Workload-Aware Credit Scheduler for Improving Network I/O Performance in Virtualization Environment

Haibing Guan, *Member, IEEE*, Ruhui Ma, and Jian Li, *Member, IEEE*

Abstract—Single-root I/O virtualization (SR-IOV) has become the de facto standard of network virtualization in cloud infrastructure. Owing to the high interrupt frequency and heavy cost per interrupt in high-speed network virtualization, the performance of network virtualization is closely correlated to the computing resource allocation policy in Virtual Machine Manager (VMM). Therefore, more sophisticated methods are needed to process irregularity and the high frequency of network interrupts in high-speed network virtualization environment. However, the I/O-intensive and CPU-intensive applications in virtual machines are treated in the same manner since application attributes are transparent to the scheduler in hypervisor, and this unawareness of workload makes virtual systems unable to take full advantage of high performance networks. In this paper, we discuss the SR-IOV networking solution and show by experiment that the current credit scheduler in Xen does not utilize high performance networks efficiently. Hence we propose a novel workload-aware scheduling model with two optimizations to eliminate the bottleneck caused by scheduler. In this model, guest domains are divided into I/O-intensive domains and CPU-intensive domains according to their monitored behaviour. I/O-intensive domains can obtain extra credits that CPU-intensive domains are willing to share. In addition, the total number of credits available is adjusted to accelerate the I/O responsiveness. Our experimental evaluations show that the new scheduling models improve bandwidth and reduce response time, by keeping the fairness between I/O-intensive and CPU-intensive domains. This enables virtualization infrastructure to provide cloud computing services more efficiently and predictably.

Index Terms—Cloud computing, I/O virtualization, Xen, SR-IOV, hypervisor, scheduling

1 Introduction

With the rapid development of cloud computing, service providers must satisfy users various requirements with improved business agility. Virtualization is one effective way of building cloud infrastructure for enterprises by utilizing their existing IT investments efficiently [1], [2], [3], [4], [5]. Moreover, it can increase the available resources and the flexibility of their management; consolidate diverse operating systems on one physical machine to the use of disparate hardware, allow IT staff to deliver a better service at lower cost and leverage a broad ecosystems with greater security and reliability.

Not only computing resources, but also network facilities are virtualized by the Virtual Machine Monitor (VMM), which allows multiple guest operating systems to access the network concurrently. The virtual network processing involves multiple trap-and-emulations, which make network I/O virtualization a heavy consumer of computing resource [6], [7], [8], especially for high-speed networks such as a 10 Gbps network connection. Thus, network I/O processing correlates highly with the resource allocation policy of VMM scheduler, owing to the heavy cost of network I/O processing. In a compelling cloud infrastructure, VMM should enable network performance of VMs to make full use of physical line rates. Furthermore, the virtual machine systems with different I/O or computing workload attributes should be scheduled differently with workload attribute awareness.

A lot of research has been done on ways to make virtualization a desirable choice for network-intensive applications [2]. The Virtual Machine Device Queue (VMDq) is a component of Intel virtualization technology for connectivity [9]. VMDq classifies packets by network adapter, which reduces the burden on the VMM [10]. However, it still needs VMM intervention for memory protection and address translation. Direct I/O is a technique that allows VMs to access hardware directly, without VMM intervention [11]. However, a network device is assigned to a particular VM and cannot be shared by other VMs. Single Root I/O Virtualization (SR-IOV) [12] is a novel solution for network virtualization, which has become a de facto standard for cloud computing infrastructure. In this model, packets are sent from Network Interface Card (NIC) to the VM without the traditional data copy in the para-virtualized driver. Meanwhile, an SR-IOV-capable device can create multiple virtual functions (VFs), and each VF can be assigned to one VM, while all VMs still share the physical resource [12].

- *H. Guan and R. Ma are with the Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {hbguan, ruhuima}@sjtu.edu.cn.*
- *J. Li is with the School of Software, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: li-jian@sjtu.edu.cn.*

Manuscript received 15 Sept. 2013; revised 26 Feb. 2014; accepted 24 Mar. 2014. Date of publication 1 Apr. 2014; date of current version 30 July 2014.
Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2314649

SR-IOV offloads almost all the driver domain's overhead to NIC hardware and achieves high I/O performance [12], [13], [14]. However, current VMM schedules I/O-intensive domains, as well as CPU-intensive domains, simultaneously and equivalently, and as a result, they can only achieve poor throughput with high latency, because the scheduling is less than optimal. We analyse two major factors that affect the I/O performance. First, an I/O-intensive domain may not get enough CPU resources in the presence of a high-performance network that costs a lot to process a large number of packets. Second, an I/O-intensive domain may not be scheduled in time. The default credit scheduler allocates credits with a fixed time slice, which is 30 ms. If the I/O-intensive domain happens to be inactive when the scheduler allocates credits, all the credits are allocated to another active domain, and network performance may be seriously affected.

