CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **28**:386–399
Published online 28 May 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3496

SPECIAL ISSUE PAPER

iShare: Balancing I/O performance isolation and disk I/O efficiency in virtualized environments

Song Wu[1], Songqiao Tao[1], Xiao Ling[2,*,†], Hao Fan[1], Hai Jin[1] and Shadi Ibrahim[3]

[1]*Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China*
[2]*Information & Communication Company, Hunan Electric Power Corporation of State Grid, Changsha, 410007, China*
[3]*Inria, Rennes Bretagne Atlantique Research Center, France*

SUMMARY

Performance isolation has long been a challenging problem for disk resource allocation in virtualized environments. While there have been many researches working on I/O performance isolation and disk utilization, none of them addresses the I/O performance isolation and disk utilization as a whole. To this end, we investigate the impact of current disk I/O performance isolation schemes on disk I/O utilization. Interestingly, our studies report that current isolation schemes bring unnecessary disk idle and reduce the overall disk I/O performance because of ignoring the disk states and characteristics of requests. Accordingly, we propose an adaptive proportional-share I/O scheduling framework, named *iShare*, in virtualized environments. *iShare* not only ensures I/O performance isolation through proportionally allocating time slices according to the weights of virtual machines but also preserves high disk efficiency by detecting disk states and adaptively adjusting the time slice size based on characteristics of requests. We implement a prototype of *iShare* on the Xen platform. The experimental results show that *iShare* ensures I/O performance isolation while improving disk I/O efficiency, compared with *Blkio* (i.e., the default I/O performance isolation method in Xen), *iShare* increases disk I/O bandwidth by 58% and slightly improves the I/O performance isolation for the sequential write applications. Copyright © 2015 John Wiley & Sons, Ltd.

Received 15 October 2014; Revised 6 March 2015; Accepted 10 March 2015

KEY WORDS: virtualization; I/O scheduling; I/O performance isolation; disk I/O efficiency

1. INTRODUCTION

Virtualization technology has become a prominent tool in cloud environments: it enables multiple virtual machines (VMs) to run on one single physical machine. For example, Amazon web [1] services rely on the Xen virtualization hypervisor [2] to provide the VM-based infrastructure as a service solution, which enables users to lease or customize their own environments and run their own applications. More and more Big Data applications are running in cloud environments for efficient and flexible resource management. However, Big Data applications are likely to cause massive I/O operations. This probably brings fierce disk I/O resource contention among VMs because the hypervisor can not provide I/O performance isolation among VMs [3, 4]. And the contention leads to unpredictable I/O completion time, thus affecting service of quality (QoS) of VMs in virtualized cloud. Therefore, disk resource allocation is required to enable I/O performance isolation among VMs.

*Correspondence to: Xiao Ling, Information & Communication Company, Hunan Electric Power Corporation of State Grid, Changsha, 410007, China.
†E-mail: xlinghust@gmail.com

Current works propose proportional sharing of storage resource in the hypervisor based on the performance requirements of VMs to ensure I/O performance isolation [5–9]. For example, *Blkio* [5] proportionally assigns I/O time slices to VMs according to the weights of VMs. Although these schemes ensure I/O performance isolation among VMs, they waste the disk resource because of reserving and limiting the disk I/O resource of VMs. The actual disk I/O resource requirement of a VM is smaller than reserved resource when the VM does not have any backlogged request. The waste of disk I/O resource greatly reduces the I/O performance of VMs in virtualized cloud. Consequently, both the industry and academia have recently paid serious attention to the impact of I/O performance isolation on disk I/O efficiency [10, 11]. For instance, Tencent Company (the leading internet company in China) observes that the existing I/O performance isolation scheme reduces disk I/O bandwidth by 40%–50% in cases with write requests, according to feedbacks of its businesses in its virtualized data center.

To further study the relationship between disk I/O performance isolation and utilization, a series of I/O-intensive experiments is conducted in different I/O scheduling methods (i.e., *CFQ* and *Blkio*) and disk states (i.e., level of disk contention). We discover two typical disk states in the virtualized data center. The overloaded state denotes I/O load capacity if virtual disk is overloaded. The underloaded state denotes I/O load capacity if virtual disk is not overloaded. By comparing bandwidths in different I/O scheduling methods, we find that ensuring I/O performance isolation does not impact the disk I/O performance in the overloaded state, but reduces disk bandwidth in the underloaded state. Even if the current served VM has no backlogged request, the disk can not serve other VMs during the current served VM's time slice. This leads to the disk idle. When service time of pending requests is short and arrival time intervals of future requests are long, the disk idle reduces I/O bandwidth of the disk. Actually, I/O intensive applications do not often lead to overloaded state. For example, the I/O access of the MapReduce [12] application neither exceeds the I/O load capabilities of device nor results in the backlogged requests in the block layer because of the overlapping CPU computation, which is beyond the scope of this paper.

Accordingly, we present an adaptive proportional-share I/O scheduling in hypervisor, named *iShare*, to support I/O performance isolation among VMs while enhancing the disk I/O performance. *iShare* can automatically detect the levels of disk contention and then use different I/O scheduling methods. When *iShare* perceives the overloaded state, it proportionally allocates dedicated time slices to VMs based on their weights to guarantee I/O performance isolation. Furthermore, once *iShare* detects the disk's underloaded state, it leverages a *dynamic isolation* (*DyIso*) algorithm to adaptively adjust the time slice size of VMs. The *DyIso* algorithm estimates the service time and arrival interval of requests of VMs to make the decision on maintaining the proportional time slice allocation. By doing this, *iShare* avoids the unnecessary disk idle and therefore ensures disk I/O efficiency. We implement a prototype of *iShare* on the Xen platform and evaluate the I/O performance of VMs and disk efficiency. Our experimental results show that *iShare* can guarantee I/O performance isolation among VMs while reserving the high disk I/O bandwidth. In contrast to *Blkio* (the default I/O performance isolation method in Xen), *iShare* not only improves the total bandwidth by 58% but also enhances I/O performance of VMs running sequential write applications.

The rest of this paper is organized as follows. In Section 2, we give a brief introduction to *Blkio*. Section 3 analyzes the impact of I/O performance isolation on disk I/O efficiency. The design of *iShare* and related algorithms are discussed in Section 4. Section 5 shows the extensive performance evaluation. Section 6 briefly summarizes the related work. Finally, Section 7 concludes the paper.

2. BACKGROUND

Linux kernel implements a control group (*cgroups*) [13] mechanism to limit and isolate resource usage (e.g., CPU, memory, and disk I/On) of process groups. The *cgroups* mechanism provides a unified interface to resource management in traditional and in virtualized environments. Therefore, the *cgroup* mechanism is widely applied in virtualized data centers. Block I/O (*Blkio*) is a subsystem of the *cgroups* mechanism. It controls access of processes to I/O on block devices. *Blkio* offers two policies: I/O throttling and proportional weight division.

