and secure management of desktop environments. Virtual desktop computing can be realized by server-side and client-side virtualization. Server-side virtual desktop computing, called Virtual Desktop Infrastructure (VDI), pushes virtual desktops into centralized server farm, to which each user connects via network; this model matches thin client computing. In this model, user experience tightly depends on network bandwidth and remote desktop protocols. In order to improve user experience, VDI solutions provide multimedia-enhanced remote desktop protocols, which offload media processing to client devices for reducing network bandwidth. In spite of such improvement, server-side VDI cannot always guarantee desktop availability for mobile users who may be disconnected from the central server.

Client-side virtualization enables offline (disconnected) VDI by allowing users to run virtual desktops on their local client devices. Since a desktop environment is entirely confined in a local device, client-side virtualization provides much better user experience and availability, regardless of network status, compared to server-side approaches. Client-side virtualization can still use a centrally managed VM by checking out the VM image from a management server to a local device if necessary. For this type of virtualization, many vendors provide various client-side solutions in the form of bare-metal hypervisors (e.g., Citrix XenClient, VMware CVP, and Virtual Computer NxTop) and hosted hypervisors (e.g., VMware ACE, MVP, and Workstation) [1]. This section describes usage scenarios of client-side virtualization and hosted I/O virtualization.

2.1 Usage Scenarios

2.1.1 Mobile Laptops

Many people use their laptops both for personal purposes and for business tasks. In this case, corporate information can be easily leaked by a malicious attack or virus, which may have penetrated the system via less secure personal applications. In order to address this problem, corporate administrators may confine sensitive information to securely isolated facilities, but the weak isolation between personal and corporate environments on a laptop makes it difficult to completely protect the information.

Client-side virtualization offers complete isolation between business and personal environments, enabling a user to securely accommodate both on a single laptop. Citrix XenClient [27] manages a business VM as a secure domain while allowing private extensions in a personal VM. Likewise, the VMware MVP enables employee-owned devices to securely isolate corporate profiles from personal ones.

2.1.2 Converged Consumer Electronics Devices

Consumer electronics devices have converged because of the increase in their computational power, memory, and storage capacity. A smartphone, for example, has integrated multiple functions such as PC-like tasks (e.g., Flash playing and downloading via Wi-Fi) with its primary functions, i.e., calling and texting. This trend enables a device user to freely extend the functionality by downloading 3rd-party applications via an online store (e.g., AppStore). In this type of devices, their primary functions must be safely protected from malicious applications, because malfunctions of major components would lead to recalls of the devices.

For the trustworthiness of extensible consumer electronics devices, client-side virtualization provides strong isolation of the intrinsic components from other foreign software components [13]. Virtualization also allows a real-time OS to run major critical tasks while using a general-purpose OS for PC-like workloads. VirtualLogix, OKL4 microvisor, and secure XenARM project [16] have promoted the development of virtualization-based embedded systems for this purpose. Recently, Andrus *et al.* proposed *Cells* [2], a prototype of Android-based smartphone virtualizaton that enables a single smartphone to have multiple securely isolated virtual phones by means of OS-level virtualization.

2.2 I/O Virtualization

One of the major roles of the hypervisor is to enable multiple VMs to share local I/O devices. I/O virtualization should conduct secure multiplexing in such a manner that

[1]Bare-metal and hosted hypervisors are also called Type-1 and Type-2 hypervisors, respectively.

an incoming I/O event is properly delivered to its target VM.

Most virtualization solutions adopt hosted I/O virtualization, where a separate trusted VM takes care of I/O devices [34, 24, 12]. In this scheme, an I/O request from a VM is forwarded to the trusted VM, which then handles it by directly accessing its target device. Since client products are equipped with versatile I/O devices [2], hosted I/O virtualization is a suitable approach for reusing existing device drivers. In the case of consumer electronics, the vendor-specific drivers can be used without porting efforts. Moreover, in many cases of client-side virtualization, there exists a trusted VM, which is a vendor-provided or corporate-managed domain, ensuring that it can securely host I/O accesses on behalf of another VM.

Client-side hypervisors must support video device virtualization for graphical user experiences. In hosted I/O virtualization, a trusted VM virtualizes the framebuffer of other VMs for video display. Furthermore, modern virtualization solutions provide 3D graphics virtualization to ensure that VMs can utilize Graphics Processing Unit (GPU) acceleration. 3D graphics virtualization enables a trusted VM to handle GPU acceleration commands requested from other VMs by directly communicating with a video device [22, 8].

3. RELATED WORK AND MOTIVATION

Many researchers have addressed the issue of how to support multimedia workloads on general-purpose OSes. Furthermore, several proposals have argued that VM schedulers should consider the workloads within a VM. In this section, we review previous work on OS scheduler supports for multimedia and enhancement of VM schedulers.

3.1 OS Scheduling for Multimedia

OS scheduling schemes for multimedia are classified into two approaches: 1) scheduling based on explicitly specified information and 2) scheduling based on dynamically monitored information. The first approach allows an OS scheduler to handle specific multimedia applications based on the explicit resource requirements that are provided by a user or an application itself. They usually dictate the timing constraints or CPU shares required for processing the multimedia. SMART [29], BVT [9], Rialto [17], and Processor Capacity Reserves [28] used such explicit information while also supporting conventional best-effort workloads.

Regarding the second approach, previous schemes automatically identify multimedia workloads by monitoring their well-known characteristics such as rate-based execution [33, 4, 11] and multimedia device accesses [36]. Steere *et al.* [33] developed a reservation-based scheduler that adjusts CPU proportion based on the progress of threads. The BEST scheduler specifically deals with periodic soft real-time tasks such as media players by monitoring periodic behaviors [4]. Etsion *et al.* [11] extended the Linux scheduler to reflect the output rates of each task by instrumenting the X server and the kernel. All of the above approaches commonly used feedback-driven CPU allocation to achieve expected behaviors. Recently, in order to ensure the quality of multimedia, RSIO [36] has assigned higher priority to a process that accesses an audio device, based on the fact that media playback typically accompanies audio output.

3.2 VM Scheduler Enhancement

Since VM schedulers are generally designed assuming that each VM is a black-box, they are agnostic about the timing requirements within a VM. VSched [25] and the Xen sEDF scheduler [7] allow a user to set a time constraint, (i.e., period, slice), for each VM. The Xen Credit scheduler implements a proportional-share scheduling with which a user can adjust the CPU share for each VM [31]. Lee *et al.* [23] enhanced the Xen Credit scheduler by adding the parameter of desired scheduling latency for a soft real-time VM. The above schedulers require a user to have a priori knowledge of the workload characteristics within each VM. In addition, such specific configurations make it difficult for the schedulers to deal with mixed and dynamic workloads in a VM.

In order to enable schedulers to reflect the requirements of tasks within a VM, prior proposals have been based on either guest OS cooperation or hypervisor-based solutions. Augier [3] allowed guest OSes to supply local scheduling data with which a VM scheduler can globally perform task scheduling on behalf of guest OSes. The guest-aware [19] and preemption-aware [35] schedulers can enhance the responsiveness of interactive workloads in a VM by providing the hypervisor with the guest-priority information and the preemption conditions, respectively. The approaches based on guest OS cooperation require modifications to the guest OSes.

