This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

VMCD: A Virtual Multi-Channel Disk I/O Scheduling Method for Virtual Machines

Huailiang Tan, Chao Li, Zaihong He, Keqin Li, *Fellow, IEEE*, and Kai Hwang, *Fellow, IEEE*

Abstract—In the era of cloud computing and big data, virtualization is gaining great popularity in storage systems. Since multiple guest virtual machines (DomUs) are running on a single physical device, disk I/O fairness among DomUs and aggregated throughput remain the challenges in virtualized environments. Although several methods have been developed for disk I/O performance virtualization among multiple DomUs, most of them suffer from one or more of the following drawbacks. (1) A fair scheduling mechanism is missing when requests converge together from multiple queues. (2) Existing methods rely on better performance of the underlying storage system such as Solid State Drive (SSD). (3) Throughput and latency are not considered simultaneously. To address these disadvantages, this paper presents a Virtual Multi-Channel of Disk I/O (VMCD) method that can be built on top of an ordinary storage utility, which mitigates the interference among multiple DomUs by using separated virtual channel (V-Channel) and an I/O request queue for each DomU. In our VMCD, several mechanisms are employed to enhance the I/O performance, including a credit allocation mechanism, a global monitoring strategy, and a virtual multi-channel fair scheduling algorithm. The proposed techniques are implemented on the Xen virtual disk and evaluated on Linux guest operating systems. Experiments results show that VMCD increases fairness by 70% approximately compared with CFQ and Anticipatory schedulers, by 30% approximately compared with Deadline scheduler; and enhances bandwidth utilization by 28% approximately compared with CFQ and Anticipatory schedulers, by 37% compared with Deadline in the case of three or more virtual DomUs running on the same physical host.

Index Terms—Bandwidth utilization, Fairness, I/O bandwidth, Virtualization, Virtual channel.

1 Introduction

The virtual machine (VM) technology has become an essential issue of the I/O performance in storage systems, especially with the fast development of cloud computing [35]. Many techniques, such as CPU, memory, and network in VMs, have been well researched, and significant progress has been made in such fields. However, disk I/O sharing is still a challenge in virtualized systems [27]. The low efficiency caused by disk contention has become one of the severest problems that affect the performance of virtualized systems.

Optimally managing and allocating I/O resource among numerous VMs is an effective and important technique to enhance disk I/O performance. The well-established resource allocation strategies in traditional VM for network I/O resources are that multiple DomUs willfully consume resources based on respective maximal requirement. Each DomU's maximal bandwidth is restricted by the credit mechanism [25][32]. However, the credit number cannot be dynamically allocated and adjusted based on actual status, which may lead to bandwidth waste in some DomUs while others cannot get enough resources. Under circumstances where numerous VMs share the same physical device, disk I/O is usually regarded as a bottleneck in overall performance [39]. In the original Xen virtualized environment [4], when a number of VMs are running concurrently, the backend manages multiple virtual I/O devices simultaneously. I/O requests from multiple DomUs are extracted from the corresponding I/O rings, and added into a shared queue waiting to be scheduled. The scheduling algorithm in current Dom0 (control domain) cannot distinguish which VM issues the request, so excessive I/O requests in a certain period of time would hamper the process of other VMs, which causes an unfair and unstable allocation of disk I/O bandwidth. In addition, the pattern of sequential access may be changed. For a request sequence of disk I/O from a certain DomU, two requests (request1 and request2) access the disk one after another. However, an I/O request needs to go through a number of paths. Especially, when the request enters into Dom0 through the front driver, access to the disk can be admitted after it has gone through Blktap (Blktap is mainly used to transform 64-bit virtual disk addresses into offsets in disk image files using the synchronization pattern) [11] and I/O scheduler. Because the Blktap and I/O scheduler are shared by all DomUs, request1 and request2 compete with I/O requests coming from other DomUs. Under such circumstances, a sequential access may be turned into a random access.

Some research efforts have been directed toward enhancing disk I/O performance in virtualized environments. In addition to the traditional disk I/O scheduling algorithms (i.e., SSTF (Shortest-Seek-Time-First) [8], SCAN [8], and LOOK (Lookahead) [36]) used in typical operating systems, these efforts focus on further exploiting the performance of SSD (Solid State Drive) [5][12][21], scheduling frameworks (i.e., two-level scheduling framework in which one level guarantees the throughput and the other meets the latency requirements [20] [40]), and scheduling algorithm (i.e., using the gen-

* *H. Tan, C. Li, Z. He, and K. Li are with the College of Information Science and Engineering, Hunan University, Changsha 410082, China. Email: tanhuailiang@hnu.edu.cn(Huailiang Tan), S12101025@hnu.edu.cn(Chao Li)*
* *Keqin Li is also with Department of Computer Science, State University of New York, New Paltz, New York 12561, USA. E-mail: lik@newpaltz.edu.*
* *Kai Hwang is with Department of Electrical Engineering and Computer Science, University of Southern California, Los Angeles, CA 90089-2562, USA. E-mail: kaihwang@usc.edu.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

eralized processor sharing principle to meet the throughput requirements [10] [29] [1]). However, the aforementioned solutions lead to either performance penalty in some irrelevant paths, or over-provisioning when multiple VMs are running simultaneously. Moreover, most of them consider throughput and latency requirements but ignore the fairness of virtual disk bandwidth allocation. Besides, although some solutions adopt the disk I/O multi-queue scheduling method to eliminate the interference from other VMs, they still fail to effectively provide the fairness of bandwidth allocation. For example, a double-layered scheduling structure is proposed in [40], with the lower layer satisfying the delay demand and the higher layer evenly allocating disk bandwidth. In this architecture, the higher layer provides an independent request queue for each DomU, and the lower layer has a single queue whose requests come from all queues in the higher layer. This structure lacks a fairness mechanism when requests from multiple queues (high layer) are pushed into a single queue (low layer), which may lead to interference and bandwidth allocation unfairness.

Due to the defects of existing I/O resources allocation schemes, we need to develop a new improved approach to overcoming the inefficiency of existing schemes. For virtualized network I/O, we designed a model named Dynamic Mapping of Virtual Links (DMVL) in [33]. For virtualized disk I/O, the policy becomes more complex, because it needs to consider two aspects of bandwidth and latency simultaneously. Taking into account the complexity, in this paper, we propose a novel method called Virtual Multi-Channel of Disk (VMCD), which sets up independent virtual channels (V-Channel) in I/O devices and reasonably provides fair and stable disk I/O bandwidth for each VM. The VMCD model is designed out of two motivations. Firstly, the I/O scheduling in Dom0 aggregates all the disk I/O requests from each VM, so fairly and stably allocating disk I/O bandwidth needs to be achieved here. Secondly, the disk I/O requests in each DomU go through a number of complicated processes, and their access pattern may be interfered by other DomUs. However, current virtualization technologies cannot provide guarantee for sequential I/O access in each DomU. Inspired by the credit mechanism in Xen that restricts the maximum bandwidth for each VM, we dynamically adjust disk I/O bandwidth of the V-Channel according to the fluctuation of credit number that can be monitored in real time at every link. A shared log is leveraged to record information such as previous utility of bandwidth, length of Dom0 request sequence in each VM, and allocation of both read and write requests. Bandwidth limitation module is optimized by the method of dynamic bandwidth adjustment based on the shared log. In addition, with the cooperation of disk I/O fair scheduling algorithm, independent I/O request sequences and disk I/O request's deadline for each VM can be guaranteed in Dom0. VMCD is completely based on pure software approaches and has no need of special disk hardware support.

The main contributions of our work are as follows.

(1) We propose a novel method of disk I/O virtual multi-channel to separate the unified management of all running VMs in Dom0, and create an independent V-Channel mapping in I/O devices for each DomU. Reasonable and fair allocations of bandwidth are applied to each V-Channel with high throughout.

(2) We improve the CFQ [3] algorithm, turning it into an algorithm of fair scheduling among VMs and using it to guarantee delay requirements for each request.

(3) Our experimental results demonstrate that VMCD can increase the fairness and stability of resource allocation and bandwidth utilization.

The rest of this paper is organized as follows. Section 2 describes the VMCD architecture based on Xen virtualized disk I/O structure. Section 3 illustrates the design and detailed implementation of VMCD components, and describes the corresponding algorithms. Evaluations are given in Section 4, and related works are discussed in Section 5. Finally, in Section 6, we summarize our results and provide a number of conclusions.

2 VMCD Architecture

We introduce the VMCD architecture model after taking a brief look at Xen virtualized disk I/O structure in this section.

2.1 Xen Virtual Disk I/O Structure

Xen [4] has been widely used in both academia and industry. One of the most striking features of Xen is para-virtualization, which delivers higher performance than full virtualization.