DOI: 10.1002/cpe

The *I/O throttling policy* sets an upper limit for the number of I/O operations performed by a specific device. Because of the changing resource requirement of application, setting a constant limitation is hard and results in poor I/O performance. Thus, this policy is not widely applied.

The *proportional weight division policy* allocates disk I/O resource based on the weights of process groups to ensure I/O performance isolation among groups. This policy is implemented in *Completely Fair Queuing scheduler* (*CFQ*) and assigns groups dedicated time slices based on the proportion of weights. Thus, each group has a set percentage of all I/O operations reserved. Compared with the I/O throttling policy, the proportional weight division policy is more practical for ensuring I/O resource isolation. In virtualized environments, *Blkio* based on *CFQ* ensures I/O performance isolation among VMs by mapping a VM to a process group and setting different weights for VMs.

3. I/O PERFORMANCE ISOLATION VERSUS DISK I/O EFFICIENCY

The hypervisor strongly relies on the I/O performance isolation schemes in order to meet QoS requirements of VMs. For instance, Xen and KVM [14] rely on an I/O performance isolation scheme (i.e.,*Blkio*) to proportionally assign the I/O time slices to VMs according to the weights of VMs. With the growing number of VMs hosted by the same physical machine, improving disk I/O efficiency is important. Virtualized data centers are therefore increasingly concentrating on an issue: how does the I/O performance isolation scheme impact the disk I/O efficiency while ensuring QoS of VMs? In this section, we first give a big picture on this issue and then provide further analysis. We carry out a series of experiments on Xen platform without and with the I/O performance isolation scheme: default *CFQ*[‡] and *Blkio*,[§] respectively.

3.1. Experimental setup

We conduct our experiment using one physical node, equipped with four quad-core 2.13 GHz Xeon processor, 4 GB of memory and one dedicated SAS disk of 146 GB (RAID5), running CentOS6.4 with kernel 3.0.57. All results are obtained using Xen version 4.0.4. Two guest VMs (named *VM1* and *VM2*, respectively,) are deployed within the physical node. The guest VM is configured with two virtual CPUs, 1 GB memory and 30 GB virtual disk, running CentOS6.4 with Linux kernel 2.6.32.

Disk states are classified into overloaded state and underloaded state in a virtualized data center. In the overloaded state, the I/O resource requirements of VMs are larger than the physical disk I/O resource. On the contrary, the I/O resource requirements of VMs are smaller than the physical disk I/O resource in the underloaded state. Our experiments cover both states: VMs running in the overloaded scenarios and in the underloaded scenarios. In each scenario, we do six typical experiments with I/O-intensive workloads. Both VMs run the same workload, and the weight ratio is set to 2:1 under *Blkio*. Table I describes these workloads generated by *fio* [15]. These workloads use synchronous I/O ways and bypass I/O buffers of VMs' operating systems to access data. 30% requests are 4 KB, 40% requests are 8 KB, and the rest are 16 KB.

3.2. Macroscopic analysis

Table II shows the bandwidths of VMs and the total disk bandwidth in the overloaded scenarios and underloaded scenarios, respectively. By comparing the total disk bandwidth under *Blkio* with that under the default *CFQ* in these scenarios, we illustrate the disk overhead[¶], which is caused by ensuring I/O performance isolation under *Blkio*, as shown in Table II.

[‡] *CFQ* is a default disk I/O scheduler in the block device driver of Xen.

[§] *Blkio* is *CFQ* based on the blkio, which uses proportional weight division policy in this section and the rest of paper.

[¶] In Table II, the symbol "+" means that *Blkio* brings disk I/O overheads and reduces disk I/O performance. The symbol "-" means that *Blkio* decreases disk I/O overheads and improves disk I/O performance.

DOI: 10.1002/cpe

Table I. The description of the workloads generated by *fio*.

Scenario	Workload	Description
overload	*O-SR*	sequential read 32 files, with total size of 4 *GB*
	O-SW	32 threads sequential write 32 files, with total size of 4 *GB*
	O-RR	32 threads random read 32 files, with total size of 4 *GB*
	O-RW	32 threads random write 32 files, with total size of 4 *GB*
	O-RRW	32 threads mixed random read and write 32 files, with total size of 4 *GB*
	O-SRW	32 threads mixed sequential read and write 32 files, with total size of 4 *GB*
underload	*U-SR*	sequential read 1 file, with size of 2 *GB*
	U-SW	32 threads sequential write 1 file, with size of 2 *GB*
	U-RR	32 threads random read 1 file, with size of 2 *GB*
	U-RW	32 threads random write 1 file, with size of 2 *GB*
	U-RRW	32 threads mixed random read and write 1 file, with size of 2 *GB*
	U-SRW	32 threads mixed sequential read and write 1 file, with size of 2 *GB*

Table II. The bandwidths of virtual machines and total disk I/O bandwidth in overloaded scenario and in underloaded scenario (KB/s).

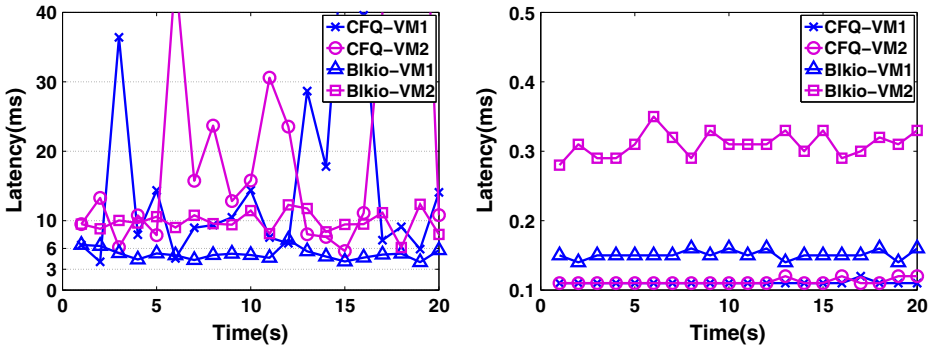
		CFQ			*Blkio*			
Scenario	Workload	*VM1*	*VM2*	Total	*VM1*	*VM2*	Total	Overhead
overload	*O-SR*	22829	20109	42938	28403	14232	42635	+0.71%
	O-SW	80176	65061	145237	105783	52876	158659	−9.24%
	O-RR	4925	4448	9373	6162	3113	9275	+1.05%
	O-RW	6716	3446	10162	6766	3431	10197	−0.34%
	O-RRW	R:2790	R:2251	R:5041	R:3141	R:1576	R:4717	+6.39%
		W:2798	W:2217	W:5015	W:3098	W:1598	W:4696	
	O-SRW	R:8118	R:7146	R:15264	R:10282	R:5173	R:15455	−1.32%
		W:8168	W:7135	W:15303	W:10367	W:5149	W:15516	
underload	*U-SR*	26765	28054	54819	58691	29975	88666	−61.7%
	U-SW	73348	73409	146757	56063	27988	84051	+42.7%
	U-RR	3676	6142	9818	6311	3147	9458	+3.7%
	U-RW	4875	5833	10708	6858	3425	10283	+4.0%
	U-RRW	R: 1747	R: 1745	R: 3492	R: 1642	R: 815	R: 2457	+29.5%
		W:1753	W:1723	W: 3476	W: 1637	W: 820	W: 2457	
	U-SRW	R:42856	R: 42338	R: 85194	R:32953	R: 16558	R: 49511	+41.2%
		W:42764	W: 43353	W: 85117	W: 32909	W:16558	W: 49467	

3.2.1. Overloaded cases. Table II shows that the overheads brought by *Blkio* are small and tolerable in the overloaded scenarios. Moreover, the proportion of bandwidth between applications reflects their weights. Hence, *Blkio* does not waste disk bandwidth while ensuring QoS of VMs in overloaded state.

