vSIP: Virtual Scheduler for Interactive Performance

Yan Sui Chun Yang Ning Jia Xu Cheng
Department of Computer Science and Technology, Peking University, Beijing, China
{suiyan, yangchun, jianing, chengxu}@mprc.pku.edu.cn

ABSTRACT

This paper presents vSIP, a new scheme of virtual desktop disk scheduling on sharing storage system for user-interactive performance. The proposed scheme enables requests to be dynamically prioritized based on the interactive feature of applications sending them. To enhance user experience on consolidated desktops, our scheme provides interactive applications with priority requests, which have less latency in accessing storage than requests from non-interactive applications sharing the same storage. To this end, we devise a hypervisor extension that classifies interactive applications from non-interactive applications. Our framework prioritizes the requests from these applications and limits the requests rate. Our evaluation shows that the proposed scheme significantly improves interactive performance of storage-sensitive application such as applications launch, Web page loading and video cold playback, when other storage-intensive applications highly disturb the interactive applications. In addition, we introduce a guest OS information transfer method, hence the efficiency and accuracy of the identification of interactive applications can be further improved.

CCS Concepts

•**Information systems** → *Storage virtualization;* •**Software and its engineering** → *Virtual machines;*

Keywords

Hypervisors, Management of Virtual Environments, Virtual I/O, Virtual Storage

1. INTRODUCTION

Virtual desktop infrastructure (VDI) [1], which is a type of virtualized environments deploying individual desktops in server farms, has drawn great attention in recent years. The desktop consolidation manages many desktops, and shares CPU, memory, network and storage resource among them. This kind of managing and sharing can lead to the significant cost savings due to efficient resource utilization and savings in management from the high density of desktop consolidation. However, so many VMs (Virtual Machines) sharing storage resource causes great rising pressures on storage services. In VDI environment, the performance of storage system becomes bottleneck for many applications running in virtual desktop, especially for the interactive applications. Interactive applications usually suffer the large degradation in performance or become unusable, because of the long latency from storage [5, 17].

CF'16, May 16-19, 2016, Como, Italy
© 2016 ACM. ISBN 978-1-4503-4128-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2903150.2903178

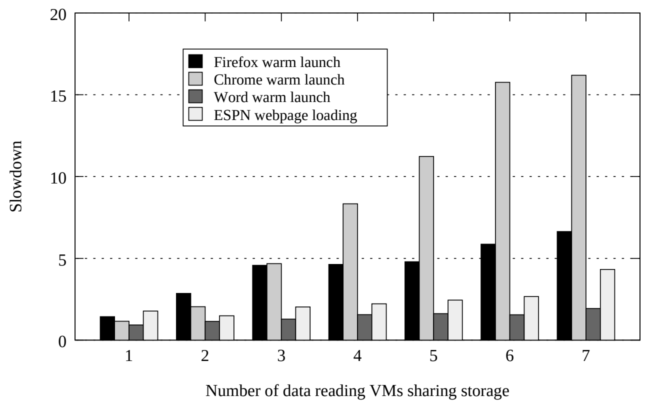

Figure 1: The significant performance loss of interactive applications under different loads on shared storage.

Interactive applications are widely used in desktops and their performances are important for users' experience. The research [3] shows that the performance metrics for interactive applications is latency not throughput for the conventional methodology of system performance measurement. In virtualization environment, there are some works on the interaction-ware CPU scheduling in VDI hypervisors, such as vAMP [11], which dynamically adjusts interactive applications and schedules them to the vCPU having more power to reduce the latency and improve users' experience. Many interactive applications in desktops are storage-sensitive, as that the latency of these applications mainly depend on the performance of storage. Storage schedulers in hypervisors provide limited support for consolidated desktops with respect to interactive performance, which introduces significant performance loss to these applications, as shown in Figure 1.

For some application scenarios, such as cold launch or multimedia cold playback, it is obvious that the performance mainly depend on the performance of storage. In some other storage-sensitive application scenarios, we cannot find the need of their storage accessing easily. For example, the warm launch of applications and Web page loading seem not to access storage, because the data has been loaded into memory or the data is accessed from network. However, the load of shared storage still significantly influences their performance, as shown in Figure 1. In this paper, we name the former sensitivity of storage as explicit storage sensitivity and the latter as implicit storage sensitivity.

We propose Virtual Scheduler for Interactive Performance (vSIP), which is capable of dynamically adjusting the requests form interactive application in order to surmount the limitation of hypervisor schedulers regarding interactive performance. Using the knowledge of applications type who send the requests, the disk schedulers in hypervisors can prioritize the requests from interactive applications and reduce the queue latency of these requests. In this way, the respond latency of interactive applications decreases and the users enjoy better experience. To this end, we introduce a hypervisor extension that guides the hypervisor disk scheduler to prioritize the requests from interactive applications, based on the estimation of workload characteristics.

The primary role of our hypervisor extension is to infer which requests come from interactive applications. It is a great challenge for the hypervisor to accurately identify these requests, because the hypervisor can access only a small set of hardware interactions. We use hypervisor-level task tracking [10] to match the requests and the applications sending them. In our extension, we need to identify the storage-sensitive interactive applications, so that the traditional methods of identification based on CPU loads are not suitable. We devise the identification of interactive applications on the basis of user input event and storage request pattern. If an application usually sends more requests after user input events than before input events, we would mark this application as an interactive application. At last, our hypervisor disk scheduler use these information to schedule storage requests, in the aware of interactive performance.

A simple method of scheduling requests from interactive applications is to prioritize all these requests to guarantee the latency and the performance of interactive applications. However, this simple method has two disadvantages. 1) If the throughput of requests having priority is very high, this method could lead to starvation of other applications. 2) The user of non-interactive application could try to lie about the interactive behaviour of his application to get performance benefit. To solve these problems, we limit the requests rate of priority requests and total requests from each VM to ensure that there is no starvation or applications/users lying of interactive features for benefit (such as clicking mouse or keyboard continuously to get better performance).