Xen VMs meet the demand of VM user for block devices by providing an interface of virtual block device. In block device drivers, the data transfer process is shown in Fig. 1. The disk access of a VM goes through the following components.
(1) *File system*. The operation of accessing the virtual disk from a DomU application needs to be encapsulated into an I/O request to virtual disk when passing through the file system layer.
(2) *I/O scheduling in DomU*. According to the real-time priority or the principle of minimizing disk seek time, an I/O scheduling link selects appropriate scheduling algorithm for I/O requests and passes the request to lower layer drivers.
(3) *FE (front-end) driver*. After receiving an I/O request, the front-end driver cannot control the hardware to schedule them directly. Instead the request must be sent to Dom0 by I/O ring, event channel, and shared memory.
(4) *BE (back-end) driver*. The back-end driver in Dom0 receives an event notification from a DomU, retrieves I/O requests from the I/O ring and the shared memory, assembles them into a complete I/O request, and submits them to the Blktap.
(5) *Blktap*. Blktap is an application that runs in Dom0 (in user space), providing an interface of the virtual disk file to the user layer. Virtual disk can be mapped to a partition (or a file) in the physical disk. The role of Blktap is to convert the 64 sector numbers of the virtual disk in I/O requests of DomUs to the offset of the virtual disk file in the physical disk.
(6) *I/O scheduling in Dom0*. Disk accesses coming from all DomUs forms real disk I/O requests, and are pushed into a request queue. After that, Dom0 uses an appropriate disk

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

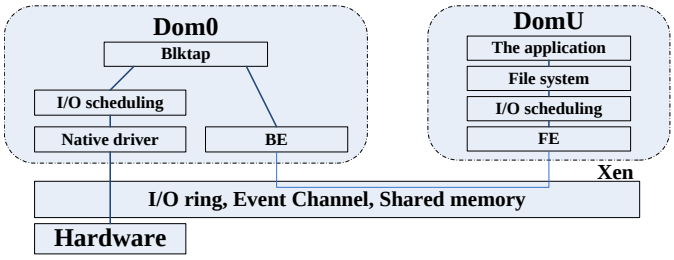


Fig. 1. Data transfer process of block device driver.

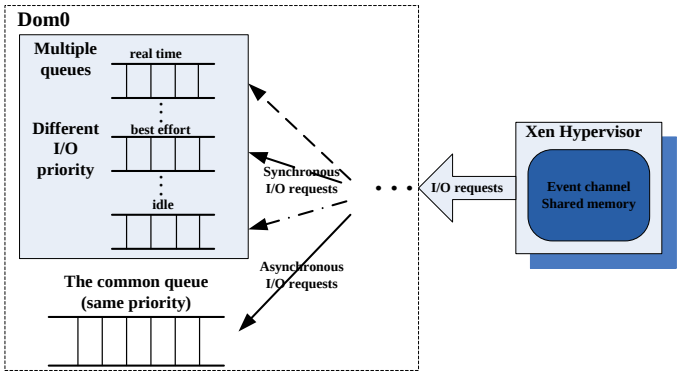


Fig. 2. The flowchart of CFQ in Xen.

scheduling algorithm for the I/O requests in the queue.
(7) *Native driver*. Finally, the native driver serves I/O requests, and returns the results.

2.2 VMCD Architecture

CFQ in Linux kernel provides a queue for each process, and fairly schedules I/O requests by polling these queues. The flowchart of CFQ in Xen is shown in Fig. 2. It divides requests into two categories (synchronous and asynchronous). Asynchronous requests from all processes with the same priority are put into one queue, while some internal queues (with different I/O priority) for synchronous request are maintained. The original CFQ in Xen cannot prevent a DomU being interfered from other DomUs.

VMCD provides a virtual link for each VM in Xen, and maps the V-Channel to the physical disk. With V-Channel, each DomU is connected directly to its own virtual block I/O device. This design is to improve the isolation of disk I/O streams of VMs by considering the following two aspects: (1) The disk I/O bandwidth allocation is unfair in the original Xen. (2) The I/O access pattern of each DomU may be changed due to the competition for a single channel among multiple DomUs.

The system architecture of VMCD is demonstrated in Fig. 3. Each V-Channel is isolated, meaning that each DomU believes it is in exclusive use of the virtual block I/O devices. Components of a V-Channel for each DomU are as follows: Application, File system, Block device layer, and Scheduling algorithm in DomU, FE driver, I/O ring, BE driver, Virtual disk control block, Native driver, and Physical I/O device in Dom0.

In order to improve the isolation of disk I/O streams of VMs, Dom0 in VMCD maintains a request queue for each DomU, and I/O requests are pushed into the corresponding queue according to the domain identity (Dom ID). To fairly allocate bandwidth for each V-Channel and fairly scheduling requests in every pending queue, Virtual Disk Control Block is designed, including the credit allocation mechanism, the global monitor module, and the disk I/O scheduling. The three components are introduced in detail in Section 3.

3 VMCD Specific Implementation

In this section, we mainly introduce the specific implementation of VMCD including the credit allocation mechanism, the global monitoring strategy, and the virtual multi-channel fair scheduling algorithm. The notations and their definitions hereafter are summarized in Table 1.

TABLE 1
Definitions of Symbols

Symbol	Meaning
B_T	Total actual bandwidth
B	Total bandwidth requirement
B_a	The actual sum bandwidth of all DomUs
B_i	The bandwidth requirement of $DomU_i$
B_i'	Actual allocated bandwidth of $DomU_i$
R_i	Number of requests of the ith pending queue
C	Amount of credits
C_i	Number of credits of the ith pending queue
T_o	Original deadline of I/O request
T_ε	The deadline after I/O request entering pending queue
T_c	Transmission time of request from creation to be pushed into pending queue in Dom0
T	Period between two consecutive replenishment event
T_f	Final deadline of request
T_ω	The waiting time of request in the pending queue
T_s	The time taken by one scheduling
L_t	Threshold
L_i	Number of requests of the ith pending queue
L_c	Number of allocated credits of the ith pending queue in the previous period
L_b	Number of dispatched credits of the ith pending queue in the previous period
L_i'	Number of credits of the ith pending queue

3.1 Definition of Fairness of Disk I/O Bandwidth Allocation

There exist many definitions of fairness in resource allocation. Different definitions are suitable for different scenarios, and multiple definitions can be chosen to quantify fairness [16][17][19]. If x_i is the resource allocated to user i, x_i is in range between 0 and 1 for Jain's index [16], where 0 denotes the minimum fairness, and 1 means the maximum fairness.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

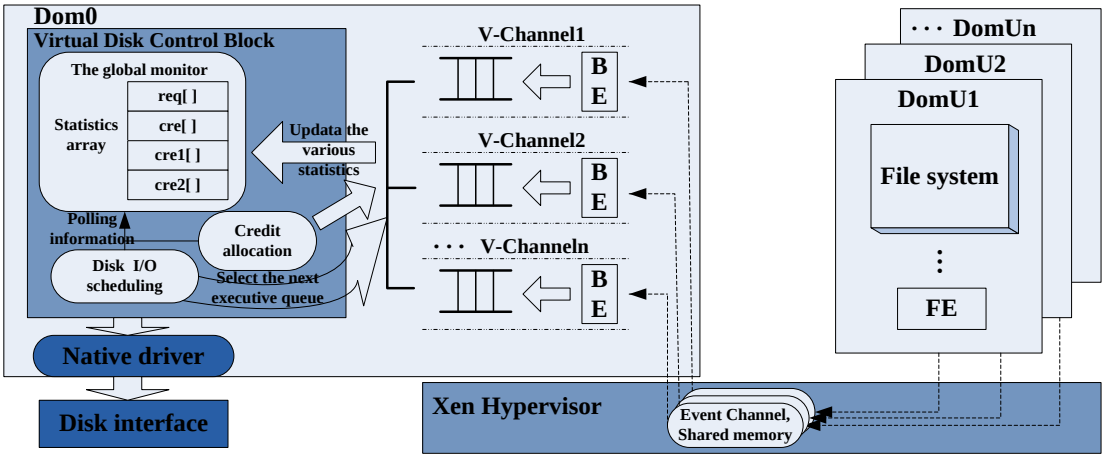


Fig. 3. VMCD architecture.

However, Jain's index often corresponds to an x where all x_i are the same, which is not appropriate for our requirement. The proportional fairness [17] corresponds to the case which owns a fixed proportion to each user, but each DomU has a variable bandwidth requirement in VM. And parameter α of α-fair utility function is viewed as another fairness measurement, in which $\alpha \to \infty$ is fairer than $\alpha = 1$, but it remains unclear what this means, e.g., is $\alpha = 3$ fairer than $\alpha = 2$ [19]? So it cannot be applied to this paper. According to the unique requirement of this paper and the definition of fairness from [28], we can quantify fairness of I/O bandwidth allocation for virtual disk. An allocation policy among multiple DomUs hosted on the same physical machine is fair only if there exists a constant θ such that for all time intervals $[t_1, t_2]$ during which any pair of VMs ($DomU_i$, $DomU_j$) meets the following equation:

$$\left| \frac{S_i(t_1,t_2)}{w_i} - \frac{S_j(t_1,t_2)}{w_j} \right| \leq \theta \tag{1}$$

where $S_i(t_1,t_2)$ and $S_j(t_1,t_2)$ represent the I/O bandwidth required by $DomU_i$ and $DomU_j$ during time interval $[t_1,t_2]$ respectively, w_i and w_j are the weights, and θ is the fairness factor. Smaller values of θ indicate better fairness. If θ is a function of the length of the time interval $[t_1,t_2]$, I/O bandwidth allocation is unfair.

