Flubber: Two-level Disk Scheduling in Virtualized Environment

Hai Jin, Xiao Ling, Shadi Ibrahim, Song Wu, Wenzhi Cao, Gabriel Antoniu

Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 00 (2013) 1–23
Future Generation Computer Systems

Flubber: Two-level Disk Scheduling in Virtualized Environment

Hai Jin[a], Xiao Ling[a], Shadi Ibrahim[b], Wenzhi Cao[a], Song Wu[a,*], Gabriel Antoniu[b]

[a]*Cluster and Grid Computing Lab, Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China*
[b]*INRIA Rennes - Bretagne Atlantique, Rennes, France*

Abstract

While virtualization enables multiple virtual machines (VMs) — with multiple operating systems and applications — to run within a physical server, it also complicates resource allocations trying to guarantee *Quality of Service* (QoS) requirements of the diverse applications running within these VMs. As QoS is crucial in the cloud, considerable research efforts have been directed towards CPU, memory and network allocations to provide effective QoS to VMs, but little attention has been devoted to disk resource allocation.

This paper presents the design and implementation of *Flubber*, a two-level scheduling framework that decouples throughput and latency allocation to provide QoS guarantees to VMs while maintaining high disk utilization. The high-level throughput control regulates the pending requests from the VMs with an adaptive credit-rate controller, in order to meet the throughput requirements of different VMs and ensure performance isolation. Meanwhile, the low-level latency control, by the virtue of the *batch and delay earliest deadline first mechanism* (BD-EDF), re-orders all pending requests from VMs based on their deadlines, and batches them to disk devices taking into account the locality of accesses across VMs. We have implemented *Flubber* and made extensive evaluations on a Xen-based host. The results show that *Flubber* can simultaneously meet the different service requirements of VMs while improving the efficiency of the physical disk. The results also show improvement of up 25% in the VM performance over state-of-art approaches: for example, in contract to the default Xen disk I/O scheduler — *Completely Fair Queueing* (CFQ) — besides achieving the desired QoS of each VM, *Flubber* speeds up the sequential and random reads by 17% and 25%, respectively, due to the efficient physical disk utilization.

Keywords: I/O scheduling, virtualization, quality of services, performance, efficiency

1. Introduction

The increasing trend towards cloud computing has elevated *Quality of Service* (QoS) of *virtual machines* (VMs) to a key issue in virtualized environments [1, 2, 3]. While virtualization enables multiple VMs — with multiple operating systems and applications — to run within the same physical server, and multiplexes the underlying hardware resources among them, it also complicates the resource allocations trying to guarantee QoS requirements of the diverse applications running within these VMs. Disk I/O allocation in virtualized environments has received less attention from the research community than CPU, memory and network allocation [4, 5, 6, 7]. Due to disk contention,

*Corresponding author. Tel.: +86 27 87543529.
Email address: wusong@hust.edu.cn (Song Wu)

virtualization fails to guarantee the latency and throughput target of each VM, regardless whether the VMM provides a good VCPU scheduler or other resource allocation [8].

Compared with sharing I/O among processes in shared storage or in the native operation system, sharing I/O among VMs has extra features:

- First, unlike general user processes, VMs have self-disk I/O schedulers to send batches of requests in a round-robin mode.
- Second, the semantic isolation leads to the disk scheduler of a *Virtual Machine Monitor* (VMM) having difficulty perceiving the characteristics of workloads, so the VMM provides service to requests from VMs using *First Come First Served* (FCFS).
- Last, the virtual disks associated with VMs are mapped into large files in the physical disk, which results in spatial locality of accesses across VMs.

Because of these special features of virtualization, the traditional solutions for resource contention in the shared storage are not appropriate for virtualized environments [9, 10, 11, 12, 13]. Therefore, some solutions in VMM have been proposed to alleviate the disk I/O contention in recent years [14, 15, 16, 17]. Some of these studies propose fair or proportional sharing of disk bandwidth among VMs consolidated on a server [14, 15]. Based on proportional sharing of disk bandwidth, other studies [16, 17] control the change of throughput for each VM in the VMM. These approaches are unable to meet both the throughput and latency targets of each VM, especially when VMs with high throughput requirement and VMs with sensitive latency requirement share a larger disk. Moreover, these studies focus on performance isolation among VMs via bandwidth control for VMs without considering the cost of disk head movement and disk I/O utilization. For example, in a simple setup with two VMs: a VM running a Data Migration with high throughput requirement (called *DM* VM) and the other VM running a Online Transaction Processing application with low latency requirement (called *OLTP* VM), throughput control may prevent *DM* VM with sequential access from grabbing bandwidth of the OLTP VM with random access, thus meeting the expected latency of the OLTP application, but at the cost of high frequent disk seek and rotational overheads. This results in decreasing the I/O performance of the two applications in the virtualized environment in spite of meeting QoS of VMs. Even worse, the presence of background processes in the VMM such as backup processes and checkpoints leads to a high number of physical disk seeks and rotational overheads which in turn lowers the performance of the VMs.

In this paper, we address the problem of allocating disk resources to guarantee specified latency and throughput targets of VMs while keeping disk I/O efficiency. To achieve the goal, we present a flexible two-level disk scheduler in VMM, named *Flubber*, which separates throughput and latency control. *Flubber* consists of a self-adaptive high-level throughput control, which regulates the arrival rate of pending requests from each VM, and a low-level latency control, which guarantees latency of requests for VMs. The high-level throughput control, called *Virtual Bandwidth Control* (VBC), using a credit-rate controller, assigns a stochastic credit value to each VM, in order to meet the throughput targets of VMs and ensure performance isolation. Moreover, when a complex relation is introduced between the bandwidth allocation and performance requirements of VMs (e.g. the case of VMs with write-dominated applications) and the coarse-grained credit assignment can hardly meet the desired throughout requirements of the VMs, the credit-rate controller dynamically adjusts the assigned credits by employing a feedback-based credit adjustment policy. For the latency requirements of VMs and disk I/O utilization, the low-level latency control, called *Virtual Latency Control* (VLC), uses a *Batch and Delay-Earliest Deadline First* (BD-EDF) mechanism, similar to the *earliest-deadline-first* (EDF) algorithm, to manage the request queue at the physical block driver. Unlike EDF, the BD-EDF mechanism uses non-work-conserving scheduling and takes into account the spatial locality of accesses across VMs and the running background processes in VMM. Therefore, VLC batches the requests from the same VM or sequential requests, without exceeding the deadlines of other VMs. Moreover, VLC delays requests from background processes and batches them, in order to guarantee QoS of VMs.

We implement a prototype of *Flubber* in Xen-based host, and evaluate the performance of VMs with *Flubber*. The results demonstrate the efficiency of our adaptive credit-rate controller: while the coarse-grained credit assignment can easily meet the desired throughput for the read-dominated applications, the feedback-based credit adjustment policy can successfully do so for write-dominated applications by adjusting the assigned credits within relatively short interval time (e.g., the credits adjustment is completed within *5* seconds in our experiments). Moreover, the

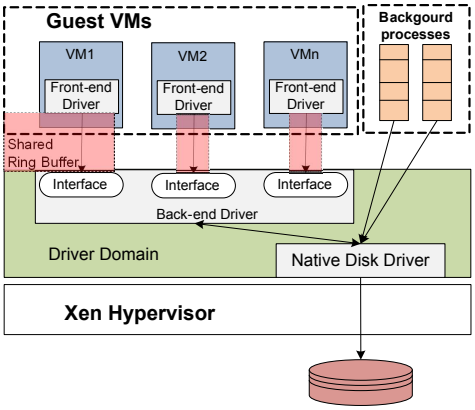


Figure 1: **Xen I/O Model:** Xen Device Driver consists of four main components (*the native disk driver, the bottom half of the split driver -called back-end driver, the top half of the split driver - called front-end driver, and a shared ring buffer*)

results also show that *Flubber* reduces the disk head seeking owing to the adopted BD-EDF mechanism: in the case of read applications, 96% of the request waits introduced by the BD-EDF algorithm lead to dispatched requests of a small distance of disk head seeking (mostly zero) without violating the latency requirements of the VMs. As a result, *Flubber* simultaneously meets the different QoS requirements of VMs while improving the overall physical disk I/O. *Flubber* shows improvement of up 25% in the VM performance over state-of-art approaches. For example, in contract to the Xen default disk I/O scheduler (CFQ scheduler) [9], besides achieving the desired QoS of each VM, *Flubber* speeds up the sequential and random reads by 17% and 25%, due to the efficient physical disk utilization.

The contributions of this paper can be summarized as follows:

- It presents a two-level scheduling framework in VMM, named *Flubber*, to achieve fine-grained throughput and latency control for different VMs running on a physical server.
- Considering spatial locality of accesses across VMs, especially when background processes are running in VMM, *Flubber* embraces a BD-EDF mechanism to keep disk I/O efficiency while guaranteeing the desired throughput and latency of VMs.
- We implement and evaluate *Flubber* in Xen-based host. The experimental evaluation demonstrates that *Flubber* can meet QoS requirements of VMs while improving disk I/O efficiency.

The rest of the paper is organized as follows. Section 2 discusses the background and related work. The design of *Flubber* and related algorithms are discussed in section 3. Section 4 details the performance evaluation. Finally, we conclude the paper in section 5.

2. Background and Related Work

In this section, we briefly introduce the Xen I/O model and discuss previous related work.

2.1. Xen I/O Model

Xen hypervisor [18] is a para-virtualizing virtual machine monitor that allows multi-VMs to run on a physical server and share its hardware resources. Xen acts as an intermediate layer between the VMs and the hardware devices, that is, only Xen has a direct access to hardware devices: it is responsible for allocating resources (CPU, memory and I/O devices, etc.) to each VM running on the same hardware device.

Xen has introduced a new I/O model, called a driver domain, which performs I/O operations on behalf of unprivileged guest VMs. As shown in Fig. 1, when guest VMs perform I/O operations, Xen receives the I/O requests from the front-end drivers of guest VMs in a *First Come First Served* (FCFS) order and informs the driver domain

to process them, then passing these requests to the back-end driver of the driver domain. In turn, the driver domain issues these requests from guest VMs to the physical driver. The driver domain treats guest VMs as general processes and handles the requests via native disk I/O scheduler, such as CFQ (the default scheduler), Deadline, Anticipatory, and Noop.

2.2. Related Works

The research literature contains a large body of work aiming to guarantee QoS in storage systems to provide throughput and latency guarantees while achieving high disk I/O utilization. Accordingly, earlier works are classified into three categories: bandwidth allocation for performance isolation and differentiation, latency-sensitive control with bandwidth allocation, and disk resource allocation for disk I/O efficiency.

2.2.1. Bandwidth Allocation for Performance Isolation and Differentiation

There have been many studies on bandwidth allocation to ensure performance isolation and differentiation between users or applications in a shared storage system. Most of these studies use proportional bandwidth sharing by allocating throughput or bandwidth in proportion to the weight of the processes/applications (such as YFQ [19], Stonehenge [10], Argon [11], Aqua [12], and SFQ(D) [13]), or by allocating different exclusive time-slices for applications by grouping these applications according to their QoS requirement (such as CFQ [9] and Cello [20]). Unfortunately, due to the abstraction layer introduced by virtualization, these solutions cannot be applied in practice in virtualized environments [21, 22, 23].

Recently, fair and proportional bandwidth sharing have been employed in virtualized environments to ensure performance isolation among VMs [14, 15]. VIOS [14] controls the coarse-grain allocation of disk time to the different operating system instances using completely fair queuing and a compensating round-robin scheduler. PARAD [15] proposes a coarse-grain control on the throughput of a VM. Based on the weight of processes, it assigns virtual clock tags to the requests, then it uses SFQ(D) to dispatch the requests to the storage system according to the order of their virtual clock tags. Similarly, a coarse-grain bandwidth control is used in *Flubber* to meet the desired throughput requirements of the VMs. But unlike all the prior work, *Flubber* uses a dynamic time interval for credit replenishment, thus avoiding wasting the disk bandwidth caused by static time intervals. Moreover, all the aforementioned solutions do not guarantee QoS of latency-sensitive applications such as remote desktop VM.

