Soft Computing (2020) 24:5987–5997
https://doi.org/10.1007/s00500-019-04551-w

FOCUS

An efficient virtual CPU scheduling in cloud computing

Joonhyouk Jang[1] · **Jinman Jung**[2] · **Jiman Hong**[3]

Published online: 27 November 2019

Abstract
In cloud computing, fine-grained virtual CPU scheduling techniques are essential in hiding physical resources from running applications and mitigating the decrease in performance upon virtualization. However, evaluating and predicting the behaviors of virtual processors is getting harder because of the diverse QoS requirements of cloud applications. In this paper, we propose a novel virtual CPU scheduling scheme to provide a high I/O performance for cloud applications. We present an evaluation function that evaluates the task characteristics of virtual machines by observing the amount of resource consumption of each virtual processor. Based on the evaluation function, the proposed scheduling scheme controls the priorities of virtual machines adaptively for fair distribution in handling I/O requests. Because our scheme evaluates both CPU-intensiveness and I/O-intensiveness of virtual machines, it provides high performance in terms of responsiveness even for various types of tasks. We implemented and experimented the proposed scheme on a virtual machine monitor. The experimental results showed that the proposed scheme increases the responsiveness and I/O bandwidth of virtual machines.

Keywords I/O virtualization · Virtual machine scheduler · Performance

1 Introduction

Due to the increasing number of applications on cloud system and their varying purposes and characteristics, demands for virtualization techniques are increasing. Many existing cloud systems consolidate their servers using virtualization techniques to enable different applications in sharing server resources efficiently (Jain and Paul 2013; Moreno-Vozmediano et al. 2012; Beloglazov and Buyya 2010). The virtualization technique integrates multiple physical resources into a logical resource or separates a physical resource into multiple logical resources to increase resource efficiency. In a virtual machine monitor (VMM), a virtualization layer virtualizes physical resources to operate multiple systems on a single machine. Examples of VMMs include Hyper-V (Velte and Velte 2010), VirtualBox (Watson 2008), Xen (Barham et al. 2003), and KVM (Kivity et al. 2007).

A VMM manages virtual machines just as an operating system does process. As an operating system provides processes with abstracted physical resources, a VMM provides virtual machines with virtualized processors, memory, and I/O. A virtual machine operating in a physical machine is provided a virtual processor instead of a physical processor. While an operating system selects a process to consume CPU resources in each scheduling decision, a VMM selects a virtual processor to consume CPU resources based on the VMM's own scheduling policy.

Xen hypervisor, one of the most well-known VMMs, uses the credit scheduler for a fair scheduling of virtual machines. Credit scheduler works based on the credit, which represents CPU resources. Because applications with various requirements run on a physical machine, a VMM should efficiently virtualize physical resources and fairly allocate resource

Communicated by B. B. Gupta.

Joonhyouk Jang and Jinman Jung have contributed equally.

This paper is an extension version of the conference paper: A Virtual CPU Scheduling Model for I/O Performance in Paravirtualized Environments, J. Jung, J. Park, S. Kim, M. Heo, J. Hong, In Proceeding of the International Conference on Research in Adaptive and Convergent Systems, pp. 20–23. ACM.

✉ Jiman Hong
jiman@ssu.ac.kr

Joonhyouk Jang
jhjang@useed.co.kr

[1] USEED Inc., Siheung-si, Korea

[2] Hannam University, Daejeon, Korea

[3] Soongsil University, Seoul, Korea

shares to achieve a high performance, reliability, and fairness.

However, credit scheduler cannot make ensure high fairness and predictable performance in terms of I/O because its main purpose is providing each virtual machine a fair share of CPU resources; however, boosting, which is applied to credit scheduler to improve the responsiveness of virtual processors for I/O events, is not efficient enough in supporting I/O virtualization.

In the case of boosting, if a virtual processor is idle, which means that the virtual processor rarely consume CPU resources and has a pending I/O event, the virtual processor will be boosted by temporally acquiring the highest scheduling priority. Because a virtual processor with the highest priority preempts virtual processors with lower priorities, the responsiveness of virtual machines is increased. However, boosting is not well suited in certain cases (Ongaro et al. 2008), where it cannot benefit virtual processors with CPU-intensive or mixed workloads because they continuously consume credits and have a number of runnable tasks. When CPU-intensive and I/O-intensive virtual machines are running together on a VMM, fairness in processing I/O events cannot be achieved. In addition, if most of the virtual machines run latency-sensitive applications that provide network-based services, the virtual processors of the virtual machines are fairly boosted, but this situation leads to performance degradation as boosting fundamentally disturbs the scheduling priority of virtual processors.

As described earlier, the credit scheduler has clear drawbacks with respect to I/O virtualization (Iosup et al. 2011; Cherkasova and Gardner 2005; Nagarajan et al. 2007). To improve the I/O performance of credit scheduler, various scheduling policies have been proposed based on priority modification or time slice modification of a virtual processor. For example, the highest priority is assigned to the privileged virtual machine that is in charge of I/O virtualization or adjusting the amount of time slice of the boosted virtual processors (Ding et al. 2014). Further, some studies focused on the virtual I/O in a VMM by optimizing event channels or minimizing context switching overhead between the receiver of an I/O request and the privileged virtual machine (Xi et al. 2011; Williams et al. 2012). However, most of the existing solutions to this problem assume that each virtual machine is either latency-sensitive or not. A virtual machine that runs both CPU-intensive and I/O-sensitive applications can exist, but such workloads are out of the sight of these studies. A virtual machine with mixed workloads may be latency-sensitive similar to I/O-bound virtual machines, but it is rarely boosted because its CPU-intensive applications aggressively consume CPU resources.