The main contributions of this work are summarized as follows:

First, we analyse the network performance bottlenecks, using a high system workload on the Xen VMM platform. Experimental results demonstrate that one of defects is that an I/O-intensive domain normally lacks credits to perform I/O. The other is that an I/O-intensive domain is more likely to be inactive at credit allocation time, and may wait a relatively long time to handle the inputs. As tested, bottlenecks in the network virtualization framework degrade application performance, which motivate our further development and implementation of network virtualization.

Second, we propose a workload-aware network virtualization model to monitor virtual machine behaviour, and propose two new optimizations based on it: *Shared Scheduling* and *Agile Credit Allocation*. Since an I/O-intensive domain often lacks credit in the presence of network traffic bursts, our *shared scheduling* provides extra credits that are shared by CPU-intensive domains to I/O-intensive domains. This approach ensures that I/O-intensive domains acquire adequate resources. *Agile Credit Allocation* adjusts the total credits adaptively, according to the number of I/O-intensive domains, which shortens the waiting time for an I/O-intensive domain when it is removed from the active list during scheduling.

Finally, we implemented the proposed model and optimization, obtained extensive performance data to validate and quantify the benefits of proposed approaches on actual Xen virtualized systems. Note that our proposed shared scheduling and allocation schemes are generic enough, and can extend to all VMMs with proportional shared CPU scheduling algorithms, such as Vmware ESXi [15] server and KVM [16], etc.

The rest of this paper is organized as follows. We first introduce Xen's credit scheduler, I/O model and the SR-IOV model in Section 2. Following this, in Section 3, we analyse the network performance under heavy workload. Section 4 presents the design of our workload-aware credit scheduling model, and describes the two optimizations in detail. In Section 5, we evaluate the performance of our solutions with several testing scenarios. Section 6 introduces related work about I/O scheduling, and we give our conclusions in Section 7.

2 Background

2.1 Credit-Based Scheduler

The default scheduler in the current version of the Xen VMM is the credit scheduler. The problem is to determine which virtual CPU (*v*CPU) should run on each physical CPU (*p*CPU), where a *p*CPU is a real CPU in a machine and a *v*CPU is a simulated CPU in a VM. Each processor core can provide 300 credits in each scheduling slice (30 ms); the scheduler allocates these credits to domains according to each domain's weight. There are three *v*CPU statuses for a domain: *under*, *over* and *boost*. *under* indicates that a *v*CPU still has credits to use, while *over* means that the *v*CPU has exhausted its credits. A *v*CPU which has *boost* status has the highest priority; when an event is sent to an idle *v*CPU, this *v*CPU will be awakened, and if its priority is *boost*, it will preempt the *v*CPU which is running to achieve low I/O response latency. A run queue maintains *v*CPUs with *boost* on top, followed by *v*CPUs with *under*, while *v*CPUs with *over* are last. The scheduler picks a *v*CPU at the head of the run queue as the next *v*CPU, and *v*CPUs that have the same priority will be scheduled in First In First Out (FIFO) mode. When a *v*CPU has enough credit, it is allowed to run for three tick periods (30 ms), the scheduler debits 100 credits every 10 ms, and credits are redistributed every 30 ms. When it comes to multi-core architecture, if there is no *v*CPU with *under* in a *p*CPU's run queue, it will steal a runnable *v*CPU from another *p*CPU. This load-balancing mechanism guarantees that no *p*CPU is idle when there is runnable *v*CPU in the system. The priorities are implemented as different regions of the same queue, with one queue of task per physical processor. The front part of the queue is *boost* priority, the next part is *under* priority, followed by *over* priority. When a task enters the run queue, it is simply inserted into the end of the corresponding region of the queue, and the scheduler picks up the task from the head of the queue to execute.

2.2 I/O Virtualization Model of Xen

In order to guarantee the safety of the system, the Xen virtual machine makes use of Isolated Driver Domains (IDD) to perform I/O operations, while the user domains (domainU)use virtual device drivers for transparent I/O access. In this model, a device driver is divided into three parts: frontend driver; backend driver; and native driver. The frontend driver is included in domainU; the backend driver and native driver run in domain0. The frontend driver transfers domainU's I/O requests to the backend driver, which interprets the request, and maps it into a physical device. Then the native device driver controls hardware to finish the I/O operation. The I/O requests between the frontend and backend drivers are passed through an I/O ring, implemented in memory shared between domainU and domain0. The I/O ring just contains the descriptive information for I/O requests, and I/O data is not put in the I/O ring.