Hypervisor-based solutions are to rely on heuristics to improve responsiveness without any guest OS involvement. The boosting mechanism of the Xen Credit scheduler favors a VM that consumes less CPU resources due to the block-and-wake behavior that is usually observed with I/O activities [31]. Similarly, network-intensive VMs can be preferentially handled by monitoring network traffics for each VM [14]. In our previous work, the task-aware scheduler improves responsiveness of interactive tasks mixed with other ones in a VM, by inference based on CPU usage and the scheduling patterns of each task [20, 21]. Since the existing schemes are developed based on server consolidation, they ensure static fair-share CPU allocation among VMs instead of the dynamic adjustment of CPU share based on multimedia workloads.

3.3 Motivation

Although prior OS-based methods for multimedia support are effective on native OSes, guaranteeing multimedia quality is hard when OSes are indirectly serviced by the hypervisor. In a virtualized system, a VM scheduler provides each VM with a certain proportion of physical CPU, with which each guest OS schedules its tasks locally. If a VM scheduler does not guarantee proper CPU allocation, the scheduling efforts of an OS for handling multimedia are futile. For the scheduling of the OS to be effective, a VM scheduler should allocate appropriate CPU share for a guest OS to support good multimedia quality.

In order to achieve transparent support for multimedia on a virtualized client, our work is aimed at a hypervisor-based solution that does not rely on OS involvement or specific configurations of resource requirements. Since an OS directly controls applications for resource management, it is possible to modify the OS scheduler to give the hypervisor some hints about multimedia workloads. In the virtualization domain, however, modifying OS is deemed to be cumbersome and makes deployment difficult, since a hypervisor may support various OSes and different kernel versions. In

addition, since most OSes have much larger trusted computing bases (i.e., size of source codes) than the hypervisor, OSes are more likely to be compromised, thereby leading to the abuse of the interface used for hinting. In light of the need for practicality and reliability, we chose hypervisor-based support for multimedia.

4. VM SCHEDULER ANALYSIS OF MULTIMEDIA WORKLOADS

This section describes the mechanisms of the Xen Credit scheduler and presents the experimental analysis on the quality of multimedia applications.

4.1 Credit Scheduler

The Credit scheduler is the default scheduler of the Xen hypervisor [5] and has been actively evolving towards client-side virtualization [30, 10]. Being a proportional share scheduler, it distributes CPU share, or *credit*, to active VMs according to the weights specified by a user. Each active VM is given a certain amount of credits every 30ms, and consumes them every 10ms while running. The scheduler allows a dispatched virtual CPU (vCPU) to run during a timeslice of 30ms. For proportional CPU sharing, the scheduler maintains two states depending on whether a vCPU has remaining credits. Since a vCPU that has remaining credits (UNDER state) is eligible to continue running, it is always picked out of its runqueue ahead of those that have exhausted their credits (OVER state). The vCPUs in the same state are maintained in a round-robin manner.

In order to supplement a static timeslice (30ms), which can degrade I/O responsiveness, the Credit scheduler provides preemption for a vCPU that has been blocked in the UNDER state when it is awakened by its I/O response; a vCPU becomes blocked when it has no runnable task. In order to support preemption, such a vCPU enters the BOOST state, which has the highest priority, and preempts the vCPU currently running in the UNDER or OVER states. Since an I/O-bound workload mostly waits for I/O responses with being blocked, preemption support significantly improves I/O responsiveness [31].

4.2 Experiments

In order to identify the issues that arise from a virtualized client in terms of multimedia support, we present experimental results on the quality of multimedia applications running under various workloads.

4.2.1 Experimental Setup

We evaluated multimedia quality in cases where a multimedia application in a VM competes with various workloads in another VM. We investigate the effect of a multimedia workload depending on whether it runs in a VM that has the capability of direct I/O or not (indirect I/O). Multimedia workloads with direct I/O present near-native performance by avoiding indirect emulation. For example, XenClient can give a VM direct I/O privilege for performance enhancement, especially high-definition user experience [27]. Xen allows a VM to have the privilege of direct I/O access. The VM is called an *isolated driver domain (IDD)* [12]; *domain0* is the default IDD, but any other trusted VM can take on this role. An untrusted VM can also be given direct I/O privilege, called I/O passthrough, with the aid of IOMMU hardware such as Intel VT-d [1]. Since our hardware does not have the IOMMU facility, multimedia workloads with direct I/O were experimented in the domain0. In this paper, a VM with direct I/O is generally referred to as an IDD, while a VM with indirect I/O is called a guest domain; the terms VM and domain are used interchangeably.

For the scheduler setup, we ensure that each VM receives equal CPU allocation by assigning the same weight to the VMs. In addition, the schedulers allow VMs to use idling CPU resources beyond their given allocation (i.e., work-conserving mode) [2].

Our evaluation was conducted on a laptop PC comprised of an Intel Core 2 Duo 2.16 GHz CPU, an Intel GMA 950 integrated GPU, an Intel high definition audio controller, and 2 GB DDR2 RAM; each VM has the same 1 GB RAM. We used Xen-3.4.0 as the hypervisor and the Ubuntu 8.04 distribution with the Linux-2.6.18.8 kernel as an OS. The network and disk I/O of the guest domain are handled via a software bridge and a file-backed disk image, respectively. In order to interact with the graphical interface of the guest domain, Xen uses a *vncviewer* to show the video output generated by the guest domain via a *virtual framebuffer*. For 3D graphics acceleration, we used *VMGL* [22], a virtualized OpenGL implementation that provides a guest domain with hardware acceleration for 3D graphics. VMGL forwards OpenGL commands requested from the guest domain to a stub in an IDD, which then sends the commands to a hardware acceleration unit. We enabled only one core in order to investigate the effect of scheduling in the case where a multimedia application time-shares a CPU core with other workloads.

4.2.2 Measurement Methodology

We modified the VLC player to record timestamp every displayed frame except dropped ones. By post-processing the record, we obtained real frame rates as time progresses. In order to measure the frame rate of Quake3, we interposed in every call to *glXSwapBuffers()* whose function is exchanging front and back buffers in the OpenGL library; for hooking the function call to the library, we used the `LD_PRELOAD` facility in Linux. Finally, we used the XenMon utility [15] for measuring CPU usage of each VM every second.

4.2.3 Workloads

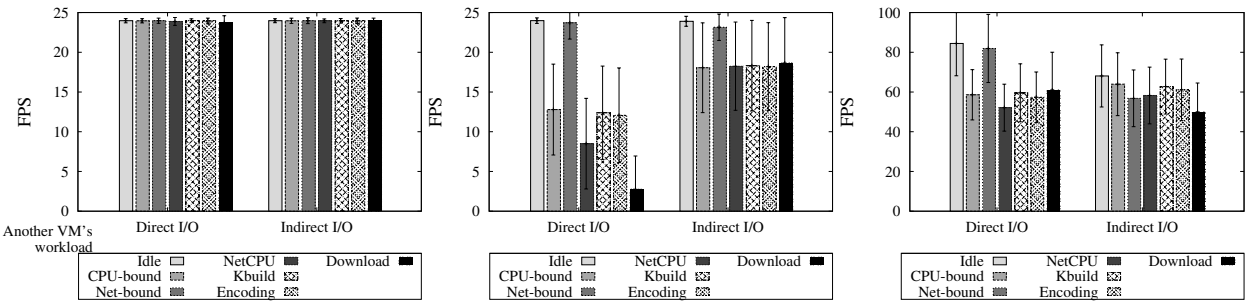
For the multimedia workloads, we chose the following media player and 3D game applications.