In this paper, we propose a novel virtual machine scheduler based on a loan and redeem system. For the purpose of explicitly control boosting, two types of representation of CPU resources are presented. The *under credit* of a virtual processor is periodically recharged according to its proportional share of the virtual processor, and it is consumed during non-boosted execution. The *boost credit* of the virtual processor is periodically recharged according to its credit rating, and it is consumed during boosted execution. The characteristics of the applications running on the virtual processor are reflected in the credit rating. The credit rating of each virtual processor is decided by how often it is boosted and how long the processor runs by boosting; the behavior of a virtual processor in boosted execution reflects the characteristics of the applications running on it. In addition, after a virtual processor consumes all of its *boost credits*, it has to repay the loan before receiving further *boost credits*. The repayment is proportional to the remaining *under credit* of the virtual processor, which means that less active virtual processors will have another chance to be boosted earlier than more active virtual processors.

To avoid the burden of distinguishing and tracking tasks on a virtual processor, we infer the task characteristics from their resource consumption patterns. Therefore, the proposed scheme uses a graded approach to reflect the dynamic behavior of a virtual machine instead of simply classifying I/O-intensive and CPU-intensive virtual machines. By changing the amount of *boost credit* to be given to a virtual processor with respect to its behavior, the chances of boosting are fairly distributed among virtual processors with different types of workloads.

The proposed scheme was implemented and evaluated in Xen hypervisor 4.6, and the results were compared with those of credit scheduler and a scheduler with aggressive boosting. The network response time, network bandwidth, and CPU utilization were evaluated experimentally. The experimental results demonstrate that the proposed scheme achieves a high I/O performance using credit scheduler with a low overhead in CPU utilization.

The rest of this paper is organized as follows. In Sect. 2, related works are introduced. In Sect. 3, a model of the virtual machine scheduler will be introduced. In Sect. 4, we describe the proposed scheme. In Sect. 5, the experimental results are discussed. Our conclusions are presented in Sect. 6.

2 Related works

2.1 Cloud computing

Cloud computing offers many benefits such as scalability, efficiency, on-demand services although there are security and privacy challenges to overcome (Zkik et al. 2017). It provides services to the customers at the lowest cost in their service level agreement between the cloud service provider and the cloud customer (Ratten 2015).

The low cost is due to the successful use of multilayer virtualization enabling dynamic elastic resource-sharing between different services (Hassan et al. 2019). The virtualization also enables cloud servers to provide a number of services simultaneously to cope with the different customer's demands (Manasrah et al. 2019). The scheduling in the virtualization plays a key role in the performance of cloud computing and user experience of the entire cloud service (Sadashiv and Kumar 2018).

Many investigations have been performed on improving the I/O performance of virtual machines on a VMM. In Chen et al. (2010), the overhead of the performance of a virtualized I/O on a VMM is analyzed and it is shown that the performance of latency-sensitive applications depends on the scheduling latency and I/O model of the VMM. In Cherkasova and Gardner (2005), a model to estimate the resource usage of applications in a virtualized environment is proposed.

The authors in Gordon et al. (2012) and Zeng et al. (2015) deal with optimization of I/O model in a VMM. In Gordon et al. (2012), the authors achieved high I /O performance by reducing context switching between a guest and host domain that are occurred by interrupts during I/O processes. XCollOpts presented in Zeng et al. (2015) focuses on the load balancing problem in assigning VCPU to PCPU and optimizing the virtualized I/O process for small packets to preventing premature preemptions.

2.2 I/O performance for virtual scheduling

Studies have been conducted to improve I/O performance by controlling the time slice of a latency-sensitive domain (Ding et al. 2014; Xu et al. 2012). Ding et al. (2014) proposed a dynamic time slice for credit scheduler in Xen hypervisor. They reported an increase in scheduling latency with an increase in the number of virtual processors. Because credit scheduler provides a fixed time slice regardless of the scheduling priority, the scheduling latency and I/O performance decrease in proportion to the number of virtual processors. Dynamic time slicing adjusts the amount of time slice depending on the number of virtual processors to ensure a predictable performance, regardless of the number of virtual processors. Xu et al. (2012) proposed a scheme to reduce the time slice of latency-sensitive domains; in addition, the scheme can run latency-sensitive domains much frequently than CPU-sensitive domains.

In addition to adjusting the time slice of virtual processors, some studies focused on controlling the priority of virtual processors to improve the I/O performance of virtual machines. Kim et al. (2008) proposed defining a priority that is higher than the boost priority to decrease the scheduling delay, while many virtual processors are executed with the boost priority at the same time. Yang et al. (2014) estimated the delay time in scheduling using a queueing theory and increased the priority of a virtual processor that is supposed to have long scheduling delays.

Little work has been done to provide an efficient solution for virtual machine scheduling with the mixed workloads, which are increasingly common. In the studies cited above, it was assumed that a virtual machine is either latency-sensitive or not, but not both. However, Kim et al. (2009) and Bai et al. (2010) suggested that a virtual machine can have composite workloads because multiple tasks can run simultaneously. In Kim et al. (2009), authors proposed that a domain can report the priority of a running task or a blocked task to a VMM. Because a blocked task has higher priority than an unblocked task in an operating system, a domain that has blocked tasks is preferably scheduled. In spite of the improvement in I/O performance due to task awareness, it is complicated and requires quite a large overhead for a VMM to be aware of the tasks running in guest operating systems, including inter-domain communications. However, we propose that a high I/O performance can be achieved by optimizing a virtual machine scheduler without additional overheads.

Our scheduler differs from these approaches in several fundamental ways. First. Our scheduler enables VM to adjust the boost frequency according to reflect the dynamic behavior of a virtual machine. Previous studies use a method of defining new priorities that is higher than the boost priority or controlling time slice. Second, we focus on the I/O performance of a virtual machine that runs both CPU-intensive and I/O-sensitive applications, which are increasingly common. However, most of the existing solutions to this problem assume that each virtual machine is either latency-sensitive or not. Finally, our scheduler uses an efficient solution without inter-domain communications.

3 Xen hypervisor

In this section, we present an overview of the Xen hypervisor. With respect to I/O performance, its I/O virtualization and scheduling algorithm features are described.