The proposed scheme is evaluated based on a KVM/SPICE VDI environment. Our evaluation shows that vSIP decreases 38% or 48% cold launch time of interactive applications mixed with one data reading VM or one operating system (OS) booting VM, compared with using Linux default CFQ (Completely Fair Queuing) scheduler. Compared with running interactive applications alone, our scheme increases only 9% or 17% cold launch time of interactive applications. For multimedia playback, vSIP guarantees a small drop frame rate in all experiments, while CFQ scheduler introduces a drop frame rate up to 39.1%, when there are five VMs running date read. For warm launch and Web page loading, vSIP also guarantees small performance loss, when the interactive applications are mixed with data reading. When the interactive applications are mixed with system OS booting, vSIP introduces some performance loss compared with interactive applications running alone. Because there are some write and flush operations in the system OS booting and such operations introduce great latency to the interactive requests, even when they get a high priority.

The specific contributions of our work are as follows:

- We propose a method of identifying interactive applications, which are storage-sensitive. This method identifies storage-sensitive interactive applications better than other systems which focus on CPU-intensive interactive workloads.
- We show a scheme of virtual desktop disk scheduler, which lowers the queue latency of requests from interactive applications and considers the problem of lying and starvation, which other systems do not do.
- We point out that may interactive applications are storage-sensitive and show their performance improvement under our scheme.

Next, in Section 2 we present an overview of background and discuss related work. Then, we show the details of vSIP in Section 3. We present the experiment result in Section 4. Finally, we conclude in Section 5.

2. BACKGROUND AND RELATED WORK

This section describes storage-sensitive applications in desktops and existing disk scheduling algorithms in virtualization environment. We then argue why the hypervisor scheduler should be enhanced for user-interactive performance in the consolidated desktops.

2.1 Storage-Sensitive Scenarios in Desktop

There are many storage-sensitive scenarios in desktop environments. We can classify the sensitivity into two classes, explicit sensitivity and implicit sensitivity. The former comes from the explicit logic need for storage accessing, such as files cold loading or applications cold launch. The latter sensitivity is implicit and it is hard to find the logic for storage accessing from intuition. However, the performance of scenarios having this sensitivity still depend on the storage system, such as applications warm launch or Web pages loading.

For the explicit sensitivity, it is obvious that the performance of cold launch or loading depends on the performance of storage, because they usually spend most of the execution time on accessing storage. In the implicit sensitive scenarios, the time spent on storage accessing contributes a less proportion of total execution time. However, these scenarios are still sensitive to storage performance. The research [3] shows that the performance metrics of interactive applications should be latency, but not throughput. For example, a user browses Web pages for ten minutes. Although the user just clicks mouse to load a new page several times and these interactions take total several seconds, the time cost in interactions is still very important for the user's experience.

If an interaction leads to implicit storage accessing and the bad storage performance increases the respond latency, the user experience would decrease significantly.

Warm launch of applications is a typical implicit storage sensitivity scenario. Intuitively, warm launch causes no storage accessing, because all data needed has been loaded into memory. However, many applications still read data from storage in warm launch. We choose five widely used interactive applications including Firefox, Chrome, Word, Excel and PowerPoint to show this implicit storage sensitivity in warm launch. We build several VMs, one running these interactive applications, others running storage-intensive workloads. These VMs share the storage and we make sure that other resources are sufficient, such as CPU or memory. We choose two storage-intensive workloads, data reading and OS booting. The relative warm launch time of the above five interactive applications mixed with different overhead is shown in Figure 2 and Figure 3, using the warm launch time when running alone as baseline. We find that the warm launch time increases, when there are more overhead VMs.

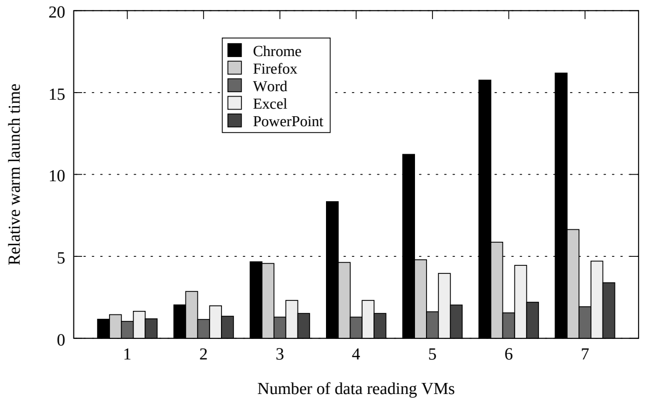

Figure 2: The relative warm launch time of interactive applications mixed with data reading. The more overhead VMs, the poorer performance.

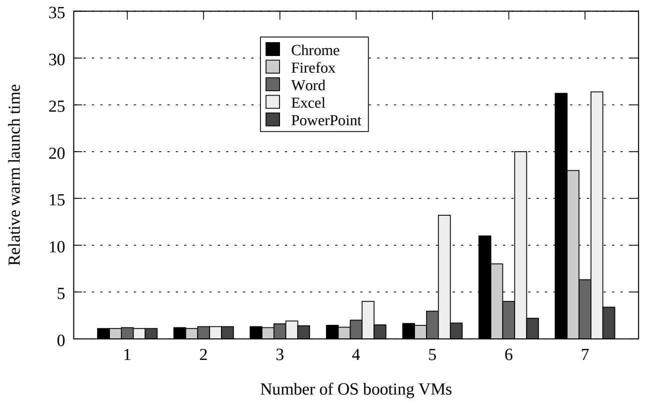

Figure 3: The relative warm launch time of interactive applications mixed with OS booting.The more overhead VMs, the poorer performance.

Web pages loading is another typical implicit storage sensitivity scenario. If the browser has been loaded into memory and the Web page is accessed from network, there seem to be no need to access the storage. We choose Bbench [8], which loads Web page and measures the loading time automatically to measure the performance of Web pages loading. The same as above, the VM running web browser shares storage with the VMs running overhead workloads including data reading and OS booting. An interactive session of Web browsing from Bbench consecutively visits ten Websites: Amazon, BBC, CNN, Craigslist, eBay, ESPN, Google, MSN, Slashdot, and Twitter. We relaunch the Web browser before each time Bbench running. We deploy the Web page data of Bbench in a special virtual disk and intercept the requests to this disk to make sure all data of Web pages has been loaded into memory. The relative results of geometric average of loading time mixed with different overhead workloads are shown in Figure 4. We also find that the Web page loading time increases, when there are more overhead VMs.