- Media player. Video playback of a media player requires soft real-time features, since its quality relies on the timely display of each video frame. Most media players typically drop a frame when a playback process misses the deadline for displaying the frame, largely due to CPU contention. Therefore, displayed Frame-Per-Second (FPS) is used to evaluate how well video playback is supported. We used VLC, an open-source cross-platform media player. We played 85-sec low-resolution (640×354) and high-resolution (1280×720) video files, which have the same video contents encoded with H.264 and frame rate (23.976 FPS).

[2]Non-work-conserving mode is appropriate for server consolidation environments that regards strict performance isolation among VMs as a crucial factor

Workloads		Description	CPU usage (%)	
			Direct I/O	Indirect I/O
Multimedia	VLC-LowRes	640x354 video playback (*vlc*)	17	24 (+18)
	VLC-HighRes	1280x720 video playback (*vlc*)	62	56 (+24)
	Quake3	Quake III arena demo play	72	29 (+57)
Competitor	Idle	No workload	0	0 (+0)
	CPU-bound	CPU-hungry loop	100	100 (+0)
	Net-bound	TCP streaming benchmark (*iperf*)	11	5 (+15)
	NetCPU	Net-bound with CPU-bound	100	85 (+14)
	Kbuild	Build Linux 2.6.34 kernel (*make*)	94	96 (+2)
	Encoding	Encode a video (*mencoder*)	100	98 (+2)
	Download	Download a file from Web (*wget*)	41	28 (+40)

Table 1: Evaluated workloads description (in the case of indirect I/O, CPU usage consumed by an IDD is separately indicated in the parenthesis)



(a) VLC-LowRes (b) VLC-HighRes (c) Quake3

Figure 1: Average frame rates of the Credit scheduler

- 3D game. 3D game applications are also prevalent multimedia workloads that need a relatively high frame rate and exploit hardware acceleration for complex rendering. Most 3D game applications commonly adjust their frame rates depending on the available CPU. Accordingly, FPS is used as a metric for evaluating 3D game quality. We chose *Quake III Arena*, which has been widely used for evaluating how well hardware and system software support 3D game capability. For the evaluation, we used a publicly available demo that runs for 68 seconds.

For workloads that compete with the multimedia one for CPU, we used synthetic and real workloads. Table 1 shows the description and CPU consumptions for each evaluated workload. In the case of indirect I/O, CPU usage for I/O processing is indicated in the parenthesis. For networking, a separate machine is connected to the virtualized laptop via a 100 Mbps Ethernet switch.

4.2.4 Results

Figure 1 shows the frame rates achieved when each multimedia workload competes with various ones in another VM. The results indicate the average and standard deviation of FPS samples measured every second during five runs. Since VLC-LowRes consumes less CPU than its allocation (50%), it shows good playback quality when running with competing workloads. We will focus on the extent to which VLC-HighRes and Quake3 are affected by competing workloads, since they have higher CPU requirements than the allocated ones.

As shown in Figure 1(b) and Figure 1(c), the frame rates of VLC-HighRes and Quake3 are significantly degraded when running with CPU-hungry workloads (CPU-bound, NetCPU, Kbuild and Encoding). In addition, VLC-HighRes suffers from large variation while competing with those workloads. Note that NetCPU in a guest domain more degrades frame rates in an IDD because indirect network I/O impedes the multimedia workloads. Meanwhile, multimedia workloads with indirect I/O suffers less degradation from CPU-hungry workloads than that with direct I/O, since their CPU requirement become lower by offloading display operations to an IDD; VLC-HighRes with indirect I/O requires 56% CPU, whereas that with direct I/O requires 62%.

In particular, the VLC-HighRes with direct I/O suffers from severe frame drops when competing with the downloading workload. The downloading in a guest domain consumes a large proportion of the IDD's CPU for indirect I/O, whereas its own CPU consumption is relatively low. In this case, the boosting mechanism of the Credit scheduler, which is described in Section 4.1, allows a guest domain to frequently preempt an IDD when an incoming network packet is destined for the guest domain, since the IDD mostly exhausts its CPU allocation for its multimedia workload and network processing. Accordingly, frequent preemption by a guest domain significantly interferes with timely display in the video playback process.

5. VM SCHEDULER FOR MULTIMEDIA

In order to enhance the VM schedulers for good multimedia quality, we enable the hypervisor to identify multimedia workloads and their quality so that the hypervisor can achieve the multimedia-aware CPU distribution. This section describes our hypervisor-level multimedia identification method, feedback-driven CPU allocator, and scheduler implementation.

5.1 Hypervisor-level Multimedia Identification

As mentioned in Section 3.3, our goal is to enable the

hypervisor to automatically identify multimedia workloads without OS cooperation or explicit information on their resource requirements. As with prior OS solutions without resource specifications, a feedback-driven mechanism is needed to achieve the expected quality on the fly. A feedback-driven mechanism requires a quality estimation method in order to apply positive or negative feedbacks for adjusting CPU allocation. For example, Steere *et al.* [33] used the progress of threads as a metric for a feedback-driven CPU allocator by monitoring the producer-consumer buffer. Etsion *et al.* [11] used output production rates as a multimedia quality metric for their positive feedback loop.

Accordingly, our hypervisor-based solution needs multimedia identification and quality estimation methods in order to provide feedback. We notice that multimedia workloads commonly have media output such as video and audio to which a user is sensitive. In this type of workloads, a frame rate is used as a metric for video display quality, which is evaluated by human vision. We use the media output behaviors and the displayed frame rates to identify multimedia workloads and quality, respectively.

Frame rate estimation, however, is challenging for the hypervisor, which provides only low-level abstractions of underlying hardware. Indeed, better estimation can be done by a software display interface such as the X server [11], because it is a layer to which multimedia applications directly send display requests. For OS-independent and non-invasive implementation, which is our primary design principle, we enable the hypervisor to derive frame rates from low-level hardware events. In accordance with the display types, our solution considers two cases: memory-mapped display and GPU-accelerated display.

5.1.1 Memory-mapped Display

A video device uses video memory, which resides on its chip or mappable system memory, to conduct display operations. The majority of video memory is comprised of the framebuffer that stores frames to be displayed. In order to allow software to access the framebuffer, memory-mapped I/O is used via a virtual memory region. For example, the X server maps framebuffer to its virtual address space and directly writes frame contents to the address space for display. General display operations such as video playback use this type of memory-mapped interface.

With respect to memory-mapped display, the hypervisor can monitor the frequency of framebuffer writes in order to estimate the frame rates. For monitoring framebuffer writes, the hypervisor must be notified when framebuffer is updated. However, (virtual) framebuffer is allowed for direct updates via a memory-mapped interface without hypervisor intervention. Therefore, the hypervisor needs to interpose intentional notification at framebuffer writes transparently.