3.2.2. Underloaded cases. We observe that although *Blkio* ensures I/O performance isolation in the underloaded scenarios with write operations, it reduces disk bandwidth and I/O performance of VMs significantly. For instance, compared with default *CFQ*, *Blkio* reduces disk bandwidth by 42.7% in the *U-SW* experiment and by 41.2% in the *U-SRW* experiment. In contrast, the disk bandwidth under *Blkio* increases by 61.7% compared with that under default *CFQ* in the *U-SR* experiments. Therefore, *Blkio* strongly affects disk bandwidth in underloaded scenarios, especially the ones with write operations.

In order to provide deeper analysis of the aforementioned phenomenon in underloaded scenarios, we trace latencies of VMs' requests in the *U-SW* and in the *U-SR* experiment, as shown in Figure 1. First, compared with latencies of read requests, the latencies of write requests are very small and the fluctuations of their latencies are stable. Second, *Blkio* reduces the average latency of requests

DOI: 10.1002/cpe



(a) Reads requests (b) Write request

Figure 1. Latency of sequential reads and writes under *Completely Fair Queuing* and *Blkio*.

and smoothes out the fluctuations of latencies in the *U-SR* experiment. By contrast, *Blkio* increases the average latency of requests in the *U-SW* experiment.

3.3. Microscopic analysis

Through the aforementioned experiments, we observe that the current I/O performance isolation scheme (*e.g., Blkio*) strongly impacts disk bandwidth in underloaded scenarios compared with overloaded scenarios. Furthermore, this scheme leads to significant degradations in the total disk bandwidth and I/O performance of VMs in the underloaded scenarios with write operations. The key reason for this phenomenon is that in virtualized environments with disk I/O underloaded, *the I/O performance isolation scheme introduces a waiting time, even if pending requests from VMs exist in the hypervisor*. Later in the section, we will discuss this issue in more detail.

3.3.1. Underloaded cases—the introduction of the waiting time. *Blkio*, as well as other I/O isolation schemes, allocates a fixed time slice to each VM based on the weights of VMs to reduce I/O contention among VMs and guarantee the I/O performance isolation. But unlike the disk I/O overloaded scenarios, physical I/O resource can meet the I/O resource requirements of VMs in underloaded scenarios. So VMs do not have backlogged requests when the disk I/O is underloaded. But *Blkio* waits for the next request from the current served VM, ignoring pending requests from other VMs till the VM's time slice is over, when preparing to dispatch requests. As a result, *Blkio* leads to waiting time in the underloaded scenarios. For example, as shown in Figure 2, at *t2*, the request *a1* from *VM1* is completed and no pending request is from *VM1*. Because the time slice of *VM1* is not over, *Blkio* continues to wait for the next request from *VM1* until the expiration of its time slice, instead of serving the request *b1* from *VM2*. After the time slice of *VM1* is expired at t5, *Blkio* serves pending requests from *VM2* (*b1, b2, b3*). Therefore, the state of disk is idle from *t2* to *t3* and from *t4* to *t5*, although block layer has pending requests from *VM2*.

The aim of I/O performance isolation is to meet I/O resource requirements of VMs. However, physical I/O resource can meet the I/O resource requirements of VMs in the underloaded scenarios. Thus, ensuring I/O performance isolation is not necessary in these scenarios. Instead, because VMs do not have backlogged requests in underloaded scenarios, I/O performance isolation brings the waiting time and the disk idle, which may waste disk bandwidth and lower the I/O performance of VMs (Figure 2).

3.3.2. Underloaded cases—the waiting time impacts disk I/O efficiency. We further analyze how the waiting time affects the disk I/O efficiency. As shown in Figure 4(b), if the service time of the request *b1* is smaller than that of the request *a2* or if *a2* from *VM1* arrives after the end of the time slice of *VM1*, the waiting time for *a2* is unnecessary and results in a low disk I/O utilization. Instead, if the service time of *b1* is larger than that of *a2*, the waiting time for *a2* reduces the disk seek time between *VM1* and *VM2* and thus improves disk I/O utilization. Consequently, in virtualized

DOI: 10.1002/cpe

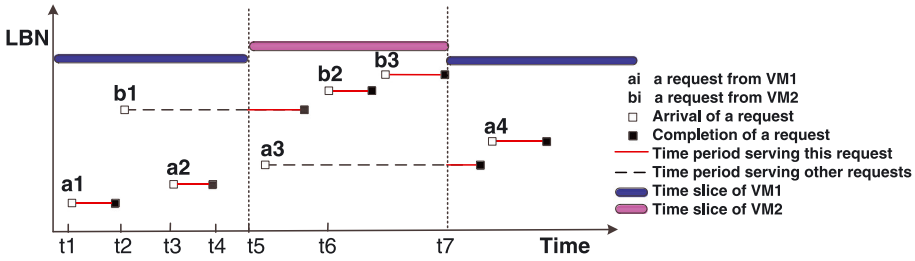


Figure 2. I/O scheduling with the I/O performance isolation scheme.

environments with disk I/O underloaded, the disk I/O overhead introduced by the I/O performance isolation is stated as:

$$O(R_i, R_j) = \begin{cases} Twait(R_i) + Tservice(R_i) - Tservice(R_j); & TA(R_i) \leqslant TE(VM_n) \\ Twait(R_i); & TA(R_i) > TE(VM_n) \end{cases} \quad (1)$$

where R_i is the next request from the current served VM (VM_n), R_j is the pending request from other VMs, $Twait(R_i)$ is the waiting time for R_i, $Tservice$ is service time of request in the device, $TA(R_i)$ is the arrival time of R_i, and $TE(VM_n)$ is the end of time slice of VM_n.