3.1 Isolated driver domain

The Isolated Driver Domain (IDD) model is used for virtualizing I/O interrupts in Xen hypervisor (Barham et al. 2003). Xen's paravirtualization includes a privileged virtual machine, $domain0$, which is in charge of I/O virtualization. Other untrusted virtual machines do not have direct permissions to get access to H/W resources. Untrusted virtual machines, $userdomains$, process their I/O requests through $domain0$ instead of processing the requests directly.

A $userdomain$ does not have real device drivers in this model. Instead, they use front-end drivers. Through

I/O Ring, which is a shared memory among domains, the front-end driver delivers I/O requests from *userdomains* to *domain*0's back-end driver. Afterward, the back-end driver delivers the I/O requests to the real device driver. In the opposite direction, when an I/O request is finished, back-end driver receives an I/O interrupt. Then it generates a virtual interrupt for the front-end driver of the associated virtual machine. When the virtual interrupt is delivered to the front-end driver, an event handler of the I/O event is operated.

Although a system can be protected from device driver faults in this driver model, the propagation of I/O events described above is not instantly processed. To process an I/O request when it is passed from a front-end driver in a *userdomain*, the *domain*0's back-end driver should wait until *domain*0 is scheduled by the virtual machine scheduler. In addition, the front-end driver receives the front-end driver's response only when the associated virtual machine is scheduled. Because *domain*0 does not have a higher priority than other virtual machines, scheduling latencies in processing I/O requests increase as the number of virtual machines in a VMM increases. In addition, scheduling latencies also increase in number as the number of boosts in a VMM increases; this is because the priority of a boosted virtual processor is higher than that of *domain*0 in credit scheduler; *domain*0 is more frequently preempted by a boosted virtual processor in this case. Thus, the I/O performance of a Xen VMM is highly dependent on the scheduling policy and number of scheduling entities.

3.2 Credit scheduler

The Xen hypervisor includes RTDS (Real Time Deferrable Server), ARINC 653, credit, and credit2 schedulers. RTDS is a soft real-time scheduler for multicore environments, and the ARINC 653 scheduler is hard real-time for single core environments. Credit and credit2 scheduler, on which our work is focused, is a general-purpose, proportional share system.

Credit scheduler operates based on a credit system. In a given time period, credits are given to all virtual machines residing in a VMM and each virtual machine consumes its credits while it is running. In the default configuration, the credits of a running virtual processor are reduced by 10 over every millisecond. In each time slice (30 ms is the default slice), credit scheduler determines the amount of credit that each virtual machine can be given using two parameters, *weight* and *cap*. The *weight* of a virtual machine refers to the proportional share of a physical processor that a virtual machine can be assigned. *cap* represents the upper bound of the utilization of a physical processor for a virtual machine, which is represented as a percentage.

A physical processor has a local run queue of virtual processors. In a run queue, runnable virtual processors are sorted by their scheduling priorities, which are BOOST, UNDER, and OVER (in the given order). In a run queue, when a virtual processor has to be inserted in, it is inserted at the end of other virtual processors of equal priority and in every scheduling decision, credit scheduler picks a virtual processor to run in a round-robin fashion. A virtual processor is assigned a priority of UNDER If its credit is positive. On the other hand, a virtual processor is assigned a priority of OVER if its credit is negative. If a virtual processor has UNDER priority, it implies that the virtual processor did not fully consume its physical processor share. In contrast, a virtual processor of OVER priority implies that CPU resource consumption of the virtual processor exceeds the amount allocated by scheduler.

In addition, if the credits of a virtual processor exceed a pre-defined threshold value or the virtual processor has no runnable tasks, the priority of the virtual processor becomes IDLE and the virtual processor is removed from the run queue after discarding all its credits. An idle virtual processor should receive an I/O event to be awakened. I/O-bound virtual processors are more likely to be idle because they consume less credit. As a result, virtual processors with CPU-intensive workloads are allocated more CPU resources than idle virtual processors that are less CPU-intensive, thus resulting in an inefficient use of CPU resources.

However, in this policy, I/O-bound virtual processors have a less chance of running. Therefore, the boost mechanism is applied to credit scheduler for the purpose of improving the responsiveness of I/O-bound virtual processors. When an idle virtual processor receives an event, it is awakened and its priority is temporally changed to BOOST. Because BOOST is the highest priority, the boosted virtual processors preempt currently running virtual processors of a lower priority. After the event is handled, the priority of the boosted virtual processors returns to UNDER. In addition to the boost mechanism, credit scheduler uses another global parameter called *ratelimit*, which is a specific time in which the virtual processor is not preempted; *ratelimit* helps latency-sensitive applications by preventing them from aggressively preempting each other.

4 Virtual machine scheduling based on task characteristics

As described in Sect. 3, the boost mechanism is useful in improving the I/O performance of virtual machines for latency-sensitive applications but arbitrarily changing virtual processor's priorities disrupts the fundamental of priority-based scheduling policy. To fairly assign the boosting chances of a virtual processor, the behavior of virtual machines with various task characteristics should be taken into account.

In this section, we discuss the behavior of virtual machines from the perspective of credit consumption and I/O fre-

quency. Based on the discussion, we design an evaluation function for a virtual machine based on its task characteristics. In addition, using the evaluation function, we propose a novel boost mechanism for credit scheduler, which controls the boost frequency.

4.1 Virtual machine behavior

The boost mechanism of credit scheduler divides the characteristics of a virtual machine into two categories, CPU-intensive and I/O-intensive. Only an I/O-intensive virtual processor, which is the main topic of many studies (Iosup et al. 2011; Cherkasova and Gardner 2005; Nagarajan et al. 2007), tends to be inactive and is boosted afterward.