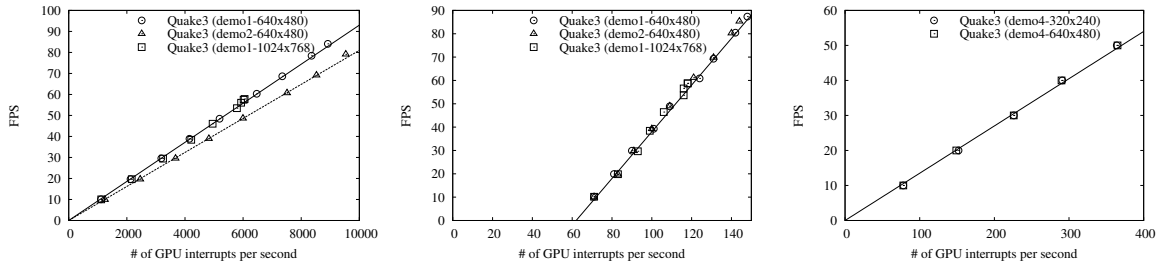
To that end, we enable the hypervisor to write-protect a virtual address region that is mapped to the (virtual) framebuffer memory. A memory protection mechanism notifies the hypervisor of a write attempt to a write-protected region via a page fault exception. In order to write-protect the framebuffer, the hypervisor inspects memory mapping to the framebuffer and disables the write permission of the corresponding virtual addresses [3]. In this manner, the hypervisor identifies a framebuffer write, which raises a page fault exception that invokes the page fault handler in the hypervisor. In order to alleviate page fault overheads, we use sampling-based write protection, by which the hypervisor write-protects one page out of N pages. The hypervisor regards a batch of successive writes to the framebuffer as one frame display. Accordingly, K frame display during one second represents K FPS.

The hypervisor maintains monitored frame rates per task in order to selectively estimate the frame rate of a multimedia application. Note that the aggregated frame rate of a VM cannot represent the multimedia quality. For example, when video playback and kernel compilation generating display output are concurrently run in a single VM, the VM's total frame rate cannot be used as a metric for the video playback quality. In order to ensure that the hypervisor can identify per-task frame rates, we use an existing task tracking technique on the basis of address space switches in which the hypervisor is involved for memory virtualization [18, 20]. In our implementation for x86 architecture, the hypervisor can identify a currently running task by checking a CR3 value, which is the root of the current address space, when a page fault occurs at framebuffer. This simple technique enables the hypervisor to associate a framebuffer write with a displaying task in OSes that use a kernel-level display interface such as Windows GDI, because the write occurs in the context of the displaying task.

Unix-like OSes, however, use a user-level display interface, an X server, to handle display requests from multiple tasks, thereby making per-task frame rate estimation complicated. In this type of system, framebuffer writes always occur while the X server is running. Since inter-process communication is carried out solely by the kernel, the hypervisor has difficulty in accurately identifying which task sends a display request to the X server. For lightweight tracking, we use a heuristic that exploits a kernel scheduling policy for interactive tasks [20, 36]. Since the X server is an event-driven and interactive task, it is very likely to be scheduled immediately after a task sends it a display request. Based on the heuristic, we regard a task previously scheduled ahead of the X server as a display requester when framebuffer writes occur. In order to improve accuracy, the hypervisor maintains a recently displayed region for each displaying task based on the written framebuffer pages. With this information, the hypervisor can reduce misidentification by filtering the other region updates. When framebuffer writes occur, the hypervisor increments the display count of the previously scheduled task if the written pages correspond to its recently displayed region. Otherwise, the hypervisor also checks the next scheduled task based on the updated region, since the X server has a request-response protocol.

Finally, we regard a displaying task that has accessed audio as a multimedia application in order to exclude non-multimedia ones that generate display output. This filtering is reasonable considering that video playback applications typically accompany audio output as well [36]. Like video devices, an audio device access is done via memory-mapped I/O, so that the access can be monitored by the hypervisor via write-protection. The hypervisor only considers synchronous audio access in a task context in order to associate the access with an accessing task; an asynchronous access by an audio interrupt handler takes place in an interrupt context, where is no relationship between the access and any task [6]. In order to exclude asynchronous audio

[3]The hypervisor validates all memory mapping of VMs before granting permission for secure memory virtualization.



(a) Intel GMA 950 (b) NVidia 6150 Go (c) PowerVR SGX540

Figure 2: Correlation between GPU interrupt and frame rates

accesses, we simply filter audio buffer writes after an audio interrupt.

5.1.2 GPU-accelerated Display

Complex graphical operations such as 3D rendering are typically assisted by GPU acceleration. In this case, a 3D application sends a GPU command stream that contains display information to a video device. In the case of Unix-like OSes, the Direct Rendering Infrastructure (DRI) [32] allows user-level applications to communicate with the GPU without the intervention of the X server. Obviously, GPU-accelerated display bypasses the memory-mapped interface.

For estimating GPU-accelerated display, the hypervisor can monitor a GPU command queue by using write-protection; a GPU command queue is shared between the CPU and GPU for communication via Direct Memory Access (DMA) operations. This type of explicit tracking enables the hypervisor to identify what GPU command is issued, so that a frame rate can be estimated by inspecting the GPU commands related to frame display. The write-protection for the GPU command queue, however, could incur substantial overheads, since most 3D games show high frame rates, which entail a considerable number of GPU commands.

Instead, we exploit the rate of interrupts raised by a video device for lightweight estimation of a 3D game's frame rate. 3D applications require fine-grained synchronization with the GPU to manage GPU command streams. The application examines the command execution progress for flush operations and GPU-accessible memory management. For example, an application needs to be notified of when submitted commands have been completed for a flush operation. This type of notification is also needed when an application decides to reuse a memory buffer that has been referenced by the GPU. Such a notification operation is typically implemented as an interrupt signaled by a video device. In this regard, we hypothesize that GPU interrupts are proportionally raised with the increase in frame rate, since higher frame rate needs more notifications by quickly using GPU resources.

Figure 2 shows the correlation between GPU interrupt and frame rates for Quake3 with different resolutions, demos, and GPUs; in addition to our target hardware, we also collated the data from the NVIDIA GeForce Go 6150 in HP Pavillion tx2000 tablet PC and the PowerVR SGX 540 in Samsung Galaxy S smartphone [4]. We repeatedly ran demos by changing the maximum frame rate. As shown in the figure, frame rates are linearly in proportion to GPU interrupt rates. This proportional GPU interrupt rate can provide effective feedback that indicates whether a frame rate is being degraded. Since our feedback-driven CPU allocator, which is described in the following section, operates based on such feedback, a GPU interrupt rate is used as an alternative estimate of GPU-accelerated display quality.

The hypervisor can simply monitor GPU interrupt rates because every interrupt is delivered to the hypervisor first. In the case of indirect GPU accesses of a guest domain, VMGL redirects hardware GPU interrupts to the domain as virtual interrupts. With the aid of VMGL, the hypervisor identifies whether incoming GPU interrupts are destined for a guest domain.