Similar to the fairness evaluation model for virtualized network [33], the disk I/O bandwidth allocation fairness in this paper can be evaluated according to formula (1). Firstly, the evaluation of fairness index is divided into two classes. The first class is that all DomUs run the same application simultaneously, that is, the demands of bandwidth of all DomUs are equal. The second class is that different applications run on multiple DomUs, namely the demands of bandwidth of every DomUs are unequal. The fairness parameter of disk I/O bandwidth allocation for $DomU_i$ is indicated by λ_i, which indicates the difference between the actually acquired disk I/O bandwidth and fairly allocated disk I/O bandwidth for $DomU_i$. Larger λ_i implies worse fairness. For the first class, $\lambda_i = |B_i' - ave|$, where ave denotes the average bandwidth for each DomU. For the later, DomUs are divided into two classes based on actual bandwidth requirements. In the first class, DomUs' actual bandwidth demands are less than the average value, $\lambda_i = \frac{|B_i' - B_i|}{B_i}$. In the second class, DomUs' actual bandwidth demands are greater than or equal to the average value, $\lambda_i = \frac{|B_i' - \beta_i B_a|}{\beta_i B_a}$, where β_i denotes the expected bandwidth ratio for $DomU_i$ ($\beta_i \neq \beta_j$). The fairness evaluation index ϕ can be obtained from

$$\phi = \sum_{i=1}^{N} \lambda_i \tag{2}$$

by accumulating the fairness parameters of all DomUs. Smaller values of ϕ indicates better fairness.

3.2 Fair Bandwidth Allocation Mechanism

In order to allocate disk I/O bandwidth to DomUs fairly, and take into account throughput and latency requirement simultaneously, credit allocation strategy and global monitoring are designed as follows.

3.2.1 Credit Allocation Mechanism

VMCD maintains an independent channel for each DomU in Dom0, and the disk I/O requests of DomUs are pushed into the corresponding pending-queue according to the Dom ID, then are dispatched to the native driver to access the physical disk. Inspired by the idea of credit-based scheduling that is used to guarantee CPU resource scheduling fairness in Xen hypervisor, we extend it to support fair allocation of disk I/O bandwidth. One credit will be consumed if a disk I/O request is dispatched, the condition for a pending queue to be scheduled is that it must have available credits. Thus, the fairness of disk I/O bandwidth allocation will be affected by the credit allocation mechanism.

Our design is related to the amount of credits and the replenishment policy. Credit represents abstractly the shared disk I/O bandwidth. Fair credit allocation to each DomU's pending queue is based on the proportion of their requirement. The total actual bandwidth is denoted by B_T, and B means the bandwidth (throughput) requirement of all DomUs. R_i is the number of requests of $DomU_i$'s pending queue in Dom0.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

Therefore, the bandwidth requirement B_i of DomU$_i$ can be expressed as

$$B_i = \frac{R_i}{\sum_{j=1}^{N} R_j} \times B \tag{3}$$

where N is the number of DomUs issued disk I/O requests.

The required bandwidth B for all DomUs will be proportional to the total actual bandwidth B_T according to the credit principle, that is, $B_T = \gamma \times B$, where γ is a proportional factor. $\gamma > 1$ denotes that the actual bandwidth is enough; $\gamma < 1$ represents that the actual bandwidth is insufficient. Therefore, according to formula (3), the obtained actual bandwidth of *DomU$_i$* is

$$B_i' = \frac{R_i}{\sum_{j=1}^{N} R_j} \times B_T. \tag{4}$$

Because credit implies the abstract bandwidth, C_i is proportional to B_i or B_i'. That is to say, C_i is proportional to B_i only if $B < B_T$, and proportional to B_i' if $B \geq B_T$. So, C_i can be calculated by the following equation according to formula (3) and (4), as follows:

$$C_i = \frac{R_i}{\sum_{j=1}^{N} R_j} \times C \tag{5}$$

where $C = \alpha \times B_T$ (α is the proportional constant) if $B \geq B_T$, and $C = \beta \times B$, (β is also the proportional constant), $B < B_T$.

With the proposed credit allocation mechanism in this paper, when assigning the disk I/O bandwidth to each DomU, the requirements of other DomUs' bandwidth would be considered reasonably. Fair and reasonable bandwidth allocation to each DomU proportionally is conducted according to the number of requests in these pending queues. Using credits allocation mechanism can easily mitigate the bandwidth competition among DomUs, and can fulfill efficient global adjustment across all DomUs for satisfying the fairness of bandwidth allocation. Each request in the pending queue is dispatched to native driver, which leads to one credit being consumed. If credits of *DomU$_i$* have been exhausted, the requests in the corresponding pending queue will wait for credit reallocation.

The credit allocation strategy also includes the credit replenishment mechanism used to reallocate credit. It ensures the latency of requests and increases the bandwidth utilization. In our design, the replenishment event will be activated in the following two conditions.

Condition (1): A certain period has elapsed. The time period denoted by T is not fixed, which is different when credit is reallocated to DomUs each time. The method of dynamic adjustment is to set T as the deadline of a disk I/O request, whose current deadline is the longest among all queues at the moment of allocating credit. This design has two considerations: (a) ensuring fair distribution of the disk I/O bandwidth among DomUs according to their requirements; (b) combining with the virtual multi-channel fair scheduling algorithm to ensure the latency of disk I/O requests.

Condition (2): The credits of a DomU's pending queue are exhausted, and simultaneously there are new incoming requests and available spare bandwidth. The activation of

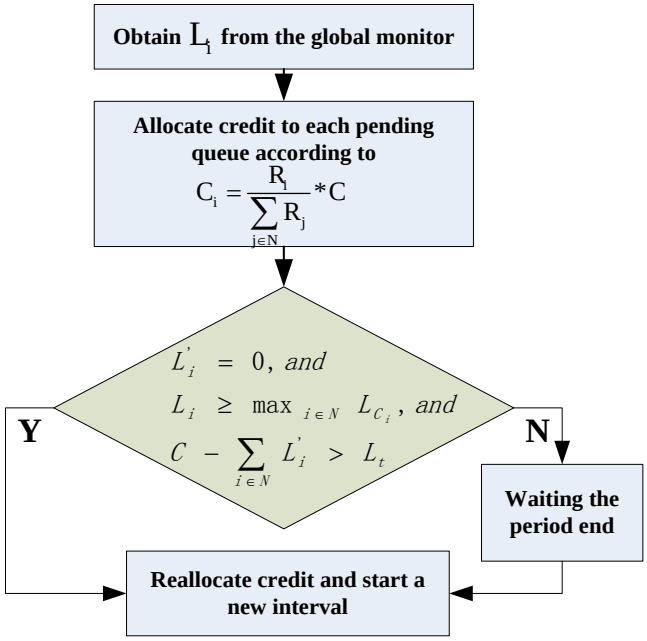


Fig. 4. Credit allocation process.

reallocation event also must satisfy the condition that the number of newly-coming requests is greater than the largest number of credits consumed by all pending queues in the previous time period. The advantages of this design are: (a) to ensure the backlogged pending requests as few as possible if the disk has spare bandwidth; (b) to avoid the occurrence of when the number of allocated credits of one DomU's pending queues in the previous interval is inadequate. And after the credits are used up, a lot of requests (that could not be completely dispatched during one period) are crowded into this queue suddenly, which can cause the decline of the whole performance. Credits consumed by one queue during the current period can be predicted according to the number of credits of a queue, which consumes the most credit in the last interval.

T varies each time upon the occurrence of condition (1). All disk I/O requests are assigned a deadline when they are generated, and sorted in *fifo_list* by it. The deadline of the same type of requests is equal. We denote the original deadline by T_o, and T_c represents the transmission time of requests from the creation to the pending queue in Dom0. After the disk I/O request being pushed into pending queue, the deadline T_ε can be expressed as

$$T_\varepsilon = T_o - T_c. \tag{6}$$

According to the description of condition (1) above, the time period T between two consecutive replenishment event can be expressed as

$$T = \max_{1 \leq i \leq N} T_{\varepsilon_i} \tag{7}$$

where N is the number of requests in all pending queues.

Based on the description above and combining with the global monitoring strategy, Fig. 4 shows the credit allocation process.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

3.2.2 Global Monitoring Strategy

The global monitoring strategy provides all information for the virtual multi-channel scheduling and credit allocation mechanism. It collects various statistics from the V-Channel of all DomUs as follows.

(1) The current sparseness status of disk I/O bandwidth.

(2) The number of requests of each pending queue, L_i.

(3) The number of allocated credits of each pending queue in the previous period, L_c.

(4) The number of dispatched credits of each pending queue in the previous period, L_b.

(5) The number of credits of each pending queue, L'_i.

(6) The deadline after request entering pending queue, T_ε.

(7) The waiting time of request in the pending queue, T_w.

All defined statistics above are used as indicators by the scheduler when dispatching disk I/O requests. According to the definition, their calculation methods are as follows. (1) Sparseness status. We set the threshold L_t to decide whether the bandwidth is fully utilized. Let $L_t = \min_{i=1}^{N} L_{C_i}$, where N is the number of pending queues. Thus, if the number of spare credits currently is more than the threshold L_t, the sparseness status is true, otherwise it is false. (2) L_i. Array *req[n]* is used to record it. The number of requests of each pending queue is written into *req[n]* correspondingly. During scheduling, when a request of $DomU_i$ is pushed into the pending queue in Dom0, *req[i]* will be incremented by one; and when a request of $DomU_i$ is dispatched from the pending queue, *req[i]* will be decremented by one. Equality L_i = *req[i]* can be obtained. (3) L_c. Another array *cre1[n]* is used to store L_c. Obviously, *cre1[i]* = $C_i = \frac{R_i}{\sum_{i=1}^{N} R_j} \times C$, where C_i is the number of allocated credits to $DomU_i$*'s* pending queue in the previous interval. When the current interval ends, the corresponding value in the *cre1[n]* is replaced by the number of allocated credits to $DomU_i$*'s* pending queue in the just-finished interval. (4) L_b. Array *cre2[n]* is used to record it. During the previous interval, when a request in $DomU_i$*'s* pending queue is dispatched, *cre2[i]* is increased by one. (5) L'_i. Array *cre[n]* is used to store it. After every credit distribution, *cre[i]* = $C_i = \frac{R_i}{\sum_{i \in N} R_j} \times C$, where C_i is the number of allocated credits to $DomU_i$*'s* pending queue in this time. During this interval, when a request in $DomU_i$*'s* pending queue is dispatched, the value of *cre[i]* is decreased by one. So, if the credits of $DomU_i$ are exhausted, *cre[i]* would be decreased to zero. (6) T_ε. It can be calculated according to formula (6). (7) T_w. It is used to compute the final deadline T_f of each request. T_f can be expressed as

$$T_f = T_\varepsilon - T_w. \tag{8}$$

The latency of disk I/O request can be ensured as long as T_f is greater than zero.