For example, a virtual machine that runs network-based applications sleeps most of the time. The virtual processor of the virtual machine has tasks that check the incoming and outcoming packets. The virtual processor is often blocked by an I/O after running for relatively short times and is periodically awakened by the boost mechanism. As credit scheduler assigns credits to every virtual processor in each scheduling period and an I/O-intensive virtual processor does not consume all its credits while it is running, the remaining credits of the virtual processor are accumulated. When the credits exceed a threshold or all the tasks of the virtual machine are blocked, the machine becomes inactive and is removed from the run queue.

In summary, when remaining credit of a virtual processor increases by running I/O-intensive workloads or has no tasks to run, it has high possibility to be inactive. In contrast, in a CPU-intensive virtual processor, there exists a runnable task most of the time. The virtual processor is much less likely to be inactive when compared to I/O-intensive virtual processors. Compared to the virtual processor of I/O-intensive tasks, the virtual processor of CPU-intensive tasks consumes credit steadily and is less likely to be blocked by I/O. Thus, this processor is hardly boosted and cannot consume a fair share of the CPU resources when compared to I/O-intensive virtual processors in some cases.

However, even a CPU-intensive virtual processor can work on I/O-intensive tasks. For example, if a virtual machine runs CPU-intensive tasks and a few I/O-intensive tasks simultaneously, its virtual processor rarely becomes inactive. When an I/O-intensive task is running on a virtual processor, a fast response is required, but a steady credit consumption of the virtual processor prevents it from being boosted. Therefore, a virtual processor with composite workloads faces a lack of fairness in I/O-event handling in the current credit scheduler; the same problem was found in other research studies (Chen et al. 2010; Xu et al. 2012; Qu et al. 2015).

As a simple solution to this problem, a scheme was proposed to fairly boost virtual processors, regardless of their activeness. However, this solution is not efficient because a virtual machine of the boost priority cannot receive preference in scheduling decisions if most of virtual processors in the system are boosted.

The boost frequency should be controlled taking into consideration the task characteristics of the virtual processor to guarantee the fair and stable I/O performance for virtual machines. Therefore, in this study, we propose a graded boost mechanism that controls the boost frequency of virtual processors depending on their behavior in order to improve the I/O performance of active virtual machines. First of all, we categorize the task characteristics of virtual processors according to their behavior as inactive, CPU-intensive, I/O-intensive, and composite.

An inactive virtual processor is neither CPU-intensive nor I/O-intensive. An inactive virtual processor has the least possibility of monopolizing CPU resources because in most cases, it runs for a short time when it is boosted and then yields the CPU resource. A CPU-intensive virtual processor has the least need for boosting among the four categories. Because a CPU-intensive virtual processor has the least number of I/O events, the effect of boosting a CPU-intensive virtual processor on the I/O performance is not significant. An I/O-intensive virtual processor is the main target of the boost mechanism in credit scheduler. It requires frequent boosting for a high I/O responsiveness. A composite virtual processor is I/O-intensive as well as CPU-intensive. In spite of the benefit of the boost mechanism being lesser for a composite virtual processor when compared to an I/O-intensive virtual processor, the I/O responsiveness of the composite virtual processor can be improved by the boost mechanism. However, a composite virtual processor should be carefully boosted because CPU resources can be monopolized when composite virtual processors are boosted. Therefore, the frequency order is as follows-inactive, I/O-intensive, CPU-intensive, and composite virtual processors.

4.2 CPU-intensiveness and I/O-intensiveness

In this section, we describe a method to evaluate task characteristics using an additional credit and redeem mechanism. As shown in Fig. 1, the proposed scheme gives in a virtual processor with pending I/O events additional credits, named *boost credits*, before boosting the virtual processor. For each virtual processor, *under credit* and *boost credit* are managed separately. In contrast to *under credits* that are consumed during normal (non-boosted) execution, *boost credit* is deducted only when a virtual processor is executed by a scheduling decision of the boost mechanism. After a virtual processor has consumed all of its *boost credits*, the virtual processor enters a deduction phase.

In the deduction phase, boosting the virtual processor is not allowed until it repays all the *boost credits* originally

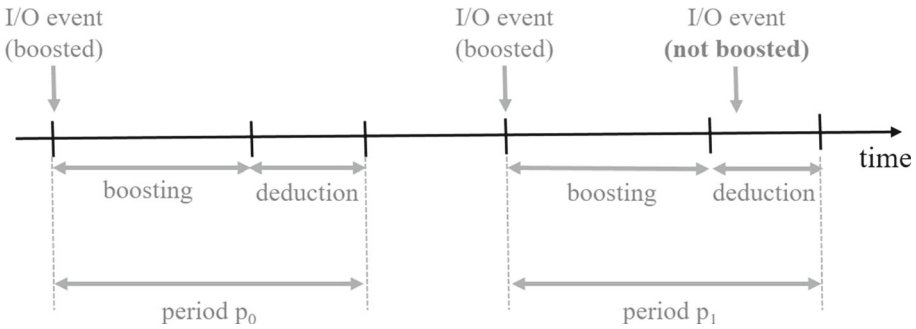

Fig. 1 Loan/deduction period

loaned to it. In addition, in the deduction phase, a portion of the *under credit* given to a virtual processor are paid to deduct *boost credit*, which were given to the virtual processor on loan.

Based on the mechanism described above, the pattern of credit consumption and deduction in a virtual processor is evaluated to measure the characteristics of its tasks. If a virtual processor consumes its *boost credit* quickly, it is considered CPU-intensive as well as I/O-intensive. If a virtual processor deducts its *boost credit* quickly, it is considered inactive. The proposed scheme evaluates the characteristics of a virtual processor and adjusts the *boost credit* that can be given to the virtual processor; therefore, the boost frequency is controlled according the task characteristics.

We evaluate the CPU-intensiveness and I/O-intensiveness of a virtual processor during the process described in Sect. 4.2. The evaluation process is described below:

4.2.1 Notations

Table 1 shows the notations used in this section.