5.2 Feedback-driven CPU Allocator

On the basis of the estimated frame rates, the hypervisor can adjust CPU allocation to sustain the expected multimedia quality. The existing VM schedulers typically support proportional-share CPU distribution [7]. By using this property, the hypervisor can dynamically adjust the proportion of CPU allocated to a VM that runs multimedia applications. To this end, we implement a multimedia manager, which is a feedback-driven CPU allocator for multimedia, in the hypervisor. The multimedia manager is VM scheduler-independent and uses a common interface to adjust the CPU share of each VM. Only actual share adjustment is implemented in a VM scheduler. This separation will support the simple integration of future VM schedulers with the multimedia manager.

The multimedia manager periodically performs feedback-driven share adjustment by monitoring the estimated frame rates for each VM. For feedback control, it uses *Desired Frame Rate* (DFR), which indicates the level of expected multimedia quality. Fundamentally, the multimedia manager follows three principles. Firstly, a VM that has a multimedia application seeks to acquire enough CPU share to satisfy DFR. Secondly, DFR is adaptively changed based on frame rates observed on the fly. This principle is essential because we do not rely on user-defined parameters. The DFR is automatically adapted with the reasonable initial values for memory-mapped and GPU-accelerated display. Finally, the multimedia manager performs CPU share adjustment in a static unit. VM-based share distribution cannot use a strict knee model [11] to decide exactly how much share needs to be allocated to achieve a certain rate. Since a VM's share is locally redistributed to each task by the task scheduler, the hypervisor cannot predict the extent to which the multimedia quality is improved after allocating additional share. Moreover, hosted I/O virtualization increases the difficulty of the prediction, because CPU consumption for real I/O is shifted to an IDD.

[4]We used the Android-version Quake3 with a demo (four) found in http://code.google.com/p/kwaak3/.

Algorithm 1 Feedback-driven CPU adaptation for memory-mapped display

```
1: fps ← GetFpsEWMA(multimedia-task)
2: if current-time − multimedia-task.start-time < FAST-START-THRESHOLD then
3:   if fps.current ≤ fps.previous and fps.current < fast-scaled-DFR then
4:     vcpu.share ← vcpu.share × 2
5:   end if
6: else
7:   if fps.current ≤ fps.previous and fps.current < DFR then
8:     if vcpu.prev-increased-share = false then
9:       vcpu.share ← vcpu.share + SHARE-UNIT
10:      vcpu.prev-increased-share ← true
11:    else
12:      vcpu.negative-count ← vcpu.negative-count + 1
13:      if vcpu.negative-count ≥ 3 then
14:        vcpu.share ← MAX(vcpu.share / 2, vcpu.original-share)
15:        vcpu.negative-count ← 0
16:        if fps.current < DFR then
17:          DFR ← fps.current
18:        end if
19:      end if
20:      vcpu.prev-increased-share ← false
21:    end if
22:  else
23:    vcpu.negative-count ← 0
24:    vcpu.prev-increased-share ← false
25:  end if
26:  if fps.current × TOLERANCE-RATIO > DFR then
27:    DFR ← fps.current × TOLERANCE-RATIO
28:  end if
29: end if
```

Based on the principles, we devise the simple algorithm of feedback mechanism based on frame rate changes in response to share allocation. Algorithm 1 describes the mechanism for memory-mapped display; the same algorithm is applied for GPU-accelerated display except fast-start, which is mentioned later. Basically, the multimedia manager periodically increases share when the current frame rate is less than expected (< DFR) without any increase (≤ previous frame rate); line 3 and 7 check this condition. The frame rate is maintained as Exponential Weighted Moving Average (EWMA) [5] in order to reduce the amount of fluctuation. The multimedia manager regards the initial period of a multimedia task as the fast-start phase. When a video playback task starts, it typically requires sudden increases in CPU load for initialization and decoding for display. For achieving responsiveness of share adjustment in the fast-start phase, the multimedia manager exponentially increases share with a short interval (line 3∼5); by default, it uses a 200 ms interval during the fast-start period that lasts for one second after a multimedia application starts. After the fast-start period, the multimedia manager linearly increases share with a one second period in the normal phase (line 7∼28).

In the normal phase, the multimedia manager additionally checks whether there has been no improvement in the frame rate even when share was increased in the previous period. If this is the case, it assumes that the increased share has not positively affected multimedia quality. This situation could arise in the following cases: First, the frame rate is saturated before reaching the DFR. This case occurs when the current DFR is larger than the actual frame rate. Second, the display rate does not rely on the CPU share increase. For example, an audio player could update its progress bar or elapsed time while playing music. Third, CPU-bound tasks in the same VM compete with a multimedia application for a given CPU share of the VM. In the first two cases, the multimedia manager needs to decrease the DFR and the share. Considering the third case, however, drops in the DFR and the share could worsen multimedia quality.

In order to achieve a favorable multimedia quality, the multimedia manager ensures that a VM has multiple chances to receive additional share even when the previously increased share is ineffective, considering the phase of local CPU contention such as the third case; by default, three chances are given. Instead, if the frame rate has not been improved after the multiple chances, its share is reduced by half and the DFR is decreased to the current frame rate (line 14∼18). The drop in the share and the DFR addresses the case of over-estimation of the DFR (the first case) and the case of CPU-independent display rates (the second case). Finally, under-estimated DFR needs to be increased based on observed frame rates. To this end, when DFR is less than the current rate applied by the tolerance ratio, it is updated to that rate (line 27). The original share is eventually returned when a video playback task is no longer scheduled for one second and no video interrupt is raised.

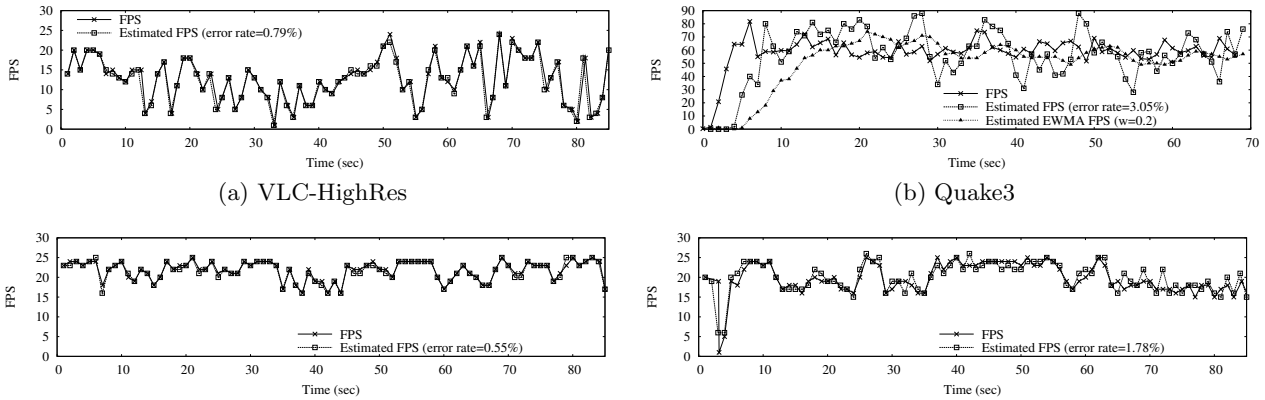
Although DFR is automatically adapted based on observed frame rates, reasonable initial DFR, which is around real FPS, is beneficial for reducing its adaptation period. For example, the initial DFR for memory-mapped display can be preferably set to 20∼30, considering that most movies have a rate of 23.976 to 29.97 FPS. The initial DFR of GPU-accelerated display, however, cannot be simply determined based on common knowledge, since a frame rate is estimated from a GPU interrupt rate. In order to find a good initial value of GPU-accelerated DFR, profiling can be performed using prevalent 3D benchmarks (e.g., Quake3) at the hypervisor installation time. Also, the hypervisor can manage initial DFR values for well-known GPUs.