The global monitoring strategy plays the role of a server in the virtual multi-channel scheduling. It provides all the required information for the credit allocation mechanism. The virtual multi-channel scheduling can be designed based on the state of all running VMs from the global monitoring strategy.

Algorithm 1 The virtual multi-channel fair scheduling

```
1) Initialize the sum of allocated tickets: L ← 0;
2) for i ← 1 to n do
3)     Inquiry L'_i and L_i in global monitoring strategy;
4)     if(queue i is not empty and the corresponding credit is
   not empty) then
5)         l_i ← L'_i;
6)         Get the sum of l_i allocated:L ← L + l_i;
7)     end if
8)  end for
9) Get a random number k(0 ≤ k < L);
10) Clear temp sum S_i ← 0 //tickets sum from the first to the
   ith queue;
11) for i ← 1 to n do
12)     S_i ← S_i + l_i;
13)     if k < S_i then
14)         ith queue is the lucky one, break;
15)     end if
16)  end for
17) Get lucky queue number i, and dispatch 4 requests in ith
   queue;
18) Update L'_i and L_i in global monitoring strategy;
19) for i ← 1 to n do
20)     Inquiry T_f of queue head request;
21)     if T_f < T_s then //T_s is the time for one scheduling
22)         Dispatch it;
23)         Update L'_i and L_i in global monitoring strategy;
24)     end if
25)  end for
26) End
```

3.3 The Virtual Multi-channel Fair Scheduling Algorithm

VMCD keeps independent I/O queues for each DomU and requires a fair scheduling scheme among DomUs.

To better serve VMCD, we propose a scheduling algorithm that can perform fairly scheduling of disk I/O requests from pending queues. According to the requirement of VMCD, the algorithm needs to meet two requirements as follows:
(1) Fair scheduling among multiple DomUs according to their required bandwidth.
(2) Guaranteeing the latency of every request from all pending queues.

Therefore, we revise and extend the CFQ algorithm for process scheduling in Linux kernel, making it become a fair scheduling algorithm for VM. Then, we combine it with the lottery algorithm and finally add some proper mechanisms to guarantee low latency. The three statistics introduced previously, namely, L'_i, L_i, and T_f, provide the basis for the algorithm. Once these statistics are obtained, we can allocate lottery to each queue according to L'_i, and the latency of request is ensured based on T_f. Algorithm 1 shows the procedure of virtual multi-channel fair scheduling.

Let n denote the number of running DomUs (i.e., the number of pending queues), and l_i denote the number of tickets allocated to pending queue i. According to the time instance when the last request is dispatched in the previous interval and the first request in this interval, T_s can be calculated. The algorithm's main steps are as follows. Firstly, if there are requests in the pending queue and spare credits, the algorithm allocates lottery to each running pending queue according to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

the number of credits in the queue currently. Because credit is the passport of the request, allocating lottery to the pending queue based on the number of credits instead of requests is the best choice. Secondly, generating randomly a number is equivalent to extracting the lottery from the allocated. After obtaining the sum of allocated tickets of all pending queues, we generate a random number to obtain a lottery. If the extracted lottery is located in a certain queue, the first four requests of this queue will be dispatched. The strategy of CFQ that dispatches four requests each time is also reserved, and this can improve the performance of disk I/O. Finally, at the end of completing the request dispatched, the algorithm can ensure the latency of pending requests by polling the deadline of the first request in the pending queue, and if the deadline is less than the time of the first scheduling, the request will be dispatched. The algorithm can guarantee the fairness of disk I/O bandwidth allocation by fairly distributing lotteries.

4 Experimental Evaluation of VMCD

In this section, we run a suite of experiments to evaluate our VMCD on the Xen-hosted platform using a series of data access patterns, which demonstrates that VMCD has several advantages on resource allocation compared to other approaches.

4.1 Experimental Setup

Our experiments were conducted on two physical computers, and each had an Intel Pentium G630 CPU running at 2.70GHz, 6GB of RAM. We used a Serial ATA disk drive (7200 RPM, 1TB) and SSD for all tests. As needed, one of them was configured as the server, which ran a modified Xen hypervisor on Linux domains. The server was configured with one or more DomU(s), and one Dom0. Dom0 and all DomUs were configured with 512MB memory and a single virtual CPU. The kernel version 3.8.0 and Xen 4.1 was used. The other machine was used as the monitor and ran Windows 7 operating system with the test tool–Iometer [13]. We divided the entire experiments into two parts. The first part was tested in original Xen with CFQ, Deadline, and Anticipatory scheduler, and the second part was done in the VMCD. The various disk I/O performances of VMCD are compared with those of original Xen with the same software configuration and the Linux kernel.

4.2 Fairness of Bandwidth Allocation

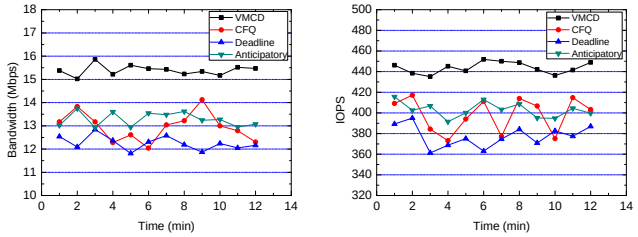
In this subsection we evaluate the fairness of bandwidth allocation of VMCD.

4.2.1 *Identical Workloads Running within the VMs*

Firstly, we measure the fairness of bandwidth allocation by transmitting the same requests to every DomU. Under CFQ, Deadline, Anticipatory, and VMCD, all DomUs are assigned the same weight and bandwidth. Our system is evaluated in the case where variable DomUs issue simultaneously disk I/O requests with different request block size and different I/O read-write type. The used access patterns (request block size and I/O read-write patterns) in this test are 512B-Sequential-Read and 128K-Random-Write. Three groups of experiments corresponding separately to the number of DomUs from 2 to 4 are conducted, and each group is repeated 10 times. Fig. 5 shows the fairness index of disk I/O bandwidth allocation, which is obtained with formula (2) using CFQ, Deadline, Anticipatory and VMCD respectively.

Fig. 5(a) shows the fairness of bandwidth allocation when the access pattern is 512B-Sequential-Read, and Fig. 5(b) shows the fairness of another access pattern of 128K-Random-Write. We can observe that VMCD improves fairness by 70% compared to CFQ and Anticipatory, by 26% compared to Deadline. The reason is that VMCD reduces the interference of each DomU by V-Channel, and can reasonably allocates credits to pending queues by adopting the fair scheduling algorithm. Meanwhile, Fig. 5 also shows that the fairness of Deadline is better than CFQ and Anticipatory. The explanation is that deadline adds timeout mechanism to the traditional NOOP scheduling, which is embodied in two aspects: (1) selecting the request timeout; (2) selecting the first request after scanning the last one in "elevator" if there are not any request timeout. Based on these, Deadline obviously enhances the fairness of bandwidth allocation. The similar results of both Fig. 5(a) and Fig. 5(b) indicate that VMCD is adaptive to each type of request block size and read-write pattern. But we can see that the fairness decreases gradually with the increasing number of virtual DomUs. The potential reason is that the resource competition among channels is more intensive with the increasing DomUs, which makes it more difficult to guarantee fair bandwidth allocation among DomUs.

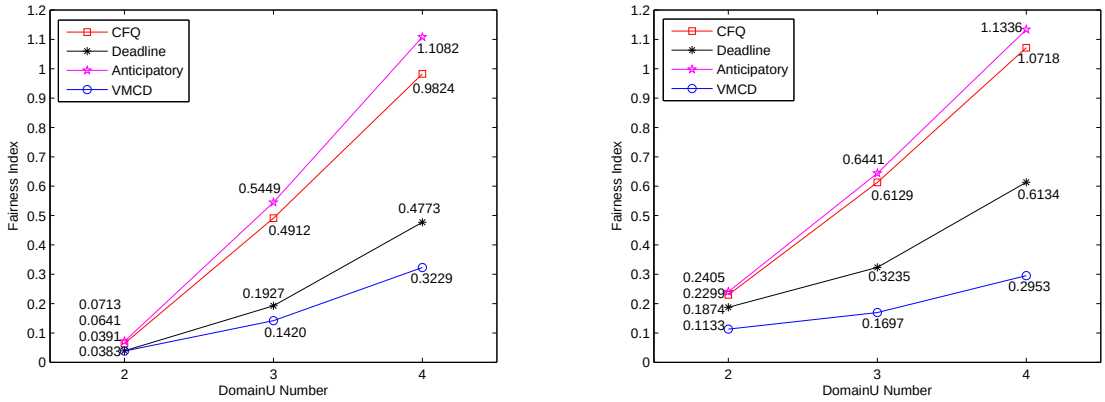
The stability of bandwidth allocation is also tested by two experiments under the circumstances that three DomUs are concurrently running with the same workload. The access patterns used in the two experiments are 512B-Sequential-Read and 64K-Random-Write. The aggregated throughput (bandwidth) and IOPS (I/O Operations per second) of three DomUs can be obtained respectively. Fig. 6 shows the stability of disk I/O bandwidth allocation in CFQ, Deadline, Anticipatory, and VMCD. As shown in Fig. 6(a), all of three DomUs obtain relatively smooth bandwidth approximate to 15.5 Mbps with VMCD, while the results from the other three schemes fluctuate between 12Mbps and 14Mbps. Similarly, Fig. 6(b) shows that VMCD approaches 440 for IOPS while others vibrate between 360 and 420.