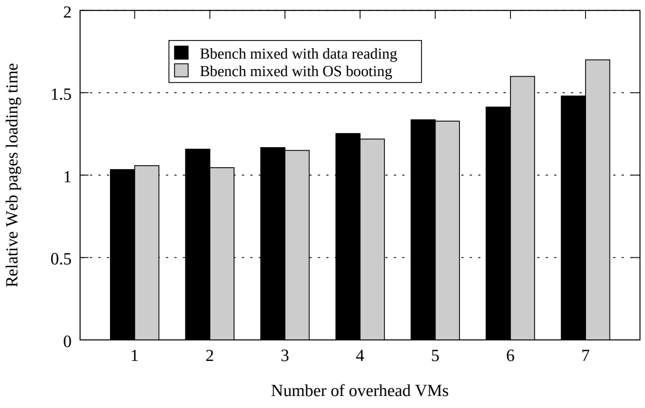

Figure 4: The relative Web page loading time mixed with different overhead workloads. The result comes from the geo-average of Bbench. The more overhead VMs the poorer performance.

We collect traces of these implicit storage sensitivity scenarios. Figure 5&6 shows the statistical result of read requests and read data from ten traces of Word warm launch and Chrome Web warm browsing of Bbench. We find that these scenarios actually send storage read requests.

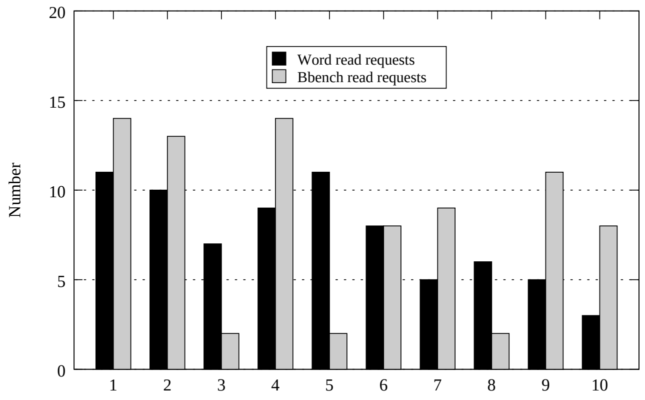

Figure 5: The statistical result of read requests from ten traces of Word warm launch and Chrome Web warm browsing from Bbench.

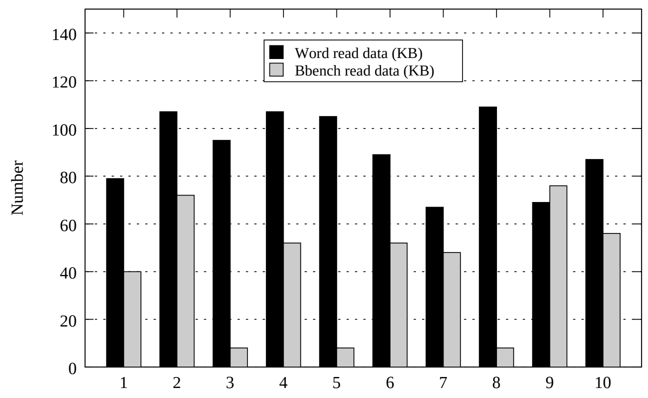

Figure 6: The statistical result of read data from ten traces of Word warm launch and Chrome Web warm browsing from Bbench.

The details of these experiments setup are the same as the experiment setup in Section 4, which we will describe latter in this paper.

2.2 Disk Scheduling Algorithm in Virtualization Environment

Strong isolation is one of the most important needs for the hypervisor disk scheduler under the consolidation of VMs having diverse requirements. Hence, the hypervisors usually use QoS-based I/O resource allocation methods, which can be divided into two broad areas. First is the class of proportional bandwidth fairness algorithms that provide proportional allocation of I/O resources, such as Stonehenge [9] and Aqua [18]. Second is the class of QoS algorithms that provide support for latency-sensitive applications along with proportional sharing. These works include SMART [15], mClock [7] and others. These methods regard VMs as black box and schedule requests in the granularity of VM in virtualization environment. The requests from one VM are regarded as one requests flow and share the same configuration or scheduler parameters. Some other researches trace the thread running in guest OS [10] or transfer semantic information directly [13]. These methods regard VMs as white box and schedule requests based on semantic information from guest OS, however, they do not involve the identification of interactive applications.

2.3 Why Hypervisor Disk Scheduler Supports for Interactive Performance

There are many interactive-aware scheduling researches in native OS, which relies on allocating more resource (such as CPU slice) to interactive applications to improve their performance and users' experience. In order to distinguish interactive applications from others, researchers have proposed various classification methods such as user-driven [14, 22], application-directed [15], middleware-assisted [4], and OS-level [23] schemes.

The high density of desktop consolidation leads to the scenario that a large number of VMs share storage resource. The requests from different applications running in the same VM are scheduled into one requests queue and regarded as one single flow in the hypervisor disk scheduler. This two level scheduler construction destroys the guest OS interaction-aware scheduling results. Even the guest OS finds the requests from interactive applications and prioritizes them. In the hypervisor scheduler, a request from the non-interactive running in another guest OS can still block the interactive requests, although these requests have high priority in the purpose of improving user experience in guest OS.

3. VIRTUAL SCHEDULER FOR INTERACTIVE PERFORMANCE

This section presents vSIP, which dynamically identifies the requests from interactive applications and prioritizes these requests. As we describe above, prioritizing requests can provide lower latency and better users' experience. In designing vSIP, we consider several challenges with regard to the different characteristics of the hypervisor from OSes and disk resource from CPU resource.

Firstly, requests from one VM can come from different applications with different interactive features. Secondly, available information for identifying workload characteristics is restrictive at the hypervisor. Thirdly, a large number of virtual machines share storage resource in some cases, hence the throughput of interactive applications could be so high that prioritizing them may cause starvation of other applications or VMs. Fourthly, storage-sensitive interactive applications have special behaviour features. Lastly, the users can always lie about the interaction of their applications under any heuristic method of identification. We need to ensure that the users have no motivation to lie. That is, the users cannot get benefit by lying about their interactive features.