In our current prototype, we conservatively set the initial values of the multimedia manager. The memory-mapped and GPU-accelerated display DFRs are initially set to 20 FPS and 7700 GPU interrupts per second, corresponding to 70 FPS of Quake3, respectively. While the EWMA weight of the GPU-accelerated display is 0.2 as a result of the large amount of fluctuation shown in Figure 3(b), the weight of the memory-mapped display is 0.8 for agile adaptation of video playback. We set the tolerance ratio to 0.8 for increasing the DFR. This default set of values are reasonable in practice, and furthermore, the DFR is well adapted to the real frame rate on the fly (See Figure 5(b)).

5.3 Scheduler Implementation

Our scheduler-dependent implementation is based on the Credit scheduler. As mentioned earlier, any proportional-share scheduler works with the multimedia manager. The advantage of the Credit scheduler is the low latency support achieved by means of the boosting mechanism, described in Section 4.1. With this mechanism, a VM that has remaining share is eligible to preempt the currently running one when it awakens in response to an incoming event. Accordingly,

[5] $EWMA_t = (1 - w) \times EWMA_{t-1} + w \times Value$, where w is the weight of an observed value and t is the current time



(a) VLC-HighRes

(b) Quake3

(c) VLC-LowRes + Kbuild

(d) VLC-LowRes + DirListing

Figure 3: Accuracy of frame rate estimation (each workload is running in an IDD while competing with a CPU-bound workload in a guest domain)

responsiveness of multimedia workloads is improved as additional share allocated by the multimedia manager increases the chances of boosting.

In hosted I/O virtualization, as shown in the Download case of Figure 1(b), preemption by a guest domain could result in deterioration of the multimedia quality in an IDD. Simply disabling preemption by a guest domain, however, could increase latency when the guest domain runs multimedia applications. In order to address this problem, we divide the original BOOST state into MMBOOST and IOBOOST depending on the types of incoming event. A VM enters the MMBOOST state when it receives a multimedia-related event within the UNDER state while running a multimedia application. In response to other types of I/O events, IOBOOST is given in the original manner of the Credit scheduler. For non-preemption and low latency of multimedia workloads, MMBOOST is assigned a higher priority than IOBOOST and is given when a VM is not only blocked but also waiting on the run queue. As a consequence, a guest domain with normal I/O operations cannot interfere with multimedia workloads in an IDD, since the scheduler disables preemption by a guest domain in the IOBOOST state.

Currently, we assume video, audio, one-shot timer, and virtual framebuffer events as multimedia-related events, since they are typically accompanied by multimedia workloads. Several OSes use a one-shot timer to adjust the timer clock frequency during an idle period in order to achieve energy efficiency [6]. Before entering the idle state, an OS stops its periodic timer and sets a one-shot timer that will expire at the earliest time a process waits for. Since video playback uses timer for timely display of a frame, we classify the one-shot timer event as a multimedia-related event.

5.4 Discussion

Our proposed solution is applicable to non-virtualized native OSes and hosted hypervisors, although the current prototype is implemented in the Xen bare-metal hypervisor. Since our identification techniques use only low-level hardware interaction such as framebuffer writes and GPU interrupts, it can be simply adopted in native OSes. Instead, the multimedia manager should be tailored to native OS schedulers. For example, the Linux CFS scheduler assigns a certain amount of CPU share for each priority [26]. In this case, the multimedia manager can increase the priority of a multimedia task that demands more quality. Similarly, our solution can be extended to a hosted hypervisor, which allows a native OS to run a VM as a task dictated by its OS scheduler.

The current algorithm of the multimedia manager assumes that multiple multimedia tasks that involve both video and audio activities do not populate a single VM at the same time. In order to support multiple multimedia tasks mixed in a VM, the algorithm can be refined. In a multimedia-friendly way, the multimedia manager can give a VM enough CPU share to make its all multimedia tasks satisfy their DFR. To this end, DFR, previously increased share, and negative count should be maintained per task.

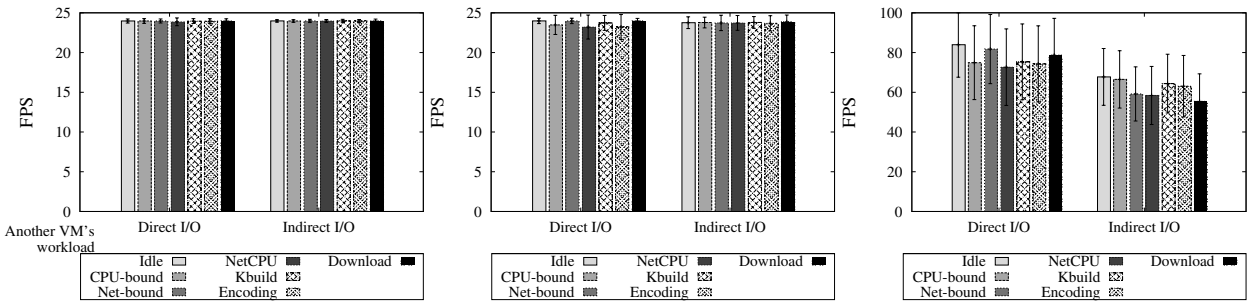
6. EVALUATION

Our multimedia enhancement scheme was evaluated over the same environment that was described in Section 4.2. We call the Credit scheduler with the multimedia manager *Credit-MM*. The initial weight and SHARE-UNIT of the multimedia manager are set to 256, the default weight of the Credit scheduler. Framebuffer write-protection sampling is done for one out of 128 pages.