Assume that n virtual processors, $V = \{v_1, v_2, \ldots, v_n\}$, exist in a system. For a virtual processor v_i, time interval p_j and r_j where p_j is from p_{j1} to p_{j2}, r_j is from r_{j1} to r_{j2}, $p_{j2} = r_{j1}$. In p_j, v_i is boosted m times: $B = \{b_1, b_2, \ldots, b_m\}$ where b_k runs from b_{k1} to b_{k2}. In r_j, v_i is normally executed l times: $R = \{r_1, r_2, \ldots, r_l\}$ where r_k runs from r_{k1} to r_{k2}.

4.2.2 Evaluation of task characteristics

When an I/O event of a virtual processor is pending, the virtual processor is given $CREDIT\_LINE$ *boost credit* and boosted. The *boost credit* is deducted during the boosted running time but not during the normal running time. From the time that the virtual processor is given *boost credit*, the time until all the *boost credit* is consumed, p, and the number of boosts for the virtual processor, n, are measured. The CPU-intensiveness of the virtual processor is evaluated using (1), and the I/O-intensiveness of the virtual processor is evaluated using (2).

Table 1 Notations

Notations	Descriptions
n	Number of virtual processors in the system
V	Set of virtual processors in the system
v_i	ith virtual processor in the system
p_j	jth time interval from p_{j1} to p_{j2}
r_j	jth time interval from r_{j1} to r_{j2}
m	Number of boosted executions during p_j
b_k	A boosted execution during p_j
l	Number of normal executions during r_j
R	Sequence of normal executions during r_j
r_k	A normal execution during r_j
$cb(v_i, p_j)$	CPU-intensiveness of v_i in p_j
$ib(v_i, p_j)$	I/O-intensiveness of v_i in p_j
$ap(v_i, p_j)$	Activeness of v_i in p_j
$C(v_i, p_j)$	Amount of *boost credit* to be given to v_i in p_j
α	Loan rate of v_i in p_j

$$cb(v_i, p_j) = \sum_{k=1}^{m} \frac{b_{k2} - b_{k1}}{m} \tag{1}$$

$$ib(v_i, p_j) = \frac{m}{p_{j2} - p_{j1}} \tag{2}$$

Here, cb indicates the *boost credit* consumed when the virtual processor is boosted. It also represents the CPU-intensiveness of the tasks while processing I/O events. Therefore, cb is used to evaluate the CPU-intensiveness of a virtual processor. Meanwhile, ib is the number of boosts received by the virtual processor during p. Because the number of boosts is equal to the number of I/O events that are delayed, ib represents the number of I/O events that occurred. Further, it represents that this virtual processor is I/O-intensive. Therefore, ib is used to evaluate the I/O-intensiveness of the virtual processor during p.

From (1) and (2), the characteristics of a virtual processor are represented by multiples of cb and ib in (3).

$$ap(v_i, p_j) = cb(v_i, p_j) \cdot ib(v_i, p_j) = \sum_{k=1}^{m} \frac{b_{k2} - b_{k1}}{m} \cdot \frac{m}{p_{j2} - p_{j1}} = \frac{\sum_{k=1}^{m} b_{k2} - b_{k1}}{p_{j2} - p_{j1}} \quad (3)$$

Here, ap represents the additional credits consumed by the virtual processor per millisecond. A virtual processor with a high ap means that the virtual processor is CPU-intensive and also I/O-intensive.

4.3 Application

In this section, we describe the proposed credit and redeem mechanism. The main idea of the proposed scheme is adapting the boosting frequency according to the task characteristics.

To adaptively control the boost frequency of a virtual processor according to its task characteristics, the *boost credit* given to a virtual processor is determined by ap in the previous period. As shown in Fig. 2, credit deduction starts after a virtual processor consumes all the *boost credits* it received. A virtual processor cannot be boosted until the entire *boost credit* is deducted. These credits are deducted in terms of amount of runtime of the task after it is scheduled normally. The remaining credits of CPU- and I/O-intensive tasks are lower, and the deduction time is longer than those of a task that is not CPU- and I/O-intensive.

$$C(v_i, p_j) = \alpha \cdot \frac{\sum_{i=1}^{n} ap(v_i, p_{j-1})}{ap(v_i, p_{j-1})} \quad (4)$$

A high ap of a task means that its characteristics are close to being CPU-intensive as well as I/O-intensive. Because these types of tasks can monopolize CPU resources by frequent boosting, the boost frequency of a task with a high ap is supposed to be low. Accordingly, the amount of *boost credit* given to a virtual processor and which should be deducted is inversely proportional to the cm in the previous period.

A task with a high ap in the previous period is given lower *boost credit* in the current period. In contrast, a task with a low ap in the previous period is given higher *boost credit* in the current period. Thus, the boost frequency of a virtual processor can be adaptively controlled according to task characteristics.