Based on these issues, we have the following design goals in mind:

1. **A simple hypervisor extension aiding the hypervisor disk scheduler.** Our hypervisor extension can identify interactive applications based on user input event and storage request pattern.

2. **A requests rate limiter.** We build a requests rate limiter to avoid the starvation of non-interactive applications or the motivation for non-interactive application user lying.

3. **An optional semantics transfer method.** As an option, we use virtio [19] to transfer the semantics information between guest OS and hypervisor. For example, we transfer the program names of applications running and use these information to identify interactive applications based on a white list.

3.1 Which Applications are Interactive

One of the biggest challenge for vSIP is the identification of interactive applications. To put it simply, any application getting input from human users and giving output to human users, who waits for the output, is an interactive application [2]. The time from the input to the last output is the respond time of the interactive application, which is usually short. When a user only clicks the mouse, a desirable latency is less than 100ms [20]. If the time is very long, the application would not be an interactive application. For example, when a user starts a disk scanner, he would expect a respond time of tens of minutes. This idea is also implicitly used in existing interactive-aware scheduling researches [14, 22, 15, 4, 23, 11]. They usually mark the applications having shorter latency as interactive applications and the applications having longer latency as non-interactive applications or background applications. We also mark these applications in this way in the reset of this paper, as that a Web browser is an interactive application and a disk scanner is a non-interactive application.

We identify interactive applications based on the above definition. Before discussing our method of identification, we look back possible ways proposed in previous work.

3.1.1 Limitations on Alternatives

The I/O-bound task tracking, which is first proposed in [12], marks the task having short time quanta in response to I/O events as an I/O-bound task. This method of identification is based on the traditional heuristic of distinguishing I/O-bound and CPU-bound tasks. Antfarm [10] makes it easy to track task switch, which makes this method a viable solution for the hypervisor to identify interactive tasks. However, modern interactive applications are not always I/O-bound tasks having short time quanta, which leads to the invalidation of this method [11]. Even worse, most of the storage-sensitive applications would be marked as I/O-bound applications in some cases.

Another way to identify interactive applications is based on the user input events and inter-process communications (IPCs) [23]. This method tracks the user input event and its delivering trace to identify all tasks involved in an interaction event. The identification result of this method is accuracy and comprehensive, however, this technique heavily relies on OS-level structures and information for tracking various types of IPCs, which leads to a considerable modification to guest OS.

vAMP [11] uses user input event and CPU load to find interactive tasks. This method detects a set of tasks that have been generating non-trivial CPU loads before a user input event as background tasks and non-background tasks are regarded as potentially interactive workloads. This technique is very powerful in CPU-intensive cases, however, it is unsuitable for the identification of storage-sensitive interactive applications. Because it is hard to estimate the CPU loads of storage-sensitive applications. For example, we compare the CPU loads of three applications including data reading, file MD5 generating and file encryption compression. These three applications are all storage-sensitive and non-interactive, while they respectively have low CPU loads, common CPU loads and high CPU loads. Hence, that method of identification based on CPU loads would be confused in such cases.

3.1.2 Our Method

Considering the strength and limitation of the previous works and the discussion above, we propose *storage-sensitive interactive applications identification* based on user input events and storage request pattern. The proposed scheme identifies the potentially interactive feature of requests at the moment of a user input event and then regards the interactive applications using these information. To this end, our scheme counts the storage requests from each application before and after the user input events, respectively. If a request occurs after the input event, it would be marked as a potentially interactive request. Vice versa, we mark the requests before input event as potentially non-interactive requests. If an application has much more potentially interactive requests than potentially non-interactive requests, we would mark it as an interactive application and the requests from it would be scheduled as interactive requests.

The rationales behind the proposed scheme are as follows: Firstly, a user input event typically initiates a user-interactive workload, so-called an *interactive episode* and interactive applications usually send requests in interactive episodes. Secondly, non-interactive applications send requests without regarding user inputs. Finally, identifying interactive applications based on the comparing of two kinds of requests introduces better adaptability and stability than methods based on the count of one kind of requests.

In order to identify interactive applications based on disk request pattern, we need to match the request and the application sending it in VMM. Antfarm [10] is the most widely used method in tracking the threads in guest OS, which monitors CR3 to track threads switches. We use this method to match storage requests and applications in our framework.

The algorithm of the identification of interactive applications is illustrated in Figure 7 and performs as follows. Requests are marked as potentially interactive requests within a certain time window, named *post-input period*, after the user input event. We also mark the requests before the user input event as potentially non-interactive requests within *pre-input period*. Usually we set equal length to these two periods. Then we count the two types of requests from each application, and compare the counts of them to identify interactive applications. As the example in Figure 7, although application A has more potentially interactive requests than application B, we still identify application B as an interactive application. Because B has much more potentially interactive requests than potentially non-interactive requests, while A has the same counts of two types requests.

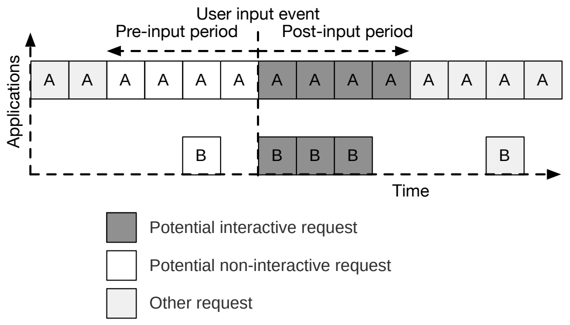

Figure 7: The identification of interactive applications based on user input and storage request pattern. Although application A has four potentially interactive requests more than application B who has three, we still identify application B as an interactive application. Because B has much more potentially interactive requests than potentially non-interactive requests(one), while A has the same count of two types requests(four).

3.2 Implementation and Analysis

Except the method of identification described above, the implement of vSIP has another two components: a pre-VM rate-limiter providing control to storage accessing and an optional guest OS semantics information transfer method.