Another branch of related work concerns bandwidth allocation using feedback control [24, 25, 26]. Triage [25, 26] uses the latency as a metric and adjusts bandwidth allocation using feedback. However, the feedback control paradigm does not fairly distribute the spare bandwidth between applications. Thanks to the dynamic interval time for credit replenishment, *Flubber* addresses the fair distribution of spare bandwidth. Also, the latency deadline of latency-sensitive applications is not considered in those previous studies.

2.2.2. Latency-sensitive Control with Bandwidth Allocation

The notion of latency-sensitive control with bandwidth allocation has been widely studied in native storage systems [27, 28, 29]. Pclock [27] and SARC/AVATAR [28] have implemented an EDF scheduler in the request queue to provide latency and throughput guarantees. However, besides that their solutions are not suitable for VMM because of the features of virtualization [8, 21], their solutions hardly provide good throughput in the disk I/O because they do not consider locality of accesses. *Flubber* uses similar approach to that used in SARC/AVATAR [28], decoupling the throughput and latency control into a two-level framework scheduler. However, unlike SARC/AVATAR, *Flubber* targets a different storage environment and it intends to improve the disk I/O utilization. Furthermore, SARC/AVATAR was evaluated based on simulations while *Flubber* is built and evaluated in Xen-based hosting platform.

Recently, some studies have focused on supporting latency-sensitive applications in virtualized environments [16, 17, 22]. DVT [22] has controlled the change of service latency in a VMM in order to improve the performance of VM, but when applying their solution in the disk I/O scheduler of VMM, it brings only a small performance improvement to the VMs because it ignores disk I/O efficiency (i.e., meeting the latency-sensitive requirements of VMs brings disk seek and rotational overheads). *Flubber* approaches the same problem to provide reasonable QoS guarantees to the VMs, but keeping in mind high disk utilization by waiting a small amount of time for requests with minimal (zero) seek distance without violating the latency deadlines of VMs.

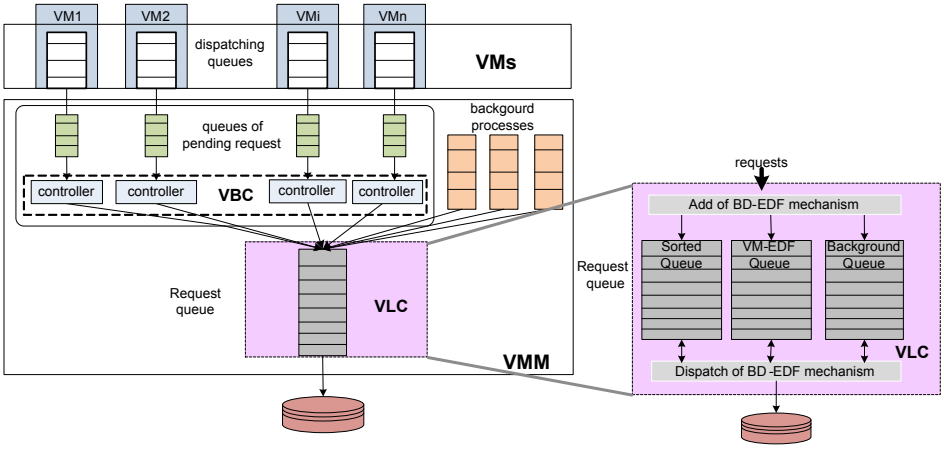


Figure 2: Two-level scheduling framework of *Flubber*

2.2.3. Disk Resource Allocation for Disk I/O Efficiency

Few publications have studied the tradeoff between disk I/O efficiency and fair sharing of disk resource [30, 31] in native systems. Their focus is on providing high bandwidth to applications with high throughput requirements and sequential access. However, these studies have ignored the latency target. In virtualized environments, few studies have focused on keeping high I/O efficiency while guaranteeing latency and bandwidth targets for different VMs. In addition, few studies have demonstrated the impacts of background processes on the QoS of VMs, although the background processes in VMM such as checkpoint, backup and defragment, may steal bandwidth from other VMs and lead to frequent physical disk head seeking. CFQ, by setting a different class of priorities for both background process and VMs, may alleviate the impacts of VMs' bandwidths but not latency.

3. Architecture of *Flubber*

The goal of *Flubber* is to practically provide both throughput and latency guarantees to VMs, while enhancing physical disk I/O utilization. Our design, therefore, intends to address the following issues:

1. Achieving different latency and throughput requirements of VMs deployed in a server;
2. Finding an optimal tradeoff between physical disk I/O efficiency and VM's QoS;
3. Avoiding frequent physical disk head seek and rotation overheads when background process is performed.

Accordingly, we propose a two-level scheduling framework in VMM to control throughput and latency separately for VMs. Taking disk I/O efficiency into consideration, we design a BD-EDF mechanism to exploit the spatial locality of VMs access. In this section, we introduce the two-level scheduling framework and describe how the two levels can guarantee QoS of VMs. Then, we detail the BD-EDF mechanism.

3.1. Two-level Scheduling Framework

In the virtualized environment, multiple VMs run on a server with a prescribed performance-based *service level objective* (SLO). In order to meet the desired throughput for different workloads within VM and steady latency for the disk scheduler of VM (i.e. CFQ, Deadline, Anticipatory, and Noop), the SLO specification for VM_i is presented as a tuple $< B_i, L_i >$, where B_i is the total required throughput by VM_i and L_i is the maximum allowed latency for a request from VM_i. Therefore, based on the abstraction of the I/O model in Xen, we opt for a flexible two level scheduling framework, called *Flubber*, separating throughput and latency control to meet SLO of VMs while keeping transparent communication between VMM and both VMs and the physical disk, as shown in Fig. 2.

The high-level throughput control, named *Virtual Bandwidth Control* (VBC), is responsible for regulating the pending requests from the VMs and aims at meeting the desired throughput requirements of VMs and enforcing performance isolation among VMs. VBC therefore shapes the arrival rate of pending requests from the dispatching

queue of each VM by a credit-rate controller. The credit-rate controller allocates a credit amount to the dispatching queue of each VM in VMM. The amount of allocated credits indicates how many pending requests from VM can be dispatched to the low-level latency control. Moreover, when a complex relation is introduced between disk resource allocation and specified SLO of VMs such as in write-dominated applications, VBC employs a feedback-based credit adjustment policy to dynamically set the most appropriate credit amount for a VM, thus, maximizing the throughput of each VM and ensuring performance isolation. The low-level latency control, called *Virtual Latency Control* (VLC), is responsible for satisfying latency target of pending requests from VMs and enforcing disk I/O utilization. VLC, by the mean of the BD-EDF mechanism, dispatches pending requests from VMs based on their deadlines and *logical block addresses* (LBAs). Considering the spatial locality of accesses across VMs, the BD-EDF mechanism, in a non-work-conserving mode, tends to batch pending requests from the same VM, without exceeding the deadlines of other pending requests. Besides, given that requests from background processes in VMM may introduce high disk head seek and rotational overheads, VLC delays these requests and then batches them. These ensure latency requirements of VMs while improving disk I/O efficiency.

3.2. High Level Scheduler - VBC

VBC selects distributed credit-rate controllers to manage the dispatching queue of each VM. Using credits can easily capture a relative rate and limit the bandwidth competition among VMs. Consequently, this enables efficient global coordination across all VMs for satisfying SLO requirements. Each pending request consumes one credit when dispatched to VLC. However, if VM has already used all the credits, the requests are queued in the corresponding dispatching queue of VM, waiting for credit replenishment.

3.2.1. Credit Replenishment Policy

Static time interval of credit replenishment enforces fairness across VMs at cost of the disk bandwidth. Instead, VBC dynamically controls the length of interval of the credit replenishment for all VMs. Thus, the replenishment event occurs:

1. Over the turn-round among VMs: when credits of all VMs are used up;
2. When all VMs hardly send pending requests and pending queues are empty;
3. When a new VM is created or an existed VM is destroyed.

The first replenishment has similar goal as the replenishment policies used in VIOS [14] and SARC [28] (i.e. to ensure fair distribution of the disk among VMs according to their requirements). The second and third replenishment events ensure that the backlogged pending requests are few if disk has spare bandwidth. When the replenishment event occurs, each VM is replenished to its full credits considering throughput target of the VMs. The length of interval of replenishment is strongly depends on the amount of allocated credits to each VM: large credit amount leads to long interval which reduces performance isolation among VMs, while small credit amount leads to short interval which lowers disk bandwidth utilization [30]. Thus, setting a suitable credit assignment for each VM is very important in order to balance physical disk bandwidth utilization and the performance requirement of VM.

3.2.2. Credit Assignment

Upon the occurrence of a replenishment event, VBC allocates credits to each VM for next interval. In order to satisfy QoS requirements of VMs, a coarse-grained credit assignment among VMs is applied. Similar to the credit assignment in SARC [28], the coarse-grained credit assignment, used in *Flubber*, is strongly related to the throughput requirement of VMs. However, as VBC dynamically controls the length of interval of the credit replenishment and considering the relationship between physical disk bandwidth utilization and performance isolation of VMs, we introduce an interval factor β for credit assignment and use the credits proportion to find the credit amount of VM_i. The credit amount denoted by a_i and stated as:

$$a_i = \beta \times c_i, \frac{c_i}{c_j} = \frac{B_i}{B_j} \tag{1}$$

where $< B_i, L_i >$ is expected SLO of VM_i, the fraction $\frac{c_i}{c_j}$ is in the lowest terms (i.e. the greatest common divisor $gcd(c_i, c_j) = 1$).

Moreover, in order to increase the disk utilization while ensuring performance isolation between VMs, the interval factor β is bounded by the maximum interval of replenishment *max_interval* ($\beta \in [0, \frac{max\_interval}{\sum_{i=1}^{n} c_i^0}]$) which represents the throughput capacity of physical disk.

Feedback-based credit adjustment policy. When a complex relation is introduced between the bandwidth allocation and performance requirements of VMs (e.g. the case of VMs with write-dominated applications) and the coarse-grained credit assignment can hardly meet the desired throughout requirements of the VMs, the credit-rate controller dynamically adjusts the assigned credits by employing a feedback-based credit adjustment policy.

The feedback-based credit adjustment policy normalizes the achieved performance of each VM during the last interval to indicate the deviation from their expected performance. To meet both throughput and latency requirements of VMs, the normalized performance indicator selects the worst deviation based on the achieved throughput and latency and stated as:

$$Ind_i^k = \max\{\frac{B_i - b_i^k}{B_i}, \frac{l_i^k - L_i}{L_i}\} \tag{2}$$

where Ind_i^k is the normalized performance indicator of VM_i during the interval k. b_i^k and l_i^k are achieved average throughput and latency during the interval k, respectively, and $< B_i, L_i >$ is the expected SLO of VM_i.

The value which the indicator returns decides whether there is a need for adjusting the allocated credits for VMs: zero value indicates that the required SLO is achieved and negative value indicates that the current credits have led to exceeding the SLO objectives, in both cases no action will be taken and the current credit assigned will be used for later intervals. However, if the normalized performance indicator returns a positive value (i.e. SLO violation has occurred), then the feedback-based policy will adjust the credit amount assignments as follows:

- The feedback-based credit adjustment policy updates the proportion of credits between VMs to compensate the deficiency of the allocated credits in the last interval.

$$\frac{c_i^{k+1}}{c_j^{k+1}} = \frac{a_i^k + Ind_i^k \times \beta}{a_j^k + Ind_j^k \times \beta} \tag{3}$$

  The fraction $\frac{c_i^{k+1}}{c_j^{k+1}}$ is in lowest terms (i.e. $gcd(a_i^{k+1}, a_j^{k+1}) = 1$).