Based on (1), the impact of the waiting time on disk I/O performance is due to characteristics of requests, including the service time and the arrival time. For instance, because of spatial locality among VMs, the service time of read requests from the same VM is smaller than that of pending read request from other VMs. The introduction of waiting time can exploit this locality to batch requests from the current served VMs and reduce the disk seek time across VMs. Hence, as shown in Figure 1(a), compared with default *CFQ*, *Blkio* reduces the latencies of read requests and stabilizes their fluctuations. Instead, as shown in Figure 1(b), because of the impact of disk cache, latencies of write requests from different VMs are very small and almost the same *CFQ*. Therefore, the waiting time that *Blkio* brings increases the latency of write requests. Additionally, when the arrival time of next request is too long (i.e., larger than the remaining time slice), the introduction of waiting time also reduces disk I/O utilization.

3.3.3. Summary. The existing I/O performance isolation scheme (i.e., *Blkio*) cannot dynamically adjust time slice according to the disk state (i.e., underload and overload) and the characteristics of requests. This brings the unnecessary disk idle, thus wasting disk I/O resource in underloaded scenarios when service time of pending request is not greater than that of next request from the current served VM or arrival time intervals of next requests are long. Therefore, we need to dynamically adjust time slice among VMs according to disk state and characteristics of requests to enhance the disk I/O utilization and guarantee the I/O performance isolation.

4. *ISHARE*: EFFICIENT I/O SCHEDULING SYSTEM

We present an adaptive proportional-share I/O scheduling in hypervisor, named *iShare*, to improve disk I/O efficiency while ensuring I/O performance isolation among VMs. Similar to the state-of-the-art I/O performance isolation schemes (e.g., *Blkio*), *iShare* allocates time slices to VMs in proportion to their weights. However, different from them, *iShare* distinguishes disk state and then dynamically adjusts time slices of VMs, being aware of characteristics of requests, to obtain a good tradeoff between I/O performance isolation and disk I/O efficiency. As discussed in Section 3, the disk I/O overhead is due to the service time and the arrival time of requests in the underloaded state. *iShare* therefore embraces a *dynamic isolation* (*DyIso*) algorithm, which captures these characteristics of requests and accordingly adjusts proportional time slice allocation to avoid the unnecessary disk idle and ensure the high I/O throughput. In this section, we first introduce the *DyIso* algorithm and then detail the implementation of *iShare*.

DOI: 10.1002/cpe

4.1. DyIso algorithm

The aim of the *DyIso* algorithm is to avoid the unnecessary disk idle. Therefore, according to the analysis in Section 3, the *DyIso* algorithm decides whether to end the time slice of the current served VM when the block layer does not have any pending requests from the VM.

First, the *DyIso* algorithm introduces two characteristic parameters for each VM to predict service time and access interval of requests, respectively. One characteristic parameter (i.e., *await*) is the recent average completion time interval between requests from the same VM, for estimating service time of requests. The other characteristic parameter (i.e., *thinkt*) is the recent average arrival time interval between requests from the same VM, for predicting the access interval of VM. Because the impact of the latest interval on the predication is greater than that of historical ones, we adopt a time-weighted average method to compute the two characteristic parameters. After completing a request (R_i), the *await* of VM_n (i.e., $VM_n.await_i$) is updated as follows:

$$\begin{aligned} &VM_n.await_i = (totalw_i + c)/Sw_i; \\ &totalw_i = a * totalw_{i-1} + b * (R_i.endtime - R_{i-1}.endtime); \\ &Sw_i = a * Sw_{i-1} + b; \end{aligned} \tag{2}$$

where $totalw_i$ is the weighted sum of historical completion time interval and the latest completion one. Sw_i is the weighted number of completion time interval. When a new request (R_j) from VM_n arrives, the *thinkt* of VM_n (i.e., $VM_n.thinkt_j$) is updated as follows:

$$\begin{aligned} &VM_n.thinkt_j = (totalt_j + c)/St_i; \\ &totalt_j = a * totalt_{j-1} + b * (R_j.arrivetime - R_{j-1}.arrivetime); \\ &St_j = a * St_{j-1} + b; \end{aligned} \tag{3}$$

where $totalt_i$ is the weighted sum of historical arrival time interval and the latest arrival one. St_i is the weighted number of arrival time interval. In (2) and (3), a, b, and c are the weight parameters. And weight of the latest time interval (i.e., a) is much greater than that of historical one (i.e., b).[‖] So the *DyIso* algorithm can capture these changes once the characteristics of requests are changed.

Then, the *DyIso* algorithm decides whether to end this VMs remaining time slice based on the two aforementioned characteristic parameters. The algorithm is as follows:

where VM_n is current served VM. VM_j are the other VMs. *max_seektime* is the maximal disk head seek time. $VM_n.slice\_rest$ is the rest of VM's time slice. $VM.pending\_requests$ is the number of pending requests from VM.

When there is no pending request from the current served VM in the block layer and the VMs time slice is not over, the *DyIso* algorithm works (Line 8). If the current served VMs time slice is not over and the $VM_n.thinkt$ is greater than its $VM_n.slice\_rest$, the *DyIso* algorithm ends the VMs time slice and then serves pending requests from other VMs (Line 9-10). Otherwise, the *DyIso* algorithm considers the disk I/O overhead of serving other VMs and that of serving the current served VM. If the *await* of a VM with pending requests ($VM_j.awat$) is not greater than that of the current served VM ($VM_n.awat$) and is smaller than $max\_seektime$ (Line 11-14), it means that the service time of pending requests from other VMs is smaller than the sum of the arrival time interval and service time of the next request from the current served VM. And the prediction is reliable. So the *DyIso* algorithm also ends the time slice of the current served VM and serves the VM with smaller *await* (Line 15). Otherwise, the *DyIso* algorithm still maintains the time slice allocation to reduce disk seek overheads between VMs.

4.2. Implementation of iShare

We design and implement *iShare* based on *DyIso* algorithm in the block device driver of Xen. *iShare* maintains a request queue for each VM in the block layer and allocates a dedicated time slice to the VM. Then, to distinguish underloaded and overloaded states, *iShare* dynamically divides VMs

[‖]According to experiences in the experiments, when a, b, and c is set to 7/8, 32, and 1024, respectively, the *await* and the *thinkt* accurately estimate the recent average completion time interval and the recent average arrive time interval between requests from the same VM.