(a) 512B, 100% read, 100% sequential. (b) 64K, 100% read, 100% sequential.

Fig. 6. Stability of bandwidth allocation for multiple DomUs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing



(a) 512B, 100% read, 100% sequential.

(b) 128K, 100% write, 100% random.

Fig. 5. Fairness for changing requests specification and the number of DomUs.

From the figure, VMCD shows better stability (the amplitude as small as possible), and the main reason is that it can mitigate the interference among multiple DomUs by using separated V-Channel and an I/O request queue for each DomU. And the Anticipatory scheme shows better stability than the other two schedulers. Although Anticipatory is close to VMCD in stability, its performance is lower than that of VMCD. VMCD improves stability significantly in the case of multiple DomUs running on the same physical host.

4.2.2 *Different Workloads Running within the VMs*

In this part we evaluate the fairness of bandwidth allocation by transmitting different requests to each DomU. The tests in this section are divided into two cases. The first case is that the I/O sizes are variable and each DomU has the same weight. The second case is that the I/O patterns are different. Similar to the above experiment, our VMCD is also compared to CFQ, Deadline and Anticipatory. For the first case, the number of concurrent DomUs with different mixed I/O workloads are from 2 to 4. Access patterns of I/O workload including I/O sizes, read/write and sequential/random are shown in Table 2. For the second case, three concurrent DomUs with weight ratio of 1:10:30 are deployed. Various combinations of I/O patterns are shown in Table 3.

TABLE 2
The description of different I/O sizes workloads running within VMs.

VM	Workload	Description
VM1	512B-SR	Sequential read, I/O sizes is 512B
VM2	64K-SR	Sequential read, I/O sizes is 64K
	128K-SR	Sequential read, I/O sizes is 128K
VM3	512K-SR	Sequential read, I/O sizes is 512K
	256K-SR	Sequential read, I/O sizes is 256K
VM4	1M-SR	Sequential read, I/O sizes is 1M

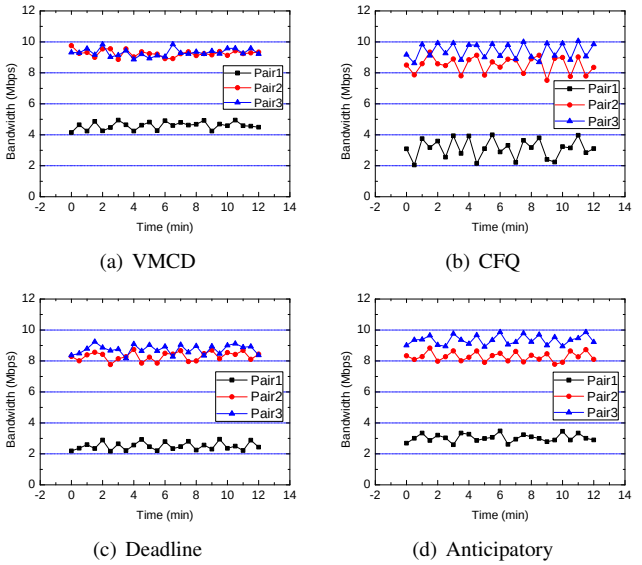
Firstly, a group of experiments is run in three DomUs using I/O size of 512B, 64K, and 512K respectively with the same I/O type. Fig. 7 shows the bandwidth allocations of VMCD, CFQ, Deadline, and Anticipatory. Three DomUs respectively acquired 4.64Mbps, 9.29Mbps, and 9.25Mbps bandwidth allocation (Corresponding to Pair1, Pair2 and Pair3 in Fig. 7(a)) in VMCD, 3.08Mbps, 8.63Mbps, and 9.18Mbps (corresponding to Pair1, Pair2, and Pair3 in Fig. 7(b)) in CFQ, 2.57Mbps, 8.60Mbps, and 8.38Mbps (corresponding to Pair1, Pair2, and Pair3 in Fig. 7(c)) in Deadline, and 3.01Mbps, 8.28Mbps, and 9.36Mbps (corresponding to Pair1, Pair2, and Pair3 in Fig. 7(d)) in Anticipatory. Fairness index of each scheme were obtained using formula (2). The experimental results are shown in Fig. 8. It demonstrates that VMCD improves fairness significantly. VMCD improves fairness by 67% compared to CFQ and Anticipatory, by 58% compared to Deadline in the case of three or more VMs running on the same physical host.

TABLE 3
The description of different I/O types workloads running within VMs.

VM	Workload	Description
VM1	I/O intensive	10IOps, Random read, 64K
VM2	I/O moderate	100IOps, Random read, 64K
VM3	I/O scarce	300IOps, Random read, 64K

Secondly, for different I/O patterns, we compare the performance of three DomUs under VMCD with that under CFQ. We test VMCD with the various combinations of I/O patterns (I/O intensive, I/O moderate, and I/O scarce) using the same experimental setup. Fig. 9 shows that the three DomUs respectively acquired 3.04Mbps, 8.48Mbps, and 16.83Mbps bandwidth allocation in VMCD, and 8.63Mbps, 8.83Mbps, and 9.04Mbps in CFQ. It can be seen that CFQ obtains approximately equal bandwidth among three DomUs without considering their actual demands. The allocation of bandwidth for CFQ is not reasonable, and its fairness index is 13.687. While VMCD reasonably allocates the disk bandwidth according to the demands of each DomUs (fairness index is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing



(a) VMCD (b) CFQ

(c) Deadline (d) Anticipatory

Fig. 7. Disk bandwidth allocations for changing I/O size to different DomUs.

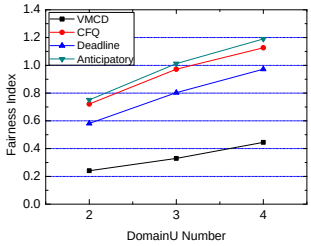


Fig. 8. Fairness for changing I/O size to different DomUs.

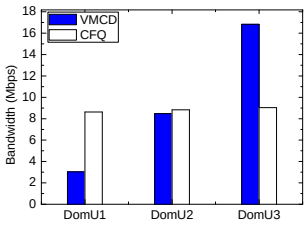


Fig. 9. Disk bandwidth allocations for changing I/O type to different DomUs.

4.404). VMCD promotes the fairness by 67.8% in comparison with CFQ in the case of various combinations of I/O patterns.

The evaluations above show that VMCD significantly improves fairness. The main reason is that credit allocation of VMCD is based on the number of credit consumed during the last time period and the newly-coming requests. Credit allocation according to bandwidth demand of each DomU makes it possible that all DomUs (with different demands to bandwidth) can obtain fair treatment.

4.3 Disk I/O Performance for Multiple DomUs

We perform several experiments to measure the disk I/O performance of VMCD, and compare the performance of multiple DomUs under VMCD with that under CFQ, Deadline, and Anticipatory.

4.3.1 Bandwidth Utilization

In this part, we evaluate the aggregated bandwidth utilization with CFQ, Deadline, Anticipatory, and VMCD respectively. Multiple DomUs' bandwidths are equally allocated, and we obtain the sum of bandwidth of all running DomUs.

Our system is evaluated using the same method as the first set of experiments, except for the difference that there are four groups of experiments with different number of DomUs ranging from 1 to 4. The workloads run within DomUs are 512B-Random-Write and 128K-Sequential-Read. Table 4 shows disk I/O bandwidth utilization in four scheduling approaches (in Mbps).

According to Table 4, the aggregated disk bandwidth utilization of VMCD is 11-22% higher than that of other schedulers used in the original Xen when there are more than one concurrently running DomUs. For the test with a single DomU, VMCD shows slightly lower bandwidth utilization than CFQ and Anticipatory in Table 4 (b), which we believe is caused by VMCD's overhead. For multiple DomUs, due to the fact that the amount of requests is increasing with the increase of concurrent DomUs, the scheduler will try to allocate disk I/O bandwidth to all DomUs if the requiring bandwidth is lower than the maximum. The aggregate bandwidth will raise with the increase of DomUs, which is shown in Table 4. But the interference from other DomUs lowers the actual obtained bandwidth. VMCD mitigates the interference between multiple DomUs by the separated V-Channel for each DomU, thus improving performance remarkably as compared with CFQ, Deadline, Anticipatory.