- Based on the new credits proportion and taking into account the limitation of the disk throughput, new credit amount will be allocated:

$$a_i^{k+1} = \begin{cases} c_i^{k+1} \times \beta & ; \sum_{i=1}^{n} c_i^{k+1} \times \beta < max\_interval \\ max\_interval \times \frac{c_i^{k+1}}{\sum_{j=1}^{n} c_j^{k+1}} & ; \sum_{i=1}^{n} c_i^{k+1} \times \beta \geq max\_interval \end{cases} \tag{4}$$

It is important to mention that the length of the next interval ($next\_interval = \sum_{i=1}^{n} c_i^{k+1} \times \beta$) should never exceed the maximal interval. This will guarantee that all the VMs will meet their throughput requirements while using the maximum of the disk throughput capacity (i.e. ensuring fair distribution of the disk throughput between VMs while meeting their throughput requirements). Accordingly, the next credit amount allocated for VM_i in the next interval $k+1$ is the product of the new credit proportion c_i^{k+1} and β if the next interval within the maximum one, otherwise, it is the product of the maximum interval *max_interval* and the fraction of the new credit proportion c_i^{k+1} out of the total credits proportion of all VMs. In summary, the feedback-based credit adjustment policy can successfully achieve the most suitable credit amount using relatively short time and does not impact the application performance.

3.3. Low Level Scheduler - VLC

VLC manages the request queue of the physical disk including pending requests from all VMs and the requests from background process. To meet latency targets of requests from VMs and enforce disk I/O efficiency, VLC employs BD-EDF mechanism that dispatches the requests to the disk based on both the deadline and the *logical block address* (LBA) of a request. VLC therefore divides the request queue into three type queues: *sorted queue*, where all requests from VMs and background process are queued according to their LBAs, *VM-EDF queue*, where the pending requests from all VMs are sorted according to their deadlines, and *background queue*, where the requests from background processes in VMM are sorted in *first in first out* (FIFO). Next, we detail how BD-EDF manages the requests within the three type queues.

Algorithm 1: Classification Algorithm

Description: Procedure invoked upon arrival of request $Request_j$.
en-queue $request_j$ *in the Sorted Queue* /* based on its LBA $request_j.address$ */
if $request_j$ *is* VM_i*'s request* **then**
  $t_j = request_j.s_i + L_i - M$
  /* **M** is maximal seek time of physical disk; L_i is the target latency of VM_i*/
  $request_j.deadline = t_j$
  $k \leftarrow 0$
  while $VM\text{-}EDF.queue[k].deadline < request_j.deadline$ **do**
    $k \leftarrow k + 1$
  end
  foreach $m = k$; end of the queue **do**
    en-queue $request_j$ *in the* $m + 1$ *position in the VM-EDF Queue*
  end
  en-queue *in the m position in the VM-EDF Queue*
end
else
  en-queue $request_j$ *in the tail of the Background Queue*
end

3.4. Deadline Target of VM and Disk I/O Efficiency - BD-EDF Mechanism

BD-EDF mechanism aims at providing latency guarantee for different VMs while ensuring efficient disk I/O utilization, by determining the most appropriate pending request to be dispatched to the physical disk device. However, as virtual disks associated with VMs are mapped into large files in physical disk, simple EDF queue may lead to frequent disk head seek and rotational overheads when I/O requests from different VMs arrive simultaneously. Moreover, when background processes are running in the VMM, the disk head seek and rotational overhead will be markedly increasing because of the frequent disk head movements when switching between the requests from the VMs and the requests from the background process. Based on the aforementioned discussion, BD-EDF uses two algorithms. The first one, called classification algorithm, determines the type of queue that a request will join. The later one, called batch and delay algorithm, selects the request that will be dispatched to the physical disk.

3.4.1. Classification Algorithm

As shown in Algorithm 1, once a request arrives in the request queue of a disk, VLC queues the request into the sorted queue based on its LBA (this will allow BD-EDF to batch requests with close LBAs). If the request is coming from a VM_i, a deadline t_j will be associated with the request and stated as:

$$t_j = request_j.s_i + L_i - M \tag{5}$$

where $request_j.s_i$ is the start time of $request_j$ in VM_i, M is the maximal seek time of physical disk and L_i is the latency target of VM_i).

The request will then be en-queued in the VM-EDF queue in accordance to its deadline. Otherwise, the request will be en-queued in background queue in FIFO mode.

3.4.2. Batch and Delay Algorithm - BD

In order to ensure efficient disk I/O utilization, the BD algorithm intends to batch requests from the same VM or requests of which LBAs are close, without violating the deadline of any request in the VM-EDF queue. Accordingly, BD algorithm introduces a waiting time — which is a tradeoff between disk I/O efficiency and latency target of VMs — for the VM of the last completed request to send new request. The waiting time is denoted by *wait_time,* and stated as:

$$wait\_time + service\_time(R_i) < deadline(selected\_pending\_request) \tag{6}$$

where *deadline*(*selected_pending_request*) is the deadline for the pending request from VM with earliest deadline, and *service_time*(R_i) is the expected time to serve the future request R_i. Finding service time is related to the size of request size and data transmission rate of physical disk. In Xen, the size of requests from VM is limited and less than 44KB, so we neglect the service time of request. Based on above, we can evaluate *wait_time* in coarse grain.

As shown in Algorithm 2, upon the completion of a request, if the VM-EDF queue is not empty and the deadline of the head request in the VM-EDF queue does not pass the current time, a waiting time is set as the in Eq. (6). Within this waiting time, BD algorithm tends to dispatch requests by exploiting the spatial locality of VMs access: it first checks in the sorted queue whether the successive request of the last completed one has a continuous LBA of the last request's LBA or at least belong to the same VM. If so, this request will be dispatched to the disk and the waiting time will be off. However, if neither the head of VM-EDF nor the successive request of the last dispatched request in the sorted queue is suitable, our scheduler will continue waiting for a future suitable request in order to reduce disk seek overheads. If any of the new arriving requests has a continuous LBA of the last dispatched request or from the same VM as the last completed request, it will be dispatched to the disk, and the waiting time is clear. If the suitable request never appears within the waiting time, the head request of VM-EDF is dispatched. Moreover, when VM-EDF queue is empty, our scheduler batches the requests from the background queue until VM-EDF queue has pending requests.

Algorithm 2: Batch and Delay Algorithm

```
Description: Dispatching a Request.
/* last_req is the last dispatched request ; next_req is the request next to last dispatched request in sorted queue
*/
if (The length of VM-EDF ! = 0) && (VM-EDF.queue[0].deadline > current_time) then
    last_req.wait_time = request_i.deadline − max_size/Trans
    /* max_size is maximal size of last request from VM; Trans is transmission rate of disk*/
end
/*procedure invoked when a waiting time is introduced*/
begin
    dispatch_req = last_req
    if (last_req.wait_time > current_time) && ((next_req.address == last_req.address + size_block) ||
    (next_req.vmid == last_req.vmid)) then
        dispatch_req = next_req
    end
    while (last_req.wait_time > current_time) && (dispatch_req == last_req) do
        /*on the arrival of new request new_req*/
        if (new_req.address == last_req.address + size_block) || (new_req.vmid == last_req.vmid) then
            dispatch_req = new_req
        end
    end
    if (dispatch_req == last_req) && (The length of VM-EDF ! = 0) then
        dispatch_req = VM-EDF.queue[0]
    else
        dispatch_req = background.queue[0]
    end
end
```

4. Performance Evaluation

4.1. Experimental Environment

Flubber is implemented on Xen 3.3.1 with kernel 2.6.18 and is evaluated via experiments on a physical server. The server is equipped with four quad-core 2.40GHz Xeon processor, 22GB of memory and one dedicated SATA disk of 1TB, running RHEL5 with kernel 2.6.18. All VMs described in this paper are running with RHEL5 with kernel 2.6.18 and configured with 512MB memory and 1 VCPU.

Table 1: Two-Class Workloads Generated Using Sysbench

Class	Workload	Description	SLO
High throughput insensitive latency	Sequential Read	16 threads sequence read 128 files of which total size is 2 GB	1000/80
	Sequential Write	16 threads sequence write 128 files of which total size is 2 GB with fsync( )	1200/150
Low throughput low latency	Random Read	8 threads random read 128 files of which the total size is 1 GB	120/60
	Random Write	8 threads random write 128 files of which the total size is 1 GB with fsync( )	600/100
	Mixed Random Read and Write	8 threads random read and write 128 files of which the total size is 1 GB with fsync( )	100/60

We conduct various experiments to illustrate the QoS guarantee and disk I/O efficiency in *Flubber*. Accordingly, we use *Sysbench* [32] to generate five applications to mimic two different classes of workloads with QoS requirements (prescribed performance-based SLO) running on VMs as shown in Table 1: bandwidth-bounded applications (sequential read and sequential write) and latency-sensitive applications (random read, random write, and mixed random read&write). The throughput and latency requirements for the sequential read application are 1000IOPS and 80ms respectively, and for the random write application are 1200IOPS and 150ms, respectively. On the other hand, the throughput and latency requirements for the random read application are 120IOPS and 60ms, and for the random write application are 600IOPS and 100ms, and for the mixed random read and write application are 100IOPS and 60ms, respectively.

We empirically discuss the effectiveness of different interval factor in credit assignment of *Flubber*. Then, after finding the best interval factor, we evaluate *Flubber* with different applications by comparing *Flubber* with three other schedulers (base CFQ, CFQ with priority settings [1], and EDF scheduler with credit controlling), in two different scenarios - with and without the running background process within the VMM. Finally, we evaluate how *Flubber* can reduce the disk head seeking due to the adopted BD-EDF mechanism and discuss the success ratio of the waiting time by using blktrace [33] to record the disk seek distance.

4.2. Experimental Study on Credits Assignment

To guarantee QoS of VMs in *Flubber*, it is important to set appropriate credit amounts to VMs. However, as discussed in section 3, the credit amount is related to the proportion of credits and the interval factor β. As the proportion of credits depends on the proportion of the throughput requirements of VMs, here, we vary the interval factor and empirically study the impact of selecting different interval factor values on the QoS guarantees of VMs. Two VMs are deployed within a server. Each VM runs different class of workloads; particularly sequential read workload is running in a VM, called *SeqR* VM, and random read workload is running in the other VM, called *RndR* VM. Given Eq. (1): the equation of credit assignment in section 3, the initial proportion of credits between *SeqR* VM and *RndR* VM is 8:1. Then, by changing the interval factor β, we get different performances of VMs as shown in Table 2.

Table 2 demonstrates the performance of the two VMs with different β. With increasing β from 1 to 32, the performance of VMs and disk bandwidth utilization are improved. However, when β is 128, the performance of *RndR* VM is lower and the fluctuation of the performance of *SeqR* VM is large, because the longer interval reduces performance isolation between VMs. Accordingly, the interval factor β is set to 32.