DOI: 10.1002/cpe

Algorithm 1: *Dynamic Isolation* algorithm

```
/*VM_n is current served VM; VM_j are other VMs;max_seektime: maximal disk head seek
time */
Input: VM_n.thinkt; VM_n.awat; VM_j.awat;
VM_n.slice_rest: the rest of VM's time slice;
VM.pending_requests is the number of pending requests from VM
output: VM_n.slice_rest
/*decide whether to wait for next requests from VM_n*/
if VM_n.pending_request == 0 && VM_n.slice_rest > 0 then
    if VM_n.thinkt_i ≥ VM_i.slice_rest then
        VM_n.slice_rest = 0;
    else
        foreach j in the number of other VMs do
            if VM_n.await_i ≥ VM_j.await && VM_j.pending_requests > 0 &&
            VM_j.await < max_seek time then
                VM_n.slice_rest = 0;break;
            end
        end
    end
end
```

into two classes — *backlogged* VMs and *non-backlogged* VMs. If the number of pending requests in VM's request queue is greater than 0, the VM is a *backlogged* VM. Otherwise, the VM is a *non-backlogged* VM. When all VMs are *backlogged* VMs, the disk state is the overloaded state. Thus, the total I/O resource requirements of VMs are greater than physical I/O resource. *iShare* strictly assigns VMs time slices in proportion to the weights of VMs and serves VMs in order of their weights, in order to ensure I/O performance isolation. Once the VMs become *non-backlogged* VMs, the disk state is the underloaded state. Thus, the physical I/O resource can meet the I/O resource requirements of VMs. In this situation, the reserved time slice of the VM wastes the disk I/O resource because of the characteristics of requests, as discussed in Section 3. So *iShare* triggers the *DyIso* algorithm to adjust the reserved time slices of these *non-backlogged* VMs and relaxes I/O performance isolation when serving these VMs.

Besides, to ensure a suitable decision, *iShare* triggers the *await* update of *DyIso* algorithm when a request is completed. And the *thinkt* updates when a request arrives. The *DyIso* algorithm uses a link between VMs' *await* values to compare their *await* values and then decides whether to keep the time slice. If the algorithm decides to end the time slice of the current served VM, *iShare* selects a *backlogged* VM and dispatches pending request from the *backlogged* VM. The *await* of the *backlogged* VM is not greater than that of the current served VM. Moreover, the position of *backlogged* VM should close to disk head, and the VM is not served in this round. Therefore, *iShare* can provide I/O performance isolation while improving disk I/O efficiency, although being aware of access characteristics of VMs.

It is important to note that *iShare* can be implemented in other virtualized platforms (*e.g.,*, KVM and Linux-VServer [16]). The *DyIso* algorithm uses hypervisor-independent parameters, including the completion time and arrival time of requests. Also, the *DyIso* algorithm is implemented as a separate algorithm at the block layer of the hypervisor. Accordingly, *iShare* is also applied in traditional shared storage environments without virtualization technology.

4.3. Example to demonstrate how iShare works

As shown in Figure 3, we take an example to illustrate how *iShare* works. Two VMs are deployed in a physical machine and their weight ratio is 2:1. Our analyses begin at *t1*. We assume that the two VMs have the same *await* and the *thinkt* of *VM1* is equal to the rest time slice of *VM1* (*VM1.slice_rest*) at *t1*.

DOI: 10.1002/cpe

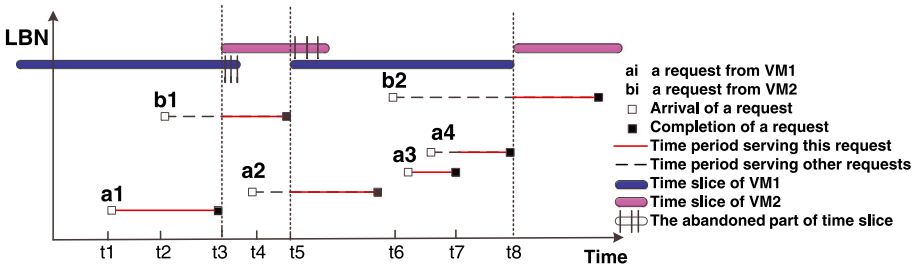


Figure 3. I/O scheduling with *iShare*.

From *t1* to *t3*, the request *a1* from *VM1* is served. *VM2s thinkt* is updated on *b1s* arrival at *t2* and *it* is smaller than *VM2.slice_rest*. At *t3*, *VM1* becomes a *non-backlogged VM*, because the request *a1* is completed and the request queue of *VM1* is empty within time slice of *VM1*. So the *iShare* use *DyIso* algorithm to handle requests. Because the *thinkt* of *VM1* is equal to *VM1.slice_rest*, *iShare* updates the *await* of *VM1* and ends the service of *VM1* when the *a1* is completed.

From *t3* to *t5*, *b2* from *VM2* is served. At *t4*, *VM1.thinkt* is updated since the arrival of *a2*. At *t5*, *b1* is completed and *VM2.await* is updated. Because *VM2.thinkt* (updated at *t2*) is smaller than *VM2.slice_rest*, *iShare* compares *VM2.await* with *VM1.await*. Processing *a1* takes shorter time than *b1*, so *VM1.await* is smaller. *iShare* switches to *VM1*.

From *t5* to *t8*, *iShare* serves *a2* from *VM1*. Even if *b2* of *VM2* arrive at *t6*, the time slice of *VM1* doesnt end because *VM1.await* is smaller than *VM2.await*. At *t7*, the request queue of *VM1* has a backlogged request *a4* within reserved time slice of *VM1*. So *iShare* does not end the time slice of *VM1* to ensure I/O performance isolation. At *t8*, because the time slice of *VM1* is expired, *iShare* switches to serve pending requests from *VM2*.

5. PERFORMANCE EVALUATION

We implement a prototype of *iShare* in Xen-4.0.4 with Linux kernel 3.0.57. In this section, we evaluate our solution with various workloads including disk I/O overloaded, disk I/O underloaded, and the changing disk I/O load. The experimental setup is the same as described in Section 3.

5.1. Evaluation with disk I/O overload

We evaluate the performance of VMs and disk under *iShare* against that under *Blkio*. To do so, we still use two VMs in the overloaded scenarios, as described in Section 3.1. The proportion of weight in *iShare* is set to 2:1 for *VM1* and *VM2*. Tables III and IV show the bandwidths of VMs and disk in the overloaded states under *iShare*, *CFQ* and *Blkio*. We illustrate the disk overhead.[**] By comparison between the performance of *iShare* and that of *CFQ* in Table III, we find that the overheads brought by *iShare* are small and tolerable in the overloaded scenarios. Moreover, the proportion of bandwidth between applications reflects their weights. By comparison between the performance of *iShare* and that of *Blkio* in Table IV, we find that both *iShare* and *Blkio* ensure the I/O performance isolation and achieve almost the same total disk bandwidth in the overloaded scenarios. This is because when the VMs are in overloaded scenario, both VMs have backlogged requests. *iShare* proportionally allocates time slices like *Blkio*. Accordingly, we can draw the conclusion: *iShare* has no additional costs (no adverse effects) and can ensure both high bandwidth and I/O performance isolation among VM in the overloaded scenario as *Blkio*.

[**] In Tables III and IV, the symbol "+" means that *iShare* reduces disk I/O overheads and improves disk I/O performance. The symbol "-" means that *iShare* brings disk I/O overheads and decreases disk I/O performance.