TABLE 4
Aggregated bandwidth with different numbers of concurrent DomUs (unit: Mbps)

Concurrent DomUs	1	2	3	4
VMCD	5.470	12.733	14.327	14.975
CFQ	5.645	10.993	11.907	12.262
Deadline	4.832	10.917	11.195	12.085
Anticipatory	5.272	10.960	11.748	12.136

(a)512B, 100% write, 100% random

Concurrent DomUs	1	2	3	4
VMCD	11.690	17.583	28.870	29.174
CFQ	11.848	15.255	25.824	25.406
Deadline	10.121	14.624	25.362	25.117
Anticipatory	12.084	15.982	26.241	26.279

(b)128K, 100% read, 100% sequential

Secondly, our test is also performed under the three concurrent DomUs, but the workloads of each DomU are different. The I/O sizes of the three DomUs are 512B, 4K, and 128K respectively. Fig. 10 shows each DomU's bandwidth allocated. We can see that the bandwidth utilization of each DomU is improved and it is remarkable with request block size of 512B (The most frequent I/O operations). The results of workloads with different I/O types running within three DomUs are shown in Fig. 9. VMCD (28.35Mbps) shows higher bandwidth utilization than CFQ (26.5Mbps). The main reason is that I/O resources cannot be fully utilized because of competition between DomUs compared to original Xen with CFQ, Deadline, and Anticipatory schedulers, while our architecture avoids competition among DomUs. At the same time, the replenishment event will be activated after meeting condition (2) (proposed in Section 3.2.1), which promotes the bandwidth utilization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

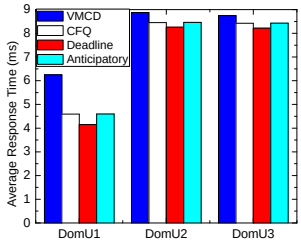


Fig. 10. Bandwidth of each DomU with different request block sizes for three DomUs (unit: Mbps).

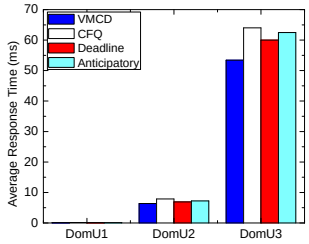


Fig. 11. ART of each DomU with different latency requirement.

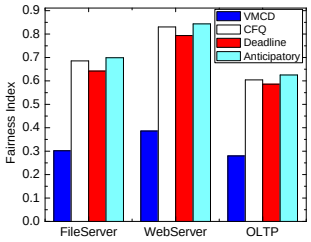


Fig. 13. Fairness for multiple DomUs with I/O intensive applications.

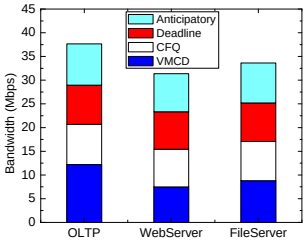


Fig. 14. Bandwidth allocation of each DomU as the different platform.

Whether the workloads are identical or different to multiple DomUs, the evaluation results above indicate that VMCD substantially improves the aggregated disk bandwidth utilization than that of CFQ, Deadline, and Anticipatory, especially when the I/O requests are frequent (The request block sizes are small). We can conclude that VMCD can effectively lower the interference among multiple DomUs by introducing V-Channel and disk I/O request queue for each Domain, and improve scheduling for I/O requests of small sizes by using multi-queue fair scheduling that maps lottery tickets to credits and contacts lottery tickets with each DomU's queue length.

4.3.2 Performance for ART

ART is the average response time of all requests of multiple DomUs. An experiment is designed in this section to evaluate the latency. Three concurrent DomUs run different workloads with diverse latency requirement. Fig. 11 shows the ART of each DomU.

From the figure, we can see that VMCD effectively reduces ART by up to 21.8%, 13%, and 14% compared with CFQ, Deadline, and Anticipatory, thus VMCD can achieve better latency than other three schedulers. The improvement of latency requirement under VMCD is due to the independent V-Channel for each DomUs that avoids the competition. The period in credit replenishment mechanism is dynamically adjusted refer to latency, and the expiring request will be dispatched in the virtual multi-channel fair scheduling algorithm. Contrasting with CFQ and Anticipatory, Deadline considers the latency when dispatching request. But Deadline also cannot ensure the latency because of multiple DomUs competing the shared bandwidth resources.

4.4 Realistic Workloads

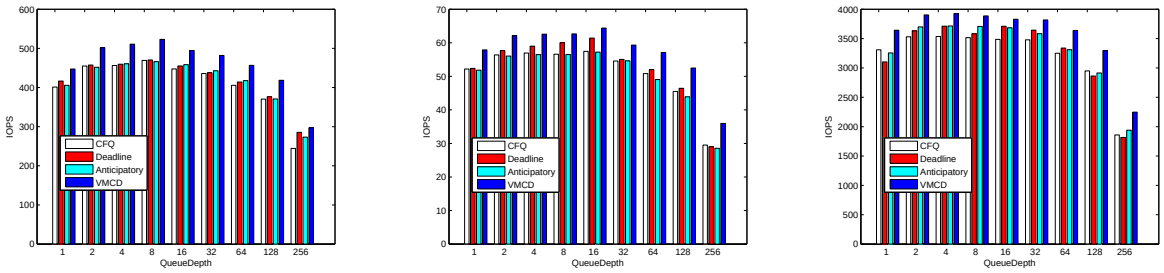
In this subsection, we test VMCD with some realistic I/O workloads (File server, Web server, and OLTP (On-Line Transaction Processing) [9]). To adequately estimate the performance achieved by VMCD against these application servers, we design two kinds of experiments.

Firstly, for each realistic workload, three Linux VMs, DomU1, DomU2, and DomU3 are deployed, and all of them are used as the same server with client applications running in another computer remotely. These I/O intensive applications are simulated by random requests, and the total size of each DomU's I/O Read/Write requests is 2GB. We change the queue depth of each DomU from 1 to 256 and collect IOPS for each experiment. The results of the three types of I/O intensive applications are shown in Fig. 12. We can observe that IOPS of multiple DomUs using VMCD is higher than the other three schedulers in all cases. VMCD shows the improvement of 11.5%, 11.2%, and 12.2% compared with CFQ, Deadline, and Anticipatory respectively for file server, 12.1%, 4.9%, and 12.5% for web server, and 11%, 5.7%, and 5.6% for OLTP. The reason is that VMCD is adaptive to each type of request block size and the read-write pattern, and these testing applications exhibit variable access patterns.

The fairness index can be calculated by allocated bandwidth of each DomU. Fig. 13 presents the results of fairness for three types of workloads. We can see that VMCD improves fairness significantly compared to CFQ, Deadline, and Anticipatory. It achieves 56%, 53%, 56.8% improvement for fairness in the case of three virtual DomUs as file server, 53.5%, 51%, 54% for web server, and 54%, 52%, 55% for OLTP. In general, the results show that VMCD improves the fairness when multiple DomUs are deployed to simulate real servers.

Secondly, three DomUs are deployed as specific real servers, which run the file server, web server and OLTP workloads respectively. And we record the allocated bandwidth under four different schedulers. Fig. 14 shows the bandwidth allocation of three DomUs as three types of different platform. The three DomUs are allocated 12.21Mbps, 7.49Mbps, and 8.8Mbps bandwidth in VMCD (the aggregated bandwidth is 28.5Mbps, similarly hereinafter), 8.48Mbps, 7.94Mbps, and 8.26Mbps in CFQ (24.68Mbps), 8.26Mbps, 7.91Mbps, and 8.14Mbps in Deadline (24.31Mbps), 8.7Mbps, 8.03Mbps, and 8.44Mbps in Anticipatory (25.17Mbps). The I/O request size of the workload OLTP usually is smaller than that of the file server and web server, and the I/O request size of the web server is the largest. So the number of OLTP requests are the most, and this leads to the maximality of disk I/O bandwidth requirement. The three schedulers of the original Xen ignore the ideal requirement of each DomUs. The reason for this is the limitations of themselves for multiple DomUs competing the shared resources. On the contrary, three DomUs obtained well-proportioned bandwidth under VMCD. These demonstrate that VMCD improves the fairness when the demands of bandwidth is unequal under realistic workloads. VMCD improves bandwidth utilization by 15% on average against

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

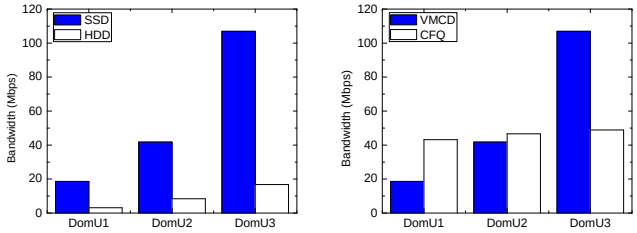


(a) IOPS of multiple DomUs running file server-like workload with different queue depth.

(b) IOPS of multiple DomUs running Web server-like workload with different queue depth.

(c) IOPS of multiple DomUs running OLTP-like workload with different queue depth.

Fig. 12. IOPS of multiple DomUs running I/O intensive applications.



(a) (b)

Fig. 15. Bandwidth of three DomUs (different bandwidth demands) on SSD.

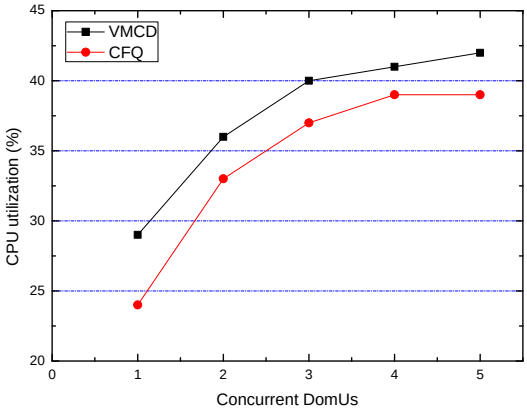


Fig. 16. The change of CPU utilization with different number of concurrent DomUs.

other three schedulers. The reason is that VMCD allocates bandwidth based on DomU' requirement and mitigates the interference among DomUs by V-Channel.