Table 2: Two-Class Workloads Generated Using Sysbench Under Different Interval Factor

β	Sequential Read				Random Read			
	avgIops	maxIops	avgdelay	MaxDelay	avgIops	maxIops	avgDelay	MaxDelay
1	476.9	570	31.6ms	46ms	58.79	68.31	125.6ms	143.6ms
2	738.5	887	21.2ms	23.4ms	93.53	107.9	83.56ms	93.57ms
8	774.4	985	2 0.49ms	26ms	131.67	146.53	60.6ms	65.1ms
32	1015.2	1267	12.05ms	20.9ms	137.5	151.49	60.07ms	68.5ms
128	1021.7	1545	20.12ms	94ms	126.6	148.5	62.64ms	73.67ms

Table 3: Execution Time in Normal Read and Write Cases

	Workload	Base CFQ	CFQ with Priority settings	EDF with credit controlling	*Flubber*
Read	Sequential Read	64.42s	58.43s	71.28s	54.16s
	Random Read	62.11s	36.59s	53.45s	46.34s
Write	Sequential Write	65.12s	61.84s	65.88s	57.67s
	Random Write	62.25s	54.98s	54.23s	47.17s

4.3. Quality of Service Guarantees with Different Applications

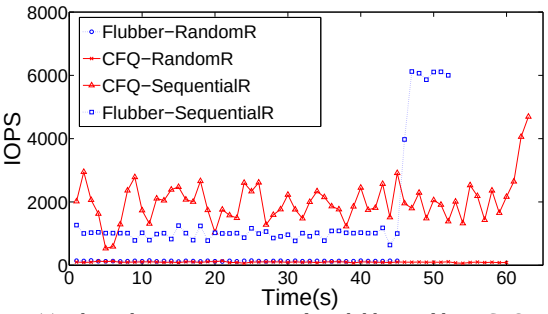
In order to evaluate *Flubber* efficiency in achieving QoS requirements of VMs and high disk I/O utilization, we compare *Flubber* with base CFQ, CFQ with priority setting, and EDF with credit controlling, when VMs running different class of workloads as described in Table 1 are deployed on a physical server. Interval factor and deadlines of VMs in *Flubber* is configured, in accordance to the empirical study described in the previous section and the latency requirement of the two classes (the interval factor β is set to 32 for all workloads). Moreover, the maximal average seek time is set to 9ms. In addition, the parameter configuration of EDF with credit controlling is the same as of *Flubber*. Under CFQ with priority settings [9], the VM with workload of high throughput and insensitive latency is set to best-effort priority and the VM with workload of sensitive latency is set to real-time priority according to different requirements of the workloads.

4.3.1. Read Applications

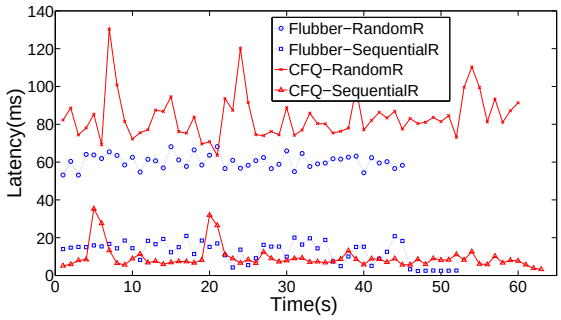
To evaluate the effectiveness of *Flubber* with read applications, we deploy two VMs (*SeqR* VM and *RndR* VM) on the server. Fig. 3 shows the throughput and latency of the two classes of read workloads achieved in *Flubber* in contrast with the three schedulers. Fig. 3a and Fig. 3b show that, besides meeting the requirements of the two-classes of applications, *Flubber* reduces the waste of the physical disk bandwidth and enforces stable throughput for VMs. In contrast, base CFQ leads to worse performance of *RndR* VM and a serious fluctuating throughput of *SeqR* VM although meeting requirements of *SeqR* VM. This is because *Flubber* prevents *SeqR* VM from grabbing more bandwidth to guarantee the latency requirement of *RndR* VM, but base CFQ only provides each VM with the same serving time-slice and never considers QoS targets of VMs. More importantly, as shown in Table 3, the response time of the applications of VMs under *Flubber* is significantly improved in contrast to base CFQ. For instance, *Flubber* outperforms base CFQ with 17% and 25% for both the sequential and random read, respectively, because of maximizing physical disk I/O efficiency with *Flubber*.

As shown in Fig. 3c and Fig. 3d, *Flubber* and real time priority ensure the latency of random read. But *SeqR* VM with best-effort priority starves because of serving *RndR* VM first, which never happens in *Flubber*. Fig. 3e and Fig. 3f show that the performance of the two workloads, especially sequential read, under EDF with credit controlling is worse than that under *Flubber*. In addition, the response time of sequential read and random read under *Flubber* are improved by 24% and 16%, respectively, in contrast to EDF with credit controlling. This is because *Flubber* reduces disk head seek and rotational overheads, and maximizes the utilization of disk I/O with the BD-EDF mechanism.

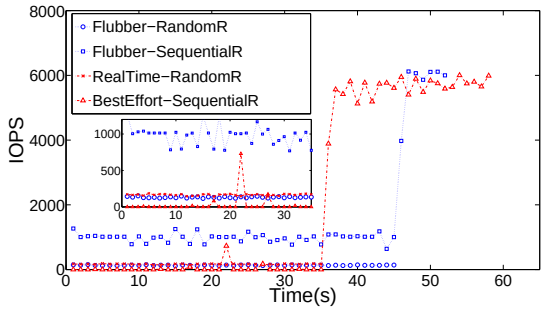
[1]In order to meet the requirements of different type of applications, CFQ scheduler provides three classes of priorities including real-time, best effort and idle state in the traditional environment. The *real-time* (RT) class is the highest priority designed for the application with sensitive time and the best-effort priority is default priority scheduler for the applications without specific IO priority. A program running with idle IO priority will only get disk time when no other program has asked for disk IO for a defined grace period.



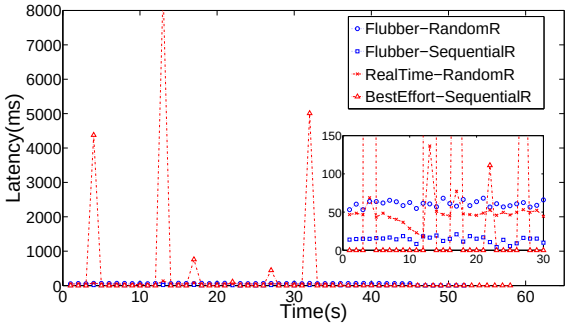
(a) Throughput comparison under *Flubber* and base CFQ



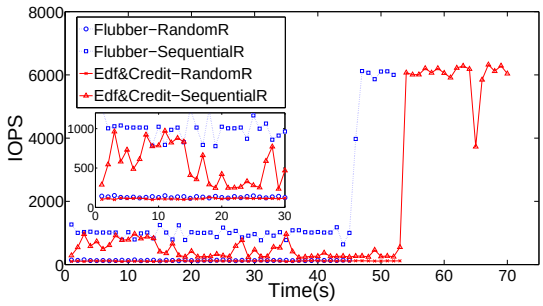
(b) Latency comparison under *Flubber* and base CFQ



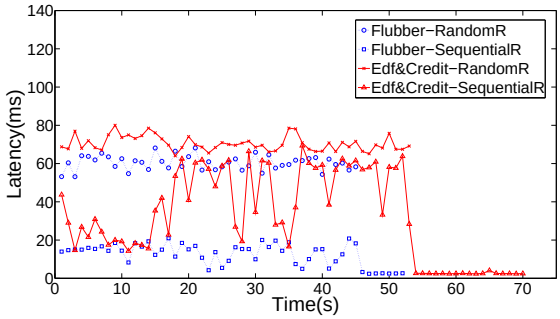
(c) Throughput comparison under *Flubber* and CFQ with priority setting



(d) Latency comparison under *Flubber* and CFQ with priority setting



(e) Throughput comparison under Flubber and EDF with credit controlling



(f) Latency comparison under Flubber and EDF with credit controlling

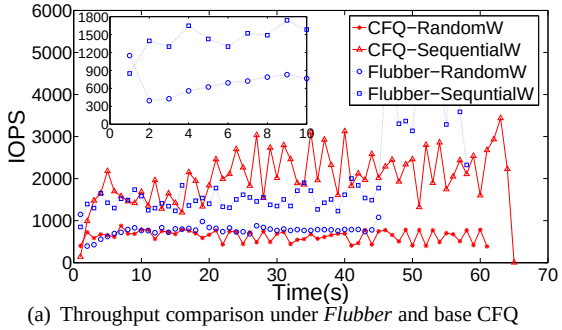
Figure 3: The achieved throughput and latency under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running two-classes of Read application

4.3.2. Write Applications

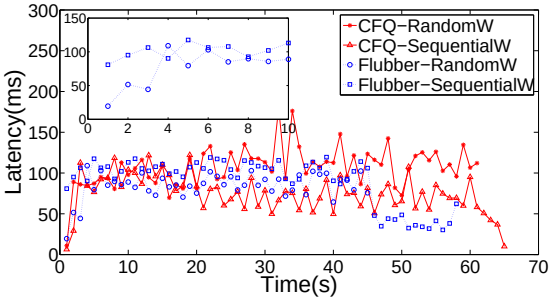
In order to validate our evaluation with different applications, we evaluate *Flubber* with write applications: two different classes of write workloads run on two VMs (*SeqW* VM and *RndW* VM).

As mentioned in section 3, unlike read applications, write applications suffer the complex relation between throughput allocation and SLO requirements. This complexity is because of the impacts of the VM cache and VMM cache when writing data [2]. As a result, it is hard to allocate the credits amount for VMs using the coarse-grained credit

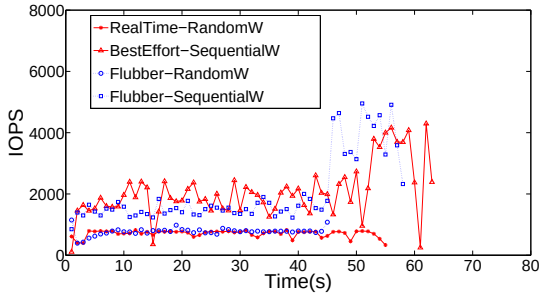
[2]In the case of write application, the write request writes the data into cache of VM and then returns, so the latency of write operation is only the caching time. When *pdflush* processes of operating system of VM and VMM scan the caches of VM and VMM (every 30s), the data will be written into disk. The aim of cache is to avoid write disk frequently and improve I/O efficient by buffering data block. To get accurate throughput and latency measurements by avoiding the caching time, we use *fsync* function for write applications in VM. The *fsync* is used for file operation and forces write request to return after transferring the data in cache into physical disk.



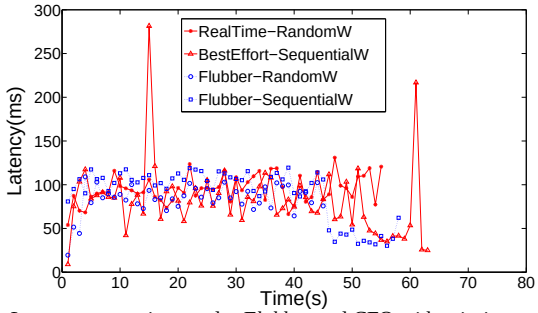
(a) Throughput comparison under *Flubber* and base CFQ



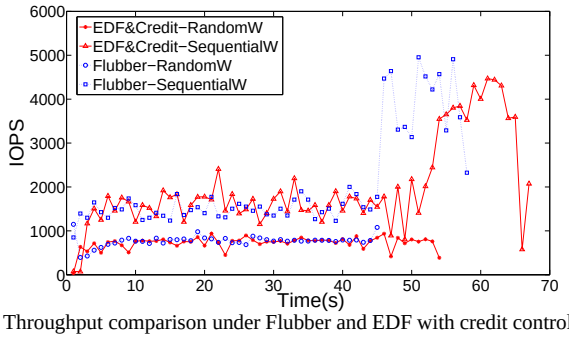
(b) Latency comparison under *Flubber* and base CFQ



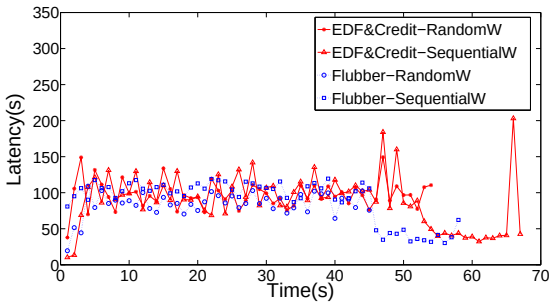
(c) Throughput comparison under *Flubber* and CFQ with priority setting



(d) Latency comparison under *Flubber* and CFQ with priority setting



(e) Throughput comparison under Flubber and EDF with credit controlling



(f) Latency comparison under Flubber and EDF with credit controlling

Figure 4: The achieved throughput and latency under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running two-classes of Write application

assignment which is based on the proportion of credits (here 2:1) and the interval factor ($\beta = 32$). Accordingly, we first zoom on the feedback-based credit adjustment policy and show how it can achieve the appropriate QoS requirement, and then similar to read applications we compare *Flubber* with three state-of-art schedulers: CFQ, CFQ with priority settings and EDF with credit controlling.