DOI: 10.1002/cpe

Table III. The bandwidths of virtual machines and total disk I/O bandwidth under *iShare* and *CFQ* (KB/s).

		iShare			*CFQ*			
Scenario	Workload	*VM1*	*VM2*	Total	*VM1*	*VM2*	Total	**Overhead**
overload	*O-SR*	28614	14461	43075	22829	20109	42938	+0.32%
	O-SW	102847	51483	154330	80176	65061	145237	+6.26%
	O-RR	6136	3120	9256	4925	4448	9373	−1.25%
	O-RW	6517	3623	10140	6766	3431	10197	−0.22%
	O-RRW	R:3084 W:3043	R:1657 W:1624	R:4741 W:4667	R:2790 W:2798	R:2251 W:2217	R:5041 W:5015	−6.44%
	O-SRW	R:10148 W:10158	R:5274 W:5256	R:15422 W:15414	R:8118 W:8168	R:7146 W:7135	R:15264 W:15303	+0.88%
underload	*U-SR*	61773	31849	93622	26765	28054	54819	+70.78%
	U-SW	66539	66590	133129	73348	73409	146757	−9.29%
	U-RR	6603	3284	9887	3676	6142	9818	+0.70%
	U-RW	5524	5517	11041	4875	5833	10708	+3.11%
	U-RRW	R: 1446 W:1447	R: 1492 W:1471	R: 2938 W: 2918	R: 1747 W: 1753	R: 1745 W: 1723	R: 3492 W: 3476	−15.96%
	U-SRW	R:35261 W:35392	R: 35505 W: 35643	R: 70766 W: 71055	R:42856 W: 42764	R: 42338 W:43353	R: 85194 W: 85117	−16.73%

Table IV. The bandwidths of virtual machines and total disk I/O bandwidth under *iShare* and *Blkio* (KB/s).

		iShare			*Blkio*			
Scenario	Workload	*VM1*	*VM2*	Total	*VM1*	*VM2*	Total	Overhead
overload	*O-SR*	28614	14461	43075	28403	14232	42635	+1.03%
	O-SW	102847	51483	154330	105783	52876	158659	−2.73%
	O-RR	6136	3120	9256	6162	3113	9275	-0.21%
	O-RW	6517	3623	10140	6766	3431	10197	−0.56%
	O-RRW	R:3084 W:3043	R:1657 W:1624	R:4741 W:4667	R:3141 W:3098	R:1576 W:1598	R:4717 W:4696	−0.05%
	O-SRW	R:10148 W:10158	R:5274 W:5256	R:15422 W:15414	R:10282 W:10367	R:5173 W:5149	R:15455 W:15516	−0.04%
underload	*U-SR*	61773	31849	93622	58691	29975	88666	+5.59%
	U-SW	66539	66590	133129	56063	27988	84051	+58.39%
	U-RR	6603	3284	9887	6311	3147	9458	+4.54%
	U-RW	5524	5517	11041	6858	3425	10283	+7.37%
	U-RRW	R: 1446 W:1447	R: 1492 W:1471	R: 2938 W: 2918	R: 1642 W: 1637	R: 815 W: 820	R: 2457 W: 2457	+19.17%
	U-SRW	R:35261 W:35392	R: 35505 W: 35643	R: 70766 W: 71055	R:32953 W: 32909	R: 16558 W:16558	R: 49511 W: 49467	+43.29%

5.2. Evaluation with disk I/O underload

Then, we still use two VMs in the underloaded scenarios, as described in Section 3.1 to evaluate *iShare*. The weight ratio is still set to 2:1 for *VM1* and *VM2*. Tables III and IV also present the VMs' bandwidths and total disk bandwidth in the underloaded scenario under *CFQ*, *iShare*, and *Blkio*.

By comparison between the performance of *iShare* and that of *CFQ* in Table III, we find that *iShare* can achieve performance isolation in *U-SR* and *U-RR* experiment. In the *U-SR* experiment, *iShare* brings 71% total bandwidth improvement. We also find that for experiments with write operations, the total bandwidths under *iShare* are smaller than *CFQ*. For example, the total bandwidth under *iShare* is 9% smaller than that of *CFQ* in the *U-SW* experiment, 16% in *U-RRW* experiment, and 17% in the *U-SRW* experiment, respectively. This is because the seek time caused by the

DOI: 10.1002/cpe

frequent time slice switch leads to disk idle. But *iShare* results in an improved total I/O bandwidth compared with *Blkio*. This is because *iShare* can end the VMs' time slices when disk idle, we discussed in Section 3 happens. When I/O-intensive applications with write operations run on VMs in the underloaded scenario, the effect is remarkable. For instance, compared with *Blkio*, *iShare* improves the total bandwidth by 58% in the *U-SW* experiment, 43% in the *U-SRW* experiment, and 19% in the *U-RRW* experiment. The bandwidth proportion between VMs is close to 1:1 under *iShare*. We observe that when VMs are running applications with write operations, time slice switch between two VMs is frequent. The frequent switch prevents the waste of time slice and significantly improves *VM2's* bandwidth. *MV1's* bandwidth decreases in the *U-RW* and *U-RRW* experiment compared with that of *Blkio*. This is because random requests have longer seek time compared with sequential requests. The frequent switch leads to decrease of bandwidth. Instead, *iShare* not only enhances total bandwidth of the disk but also significantly improves the bandwidth of *VM2*, which gets smaller time slice. In addition, for read applications in the underloaded scenario, *iShare* ensures bandwidth proportion of VMs, as well as improves disk I/O bandwidth. For example, in the *U-SR* experiment, *iShare* increases the total bandwidth by about 5% compared with *Blkio*. This is because read-only applications tend to make the parameter *await* of VMs different and the switch is mainly a result from $VMn.thinkt \geqslant VMn.slice\_rest$. This makes the time slice switch for VMs with read operations not as frequent as that for VMs with write operations. So the bandwidth proportion between VMs is still close to 2:1. Therefore, in the underloaded scenarios, *iShare* adjusts time slice according to the characteristics of requests, thus achieving high disk utilization while ensuring QoS of VMs.

5.3. Evaluation with the changing disk I/O load

Moreover, to further evaluate the effectiveness of *iShare*, we use two VMs running I/O-intensive applications with the changing disk I/O load to show that *iShare* can adaptively adjust time slice allocation with the changing disk I/O load and characteristics of requests. We evaluate *iShare* against *CFQ* and *Blkio*. The weight proportion is set to 2:1 for *VM1* and *VM2*. The access patterns of applications generated by *fio* are described in Table I. The applications' access patterns are changed every 100 *s*: *O-SW* from 0 to100 *s*, *U-SR* from 100 to 200 *s*, *U-SR* from 200 to 300 *s*, *O-SR* from 300 to 400 *s*, *U-SW* from 400 to 500 *s*, *O-SW* from 500 to 600 *s*, *O-RW* from 600 to 700 *s*.