3.2.1 Requests Rate Limiter

Our framework prioritizes the requests from interactive applications, however, there are two disadvantages in prioritizing all of these requests. The first one is the starvation of non-interactive applications, which occurs when the total throughput of interactive applications is higher than the system offerings or some fully backlogged non-interactive applications are misidentified as interactive applications. The second one is lying. That is, the users of non-interactive applications benefit by lying about the interactive feature of their applications(such as clicking mouse or keyboard continuously). To solve the above two problems, we build a requests rate limiter, inspired by qjump [6].

We give each virtual machine two parameters *weight* and *cap*, while weight limits the priority requests rate from the virtual machine and cap limits the total requests rate. We divide the entire execution process into several periods and predict the system throughput offerings in next period based on last period. For a virtual machine i, we use the prediction of system throughput, weight and cap to compute the total requests number limit and the priority requests number limit, which are named as W_i and C_i. If a VM has sent high priority requests more than W_i, it could only send common priority requests. If a VM has sent requests more than C_i, it could only send low priority requests.

Now we show how this algorithm solves the disadvantages described above. We assume that the system throughput offerings is constant, marked as max, and interactive applications care about requests latency, while non-interactive applications care about requests throughput. In the case of n VMs having the same share, we configure all *cap* as 10 and *weight* as 5 , which are regarded as default set in our framework. Hence, each VM gets max/n as C_i and $0.5*max/n$ as W_i. If a VM sends requests under common priority all the time, its throughput would match the equal share. If it tries to send high priority requests continuously, it would send half of total requests with high priority and other requests have common requests. There would not be too many requests having high priority under the limit of W_i, hence the performance of interactive applications are guaranteed. We use lower priority to limit requests rate to full the system resource utilization.

In this algorithm, the throughputs of non-interactive applications are guaranteed to avoid starvation. Even a non-interactive application is identified as interactive application, its throughput would not increase and the performance of other applications would not decrease. The false negatives, which misidentify interactive applications as non-interactive applications hurts the framework. If an interactive applications is misidentified, its performance would decrease significantly in our framework. Under our method this misidentification is rare.

3.2.2 Guest OS Information Transfer Method

As an option, we build a guest OS information transfer method with virtio [19]. Virtio is a widely used paravirtualized drivers for KVM/Linux and is chosen to be the main platform for I/O virtualization in KVM. We modify the virtio driver in guest OS and KVM to transfer extra information.

Using this method, we can transfer the names of applications running in guest OS to hypervisor and use a "white list" of application names to identify interactive applications. The rationales behind the "white list" are as follows: in a certain VDI environment, most of the applications running in guest OS are the same and these applications instances usually have the same interactive feature. Hence, we collect the identification results and add the credible interactive results to the "white list". We judge whether a interactive result is credible based on two factors: the count of identifications and the rate of interactive identifications. If there are many identifications on an application and the rate of interactive identifications is enough high, we should trust the interactive result and add it to the "white list".

Although most interactive applications typically introduce disk requests after user input events, some interactive applications send disk requests before user output events. A typical type of interactive applications sending disk requests before user output events is multimedia application, which is hard to be identified using our mechanism described above. Even worse, the widely used multimedia redirection (MMR) technology in VDI makes the problem more complex. In the VDI environment using MMR, VDI servers send the multimedia steams to user clients and these steams are decoded and played back in clients. Then the behaviour of server redirector is the same as a file reader and it is hard to excavate the interaction of the redirector.

Considering the above discussions, we use the option to identify multimedia applications based on the command name of applications. Of course, it is easy for the users to rename their applications, however, our requests rate limiter would eliminate the motivation of such lying. Besides multimedia applications, other interactive applications are identified by our heuristic method in this paper.

3.2.3 Comparing with VMM Schedulers Regarding VM as Black Box

To guarantee the performance of storage-sensitive interactive applications, a solution from intuition is to limit the requests rate of each VM. Limiting the requests rate decreases the out-standing requests number and the queue latency of requests. However, this method just alleviates the problem not solves it. At the same time, simply limiting requests rate introduces storage resource wastes, which is intolerable in storage-intensive environment.

Another solution is to use the schedulers that support reservations and limits, such as mClock [7]. However, there are three disadvantages in using this kind of schedulers. Firstly, no scheduler can handle all kinds of requests burst under reservations. In other words, for any scheduler in a constant configuration, there are always some interactive request patterns it cannot schedule under reservation. Secondly, these methods schedule requests in the granularity of virtual machines, which causes the loss of semantic information in the guest OS, as that the requests from applications having different features are mixed. The requests from interactive applications and requests from non-interactive applications are scheduled as the same. Third, these methods usually work based on the different uses of VMs, such as desktop, transaction or data copying [7]. The uses of VMs are fixed and managers can model them easily. However, all VMs supporting virtual desktops are the same to manangers in a VDI environment. It is hard to model the workload running in each VM.

3.2.4 Other Analysis

Some interactive applications pre-fetch data from storage. If the pre-fetching works perfectly and all needed data are loaded into memory before user input, our method would classify these interactive applications into non-interactive applications, which is a misidentification. However, this misidentification leads no harm, because the application needs not to access storage between users input and system output. When the pre-fetching does not work perfectly and user interaction event leads to some storage accessing, our method would work well.

One alternative for identification is to correlate the task IDs of input events with the task IDs of storage request events. That is, when a thread just receives an input event, and then immediately sends a disk read request, this disk read request should be processed at a higher priority. This method has two disadvantages. Firstly, it is hard for VMM to match the input event and the task handling the event. We can do the match in guest OS, however, it would introduce modification to guest OS. Secondly, this method can still lead misidentification. Some non-interactive applications handle user input, such as the user clicking a running anti-virus scan, while some interactive applications do not handle user input, such as Xorg.

In some interactive applications, one thread is doing input-output processing while another thread is doing background bookkeeping that needs storage accesses. Our method cannot handle the case that the bookkeeping thread works all the time, however this case is rare. If the thread only does bookkeeping sometimes, our statistics-based method would cover this "unusual behavior".

As discussed in split-level I/O scheduling [21], setting the priority of a CFQ-like disk scheduler may not be sufficient to lower the latency of an interactive request. We will improve at this point in the future work.