Zoom on the feedback-based credit adjustment policy. Fig. 4a and Fig. 4b zoom on the changes of throughput and latency of *SeqW* and *RndW* VMs during the first 10s. The coarse-grained credit allocation based on the proportion of throughput target of VMs canot achieve the expected throughput requirement of VMs. *SeqW* VM with high throughput requirement gets lower than 1000IOPS; instead *RndW* VM with low throughput requirement obtains more than 1000IOPS. This situation triggers the feedback-based credit adjustment policy of *Flubber* to adjust the credit assignment. As shown within 5s, both VMs get expected throughput and steady latency. After adjusting credits within 2s, the throughput of *SeqW* VM is more than 1200IOPS while the throughput of *RndW* VM is much lower than 600IOPS (almost 400IOPS), thus many random request from *RndW* VM are backlogged. At 4s, after several adjust-

Table 4: Execution Time and Average Throughput in Mixed Case

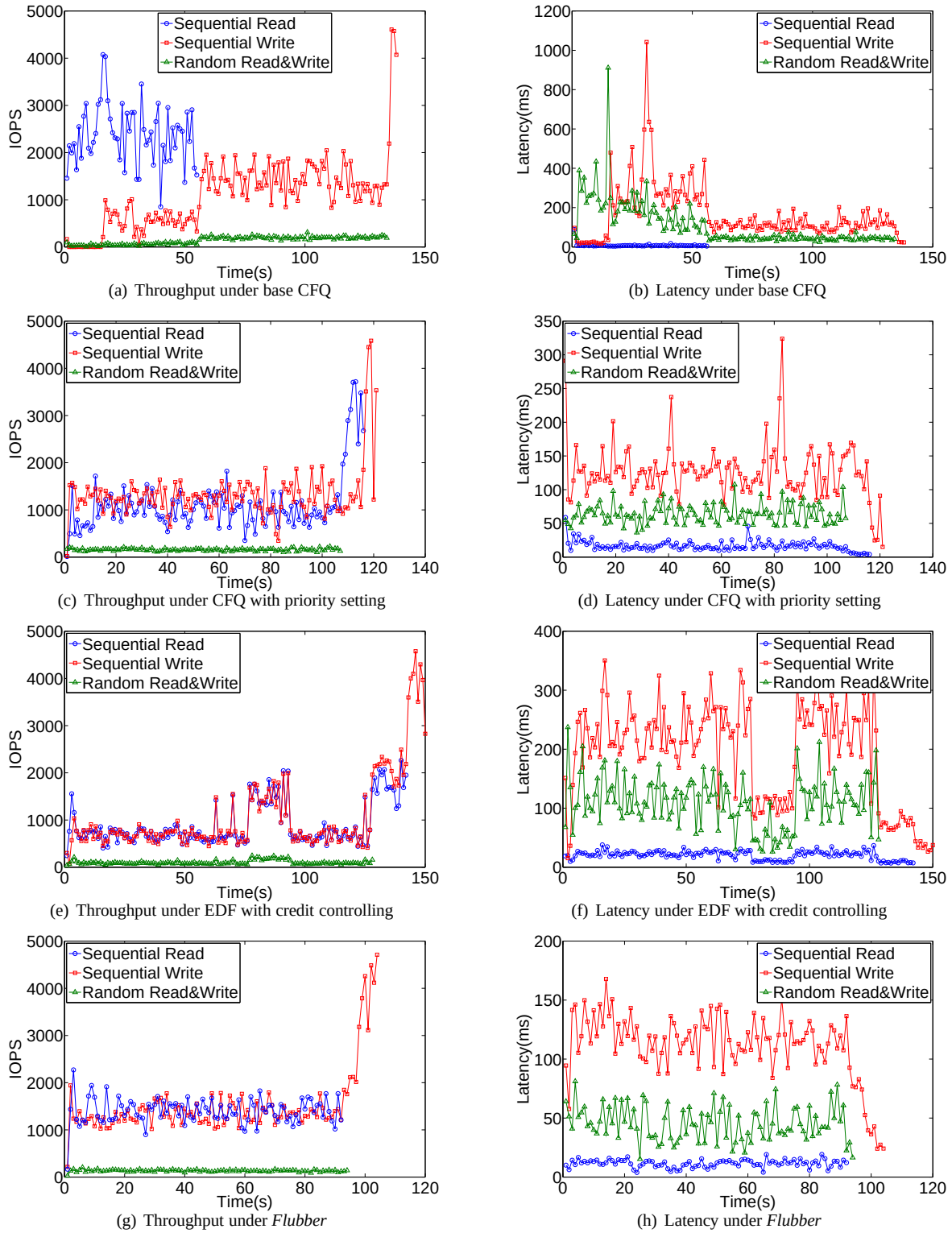
Scheduling	Sequential Read	Sequential Write	Random Read&Write	Average IOPS
Base CFQ	56.68s	134.52s	133.96s	2165.1
CFQ with Priority settings	115.84s	116.94s	107.5s	2557.1
EDF with credit controlling	142.27s	146.94s	128.55s	2026.9
Flubber	92.34s	103.57s	94.08s	2906.8

ments, the previous backlogged pending requests from *RndR* VM gets credits by increasing the proportion of credit of *RndW* VM, but this causes that many random requests write in disk simultaneously, and the latency of these request increase due to disk frequent seek. Finally at 5s, both VMs get expected throughput and steady latency. However, the feedback-based credit adjustment policy introduces a little disk seek overhead during readjusting credits but it quickly allocates appropriate credits to meet the SLO requirements of two write workloads.

QoS of write applications As described in Fig. 4, *Flubber* still ensures quite stable throughput of VMs' write-dominated workloads, in contrast with other three schedulers. On the other hand, under *Flubber* the change of latency of RndW VM is in the range of 100ms and the average latency of random write application is 86.16ms. But under the other scheduling algorithms, the latencies of two applications are often beyond the range of promised latencies and fluctuate seriously. For instance, Fig. 4b describes that the achieved latency of random write application is larger than 100ms and its average latency is 118.54ms under base CFQ, because *SeqW* VM grabs disk resource of *RndW* VM. Fig. 4d shows that the *SeqW* VM with best-effort priority, which is not starved as *SeqR* VM in scene of read operation, exhibits a higher latency jitter because of interference of *RndW* VM with real-time priority. This phenomenon never appears in *Flubber* because *Flubber* with throughput control has better performance isolation than CFQ with best-effort priority. Besides, neglecting disk seek overhead never achieves stable performance of VM, even if controlling throughput and latency. Although *RndW* VM with real time priority meets the latency requirement of random write application, the average latency of random write application is 96.93ms. As shown in Fig. 4e and Fig. 4f, the performance of two write workloads under *Flubber* is more stable than under EDF with credit controlling, although both schedulers control the throughput of two VMs to ensure performance isolation. The average latency of random write application under EDF with credit controlling is 97.35ms although meeting the latency requirement. So the average latency of random write application under *Flubber* is much lower than under other three schedulers. In addition, due to maximizing disk I/O efficiency with BD-EDF mechanism, *Flubber* outperforms base CFQ with 11% and 24% for both the sequential and random write respectively as shown in Table 3.

4.3.3. Mixed Applications

To evaluate the performance of *Flubber* further, we test VMs with mixed applications under the four schedulers: three different workloads run on three VMs deployed on a server. The sequential read application runs on *SeqR* VM, the sequential write application runs on *SeqW* VM and the random read and write application runs on *RndRW* VM. Fig. 5 shows that the achieved throughput and latency of three different workloads under *Flubber*, base CFQ, CFQ with priority setting, and EDF with credit controlling, respectively. Table 4 indicates the execution times of the three workloads and the average throughput of the server. Although base CFQ offers each VM an equal I/O time slice in block device, *SeqR* VM grabs bandwidth of other VMs, especially disk I/O resource of *SeqW VM*. This is because the hard disk prefers to handle sequential read for avoiding disk seeking and rotational overheads and the sequential read requests invoke high efficient prefetching. Base CFQ can neither meet the high throughput requirement of *SeqW* VM nor ensure the latency requirement of *RndRW* VM when *SeqR* VM runs with *SeqW* VM and *RndRW* VM. To alleviate the interference of *SeqR* VM and meet QoS of VMs, CFQ with priority setting assigns real-time priority to *RndRW* VM, first best-effort priority to *SeqW* VM, and second best-effort priority to *SeqW* VM. As shown in Fig. 5c, the throughput of sequential read application fluctuates seriously and hardly maintains 1000IOPS when three workloads run simultaneously. Besides, *RndRW* VM with the highest-priority achieves latency requirement, while the achieved latency of the sequential write application exceeds its latency bound. Fig. 5e and Fig. 5f illustrates that EDF with credit controlling hardly meets the requirements of VMs in spite of supporting performance isolation among VMs. As shown in Fig. 5g and Fig. 5h, in contrast with above three schedulers, *Flubber* guarantees throughput and latency requirements of the three different applications via the successful collaboration between VBL and VLC. *Flubber* also



(a) Throughput under base CFQ

(b) Latency under base CFQ

(c) Throughput under CFQ with priority setting

(d) Latency under CFQ with priority setting

(e) Throughput under EDF with credit controlling

(f) Latency under EDF with credit controlling

(g) Throughput under *Flubber*

(h) Latency under *Flubber*

Figure 5: The achieved throughput and latency under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running three VMs with different applications

Table 5: Execution Time with Background Process for both Read and Write applications

	Workload	CFQ with Priority settings	EDF with credit controlling	*Flubber*
Read	Sequential Read	138.639s	330.55s	117.955s
	Random Read	105.838s	84.05s	77.496s
	Backup Process	368.627s	583.464s	399.129s
write	Sequential Write	112.73s	141.95s	73.11s
	Random Write	62.64s	109.06s	57.18s
	Backup Process	1523.13s	328.018s	329.076s

keeps steady performance of applications. In addition, *Flubber* makes use of the BD-EDF mechanism to improve disk I/O efficiency of the server. For example, as described in Table 4, compared with base CFQ, *Flubber* improves the sequential write application and the random read and write application by 22% and 29%, respectively. The execution time of sequential read applications under base CFQ is shorter than under *Flubber*, but the average throughput of server under *Flubber* outperforms under base CFQ with 34%. Therefore, the above evaluation shows that *Flubber* undertakes the promised QoS, scalability and overall performance of the virtualized environment.