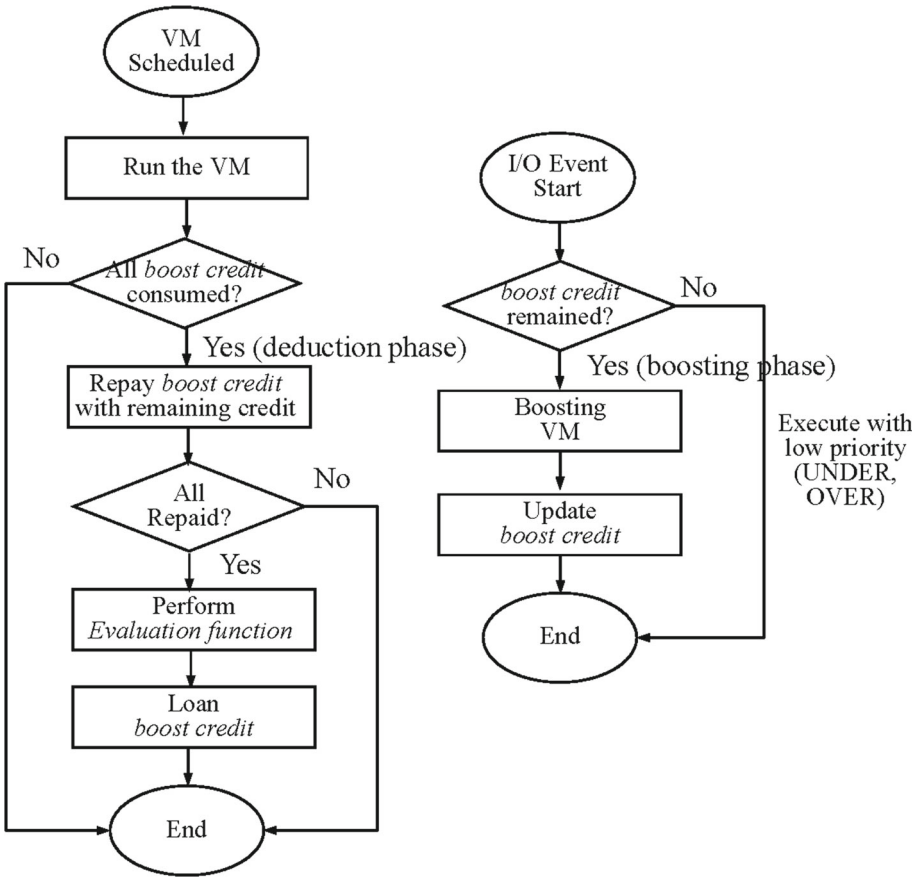

Fig. 2 Flowchart of our scheduler

4.4 Discussion

The proposed scheme applies the boost mechanism adaptively by taking into account the task characteristics in the past and the current period. The past and current characteristics of a task are integrated in ap. The current amount of the $boost\ credit$ of a task is determined by the past characteristics of the task.

The deduction of a task starts after it is scheduled normally because the current characteristics of a task can be reflected in the boost mechanism. Despite the past characteristics of a task being closer to composite, it can be boosted by fast deduction if its current characteristics are closer to inactive. In contrast, when the past characteristics of a task are closer to inactive, the task is given a large number of $boost\ credits$; it cannot receive more $boost\ credits$ if its current characteristics are closer to composite because its deduction is slow.

5 Experiments

This section presents the experimental environment and the obtained results to evaluate the performance of the proposed virtual machine scheduling scheme. To clearly show the difference between the proposed scheme and the existing scheduler, tests were conducted in an environment with multiple virtual machines performing CPU-intensive workloads, resulting in relatively long scheduling delays.

5.1 Experimental environments

The experimental environments are shown in Table 2. The server and client are physically separated to remove their influence on performance measurements.

To evaluate the network response time of the proposed scheme, we measured the network bandwidth and CPU utilization of the previous credit scheduler and the proposed scheme with multiple domains executing both CPU-intensive and I/O-intensive tasks. In Table 3, all the domains except $domain0$ are always active and execute CPU-intensive tasks. The number of virtual machines is set to 12 to extend the virtual machine scheduling delay. The proposed scheme affects I/O performance, regardless of disk I/O and network I/O because it accelerates the processing of I/O events by reducing scheduling delay. Thus, only network I/O is measured in our experiments. Four utilities, $iperf$, $hping$, $xentop$, and $stress$ are used to measure the network I/O performance (Table 4).

Table 2 Experimental environments

	Component	Description
Server	CPU	Intel Celeron 847 (Dual core 1.1 GHz)
	Memory	DDR3 8 GB
	Storage	SSD 256 GB
	Xen version	Xen 4.6 (paravirtualization)
	Host kernel	3.16
	Guest kernel	3.13
Client	CPU	AMD G-T48E 1.4 GHz
	Memory	DDR3 2 GB
	Operating system	CentOS 5.7
Network	Ethernet	100 Mbps

Table 3 Domains for measuring I/O performance

Domain id	Task	Domain	Description
Dom 0	Inactive	1	IDD
Dom 1-12	Composite	1	Test domain

Table 4 Utilities

Utility	Usage
iperf	Network bandwidth between server and client
xentop	CPU utilization of domains
hping	Network response between sever and client
stress	CPU, I/O workload generation

Table 5 Average response time

	Credit scheduler	Proposed scheduler
Response time	37.2 ms	23 ms

5.2 I/O performance

To evaluate the I/O performance of the proposed scheme, the network response time, network bandwidth, and CPU utilization of the credit scheduler and the proposed scheme are measured. The $CREDIT\_LINE$ and loan rate are set to 1000 and 20%, respectively.

The average response time of the credit scheduler and the proposed scheme is calculated to be 37.2 ms and 23 ms, respectively, which indicates an improvement of 39% in the network response time with the proposed scheduler as shown in Table 5. The cumulative distribution function of the network response time is shown in Fig. 3. When the previous scheme was employed, the response time of 35% of the data was less than 10 ms, while that of 41% and 58% of the data was less than 20 ms and 10 ms, respectively. Meanwhile, the response time of 70% of the data was less than 20 ms with the proposed scheme.

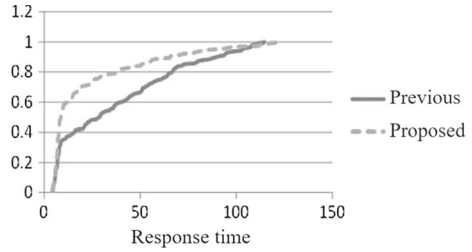

Fig. 3 TCP/IP response of the proposed scheme

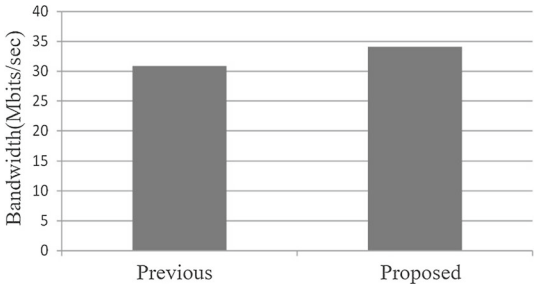

Fig. 4 TCP/IP bandwidth of the proposed scheme

Figure 4 shows the network bandwidths when the credit and the proposed scheduler were used. The average network bandwidth of the credit scheduler was 30.9 Mbps, while that of the proposed scheme was 34.1 Mbps, which indicates that the network bandwidth increased by 10% with the proposed scheme. From Figs. 3 and 4, it can be inferred that the proposed scheme resulted in a better performance than the credit scheduler in terms of response time and bandwidth by reducing scheduling delay. Domains containing both CPU-intensive tasks and I/O-intensive tasks run concurrently in a virtual environment; in the previous scheduling scheme, the latency-sensitiveness of domains is ignored because they intensively consume their credits. However, with the proposed scheme, the domains have a fair chance of being boosted according to their individual I/O-intensiveness. The difference results in a performance gap between the previous scheme and the proposed scheme in the execution environment, as shown in Figs. 3 and 4.