4.5 Portability and CPU Overhead

Since VMCD is a pure software solution, it does not rely on any special disk hardware support. To explain why VMCD does not hinder live migration and portability, we implement VMCD on SSD-based storage system. The used SSD is Intel 320 Series 40GB, and three DomUs that have different bandwidth requirements (64K, Random-Write, IOps are 10, 100, and 300) are deployed concurrently. Fig. 15(a) shows the obtained bandwidth of three DomUs on different storage devices (HDD and SSD) with VMCD. Three DomUs respectively acquired 18.58Mbps, 41.74Mbps, and 106.9Mbps with SSD, and 3.16Mbps, 8.41Mbps, and 16.75Mbps for HDD. The fairness index according to formula (2) is 4.2930 for SSD and 4.5932 for HDD. It can be seen that VMCD also exhibits good fairness on SSD. At the same time, the overall performance is much better than that of HDD (The intuitive reason is that SSD's performance is better than that of HDD). And in the SSD storage system, we also compare VMCD with original CFQ scheduling in Xen. Fig. 15(b) shows VMCD also performs better than the original Xen on SSD storage system. In addition to better fairness of disk bandwidth allocation (The fairness index is 4.293 for VMCD and 11.592 for CFQ), VMCD improves bandwidth utilization obviously, which achieves 20.5% on SSD (The aggregated bandwidth of three DomUs are 167.22Mbps for VMCD and 138.7Mbps for CFQ). The reason is that VMCD is completely based on pure software, whose virtual multi-channel method and fair multi-queue scheduler can be fully ported to higher performance storage devices.

To ensure the fairness of disk I/O bandwidth allocation, VMCD exploits the credit based disk I/O scheduling. The evaluation experiments show that VMCD can obtain good performance. However, the credit based I/O scheduling also causes extra CPU overhead, because the credit allocation mechanism needs to acquire some information such as the number of credits allocated to each pending queue through CPU computing, replenishment mechanism requires CPU to judge whether its event happens continually, and some additional data structures also need CPU to process. To evaluate the overhead of VMCD, we monitor CPU utilization of Dom0 in sequential read experiment by varying the number of DomUs from 1 to 5. As shown in Fig. 16, we compare the change of CPU utilization under VMCD and under CFQ. Because CFQ is simpler than VMCD in disk I/O scheduling, VMCD raises CPU utilization by 3% compared to CFQ when using the same number of DomUs, but brings better fairness and stability of disk I/O bandwidth allocation and higher aggregated disk I/O bandwidth utilization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

5 Related Work

Our research aims at fair disk I/O bandwidth allocation on the premise of improving performance in VMs. This section presents the related work.

At present, popular approaches to improve the overall performance of disk I/O shared among VMs can be classified into two categories: hardware-based approaches and software-based approaches. In hardware-based approaches, due to the high performance, SSD has been used in virtualized environment [21], [22], [29], [38], [39] to improve disk I/O performance of VMs. But SSD needs to erase the block before writing the new page [12]. *Write Amplification* (WA) and *Garbage Collection* (GC) are proposed in [5], [12] to reduce write operations. However, promoting disk I/O performance of VMs by using new types of disk such as SSD and PCM (Phase Change Memory) requires special hardware support, which may incur extra costs and hinder wide adoption.

This paper focuses on software-based methods. These approaches are shown in Table 5. Compared with hardware-based approaches, software-based I/O virtualization approaches can improve the performance at lower cost. Currently, the software-based schemes are used in many popular VM environments such as VMware , Xen, and KVM. Since the scheduler in VMs has significant impact on I/O performance [25], we classify the software-based approaches into two categories.

(1) Scheduling frameworks. The disk I/O multi-queue scheduling method for I/O performance virtualization is used in most frameworks. SARC/AVATAR [40], *Flubber* [20], and APA [15] have proposed a framework that has two-level hierarchy of schedulers. In SARC/AVATAR, the high level called SARC is used to meet the throughput of multiple DomUs, and the low level named AVATAR ensures the latency requirements of requests from multiple DomUs. The approach can meet the throughput and latency requirements well. There are two weaknesses in the framework. One is that I/O requests need to be classified before entering the respective queue, because the framework concerns about multiple types of application I/O requests. This complicated mechanism reduces the performance of I/O scheduling and request processing. By contrast, our VMCD is simple, in which I/O requests are pushed into corresponding queues according to their Dom ID. The other is that there is not a mechanism to guarantee the fairness when the requests from SARC enter into AVATAR. On the basis of SARC/AVATAR, *Flubber* [20] focuses on optimizing the disk I/O utilization. Admission-control-based Proportional Allocation (APA) [15] based on YFQ [7] for shared virtualization storage system was proposed. But APA ignores the latency time requirement of applications, and can only be used in prototype Performance Virtual Storage System (PVSS). Taking into consideration two factors overlooked, we design credit replenishment mechanism and poll the final deadline of requests in scheduling algorithm to ensure the latency. Unlike the two-level framework, VM-PSQ [18] was proposed to solve the problem of allocating I/O bandwidth to several VMs equally or discriminatingly in server virtualization environments. Each VM has a queue in the architecture of VM-PSQ, and requests are dispatched according to two factors (Time Slice and Schedule Token). But VM-PSQ ignores the latency of each request and the bandwidth utilization of the virtualized system. The ideas of multi-queue and performance insulation [6] [37] are also proposed, but their main research purpose targets storage systems instead of VMs. All of them face the same problem that requests from independent queues will converge on a pending queue at the end, and lack of a fair scheduling. Our VMCD alleviates the defect by selecting a queue fairly based on the virtual multi-channel fair scheduling algorithm.

(2) Scheduling algorithms. The main goal of previous scheduling algorithms is to meet the throughput and latency requirements of different users and/or classes. These scheduling algorithms can be classified into two types: proportional share scheduling and real-time scheduling.

The proportional share scheduling algorithms. Complete Fair Queuing (CFQ) [3], SFQ [10], FSFQ [29], pClock [1], and CVC [22] belong to this type. The generalized processor sharing principle was utilized in these algorithms to meet the throughput and latency requirements. However, to achieve high bandwidth utilization, they overbook the resources by capacity profiling in every VMs, which potentially leads to resource over-provisioning and causes others hungry. There exist some scheduling algorithms to achieve fairness in the other field, such as the QADP algorithm proposed in [30], which achieves fairness as well as throughput optimization in multi-hop wireless networks. QADP first assigns weight to each flow, then maps the admission price to each flow dynamically, and schedules requests among multi-flows finally. The scheduling period of this algorithm is fixed, while the credit based scheduling solution in VMCD is adjusted dynamically according to the request number of each queue. VMCD distributes and adjusts disk I/O bandwidth reasonably to each DomU based on credit and corresponding replenishment mechanism.

The real-time scheduling algorithms. There are mainly Earliest Deadline First (EDF), SCAN Earliest Deadline First (SCAN-EDF) [23], SCAN [8] [26], Anticipatory scheduling [31], etc. These algorithms rely on the service time of I/O requests to avoid deadline violation. The service time is usually computed by accurate system performance model [15]. But, because of the non-preemptive nature of the requested service [34], deadline may be missed if VMs issue synchronous requests. In our scheduling algorithm, after VMCD selects a pending queue and dispatches disk I/O requests, deadline of requests will be polled. A request will be dispatched immediately if its deadline is smaller than a specific value. VMCD does not induce deadline being missed.

6 Conclusions

This paper presents the design, implementation and evaluation of VMCD, a system that can fairly allocate disk I/O bandwidth in virtualized environment. VMCD can mitigate the interference between multiple DomUs by introducing separated V-Channel and I/O queue for each DomU. In order to better implement it, we design a credit allocation mechanism including credit replenishment mechanism for disk I/O, and the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

TABLE 5
Existing Scheduling Methods in VMs

Types	Class	Introductions
Scheduling frameworks	SARC/AVATAR	Two-level scheduling framework, which can guarantee throughput and latency simultaneously.
	Flubber	Two-level scheduling framework, focus on the disk I/O utilization.
	APA	Two-level scheduling framework, ignore the latency requirement.
	VM-PSQ	Good at performance isolation, but ignore the latency requirement and the bandwidth utilization.
Scheduling algorithms	Proportional share schedule	Use the generalized processor sharing principle to meet the throughput and latency requirements.
	Real-time schedule	Avoid the deadline violation of requests rely on the request service times.

virtual multi-channel fair scheduling algorithm. Experimental results show that VMCD substantially increases the fairness by 70% compared to CFQ and Anticipatory in the case of three or more virtual DomUs running on the same physical host, and has better stability of bandwidth allocation. VMCD improves the aggregate disk bandwidth utilization by 28% and 15% compared to CFQ in the case of small and big request block size respectively, and 37% and 18% to Deadline for small and big request block size. For IOPS and ART, VMCD has a corresponding improvement. When multiple DomUs running I/O intensive server applications, we have witnessed performance improvement of the whole system and the fairness of disk bandwidth allocation. We believe that by introducing separated V-Channel and I/O request queue for each Domain, VMCD can lower the interference between multiple DomUs and guarantee the latency of each request using virtual multi-channel fair scheduling. Our VMCD approach is completely based on software and can be easily ported to other virtualization systems.

7 Acknowledgments

The authors are grateful to the anonymous reviewers for their helpful feedback. This research was supported in part by the National Natural Science Foundation of China under grants 61272190. Dr. Kai Hwang would like to thank the support by China's 973 Basic Research Grant 2011CB302505, awarded through Tsinghua University, Beijing. He also want to acknowledge the partial support of this work by the Guangdong Innovation Research Team funding under the grant No. 201001D0104726115.