Figure 4 depicts the changes of bandwidths in VMs under *CFQ*, *Blkio*, and *iShare*. When access pattern is *O-SW* and the disk is overloaded at the beginning of 100s, we find that both *Blkio* and *iShare* can ensure the bandwidth proportion between VMs compared with *CFQ*. This is because both *Blkio* and *iShare* strictly assign time slices in proportion to the weights of VMs. At 100 *s*, the access pattern changes to *U-SR*. *iShare* not only ensures bandwidth proportion like *Blkio* but also slightly improve the bandwidth, compared with *Blkio*. This means that when the disk state changes from overloaded to underloaded, *iShare* triggers the *DyIso* algorithm, and the algorithm functions well. From 200 to 300 *s*, the disk is still underloaded but the access pattern changes. The I/O bandwidths of VMs and total disk I/O bandwidth under *iShare* are similar to that under default *CFQ*. *Blkio* preserves the bandwidth proportion between VMs. But in underloaded scenarios, physical I/O bandwidth can meet bandwidth requirements of both VMs, as shown in Figure 4(a); high disk I/O efficiency is more important for improving I/O performance of VMs. *iShare* can adaptively adjust time slice allocation with access pattern changes to improve I/O performance of VMs. From 300 to 400 *s*, the disk works in an overloaded state. The I/O resource competition among VMs is more serious, and physical I/O bandwidth cannot meet the I/O bandwidth requirements of VMs, as shown in Figure 4(a). So *iShare* strictly assigns time slices in proportion to the weights of VMs like Blkio. From 400 to 500 *s*, the I/O bandwidth under *iShare* is similar to that from 200 to 300 *s* under *iShare*. This means *iShare*'s bandwidth optimization is stable for the same workload, even if the access pattern and disk state change over time. From 500 to 700 *s*, the disk is in an overloaded state. *iShare* again focuses on I/O performance isolation and preserves the bandwidth proportion between VMs, despite the change of access pattern. According to the aforementioned results, we come to the conclusion that *iShare* can adaptively adjust time slice allocation with the changing disk I/O load and characteristics of requests to ensure I/O performance isolation among VMs and improve disk bandwidth.

DOI: 10.1002/cpe

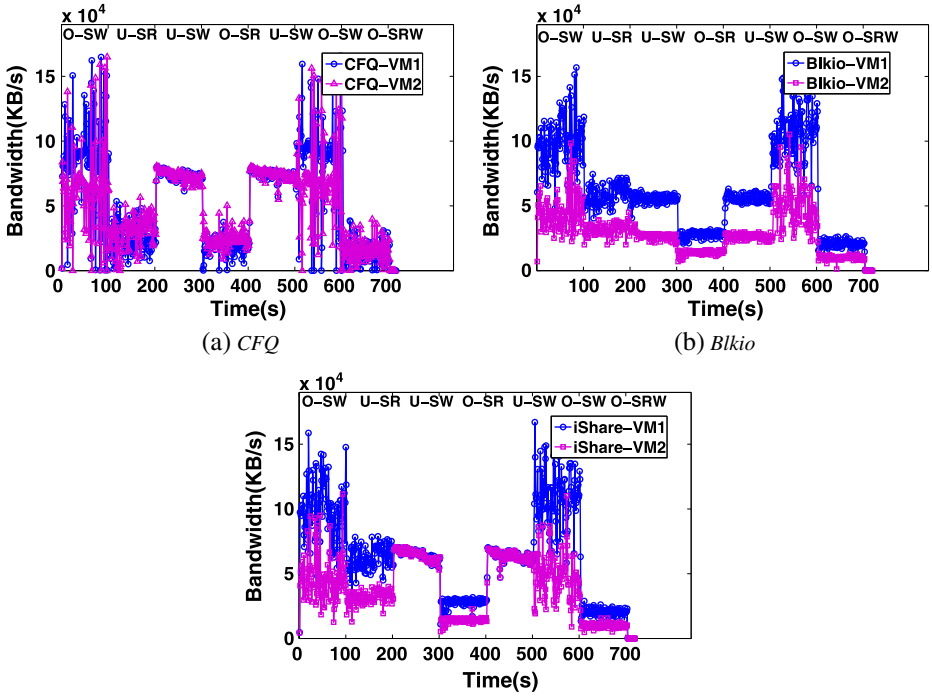


Figure 4. The changes of bandwidth of virtual machines under *CFQ*, *Blkio*, and *iShare*.

6. RELATED WORK

Ever since the advent of virtualization technology, a huge number of studies have been dedicated to ensuring and improving I/O performance of VMs. On one hand, some studies focus on improving disk I/O efficiency in virtualized environments. These studies exploit spatial locality of VM access to achieve high disk I/O utilization [17, 18]. For example, Pregather [18] build a prediction model based on the spatial locality among VMs and the sub-locality among applications to improve the disk I/O utilization. However, they sacrifice the performance of VMs with low spatial locality and thus can not ensure I/O performance of all VMs.

On the other hand, more works study I/O performance isolation among VMs to ensure I/O performance of VMs [5–8, 19–23]. These studies proportionally allocate disk I/O resource to VMs based on the weights of VMs. For example, With PARDA [21], each virtual disk can be assigned an I/O share value, which determines the relative weight of the I/Os from its virtual disk as comparison with others. VMs with higher I/O shares get higher IOPS and lower latency. mClock [8] provides additional controls of reservation and limit to control VM latency. VIOS [7] controls the coarse-grain allocation of disk time to different VMs with a CFQ and compensating round-robin scheduler. IOFlow [24] provides the static rate limit and dynamic rate limit with VMs. But it needs users to prechoose one of the limits and can not switch between the two limits with the changing access of VMs. Note that all the aforementioned solutions ignore the disk I/O overheads introduced by ensuring I/O performance isolation in underloaded scenarios. They can not adjust disk I/O resource with the changing disk state and characteristics of requests, thus leading to disk I/O overheads. Besides, [25] proposes an adaptive work-conserving scheduling to balance fairness among processes and I/O efficiency. But they only focus on all processes with backlogged requests. In addition, although some traditional disk resource schedulers can guarantee QoS and ensure high throughput in shared storage [26, 27], they need to know the information of processes in advance. Therefore, they are not applied in virtualized environments because of the transparent virtualization.

DOI: 10.1002/cpe

The industry and the academia always argue whether I/O performance isolation lowers disk I/O efficiency or not [11, 28]. For example, Lai [28] carries out a series of experiments to compare isolation with sharing. The results show that an isolation strategy loses because isolation may prevent the disk from performing any work because of the specified IOPs constraint of a VM. However, few works have deeply analyzed the issue and discussed the relation between I/O performance isolation and disk I/O efficiency [21, 29]. *To the best of our knowledge, this is the first work to analyze the overhead of the I/O performance isolation in detail and accordingly address this issue by dynamically adjusting disk I/O resource based on the disk state and characteristics of requests for ensuring I/O performance isolation among VMs while improving disk I/O efficiency*.