4.4. Different Service Requirement with Background Process

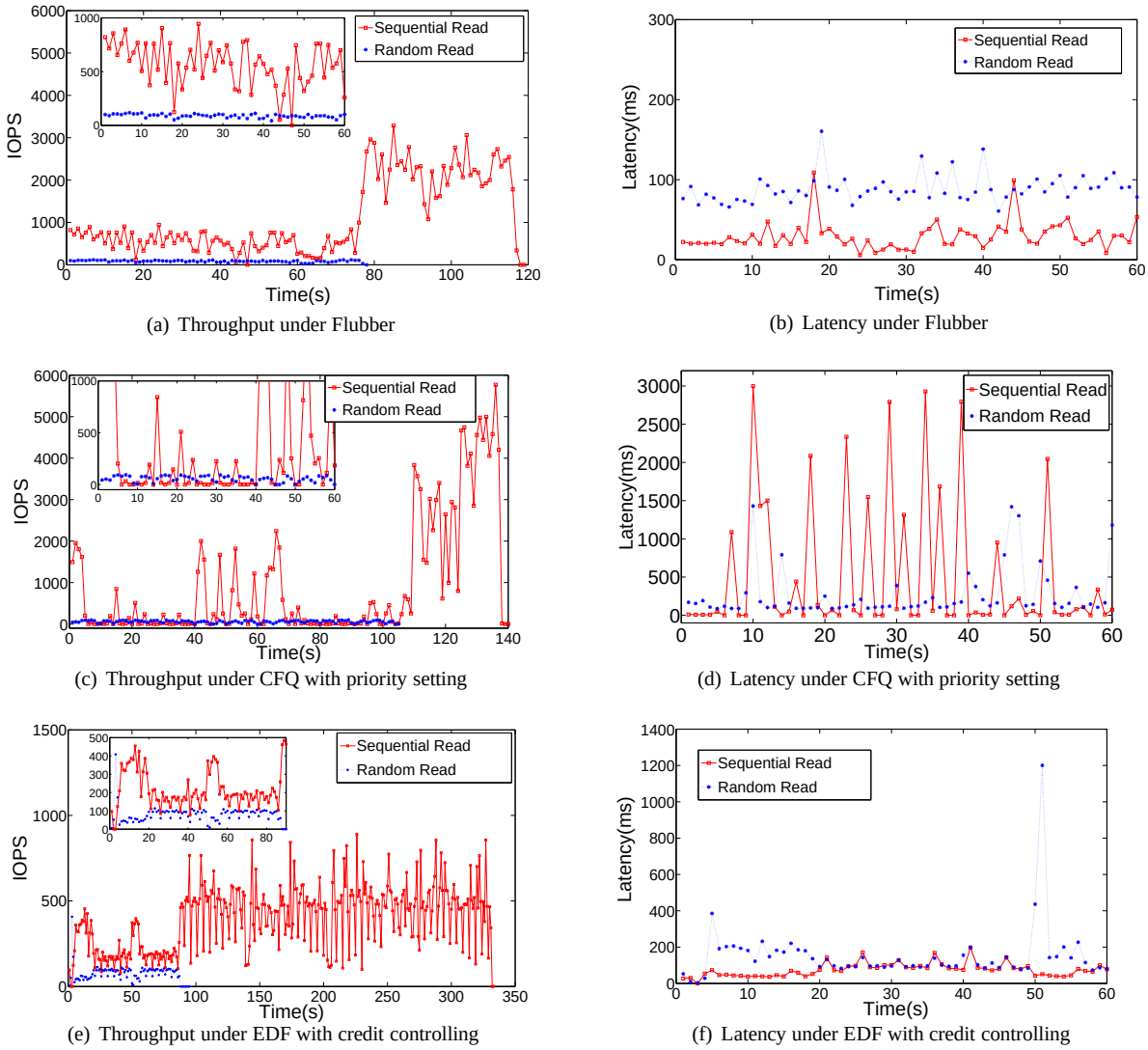
Having shown the effectiveness of *Flubber* in achieving both latency and throughput guarantees while improving the physical disk I/O efficiency in normal case, here we show how *Flubber* can maintain different requirements of VMs and disk I/O efficiency when background process is introduced. Accordingly, we have deployed two classes of VMs, and run background process — backup a VM image with a size of 12GB — in the VMM.

Read Applications. Two VMs, *SeqR* VM and *RndR* VM, are running on the server while a backup process is running in the VMM. The latency and throughput of both VMs under the three schedulers are presented in Fig. 6 and the response time of the workloads within the two VMs and the backup process are presented in Table 5. As shown in Fig. 6, *Flubber* stabilizes the latency of both workloads while improving the throughput of two VMs. In contrast with the CFQ with priority settings, both the latency and throughput of both VMs are seriously fluctuating, because the background process, at some points, grabs most of the bandwidth. Under EDF with credit controlling, the VM with sequential read suffers seriously from throughput fluctuation because of disk head seek overhead. Moreover, in contrast with CFQ with priority settings, the overall response time of the VMs is improved, for example 15% and 26% for both sequential and random read, while the background process performance is slightly degrading, less than 8%. *Flubber* also outperforms EDF with 64%, 8%, and 31% for the sequential read, random read and the background process, respectively, due to maximizing disk I/O utilization.

Write Applications. Similar to read applications, we also evaluate *Flubber* with write applications when background process runs in the VMM. We run the same two VMs with write workloads, *SeqW* VM and *RndW* VM, on the same server while backup process runs in VMM. Fig. 7 shows the achieved throughput and latency of the two write applications under *Flubber*, CFQ with priority setting, and EDF with credit controlling, and Table 5 demonstrates execution time of two write applications in VMs and background process under these schedulers. As shown in Fig. 7, *Flubber* guarantees not only the average SLO of two write applications in VMs but also guarantees stable throughput although backup process runs on VMM. The results are the same as the performance of read operation in *Flubber*. As shown in Table 5, *Flubber* improves the performance of the sequential write, the random write and the backup process by 35%, 9%, 78%, respectively, in contrast to CFQ with priority setting. The reason is that *Flubber* not only provides good bandwidth allocation but also lowering the disk head seek by BD-EDF mechanism. Due to neglecting disk seek overheads, EDF with credit controlling never avoids the two classes of write workloads from the impact of backup process although setting the deadline of the process to 500ms. As shown in Table 5, *Flubber* also outperforms EDF with 48.5%, 47.5% for the sequential write and random write, respectively, due to maximize disk I/O utilization.

4.5. Disk Head Seek and Rotational Overheads

In order to evaluate goodness of BD-EDF mechanism of *Flubber*, we use *blktrace* to trace the physical disk head seek and then return the success ratio of the waits introduced by the BD-EDF algorithm in leading to dispatched requests of a small distance of disk head seeking (mostly zero) without violating the latency requirements of the VMs.



(a) Throughput under Flubber

(b) Latency under Flubber

(c) Throughput under CFQ with priority setting

(d) Latency under CFQ with priority setting

(e) Throughput under EDF with credit controlling

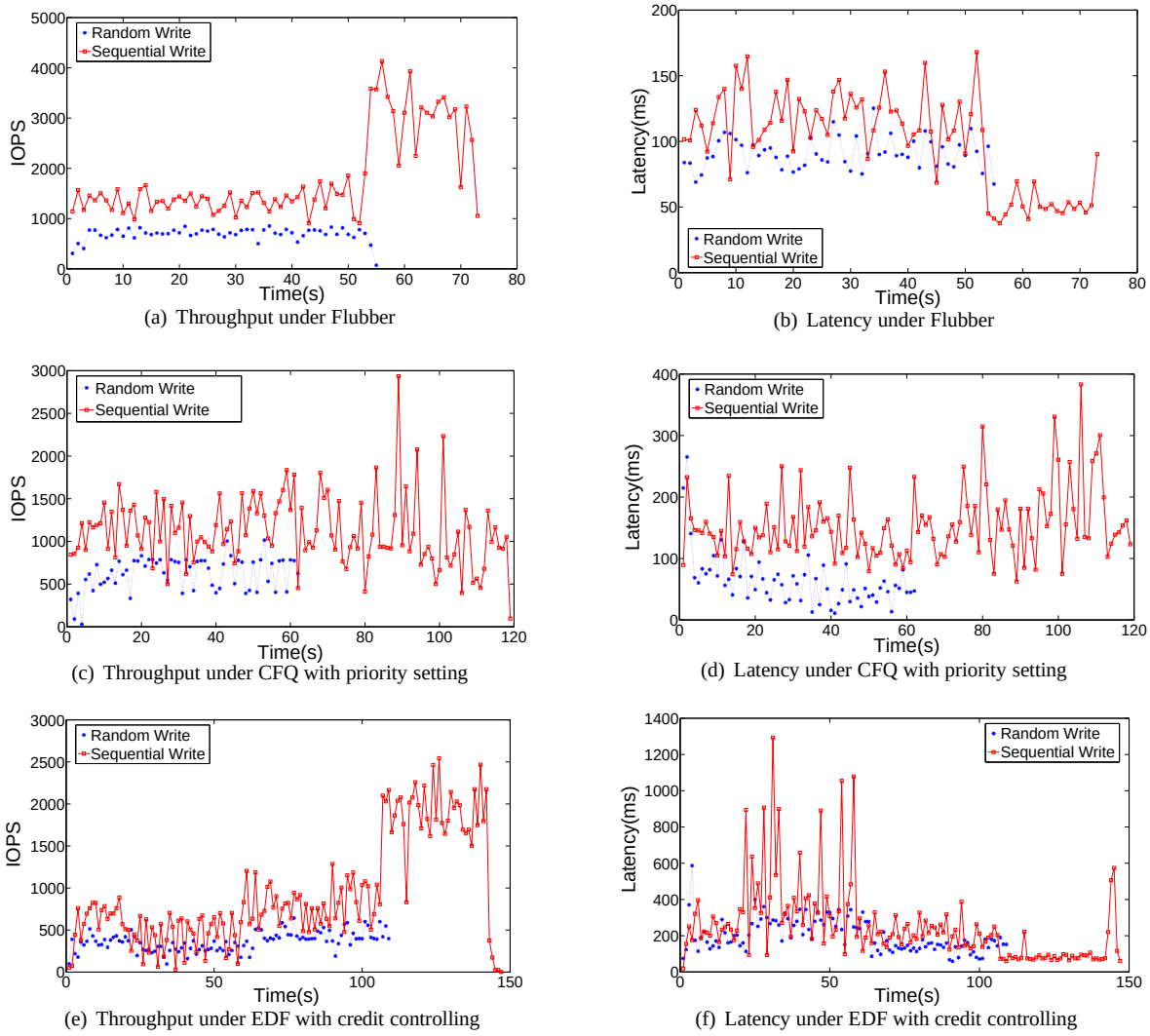
(f) Latency under EDF with credit controlling

Figure 6: The achieved throughput and latency under *Flubber*, CFQ, and EDF with credit controlling, when running two-classes of Read applications and backup runs in VMM

Read applications. Fig. 8 presents the physical disk head seek distance under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when *SeqR* VM and *RndR* VM run on the server. The disk head seek distance is different between the end address of the previous dispatched request and the start of the new dispatched request. Table 6 describes the distribution of disk seek distance, where D is disk seek distance (the seek is bounded [3] from 5×10^7 to -5×10^7 which is the space taken by the image of *SeqR*/*RndR* VM).

As shown in Fig. 8, in comparison with base CFQ and EDF with crediting controlling, *Flubber* reduces disk head seeking by delaying the read request to improve disk I/O efficiency without any QoS violation. For instance,

[3]The D can be outside the boundaries of the virtual disk (larger than 5×10^7 or smaller than -5×10^7) when: (1) The previous dispatched I/O request and the currently dispatched I/O request are not from the same VM. (2) The second is that VM process accesses the *inode* of VM image when updating the meta of image, such as the access time, the size of image, the changes of file content. The position of *inode* locates at the head of disk layout generally. (3) The third is the writing of log files (logs process of file system and operating system) which normally layouts at the start of the disk.