References

[1] G. Ajay, M. Arif, and V.J. Peter. pClock: an arrival curve based approach for QoS guarantees in shared storage systems. *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 13–24, 2007.

[2] S. Arunagiri, Y. Kwok, P.J. Teller, R. Portillo, and S.R. Seelam. Fairio: an algorithm for differentiated i/o performance. In *Proceedings of the 23rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2011)*, 2011, pp. 88–95.

[3] J. Axboe, CFQ I/O scheduler, http://mirror.linux.org.au/pub/linux.conf.au-2007/video/talks/123.pdf, 2008.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, and A. Ho. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on operating systems principles*, 2003, pp. 164–77.

[5] W. Bux, and I. Iliadis. Performance of greedy garbage collection in flash-based solid-state drives. *Performance Evaluation*, vol. 67, no. 11, pp. 1172–1186, 2010.

[6] M. Bjorling, J. Axboe, D. Nellans, et al. Linux block IO: Introducing multi-queue SSD access on multi-core systems. In *Proceedings of the 6th ACM International Systems and Storage Conference*, 2013, pp. 22.

[7] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Disk scheduling with quality of service guarantees. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 1999, pp. 400–405.

[8] Denning, and J. Peter. Effects of scheduling on file memory operations. In *Proc. AFIPS 1967 SJCC*, ACM, 1967, pp. 9–21.

[9] Filebench, http://filebench.sourceforge.net/wiki/index.php/Main_Page

[10] P. Goyal, H.M. Vin, and H.C. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *ACM SIGCOMM Computer Communication Review*, 1996, pp. 157–168.

[11] http://wiki.xensource.com/wiki/Blktap

[12] X.Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009, p. 10.

[13] http://www.iometer.org/.htm

[14] S. Ibrahim, H. Jin, L. Lu, B.S. He, and S. Wu. Adaptive disk i/o scheduling for mapreduce in virtualized environment. In *Proceedings of the International Conference on Parallel Processing (ICPP 2011)*, 2011, pp. 335–344.

[15] K. Jian, X.D. Zhu, W.W. Na, J.W. Zhang, X.M. Han, J.G. Zhang, and X. Lu. A performance isolation algorithm for shared virtualization storage system. In *Proceedings of the IEEE International Conference on Networking, Architecture, and Storage (NAS 2009)*, 2009, pp. 35–42.

[16] R.K. Jain, D.W. Chiu, and W.R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. *Technical Report, Digital Equipment Corporation*, Sep.1984.

[17] F.P. Kelly, A. Maulloo, and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[18] D. Kang, C. Kim, K. Kim, and S. Jung. Proportional disk i/o bandwidth management for server virtualization environment. In *Proceedings of the International Conference on Computer Science and Information Technology (ICCSIT 2008)*, 2008, pp. 647–653.

[19] T. Lan, D. Kao, M. Chiang, et al. An axiomatic theory of fairness in network resource allocation. In *Proceedings of the IEEE INFOCOM*, 2010, pp. 1–9.

[20] X. Ling, H. Jin, S. Ibrahim, W.Z. Cao, and S. Wu. Efficient disk i/o scheduling with qos guarantee for xen-based hosting platforms. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, 2012, pp. 81–89.

[21] X.F. Liao, J. Yu, and H. Jin. Vtrim: A performance optimization mechanism for ssd in virtualized environment. In *Proceedings of the 12th IEEE International Conference on Computer and Information Technology (CIT 2012)*, 2012, pp. 236–242.

[22] C. Li, I. Goiri, A. Bhattacharjee, R. Bianchini, and T.D. Nguyen. Quantifying and improving i/o predictability in virtualized systems. In *Proceedings of the 21st IEEE/ACM International Symposium on Quality of Service (IWQoS 2013)*, 2013, pp. 1–6.

[23] L.C. Laung and L.W. James. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[24] T. Ota and S. Okamoto. Using i/o schedulers to reduce i/o load in virtualization environments. In *Proceedings of the IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA 2011)*, 2011, pp. 59–62.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2436388, IEEE Transactions on Services Computing

[25] D. Ongaro, A.L. Cox, S. Rixner. Scheduling I/O in virtual machine monitors. In *Proceedings of the fourth International Conference on Virtual execution environments (SIGPLAN/SIGOPS 2008)*, 2008, pp. 1–10.
[26] A.L.N. Reddy and J. Wyllie. Disk scheduling in a multimedia I/O system. In *Proceedings of the first ACM International Conference on Multimedia*, 1993, pp. 225–233.
[27] Y.L. Shen and L. Xu. Efficient disk I/O characteristics analysis method based on virtual machine technology. *Journal of Software* In Chinese, vol. 21, no. 4, pp. 849–862, 2010.
[28] S.R. Seelam and 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*, 2007, pp. 105–115.
[29] X. Song, J. Yang, and H.B. Chen. Architecting flash-based solidstate drive for high-performance i/o virtualization. *IEEE Computer Society*, 2013.
[30] Y. Song, C. Zhang, Y. Fang, et al. Revenue maximization in time-varying multi-hop wireless networks: A dynamic pricing approach. *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 7, pp. 1237–1245, 2012.
[31] I. Sitaram and D. Peter. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 117–130, 2001.
[32] K.G. Srinivasa, S. Srinidhi, K.S. Kumar, et al. Game theoretic resource allocation in cloud computing. In *Proceedings of the Fifth International Conference on Applications of Digital Information and Web Technologies (ICADIWT)*, 2014, pp. 36–42.
[33] H.L. Tan, L.J Huang, Z.H. He, Y.Y. Lu, and X.B. He. Dmvl: An i/o bandwidth dynamic allocation method for virtual networks. *Journal of Network and Computer Applications*, vol. 39, pp. 104–116, 2014.
[34] P. Valente and F. Checconi. High throughput disk scheduling with fair bandwidth distribution. *IEEE Transactions on Computers*, vol. 59, no. 9, pp. 1172–1186, 2010.
[35] Weiss and Aaron. Computing in the clouds. *ACM Networker*, vol. 11, no. 4, 2007.
[36] F.W. Roland and H.Y. Zha. A look-ahead algorithm for the solution of general Hankel systems. *Numerische Mathematik*, vol. 64, no. 1, pp. 295–321, 1993.
[37] W. Wachs, M. Abd-El-Malek, E. Thereska, et al. Argon: Performance Insulation for Shared Storage Servers. *FAST*, vol. 7, pp. 5–5, 2007.
[38] D.S. Yin and J.A. Liang. A research on storage performance optimization of virtual machine. In *Proceedings of the 6th International Conference on Computer Science & Education (ICCSE2011)*, 2011, pp. 547–551.
[39] S.eehwan Yoo, C. Yoo, and H. Park. Fairness of proportional work-conserving i/o scheduling. *Electronics letters*, vol. 48, no. 12, pp. 682–684, 2012.
[40] J.Y. Zhang, A. Sivasubramaniam, Q. Wang, A. Riska, and E. Riedel. Storage performance virtualization via throughput and latency control. *ACM Transactions on Storage (TOS)*, vol. 2, no. 3, pp. 283–308, 2006.
[41] X.C. Zhang, Y.H Xu, and S. Jiang. Youchoose: A performance interface enabling convenient and efficient qos support for consolidated storage systems. In *Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST 2011)*, 2011, pp. 1–12.

Huailiang Tan received the BS degree from Central South University, China, in 1992, and the MS degree from Hunan University, China, in 1995, and the PhD degree from Central South University, China, in 2001. He has more than eight years of industrial R&D experience in the field of information technology. He was a visiting scholar at Virginia Commonwealth University from 2010 to 2011. He is currently an Associate Professor at College of Information Science and Engineering, Hunan University, China. His research interests include embedded systems, high performance I/O, and GPU architectures.

Li Chao is a M.S. Degree candidate in the Department of Computer Science and Technology at Hunan University. He received the B.S. degree in Electronics Science and Technology from Hunan University of Science and Technology in 2012. His current research interests include I/O virtualization and embedded systems.

Zaihong He received the BS degree from Hunan University, China, in 1992, and the MS degree from Hunan University, China, in 2007. She is currently a Ph.D candidate in the Department of Computer Science and Technology at Hunan University. She is also an Assistant Professor at College of Information Science and Engineering, Hunan University, China. Her research interests include big data storage system and high performance I/O.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 350 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*. He is an IEEE Fellow.

Kai Hwang is a professor of Electrical Engineering and Computer Science, University of Southern California. He is also an EMC-endowed visiting chair professor at Tsinghua University, China. He received the Ph.D. from University of California, Berkeley in 1972. His has published 8 books and 240 papers, which have been cited over 14,000 times with a citation h-index of 51. His latest book *Distributed and Cloud Computing* (with G. Fox and J. Dongarra) was published by Kaufmann in 2011 which has been translated to Chinese in 2013.

An IEEE Life Fellow, Hwang received CFC Outstanding Achievement Award in 2004, the Founders Award from IEEE IPDPS-2011 and a Lifetim Achievemnt Award from IEEE Cloudcom-2012. He has served as the Editor-in-Chief of the *Journal of Parallel and Distributed Computing* for 28 years and delivered 40 keynote speeches in major IEEE/ACM Conferences. Presently, he serves on the editorial boards of *IEEE Trans. on Cloud Computing* and *International Journal of Big Data Intelligence*. Recently, he has co-chaired the *IEEE CloudCom* held in Athens in Nov. 2012 and the *Second ASE International Conf. on Big Data Science, Social Computing, and Cybersecurity* held at Stanford University in May 27-29, 2014.