4. EVALUATION

The vSIP hypervisor extension is implemented based on KVM, a Linux module for virtualization. The testbed is under Centos 6.5 using kernel 2.6.32 and uses CFQ as the default disk scheduler. Storage requests from guest OS are handled by QEMU (version 1.2.0) running in the host Linux, while the user input events are handled by SPICE-UI component in QEMU. Once QEMU handles a disk request or a user input event, it notifies the vSIP hypervisor extension. vSIP identifies the type of application sending the request and sets priority to the request for CFQ scheduler in host OS.

The prototype is installed on Inspur Server NF5280 having four Intel quad core processors E5620 and 64GB DDR2 memory. The storage subsystem is comprised of two SATA hard devices, one of which is dedicated to OS and the other one is used as to store VM images. We use Ubuntu 12.04 with Linux kernel 2.6.32 and Windows 7 as guest OSes, and give each VM 1 vCPU and 4GB memory. We choose several interactive applications in Linux and Windows, based on the definition of interactive applications and other works [11, 16]. A SPICE client run on a separate machine connected through a 100Mbps Ethernet switch.

4.1 Measurement Methodology and Parameters

The users of VDI usually connect to the server using remote desktop protocol, which is SPICE in our KVM-based desktop environment. The performance of remote desktop is usually measured by slow motion method [16], especially for the performance of interactive applications. However, our framework just involves VDI servers not the remote display protocol nor clients. Hence, we only show the results from traditional measurement at server side. We also measure interactive applications launch time using slow motion and compare the results with the results from server side measurement. The small difference shows that the network, protocol and clients adds little to the response time of users in our environment.

Considering our system consists only one SATA hard device to store VM images, we limit the max number of VMs as eight, although there are enough CPU and memory resources for more VMs. If the system has stronger storage devices like RAID, SSDs or NVM, the additional latency between the VDI server and the users would still be negligible compared to storage latency, because the number of VM would increase under stronger storage devices.

We choose cold launch and video cold playback as representative interactive workloads having explicit storage sensitivity. We measure the performance of these workloads in guest OSes running Ubuntu 12.04. For cold launch, we measure launch time as performance metrics. We evaluate the multimedia workload using mplayer, an open-source media player. For a video playback application, the performance metric is displayed frames per second (FPS). Since a media player drops frames when it cannot meet the defined rate of a video, we measure the rate of drop frames at the server side in order to evaluate the impact of our scheme.

We choose warm launch and Web page warm loading as representative interactive workloads having implicit storage sensitivity. We measure the performance of these workloads in guest OSes running Windows 7. We measure the warm launch time using AppTimer and measure the page warm loading using Bbench.

We found two types of overhead workloads. One is data reading, the other is OS booting. The former sends read request continuously, while the latter has the request pattern including read, write, flush and idle.

All evaluations are repeatedly run 20 times and we present the average results with standard deviations in this paper. Besides disk resource, other resources such as CPU or memory are guaranteed to be the same and sufficient in all evaluations. We try to allocate more CPU and memory to each VM and such allocation introduces no performance benefit in all of our experiments.

Table 1: The vSIP parameters, their roles and default values used in the evaluations.

Parameter	Role	Default
Pre-input period	Identification of Interactive task	4sec
Post-input period	Identification of Interactive task	4sec
Inter_thresh	Identification of Interactive task	60%
Weight	Rate-limiter	5
Cap	Rate-limiter	10

Table 1 summarizes the parameters, their roles and default value for the vSIP hypervisor extension. The pre-input period, post-input period and inter_thresh are used to identify interactive task. For our method of identification, we set the same value for pre-input period and post-input period. If there is a non-interactive applications sending requests continuously, the numbers of its potential interactive requests and potential non-interactive requests would be nearly equal in such configuration. We compare the number of two types requests using the potential interactive request rate, which is based on the total number of two types requests. In the ideal situation, this rate for non-interactive applications should be 50% and we set the threshold as 60%, which is marked as inter_thresh in Table 1.

For rate-limiter, we gives each VM in our evaluation a weight of 5 and a cap of 10. In such parameters, the throughput limitations of all VMs are equal. All of interactive applications we evaluate do not achieve this limitation of weight, hence all of the requests from these interactive applications are prioritized actually.

4.2 Robustness of Our Heuristic Method

We apply our heuristic method to the disk access traces from several VDI sessions of Windows-based VMs and Ubuntu-based VMs. We collect the traces using our optional semantics information transfer method, hence the trace includes the names of applications sending requests. Our method identifies all interactive applications accurately in these traces. Confined to the length of the thesis, we only show the results of Windows-based VMs traces in Figure 8. We find that all of the interactive applications have a rate much more than 60%, including office softwares and browsers. There is a misidentification, as that verclsid.exe has a rate more than 60%. This comes from that verclsid.exe only sends an average of four requests in each sessions and three of them are marked as potential interactive requests.

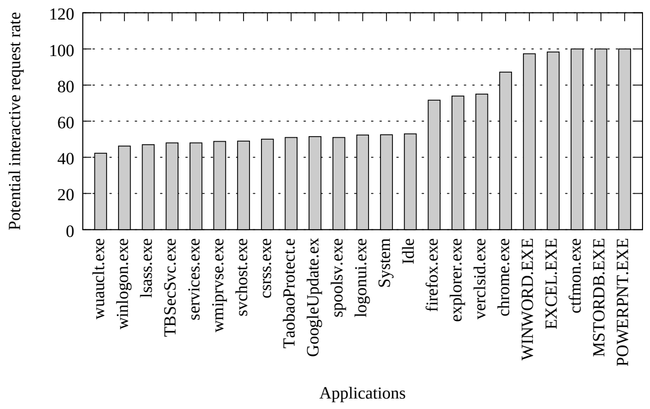

Figure 8: The potential interactive request rate of each application from our trace in Windows.