(a) Throughput under Flubber

(b) Latency under Flubber

(c) Throughput under CFQ with priority setting

(d) Latency under CFQ with priority setting

(e) Throughput under EDF with credit controlling

(f) Latency under EDF with credit controlling

Figure 7: The achieved throughput and latency under *Flubber*, CFQ, and EDF with credit controlling, when running two-classes of write applications and backup runs in VMM

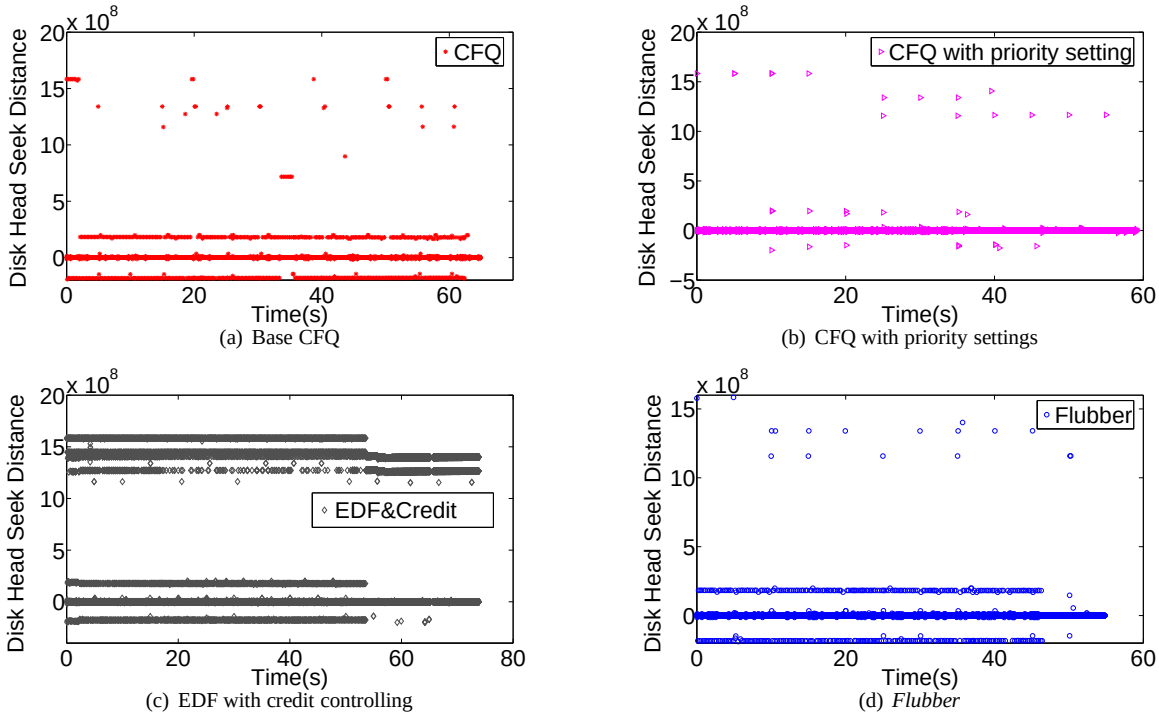
the proportion of seek distance at zero point under *Flubber* is higher than the other two base CFQ and EDF with credit controlling. This means BD-EDF mechanism batches as many requests from the same VMs as possible without exceeding their deadline, especially requests with close addresses, for reducing the disk seek overhead. However, the proportion of seek distance at zero point under *Flubber* is a bit lower than under CFQ with priority setting. This can be explained because *SeqR* VM with best-effort priority does not execute the sequential read application until the random read running within the *RndR* VM with real-time priority has finished. Unfortunately, this comes at the cost of QoS requirement of VMs (as explained earlier in section 4.3). It worths to mention that the proportion of seek distance of EDF with credit controlling outside the aforementioned boundaries is more than 40% in read case which is due to that EDF with credit controlling only focuses on deadline of VM requests and ignores the access locality of VMs, leading to high disk seek overheads when frequently switching between request from the two VMs.

In addition, we find that *Flubber* achieves 96% success ratio on waiting for a suitable future request in the normal read case by tracing requests of VMs.

Write applications. Fig. 9 shows the physical disk head seek distance under the four schedulers in the normal write

Table 6: Disk Seek Distance Distribution in Three different Cases

	Disk I/O scheduler	$d = 0$	$d \neq 0 \cap -5\times10^7 \leq d \leq 5\times10^7$	$-5\times10^7 \leq d \leq 5\times10^7$	$-5\times10^7 > d \cup d > 5\times10^7$
Read	Base CFQ	79.5%	18.41%	97.91%	2.19%
	CFQ with priority setting	81.8%	18.12%	99.92%	0.08%
	EDF with credit controlling	47.33%	12.13%	59.46%	40.54%
	Flubber	81.4%	18.56%	99.96%	0.04%
Write	Base CFQ	31.75%	53.55%	85.3%	14.7%
	CFQ with priority setting	36.78%	54.12%	90.9%	9.1%
	EDF with credit controlling	12.83%	18.97%	31.8%	68.2%
	Flubber	46.02%	48.74%	94.76%	5.24%
Mixed	Base CFQ	61.75%	34.24%	95.99%	4.01%
	CFQ with priority setting	62.02%	34.75%	96.77%	3.23%
	EDF with credit controlling	21.87%	20.08%	41.95%	50.05%
	Flubber	75.02%	22.57%	97.59%	2.41%



(a) Base CFQ (b) CFQ with priority settings (c) EDF with credit controlling (d) *Flubber*

Figure 8: Disk head seek distance under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running two-classes of Read applications

case: BD-EDF mechanism also improves the disk I/O efficiency by reducing disk head seeking in contrast with the three other schedulers: the proportion of seek distance at zero point under *Flubber* is 46% while it is 31.75%, 36.78%, and 12.83% under base CFQ, CFQ with priority settings and EDF with credit controlling, respectively. In general, the disk head seeking for write application is worse than that for read application, because the *fsync* function (which we use to avoid the impact of caches in the write case) reduces the merging of close write requests. Also, we note that the disk head seek distance is more often outside the range (decided by the VM space) for write application than in the case of read application. This can be explained because the frequent access of *inode* and frequent writing of log files.

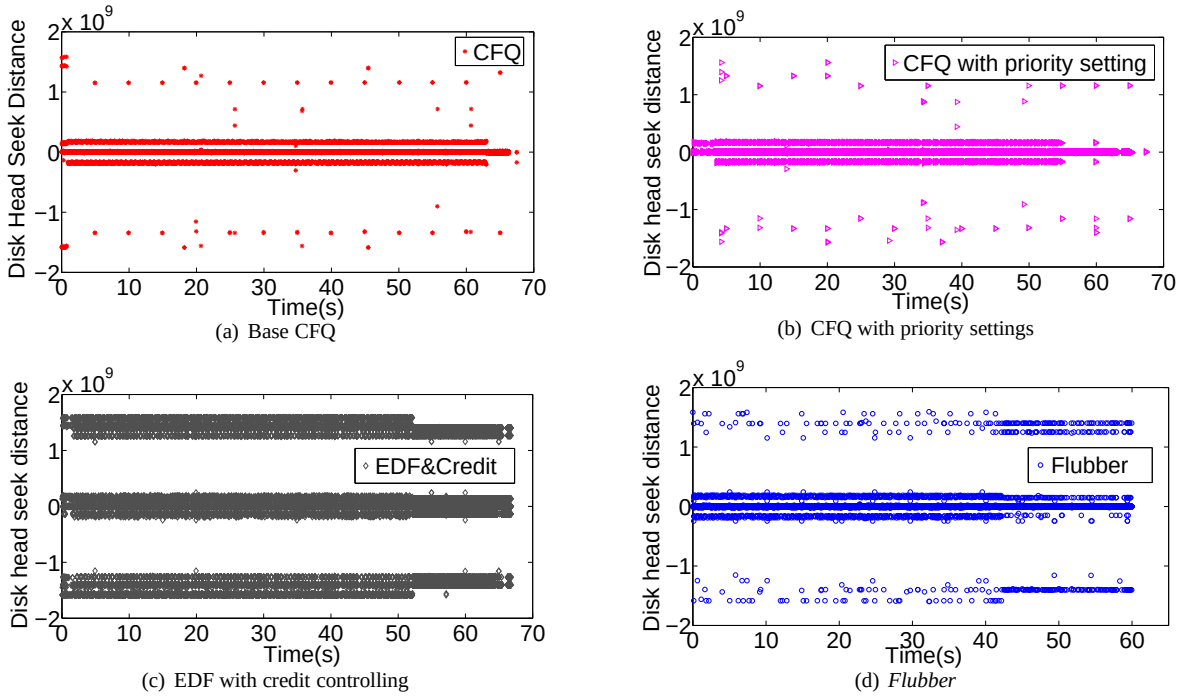


Figure 9: Disk head seek distance under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running two-classes of Write applications

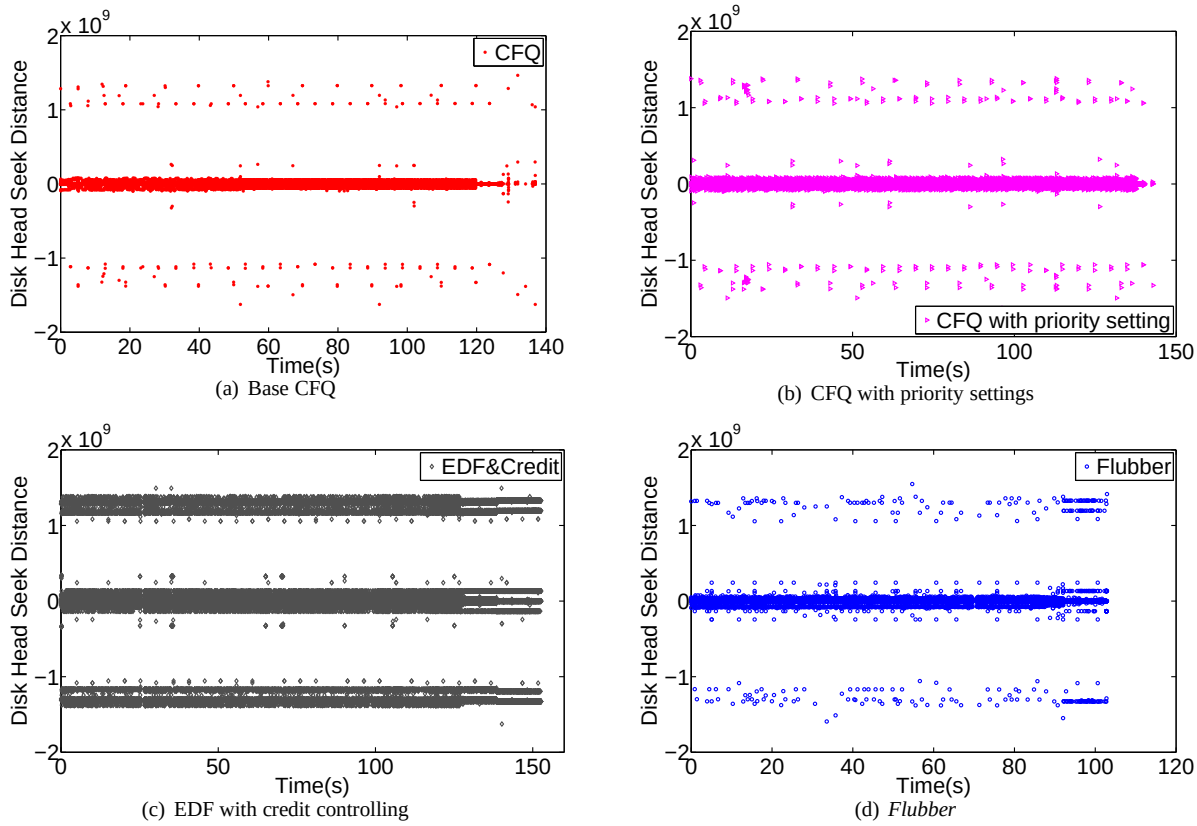
Besides, compared with in normal read case, *Flubber* only achieves 83% success ratio on waiting a suitable future request in the normal write case.