6.1 Frame Rate Estimation

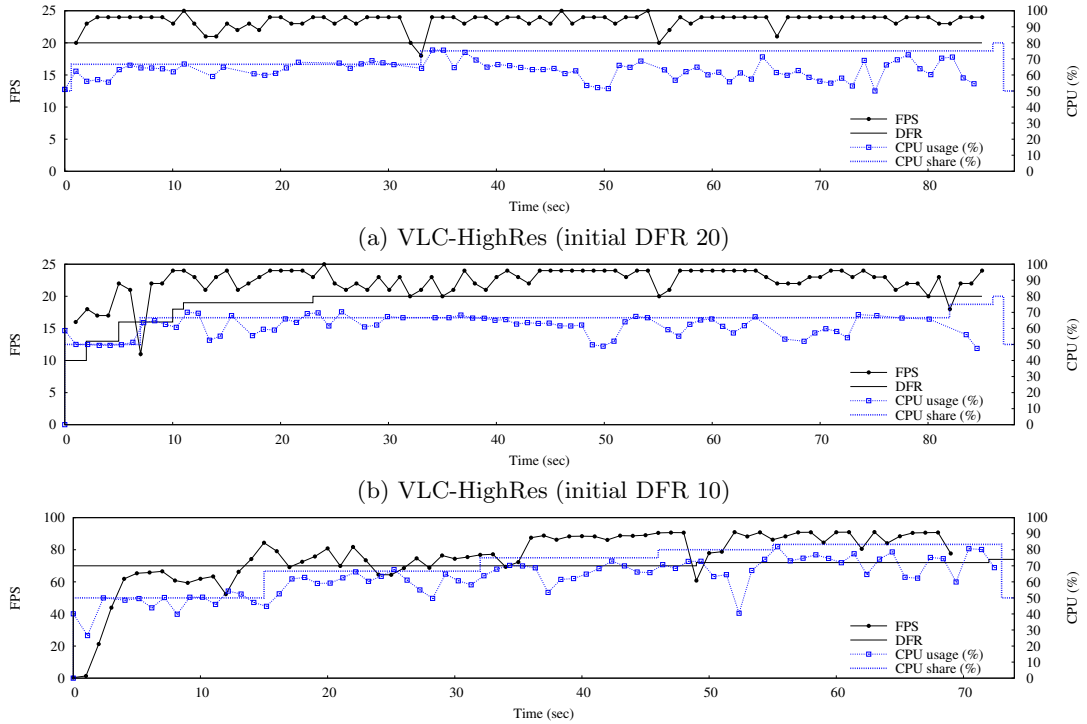
We evaluated how well the frame rate is estimated by the hypervisor. To this end, we compare the frame rate estimated at the hypervisor with the real one measured by a multimedia application. In order to evaluate how the hypervisor tracks frame rates in response to quality degradation, we ran a multimedia application with a CPU-bound workload while disabling the CPU share adjustment. As shown in Figure 3(a), the hypervisor effectively estimates the real frame rate of VLC-HighRes in an IDD with less than 1% error rate. Likewise, Figure 3(b) shows frame rate estimation for Quake3; for comparison, measured GPU interrupt rates are scaled down to frame rates by using the coefficient value (110). As can be seen in the figure, the estimated frame rate derived by video interrupts is fluctuating. The amount of fluctuation is reduced by using the EWMA with a weight of 0.2, whereby the estimated frame rate become

[6]This feature is called dynamic tick or tickless kernel and adopted in Linux.



(a) VLC-LowRes (b) VLC-HighRes (c) Quake3

Figure 4: Average frame rates of Credit-MM



(a) VLC-HighRes (initial DFR 20)

(b) VLC-HighRes (initial DFR 10)

(c) Quake3 (initial DFR 70)

Figure 5: Share adjustment and frame rate changes of Credit-MM

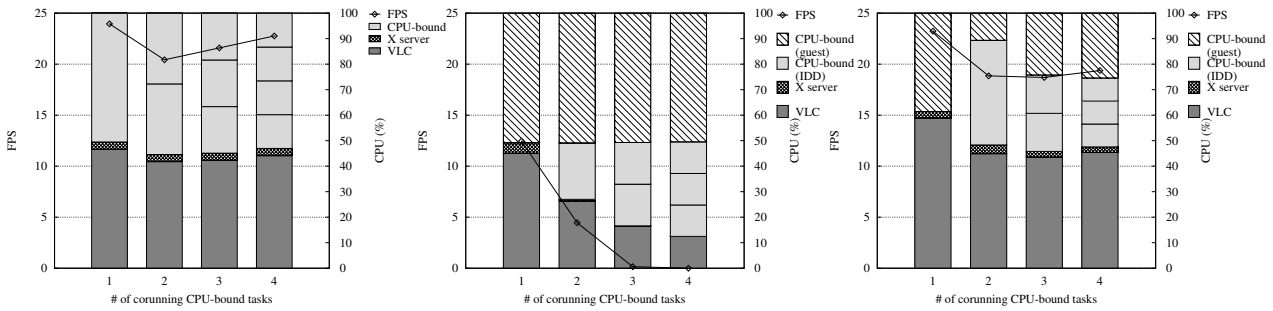
more stable similar to the real one.

For a more stressful test, we ran video playback with a workload that produces display output in the same VM; we used VLC-LowRes in order to place video playback with the other application on a screen. For workloads that generate output, we chose a kernel compilation with console output (Kbuild) and recursive directory listing (DirListing) via the *ls -alR /* command; the *gnome-terminal* is used for displaying the console output. Kbuild produces moderate output while spawning several CPU-bound tasks. DirListing, on the other hand, is very display-intensive with fine-grain communication with the gnome-terminal. As shown in Figure 3(c) and Figure 3(d), frame rate estimation is effective even while the X server multiplexes display requests.

6.2 Multimedia Quality

Figure 4 shows the multimedia quality achieved by Credit-MM. For all cases, the result shows good multimedia quality compared to those shown in Figure 1 by the multimedia-friendly CPU distribution. In particular, VLC-HighRes achieves average frame rates near the maximum FPS and much smaller variance. Our refined boosting mechanism resolves the significant frame drops of the Credit scheduler in the case of VLC-HighRes with downloading, as shown in Figure 1(b), by preventing the normal I/O of a guest domain from interfering with a multimedia application in an IDD.

Figure 5 shows the changes in frame rate, DFR, CPU consumption and CPU share as time progresses. We ran a multimedia workload in an IDD and a CPU-bound on in a guest domain. In Figure 5(a), the multimedia manager appropriately allocates the CPU share so that video playback achieves expected frame rates. As shown in the figure, accurate frame estimation enables agile adaptation of CPU share as soon as the frame rate drops below DFR (at 33 sec). Figure 5(b) depicts the automatic adaptation of DFR on the fly in the case where the initial DFR is set to be equal to 10 FPS, which is much lower than the actual one. As shown in the figure, DFR increases as a higher frame rate



(a) Native Linux (b) Credit (c) Credit-MM

Figure 6: Frame rates and CPU consumption during CPU contention: Each bar represents CPU usage of a task indicated by its label (the bars of CPU-bound tasks in the same VM are filled with the same color).

is observed. By this adaptation, more CPU share is eventually allocated when the frame rate drops below the updated DFR. Figure 5(c) shows similar adaptation for Quake3 competing with a CPU-bound one. Due to the low weight (0.2) of EWMA, CPU share is lazily allocated after the frame rate drops.

6.3 Comparison with Native Linux

We evaluated the multimedia quality and the CPU usage of each task as the CPU-bound tasks were increased. Basically, VLC-HighRes and a CPU-bound task run in an IDD and guest domain, respectively. Note that more CPU-bound tasks in a VM cannot additionally affect the multimedia workload in another VM, since the hypervisor performs VM-level CPU distribution. In this regard, we increased the number of CPU-bound tasks in the IDD in order to cause local CPU contention with the multimedia workload. Since the mediation of local CPU contention relies on the task scheduler in the OS, we compared our scheme with the native Linux. The Linux 2.6.18 version we used includes a general priority-based scheduler (the O(1) scheduler). This scheduler favors interactive and multimedia workloads based on sleeping behavior [6].