Figures 5, 6 and 7 show the CPU utilization of 12 domains with the credit scheduler, the proposed scheme, and credit scheduler with unconditional boosting, respectively. In the case of unconditional boosting, the scheduler boosts a virtual processor with pending I/O events, regardless of its priority and credits. In our experiments, it is observed that the average CPU utilization of the unconditional boosting scheduler is the lowest; however, in Fig. 7, significantly unstable results could be observed. This scheme can yield high responsiveness with low CPU utilization for some of the domains. However, the lack of fairness in this scheme causes a critical CPU monopolization problem. In contrast, the average CPU utilization

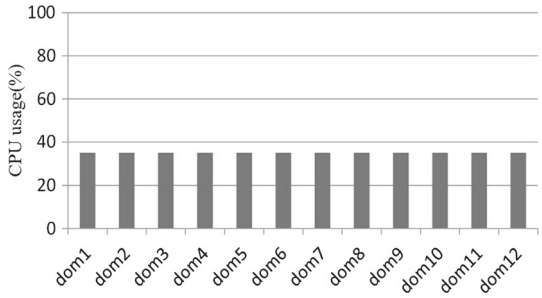

Fig. 5 CPU utilization of the credit scheduler

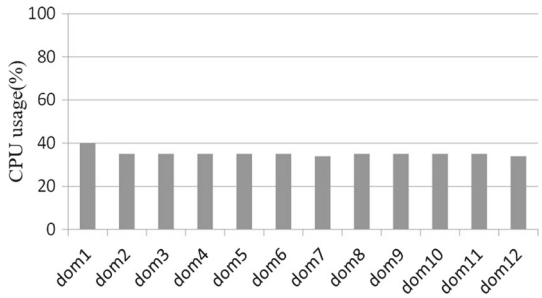

Fig. 6 CPU utilization of the proposed scheme

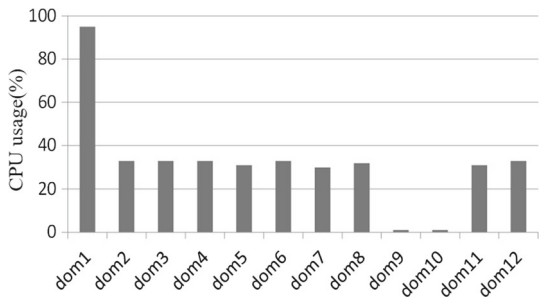

Fig. 7 CPU utilization of unconditional boosting

of the 12 domains was similar and stable in the proposed and previous schemes. This means that the proposed scheme does not hinder CPU resource management and results in a higher I/O performance when compared to the previous scheme by minimizing the negative effects caused by CPU monopolization.

5.3 Boost frequency

In addition to measuring the I/O performance of virtual machines, we measured the boost frequency of two domains executing different workloads using the proposed scheme. Relatively, the first domain is CPU-intensive and the second is I/O-intensive, but they execute identical I/O jobs. As the proposed scheme controls *boost credit* to control the boost frequency of domains, the value of $CREDIT\_LINE$ can represent boost frequency. In this experiment, boost credit

Table 6 Boost frequency of the proposed scheme

Task	*Stress* args.	DomU	credit_boost	cm
I/O-intensive	-i 1 -t 60	4	600	5
CPU-intensive	-c 1 -i 1 -t 60	4	375	9.2

and cm are measured in different task environments using stress. The experimental setup and results are presented in 6.

In Table 6, it is shown that the domains with same I/O jobs and different CPU-intensiveness and I/O-intensiveness have different *boost credit* and *ap* values. A low CPU-intensiveness results in a low *ap*, which means *boost credit* consumption per millisecond and a high *boost credit*, which represents the *boost credit* that a task can rent. In contrast, a high *ap* results in a low *boost credit*. Unlike the previous scheduling scheme, which does not consider task characteristics within domains, the proposed scheme fairly distributes I/O resources among domains with various workloads and even a CPU-intensive domain can perform I/O tasks with sufficiently low latency because the proposed scheme manages the boost frequencies of multiple domains according to the characteristics of tasks executed in each domain.

6 Conclusion

We presented a novel virtual CPU scheduler to improve the I/O performance of a virtual machine scheduler for cloud systems. In order to fairly assign the boosting chances of a virtual machines, we use a proportionally differentiated approach to adopt the boosting frequency according to the task characteristics. Our scheduler analyzes the task characteristics of a virtual machine and gives it boost credits which is consumed while boosted executions. It also controls the deduction time of boost credits in consideration of a virtual machine's behavior. The proposed scheme was implemented and evaluated in Xen hypervisor. We measured the I/O performance and CPU share of virtual machines using the proposed scheme and the existing virtual machine schedulers. The experimental results show that the proposed scheme can achieve up to 39% less response time compared to an existing scheme.

A next step in continuing the work is to study a dynamic solution for large-scale cloud system that often have a large number of virtual machines. As part of future work, we plan to analyze the effect of parameters and further study on additional dynamic adaptation techniques in a large-scale environment.

Acknowledgements This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A3B 03931258). The authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Compliance with ethical standards

Conflict of interest No potential conflict of interest was reported by the authors.