7. CONCLUSION

In this paper, we investigate the I/O performance of VMs when applying I/O performance isolation schemes in virtualized environments. Our studies reveal that the existing I/O performance isolation schemes introduce an unnecessary disk idle and thus decrease disk I/O performance in some underloaded scenarios. This degradation in the disk I/O utilization strongly depends on the service time and the arrival time of requests. Accordingly, we propose an adaptive proportional-share I/O scheduling in virtualized environments, called *iShare*, to ensure I/O performance isolation among VMs while improving disk I/O efficiency. On one hand, *iShare* proportionally allocates the time slices based on the weights of VMs to ensure I/O performance isolation among VMs. On the other hand, *iShare* detects underloaded state and then takes a *dynamic isolation* (*DyIso*) algorithm to adaptively adjusts disk resource among VMs, in order to preserve the high disk I/O throughput. The *DyIso* algorithm predicts the service time and arrival time interval of requests from VMs and decides whether to keep the proportional time slice allocation. We implement a prototype of *iShare* in the block device driver of Xen and conduct experiments to verify its effectiveness. In the future work, we are interested in using *iShare* in virtualized environments with network storage such as SAN and NAS.

REFERENCES

1. Amazon EC2, 2006. (Available from: http://aws.amazon.com/ec2/)[accessed on November 2014].
2. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP'03)*, New York, NY, USA, ACM, 2003; 164–177.
3. Pu X, Liu L, Mei Y, Sivathanu S, Koh Y, Pu C. Understanding performance interference of I/O workload in virtualized cloud environments. *Proceedings of the 3rd IEEE International Conference on Cloud Computing (Cloud10)*, Florida, USA, IEEE, 2010; 48–51.
4. Koh Y, Knauerhase RC, Brett P, Bowman M, Wen Z, Pu C. An analysis of performance interference effects in virtual environments. *Proceedings of 2007 IEEE International Symposium on Performance Analysis of Systems (ISPAS07)*, San Jose, California, USA, IEEE, 2007; 200–209.
5. Blkio. (Available from: https://www.kernel.org/doc/Documentation/cgroups/blkio-controller.txt)[accessed on November 2014].
6. Shue D, Freedman MJ, Shaikh A. Performance isolation and fairness for multi-tenant cloud storage. *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, Hollywood, USA, 2012; 349–362.
7. Seelam SR, Teller PJ. Virtual I/O scheduler: a scheduler of schedulers for performance virtualization. *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*, New York, NY, USA, ACM, 2007; 105–115.
8. Gulati A, Merchant A, Varman PJ. mclock: handling throughput variability for hypervisor IO scheduling. *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Berkeley, CA, USA, USENIX Association, 2010; 1–7.
9. Cheng G, Jin H, Zou D, Zhang X. Building dynamic and transparent integrity measurement and protection for virtualized platform in cloud computing. *Concurrency and Computation: Practice and Experience* 2010; **22**: 1893–1910.
10. Shieh A, Kandula S, Greenberg A, Kim C. Seawall: performance isolation for cloud datacenter networks. *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (Hotcloud'10)*, Boston, MA, USENIX Association, 2010; 1–7.

DOI: 10.1002/cpe

11. Zhang X, Xu Y, Jiang S. Youchoose: a performance interface enabling convenient and efficient QoS support for consolidated storage systems. *Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST'11)*, Denver, Colorado, USA, IEEE, 2011; 1–7.
12. Dörre J, Apel S, Lengauer C. Modeling and optimizing mapreduce programs. *Concurrency and Computation: Practice and Experience* 2014; **26**. DOI: 10.1002/cpe.3333.
13. Cgroups, 2008. (Available from: http://linuxcommand.org/man_pages/iostat1.html)[accessed on October 2014].
14. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: the linux virtual machine monitor. *Proceedings of the Linux Symposium*, Vol. 1, 2007; 225–230.
15. Fio, 2005. (Available from: http://linux.die.net/man/1/fio)[accessed on November 2014].
16. Linux VServer, 2010. (Available from: http://linux-vserver.org/Documentation)[accessed on November 2014].
17. Xu Y, Jiang S. A scheduling framework that makes any disk schedulers non-work-conserving solely based on request characteristics. *Proccedings of the 9th Conference on File and Storage Technologies (FAST'11)*, San Jose, CA, USA, 2011; 119–132.
18. Ling X, Ibrahim S, Wu S, Jin H. Exploiting spatial locality to improve disk efficiency in virtualized environments. *IEEE Transactions on Parallel and Distributed Computing* 2014. DOI: 10.1109/MASCOTS.2013.27.
19. Zhang Y, Ren SQ, Chen SB, Tan B, Lim ES, Yong KL. Differcloudstor: Differentiated quality of service for cloud storage. *APMRC, 2012 Digest*, Singapore, IEEE, 2012; 1–9.
20. Jin H, Ling X, Ibrahim S, Cao W, Wu S, Antoniu G. Flubber: Two-level disk scheduling in virtualized environment. *Future Generation Computer Systems* 2013; **29**(8):2222–2238.
21. Gulati A, Ahmad I, Waldspurger CA. PARDA: proportional allocation of resources for distributed storage access. *Proccedings of the 7th Conference on File and Storage Technologies (FAST'09)*, Berkeley, CA, USA, USENIX Association, 2009; 85–98.
22. Kesavan M, Gavrilovska A, Schwan K. Differential virtual time (dvt): rethinking i/o service differentiation for virtual machines. *Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC'10)*, Indianapolis, Indiana, USA, ACM, 2010; 27–38.
23. Anglano C, Guazzone M, Canonico M. Fc2q: exploiting fuzzy control in server consolidation for cloud applications with sla constraints. *Concurrency and Computation: Practice and Experience* 2014; **26**. DOI: 10.1002/cpe.3410.
24. Thereska E, Ballani H, O'Shea G, Karagiannis T, Rowstron A, Talpey T, Black R, Zhu T. Ioflow: a software-defined storage architecture. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*, Farmington, PA, USA, ACM, 2013; 182–196.
25. Gulati A, Merchant A, Uysal M, Padala P, Varman P. Efficient and adaptive proportional share i/o scheduling. *ACM SIGMETRICS Performance Evaluation Review* 2009; **37**(2):79–80.
26. Rocha PE, Bona LC. A QoS aware non-work-conserving disk scheduler. *Proccedings of the 28th IEEE Symposium on Mass Storage Systems and Technologies (MSST'12), 2012*, Pacific Grove, CA, USA, IEEE, 2012; 1–5.
27. Skourtis D, Kato S, Brandt S. Qbox: guaranteeing I/O performance on black box storage systems. *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*, Delft, the Netherlands, ACM, 2012; 73–84.
28. Lai C-A, Wang Q, Kimball J, Li J, Park J, Pu C. Io performance interference among consolidated n-tier applications: Sharing is better than isolation for disks. *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)*, AK, USA, IEEE, 2014; 134–147.
29. Hu Y, Long X, Zhang J. I/O behavior characterizing and predicting of virtualization workloads. *Journal of Computers* 2012; **7**(7):1712–1725.