Figure 6 shows the results for the native Linux, Credit, and Credit-MM. Note that the native Linux gives VLC-HighRes almost enough CPU to achieve moderate multimedia quality while remaining CPU share is fairly distributed to CPU-bound tasks. As expected, the Credit scheduler, which gives an equal CPU share to each VM, equally partitions the CPU into two VMs. Accordingly, only 50% of CPU share is redistributed to VLC-HighRes and CPU-bound tasks within the same VM, thereby leading to insufficient CPU allocation to VLC-HighRes. Figure 6(b) shows that most frames are dropped when there are more than three corunning CPU-bound tasks. As shown in Figure 6(c), on the other hand, the Credit-MM allocates additional CPU share to the VM that runs VLC-HighRes, which then acquires more CPU so that it can exhibit reasonable frame rates.

We notice two differences in the results between native Linux and Credit-MM. One difference is the CPU distribution for CPU-bound tasks. While Linux fairly allocates available CPU share to CPU-bound tasks, the hypervisor does not ensure fair allocation between CPU-bound tasks that are spread across VM boundaries. This represents an intrinsic problem with the VM-level resource management of the hypervisor. The other difference is the fact that Credit-MM achieves lower frame rates than native Linux. This is caused by the feedback-based adaptation of the multimedia manager. The feedback-based adaptation is a reactive mechanism that makes adjustment after the frame rate eventually drops.

Overhead(%)	All pages	Sampling		
		1/8 pages	1/32 pages	1/128 pages
VLC-LowRes	4.95	1.10	0.54	0.58
VLC-HighRes	3.91	1.04	0.69	0.33

Table 2: CPU overheads for different sampling ratios

6.4 Overheads

Finally, we evaluated the computational overheads of our scheme. Since the frame rate estimation that accompanies page faults is the most expensive part of our scheme, we measured overheads as varying sampling ratio. In order to show the framebuffer tracking overheads without share adjustment, we measure the CPU usage of a vCPU where a video playback workload is running with our all mechanisms enabled except the CPU adjustment. As shown in Table 2, sampling-based write protection entails negligible overhead (0.3∼1%). Even when all framebuffer pages are tracked via write-protection in the extreme case, less than 5% overhead was observed. Note that the CPU overhead actually relies on the frame rate and the CPU requirement of video playback. Nevertheless, the result for the extreme framebuffer tracking implies that the overhead of our scheme accounts for small portion of entire CPU usage of video playback.

7. CONCLUSIONS

Machine virtualization creates an additional level of resource distribution underneath traditional OSes. Since the available resources an OS believes to be real are actually distributed by the hypervisor, many OS optimizations for certain workloads may be invalidated if the hypervisor is oblivious to the internals. This paper describes how the hypervisor provides multimedia-friendly CPU distribution without the cooperation of the upper layer. Since our entire mechanism is based on low-level interactions such as framebuffer access and interrupt monitoring, it can be readily adopted by other hypervisors and even OSes. We plan to extend our scheme to utilize multicore architecture through multimedia-aware load balancing and CPU share adjustment per vCPU.

8. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0000371).

9. REFERENCES

[1] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert. Intel virtualization technology for directed I/O. *Intel Technology Journal*, 10(3):179–192, 2006.

[2] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: a virtual mobile smartphone architecture. In *Proc. SOSP*, 2011.

[3] C. Augier. Real-time scheduling in a virtual machine environment. In *Proc. JRWRTC*, 2007.

[4] S. A. Banachowski and S. A. Brandt. The BEST scheduler for integrated processing of best-effort and soft real-time processes. In *Proc. MMCN*, 2002.

[5] 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 *Proc. SOSP*, 2003.

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

[7] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three CPU schedulers in Xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, 2007.

[8] M. Dowty and J. Sugerman. GPU virtualization on VMware's hosted I/O architecture. *SIGOPS OSR*, 43(3):73–82, 2009.

[9] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Proc. SOSP*, 1999.

[10] G. Dunlap. Xen scheduler. In *Xen Summit North America*, 2010.

[11] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Process prioritization using output production: Scheduling for multimedia. *ACM TOMCCAP*, 2(4):318–342, 2006.

[12] K. Fraser, S. H, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *Proc. OASIS*, 2004.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proc. SOSP*, 2003.

[14] 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 *Proc. VEE*, 2007.

[15] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: Qos monitoring and performance profiling tool. Technical report, Hewlett-Packard, 2005.

[16] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim. Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones. In *Proc. CCNC*, 2008.

[17] M. B. Jones, D. Roşu, and M.-C. Roşu. CPU reservations and time constraints: efficient, predictable scheduling of independent activities. In *Proc. SOSP*, 1997.

[18] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. In *Proc. USENIX ATC*, 2006.

[19] D. Kim, H. Kim, M. Jeon, E. Seo, and J. Lee. Guest-aware priority-based virtual machine scheduling for highly consolidated server. In *Proc. Euro-Par*, 2008.

[20] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee. Task-aware virtual machine scheduling for I/O performance. In *Proc. VEE*, 2009.

[21] H. Kim, H. Lim, J. Jeong, H. Jo, J. Lee, and S. Maeng. Transparently bridging semantic gap in cpu management for virtualized environments. *JPDC*, 71(6):758 – 773, 2011.

[22] H. A. Lagar-Cavilla, N. Tolia, M. Satyanarayanan, and E. de Lara. VMM-independent graphics acceleration. In *Proc. VEE*, 2007.

[23] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik. Supporting soft real-time tasks in the xen hypervisor. In *Proc. VEE*, 2010.

[24] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proc. OSDI*, 2004.

[25] B. Lin and P. A. Dinda. VSched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proc. SC*, 2005.

[26] R. Love. *Linux Kernel Development*. Addison-Wesley Professional, 3rd edition, 2010.

[27] A. Mann. Citrix and Intel deliver client virtualization. Technical report, Citrix Systems, Inc., 2010.

[28] C. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: operating system support for multimedia applications. In *Proc. ICMCS*, 1994.

[29] J. Nieh and M. S. Lam. A SMART scheduler for multimedia applications. *ACM TOCS*, 21(2):117–163, 2003.

[30] N. Nishiguchi. Evaluation and consideration of the Credit scheduler for client virtualization. In *Xen Summit Asia*, 2008.

[31] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. In *Proc. VEE*, 2008.

[32] B. Paul. *Introduction to the Direct Rendering Infrastructure. http://dri.sourceforge.net/doc/DRIintro.html*, 2000.

[33] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. OSDI*, 1999.

[34] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. USENIX ATC*, 2001.

[35] Y. Xia, C. Yang, and X. Cheng. Pas: A preemption-aware scheduling interface for improving interactive performance in consolidated virtual machine environment. In *Proc. ICPADS*, 2009.

[36] H. Zheng and J. Nieh. RSIO: automatic user interaction detection and scheduling. In *Proc. SIGMETRICS*, 2010.