4.3 Explicit Storage Sensitivity

Desktop users desire low response time in application launch, while some launches cost much time, especially when the data for launch has not been loaded in memory. These cold launches send considerable disk requests for initialization during start period. We use four widely used applications in Linux including LibreOffice Impress, Mozilla Firefox, Google Chrome and GNU Image Manipulation Program (GIMP) for the evaluation in Ubuntu VM. Each application is repeatedly automatically launched and closed using a little change

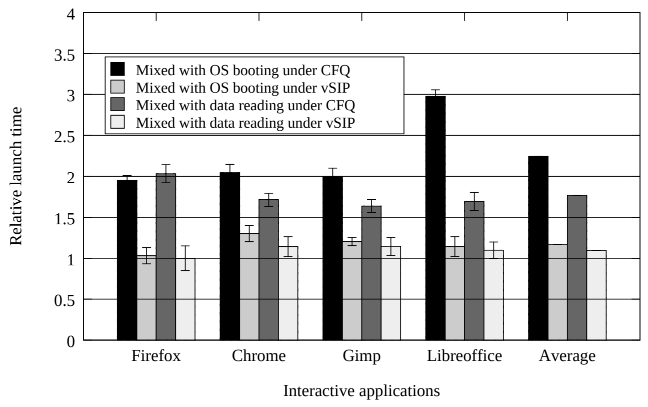

Figure 9: Interactive applications relative cold launch time mixed with different overheads. There is only one overhead VM.

to the source code. There is only one overhead VM sharing the storage running different workloads. We measure the relative launch time of interactive application as running alone for baseline and compare the results under CFQ scheduler with results under our scheme, as shown in Figure 9. vSIP decreases 38% or 48% cold launch time of interactive applications mixed with one data reading VM or one OS booting VM, compared with using Linux default CFQ scheduler. Compared with interactive applications running alone, our scheme increases only 9% or 17% cold launch time of interactive applications. There are some write and flush operations in OS booting, hence the requests from interactive applications have high priority still suffer long latency. That is why the performance of interactive applications mixed with OS booting is wore than that mixed with data reading under vSIP.

As an output-based interactive application, we evaluate a video playback workload using the mplayer media player. For a video playback workload, the requirement of disk resource relies on video contents. We choose two copies of a high definition video clip, Transformers: Age of Extinction HD. One copy has 1280x720 resolution (720p) and the other one has 1920x1080 resolution (1080p). The running time of both clips are 100 seconds under 30.0 FPS.

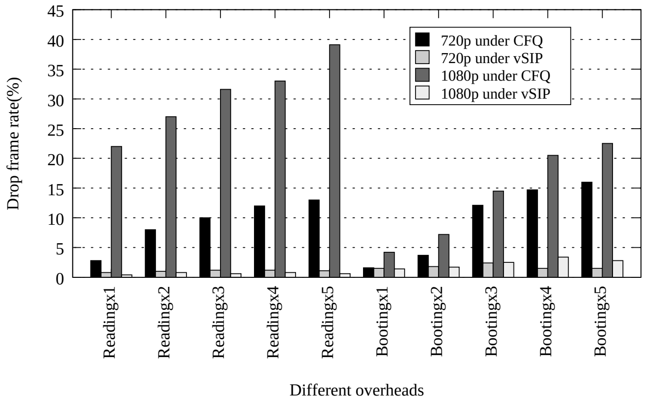

Figure 10: Multimedia player drop frame rate in different scenarios. The error is so little that we ignore it in this figure.

Figure 10 shows the video playback quality in the scenario that mplayer co-runs with different numbers of data reading VMs or OS booting VMs. The Y-axis shows the drop frame rate (= dropped frames ∗ 100 / total frames) and the X-axis shows the different overheads. vSIP guarantees a low enough drop frame rate in all cases.

4.4 Implicit Storage Sensitivity

As we have shown, application warm launch is a typical implicit storage sensitivity scenario. We choose five widely used interactive applications in Windows including Firefox, Chrome, Word, Excel and PowerPoint. Figure 11 shows the normalized warm launch time of each application in different scenarios. The overhead includes seven VMs running data reading or OS booting. When we run data reading as overhead, vSIP can guarantee the performance of warm launch. When we run OS booting as overhead, vSIP decreases the launch time compared with CFQ, however, the performance is worse than running alone.

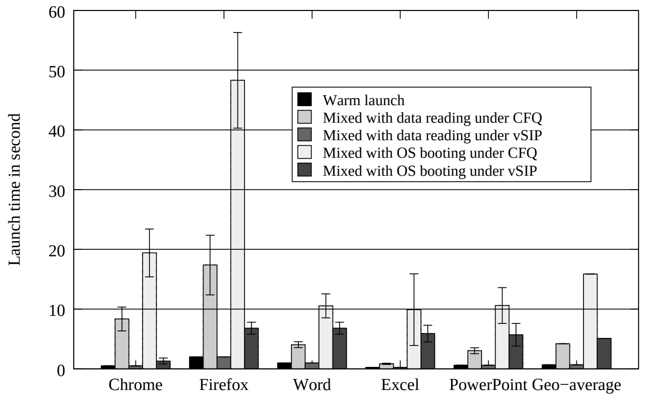

Figure 11: Application warm launch time in different scenarios. There are seven overhead VMs.

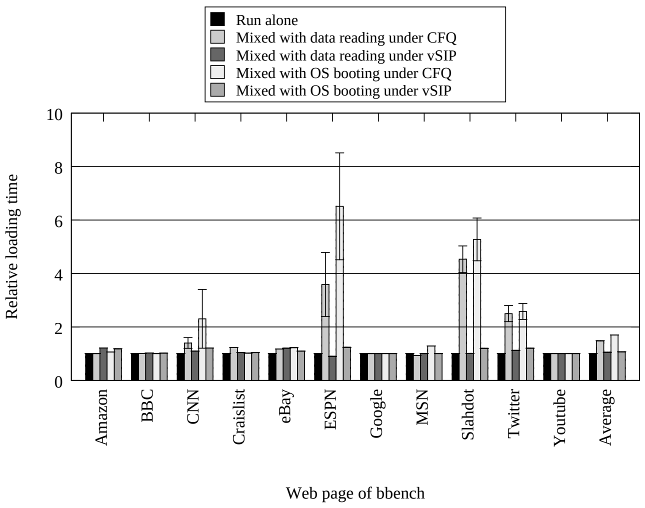

Figure 12: Relative performance of Web page loading in different scenarios. There are seven overhead VMs. We only mark the error bar of CNN, ESPN,Slahdot and Twitter.