References

Bai Y, Xu C, Li Z (2010) Task-aware based co-scheduling for virtual machine system. In: Proceedings of the 2010 ACM symposium on applied computing, SAC 2010, Sierre, Switzerland, March 22–26, 2010. ACM, pp 181–188

Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: Proceedings of the nineteenth ACM symposium on operating systems principles, SOSP 2003, Bolton Landing, NY, USA, October 19–22, 2003. ACM, pp 164–177

Beloglazov A, Buyya R (2010) Energy efficient resource management in virtualized cloud data centers. In: Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing, CCGRID 2010, Washington, DC, USA, May 17–20, 2010. IEEE Computer Society, pp 826–831

Chen H, Jin H, Hu K, Yuan M (2010) Adaptive Audio-aware Scheduling in Xen virtual environment. In: Proceedings of ACS/IEEE international conference on computer systems and applications, AICCSA 2010, Tunisia, May 16–19, 2010. IEEE, pp 1–8

Cherkasova L, Gardner R (2005) Measuring CPU Overhead for I/O processing in the Xen virtual machine monitor. In: Proceedings of the 2005 USENIX annual technical conference, Anaheim, CA, April 10–15, 2005, USENIX, pp 387–390

Ding X, Ma Z, Da X (2014) Dynamic time slice of credit scheduler, In: Proceedings of IEEE international conference on information and automation, ICIA 2014, Hailar, China, July 28–30, 2014. IEEE, pp 654–659

Gordon A, Amit N, Har'El N, Ben-Yehuda M, Landau A, Schuster A, Tsafrir D (2012) ELI: bare-metal performance for I/O virtualization. In Proceedings of the seventeenth international conference on architectural support for programming languages and operating systems, ASPLOS XVII, London, England, UK, March 03–07, 2012. ACM, pp 411–422

Hassan HA, Kashkoush MS, Azab M, Sheta WM (2019) Impact of using multi-levels of parallelism on HPC applications performance hosted on Azure cloud computing. Int J High Perform Comput Netw 13(3):251–260

Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T, Epema D (2011) Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Trans Parallel Distrib Syst 22(6):931–948

Jain R, Paul S (2013) Network virtualization and software defined networking for cloud computing: a survey. IEEE Commun Mag 51(11):24–31

Kim D, Kim H, Jeon M, Seo E, Lee J (2008) Guest-aware priority-based virtual machine scheduling for highly consolidated server. In: Proceedings of European conference on parallel processing, Euro-Par 2008, Las Palmas de Gran Canaria, Spain, August 25–29, 2008. Springer, 2008, pp Canary Island, Spain, pp 285–294

Kim H, Lim H, Jeong J, Jo H, Lee J (2009) Task-aware virtual machine scheduling for I/O performance. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE 2009, Washington, DC, USA, March 11–13, 2009. ACM, pp 101–110

Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) KVM: the Linux virtual machine monitor. In: Proceedings of the linux symposium, pp 225–230

Manasrah AM, Aldomi A, Gupta BB (2019) An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment. Cluster Comput 22(1):1639–1653

Moreno-Vozmediano RM, Montero RS, Llorente IM (2012) IaaS cloud architecture: from virtualized datacenters to federated cloud infrastructures. Computer 45(12):65–72

Nagarajan AB, Mueller F, Engelmann C, Scott SL (2007) Proactive fault tolerance for HPC with Xen virtualization. In: Proceedings of the 21st annual international conference on supercomputing, ICS 2007, Seattle, Washington, DC, USA, June 17–21, 2007. ACM, pp 23–32

Ongaro D, Cox AL, Rixner S (2008) Scheduling I/O in virtual machine monitors. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on virtual execution environments, VEE 2008, Seattle, WA, USA, March 05–07, 2008. ACM, pp 1–10

Qu H, Liu X, Xu H (2015) A workload-aware resources scheduling method for virtual machine. Int J Grid Distrib Comput 8(1):247–258

Ratten V (2015) Cloud computing technology innovation advances: a set of research propositions. Int J Cloud Appl Comput (IJCAC) 5(1):69–76

Sadashiv N, Kumar SM Dilip (2018) Broker-based resource management in dynamic multi-cloud environment. Int J High Perform Comput Netw 12(1):94–109

Velte A, Velte T (2010) Microsoft virtualization with hyper-V. McGraw-Hill, New York

Watson J (2008) VirtualBox: bits and bytes masquerading as machines. Linux J 166:2008

Williams D, Jamjoom H, Weatherspoon H (2012) The Xen-Blanket: virtualize once, run everywhere. In: Proceedings of the 7th ACM European conference on computer systems, EuroSys 2012, Bern, Switzerland, April 10–13, 2012. ACM, pp 113–126

Xi S, Wilson J, Lu C, Gill C (2011) RT-Xen: towards real-time hypervisor scheduling in xen. In: Proceedings of the ninth ACM international conference on embedded software, EMSOFT 2011, Taipei, Taiwan, October 09–14, 2011. ACM, pp 39–48

Xu C, Gamage S, Rao PN, Kangarlou A, Kompella RR, Xu D (2012) vSlicer: latency-aware virtual machine scheduling via differentiated-frequency CPU slicing. In: Proceedings of the 21st international symposium on high-performance parallel and distributed computing, HPDC 2012, Delft, The Netherlands, June 18–22, 2012. ACM, pp 3–12

Yang C, Liu J, Huang K, Jiang F (2014) A method for managing green power of a virtual machine cluster in cloud. Future Gen Comput Syst 37:26–36

Zeng L, Wang Y, Feng D, Kent KB (2015) XCollOpts: a novel improvement of network virtualizations in Xen for I/O-latency sensitive applications on multicores. IEEE Trans Netw Serv Manag 12(2):163–175

Zkik K, Orhanou G, Hajji S (2017) Secure mobile multi cloud architecture for authentication and data storage. Int J Cloud Appl Comput (IJCAC) 7(2):62–76