Mixed applications. Fig. 10 shows the physical disk head seek distance under the four schedulers in the mixed read and write case. Although the number of VMs with different I/O characters is increased, disk head seek overheads under *Flubber* is lower than under other schedulers. For instance, Table 6 presents that the proportion of seek distance at zero point under *Flubber* is 75.02%, but it is 61.75%, 62.02%, and 21.87% under base CFQ, CFQ with priority settings and EDF with credit controlling, respectively. The reason is that the BD-EDF mechanism batches as much requests with close LBAs as possible while guaranteeing deadlines of requests. Moreover, because of write request, the proportion of disk head seek distance is also outside the range (decided by the VM space). *Flubber* also achieves 91% success ratio on waiting a suitable future request in the mixed applications case.

5. Conclusions and Future Work

Virtualization has become the fundamental technique for cloud computing [34]: it enables multiple VMs with diverse applications to run within the same physical server. The current emergence of cloud computing has led to an increasing importance of QoS in virtualized environments. Cloud providers must deliver and maintain reasonable (expected) quality of services in order to attract/keep their users.

In this paper, we target the problem of ensuring the QoS of different VMs deployed on the same Xen-based hosted platform, in respect to their desired throughput and latency. We present a two-level scheduling framework, named *Flubber*, in a Xen-based hosting platform that decouples latency and throughput allocation. The high-level throughput control provides fine-grained regulation of the pending requests from the VMs with a feedback-based policy to meet the throughput requirements of different VMs and ensure performance isolation. Meanwhile, the low-level latency control, by the virtue of the batch and delay EDF mechanism, reorders all pending requests from VMs based on their deadlines. In addition, it batches them to the disk device taking into account the spatial locality of accesses across VMs. We implement a prototype of *Flubber* in a Xen-based host. We demonstrate the efficiency and benefits



(a) Base CFQ (b) CFQ with priority settings

(c) EDF with credit controlling (d) *Flubber*

Figure 10: Disk head seek distance under *Flubber*, base CFQ, CFQ with priority settings, and EDF with credit controlling, when running mixed applications

of *Flubber* through experiments that involve different classes of applications including read and write applications. Our experiment demonstrates that our solution is practical and feasible. For instance, the disk head seeking time is significantly reduced because of the adopted BD-EDF mechanism, e.g., in the case of read applications, 96% of the request waits introduced by the BD-EDF algorithm have led to dispatched requests of a small distance of disk head seeking (mostly zero) without violating the latency requirements of the VMs. *Flubber* also shows improvement of up 25% in the VM performance over state-of-art approaches, e.g., in contrast to the Xen default disk I/O scheduler (CFQ scheduler) [9], beside achieving the desired QoS of each VM, *Flubber* speeds up the sequential and random read by 17% and 25%, respectively.

In the future, we will explore different delay mechanisms to improve our system. Moreover, we want to explore the opportunities when using *Flubber* in the hybrid environment consisting of hard disk drives (HDDs) and solid state disks (SSDs) in virtualized environments: while *Flubber* can ensure the quality of service of the applications and ensure high disk utilization, SSDs can be used to response fast to some requests with critical latency requirements by re-directing the requests to them taking into account the monetary cost.

Acknowledgments

The research is supported by National Science Foundation of China under grantNo.61232008, 61133008,61073024, and 61003007, Outstanding Youth Foundation of Hubei Province under grant No.2011CDA086, and National High-Tech R&D Plan of China under Grant 2013AA01A213. This work is also supported by the Hemera INRIA Large Wingspan-Project.

References

[1] L. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 50–55.

[2] V. Stantchev, C. Schröpfer, Negotiating and enforcing QoS and SLAs in grid and cloud computing, in: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing (GPC'09), 2009, pp. 25–35.

[3] S. Ibrahim, B. He, H. Jin, Towards pay-as-you-consume cloud computing, in: Proceedings of the 2011 IEEE International Conference on Services Computing (SCC'11), 2011, pp. 370–377.

[4] H. Kim, H. Lim, J. Jeong, H. Jo, 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), 2009, pp. 101–110.

[5] D. Ongaro, A. L. Cox, S. Rixner, Scheduling I/O in virtual machine monitors, in: Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08), 2008, pp. 1–10.

[6] X. Wang, J. Zang, Z. Wang, Y. Luo, X. Li, Selective hardware/software memory virtualization, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'11), 2011, pp. 217–226.

[7] A. Menon, A. L. Cox, W. Zwaenepoel, Optimizing network virtualization in xen, in: Proceedings of the USENIX Annual Conference (ATC'06), 2006, pp. 2–2.

[8] S. K. Barker, P. Shenoy, Empirical evaluation of latency-sensitive application performance in the cloud, in: Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys'10), 2010, pp. 35–46.

[9] Completely Fair Queuing (CFQ), `http://mirror.linux.org.au/pub/linux.conf.au/2007/video/talks/123.pdf`.

[10] L. Huang, G. Peng, T. Chiueh, Multi-dimensional storage virtualization, SIGMETRICS Perform. Eval. Rev. 32 (1) (2004) 14–24.

[11] M. Wachs, M. Abd-El-Malek, E. Thereska, G. R. Ganger, Argon: performance insulation for shared storage servers, in: Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST'07), 2007, pp. 5–5.

[12] J. C. Wu, S. A. Brandt, The design and implementation of AQuA: an adaptive quality of service aware object-based storage device, in: Proceedings of the 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies (MASST'06), 2006, pp. 209–218.

[13] W. Jin, J. S. Chase, J. Kaur, Interposed proportional sharing for a storage service utility, SIGMETRICS Perform. Eval. Rev. 32 (1) (2004) 37–48.

[14] S. R. Seelam, P. J. Teller, Virtual I/O scheduler: a scheduler of schedulers for performance virtualization, in: Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07), 2007, pp. 105–115.

[15] A. Gulati, I. Ahmad, C. A. Waldspurger, PARDA: proportional allocation of resources for distributed storage access, in: Proccedings of the 7th Conference on File and Storage Technologies (FAST'09), 2009, pp. 85–98.

[16] P. Padala, K. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, Automated control of multiple virtualized resources, in: Proceedings of the 4th ACM European Conference on Computer systems (EuroSys'09), 2009, pp. 13–26.

[17] A. Gulati, A. Merchant, 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), 2010, pp. 1–7.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP'03), 2003, pp. 164–177.

[19] J. Bruno, J. Brustoloni, E. Gabber, M. Mcshea, B. Ozden, A. Silberschatz, Disk scheduling with quality of service guarantees, in: Proceedings of IEEE International Conference on Multimedia Computing and Systems, IEEE, 1999, pp. 400–405.

[20] P. J. Shenoy, H. M. Vin, Cello: a disk scheduling framework for next generation operating systems, SIGMETRICS Perform. Eval. Rev. 26 (1) (1998) 44–55.

[21] D. Boutcher, A. Chandra, Does virtualization make disk scheduling passé?, SIGOPS Oper. Syst. Rev. 44 (1) (2010) 20–24.

[22] M. Kesavan, A. Gavrilovska, K. Schwan, Differential virtual time (DVT): rethinking I/O service differentiation for virtual machines, in: Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10), 2010, pp. 27–38.

[23] S. Ibrahim, H. Jin, L. Lu, B. He, S. Wu, Adaptive disk I/O scheduling for mapreduce in virtualized environment, in: Proceedings of the 2011 International Conference on Parallel Processing (ICPP'11), 2011, pp. 335–344.

[24] D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, T. P. Lee, Performance virtualization for large-scale storage systems, in: Proceedings of the 22th International Symposium on Reliable Distributed Systems (SRDS'03), 2003, pp. 109–118.

[25] M. Karlsson, C. Karamanolis, X. Zhu, Triage: Performance differentiation for storage systems using adaptive control, ACM Trans. Storage 1 (4) (2005) 457–480.

[26] G. Soundararajan, C. Amza, Towards end-to-end quality of service: controlling I/O interference in shared storage servers, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08), 2008, pp. 287–305.

[27] A. Gulati, A. Merchant, P. J. Varman, pClock: an arrival curve based approach for QoS guarantees in shared storage systems, in: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'07), 2007, pp. 13–24.

[28] J. Zhang, A. Sivasubramaniam, Q. Wang, A. Riska, E. Riedel, Storage performance virtualization via throughput and latency control, ACM Trans. Storage 2 (3) (2006) 283–308.

[29] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, C. Maltzahn, Efficient guaranteed disk request scheduling with fahrrad, in: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems (Eurosys'08), 2008, pp. 13–25.

[30] A. Gulati, A. Merchant, M. Uysal, P. J. Varman, Efficient and adaptive proportional share I/O scheduling, Technical Report HPL-2007-186.

[31] P. Valente, F. Checconi, High throughput disk scheduling with fair bandwidth distribution, IEEE Trans. Comput. 59 (9) (2010) 1172–1186.

[32] Sysbench manual, `http://sysbench.sourceforge.net/docs/`.

[33] Blktrace User Guide, `http://www.cse.unsw.edu.au/ aaronc/iosched/doc/blktrace.html`.

[34] H. Jin, S. Ibrahim, T. Bell, L. Qi, H. Cao, S. Wu, X. Shi, Tools and technologies for building clouds, in: Cloud Computing: Principles Systems and Applications, 2010, pp. 3–20.

Vitae

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He is now Dean of the School of Computer Science and Technology at HUST. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. Jin is a senior member of the IEEE and a member of the ACM. Jin is the member of Grid Forum Steering Group (GFSG). He has co-authored 15 books and published over 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security.

Xiao Ling received her B.S. (2008) degree at National University of Defense Technology and M.S. (2011) degree at Huazhong University of Science and Technology. Currently she is a Ph.D. student at Huazhong University of Science and Technology. Her research interests are in the area of cloud computing and virtualization, focusing on virtualized I/O optimization in cloud.

Shadi Ibrahim is a postdoctoral researcher within the KerData team at INRIA Rennes, working on scalable distributed data processing in the Cloud. He holds a Ph.D. in Computer Science from Huazhong University of Science and Technology in Wuhan of China. He has several years of experience with academic research at Huazhong University of Science and Technology, INRIA research center and Microsoft research center in Asia. His research interests include cloud computing, data-intensive computing, file and storage systems.

Wenzhi Cao received his B.S. (2005) degree at Chongqing University of Posts and Telecommunications. Now he is a Ph.D. student at Huazhong University of Science and Technology. His research interest is in the area of virtualization technology, focusing on virtualized network I/O.

Song Wu a professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. from HUST in 2003. He is now the director of Parallel and Distributed Computing Institute at HUST. He is also served as the vice director of Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL) of HUST. His current research interests include grid/cloud computing and virtualization technology.

Gabriel Antoniu is a senior research scientist at INRIA. He is leading of the KerData research team at INRIA Rennes - Bretagne Atlantique. He received his Ph.D. degree in Computer Science in 2001 from ENS Lyon and his Habilitation for Research Supervision (HDR) from ENS Cachan in 2009. His research interests include: big data, grid and cloud storage, large-scale distributed data management and sharing, data consistency models and protocols, grid and peer-to-peer systems. In the area of distributed data storage, he is leading the MapReduce ANR project (2010-2013) in partnership with Argonne National Nab (USA), University of Illinois at Urbana Champaign (USA), and INRIA-UIUC Joint Lab for Petascale Computing and IBM France. He is co-leading the AzureBrain INRIA-Microsoft Research project (2010-2013) on optimized cloud storage for joint genetics and neuroimagiong analysis.