Web pages browsing is also a prevalent interactive workload as desktop users usually have their jobs done by using Web and cloud services. We use Chrome as the Web browser and insert one-second interval between visits of each Web page in Bbench. We evaluate Web pages loading time in the same scenarios as warm launch. The results are shown in Figure 12. vSIP also guarantees their performance. We can find that only CNN, ESPN, Slahdot and Twitter suffer performance loss under CFQ, and only these four page loading send storage requests in Bbench. As expected, we find a positive correlation between performance loss and requests number. For example, ESPN sends the most requests number and suffers the most performance loss in Bbench.

4.5 Performance of Non-interactive Applications

We measure the performance loss of non-interactive applications using execution time as metrics. These non-interactive applications mixed with interactive applications nearly lose no performance under vSIP compared with CFQ. In some cases, the execution time decreases.

This result comes from that vSIP only prioritizes requests from interactive applications and the number of these requests is fixed. Then, if the execution of a non-interactive application covers the execution of interactive applications, such prioritizing would bring no harm to the performance of the non-interactive application. In another way, vSIP gathers the requests from interactive applications, which introduces the reduction of disk seek time in some cases.

4.6 Mixed Workload

At last, we evaluate vSIP under mixed workloads. We build a synthetic benchmark for interactive applications. The benchmark repeatedly reads 128KB data 10 times then idles 1 second and the request pattern is random. We build three types of VMs running Ubuntu, 1) interactive VM, which runs the synthetic benchmark alone, 2) mixed VM, which runs both the synthetic benchmark and data reading, 3) overhead VM, which runs data reading alone. We use CFQ scheduler both in guest OS and host OS. In mixed VMs, the synthetic benchmark has the high priority and the data reading has the common priority in guest OS. In each evaluation, we choose mixed VM or interactive VM and run it with several overhead VMs. We measure the latency of the synthetic benchmark under different host scheduling methods including default CFQ scheduler, CFQ(P) scheduler which prioritizes requests from interactive VMs and mixed VMs, and our framework vSIP. We find that vSIP and CFQ(P) scheduler guarantee the latency of synthetic benchmark, however, if CFQ(P) scheduler prioritizes the mixed VM, other VMs would starve. Because the requests from the non-interactive application in mixed VM are also prioritized.

5. CONCLUSION

In this paper, we present the design and implementation of vSIP for improving user-interactive performance of storage-sensitive applications in consolidated desktops. To classify the disk requests, we devise a simple and effective method for identifying interactive application at the hypervisor layer. Simply prioritizing requests from interactive applications would cause many problems, such as starvation of non-interactive applications or users lying for performance benefit. To solve these problems, we build a requests rate-limiter. We show how this mechanism solves these problems in this paper. As an option, we also present a guest OS information transfer method and present a white list mechanism based on this information to improve the efficiency and accuracy of identification.

6. REFERENCES

[1] Virtual desktop infrastructure (VDI). White paper of VMware.

[2] R. M. Baecker and W. A. Buxton. *Readings in human-computer interaction*. Elsevier Science, 2014.

[3] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer. Using Latency to Evaluate Interactive System Performance. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation*, OSDI '96, pages 185–199, New York, NY, USA, 1996. ACM.

[4] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Process prioritization using output production: Scheduling for multimedia. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(4):318–342, Nov. 2006.

[5] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. Eli: Bare-metal performance for i/o virtualization. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 411–422, New York, NY, USA, 2012. ACM.

[6] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues Don'T Matter when You Can JUMP Them! In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 1–14, Berkeley, CA, USA, 2015. USENIX Association.

[7] A. Gulati, A. Merchant, and P. J. Varman. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–7, Berkeley, CA, USA, 2010. USENIX Association.

[8] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *the proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 81–90, Austin, TX, USA, 2011.

[9] L. Huang, G. Peng, and T.-c. Chiueh. Multi-dimensional Storage Virtualization. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, pages 14–24, New York, NY, USA, 2004. ACM.

[10] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking Processes in a Virtual Machine Environment. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 1–1, Berkeley, CA, USA, 2006. USENIX Association.

[11] H. Kim, S. Kim, J. Jeong, and J. Lee. Virtual asymmetric multiprocessor for interactive performance of consolidated desktops. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '14, pages 29–40, New York, NY, USA, 2014. ACM.

[12] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee. Task-aware virtual machine scheduling for i/o performance. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 101–110, New York, NY, USA, 2009. ACM.

[13] A. Krishnamurthy and S. S Kowsalya. Differentiated i/o services in virtualized environments.

[14] C. W. Mercer. Operating system support for multimedia applications. In *Proceedings of the Second ACM International Conference on Multimedia*, MULTIMEDIA '94, pages 492–493, New York, NY, USA, 1994. ACM.

[15] J. Nieh and M. S. Lam. A SMART Scheduler for Multimedia Applications. *ACM Trans. Comput. Syst.*, 21(2):117–163, May 2003.

[16] J. Nieh, S. J. Yang, and N. Novik. Measuring thin-client performance using slow-motion benchmarking. *ACM Trans. Comput. Syst.*, 21(1):87–115, Feb. 2003.

[17] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for ramclouds: Scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, Jan. 2010.

[18] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn. Efficient Guaranteed Disk Request Scheduling with Fahrrad. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 13–25, New York, NY, USA, 2008. ACM.

[19] R. Russell. Virtio: Towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, July 2008.

[20] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*, volume 3. Addison-Wesley Reading, MA, 1992.

[21] S. Yang, T. Harter, N. Agrawal, S. S. Kowsalya, A. Krishnamurthy, S. Al-Kiswany, R. T. Kaushik, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Split-level i/o scheduling. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 474–489, New York, NY, USA, 2015. ACM.

[22] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, and J. E. B. Moss. Redline: First class support for interactivity in commodity operating systems. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 73–86, Berkeley, CA, USA, 2008. USENIX Association.

[23] H. Zheng and J. Nieh. Rsio: Automatic user interaction detection and scheduling. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '10, pages 263–274, New York, NY, USA, 2010. ACM.