A VM transmits packets as follows: the frontend driver writes request information into the I/O ring through a hypercall; the backend driver handles the I/O request when domain0 is scheduled. It forwards the I/O request to the native driver, and after I/O access is completed by hardware, the backend sends a response to the frontend to finish

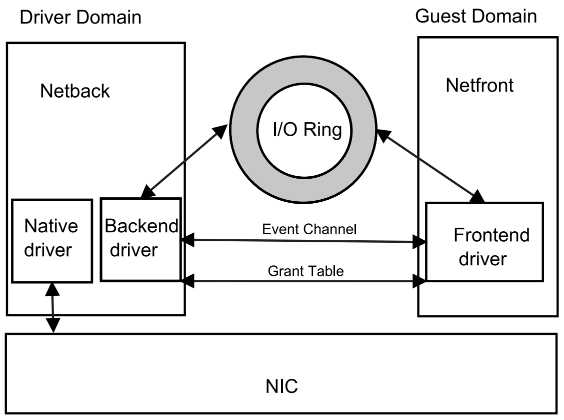


Fig. 1. Xen I/O virtualization architecture.

the I/O operation. A VM receives packets using the opposite process of transmission. Xen provides a grant table to deliver DMA data effectively between domainU and domain0. Each domain maintains a grant table to indicate which pages can be accessed by which domains, while Xen has an active grant table to record the content of each domain's grant table. When domainU needs to perform DMA, the frontend driver puts a grant reference (GR) and an I/O request into the I/O ring. After domain0 receives the request, it asks Xen to lock the page according to the GR, Xen checks the grant reference and responds to domain0. Then domain0 sends a DMA request to the actual hardware. The architecture of Xen is shown in Fig. 1.

2.3 Single-Root I/O Virtualization

Traditional I/O virtualization, such as VMDq or direct I/O, does not give high I/O performance while supporting device sharing at the same time. SR-IOV is a novel solution to this problem, which improves the performance of data transmission by eliminating the VMM intervention. An SR-IOV-capable device is a PCIe device, which has one or more physical functions (PF). A PF is a complete PCIe device that manages and configures all the I/O resources. A Virtual function is derived from a PF, which is a simplified PCIe device to implement I/O, but can not manage a physical NIC. The PF driver runs in domain0 to manage the PF, which can access all PF resources directly, to configure and manage VFs. The VF driver runs in a guest OS as a normal PCIe device driver, and accesses its corresponding VF directly. SR-IOV technology eliminates the bottleneck from the driver domain by avoiding an extra data copy, so as to greatly improve the throughput of the VM with lower CPU utilization. The SR-IOV virtualization architecture is shown in Fig. 2. SR-IOV offloads the network I/O processing from the hypervisor to specialized hardware, and device drivers VF interrupt is left as the major overhead that is still intervened by VMM, to remap it from host to guest [12].

3 Problem Illustration by Experiment

Generally speaking, timely processing of I/O tasks is necessary for a better user experience. However, VMM is not aware of the behaviour of guest domains because of the scheduling hierarchical architecture, so the credit scheduler schedules *v*CPUs equally, according to their weights. In

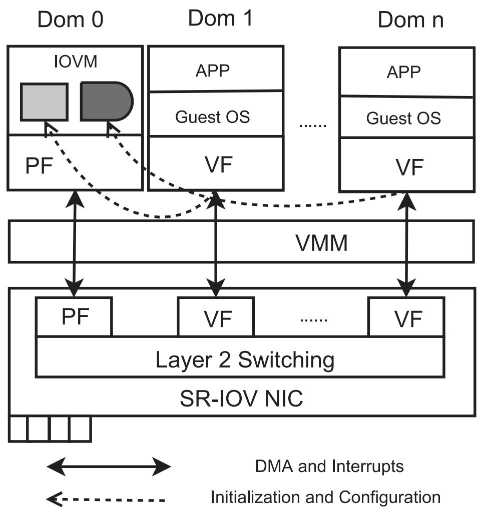


Fig. 2. SR-IOV architecture.

other words, I/O tasks do not get any special care from the scheduler, unless that VMM is able to predict the behaviour of guest domains [17], [18]. The system's computing completion time for virtual network processing affects I/O performance significantly in virtualized environment, so there is potential to improve it. We set up an experiment to analyse the performance of a high-speed network with high workload. The detailed hardware configuration of the experimental platform is described in Section 5.

3.1 Inadequate Credit

A high-performance network is different from a low-speed network, especially in virtualized environment. Specifically, a low-speed network does not require too much use of CPU resources, and I/O events just need a timely response. However, in a high-performance network, such as 10-Gigabit Ethernet, CPU resource is another important factor in network performance because it has to process lots of packets in a short time. Under the SR-IOV architecture, packets are transmitted from NIC to VM directly, without the extra data copy in domain0; the VMM just needs to handle some interrupts whose overhead is mitigated, and domain0 is no longer a bottleneck for network I/O.

Fig. 3 shows network performance change when the number of CPU-intensive domains becomes higher and higher on our testbed with quad-core as detailed in Section 5.1. In this experiment, only one VM is used as a server to receive packets from a remote native Linux, and the Netperf benchmark is used. Other VMs execute the CPU-intensive tasks, which are launched one by one to illustrate the performance change. An infinite loop is used here to burn credits.

Fig. 3a shows that when there is only one domain performing network I/O, it can achieve $\sim$5.5;Gbps. It cannot achieve higher bandwidth because only one VF is configured to the I/O-intensive domain, and it could not use more physical cores for I/O. When there are another one or two guests running CPU-intensive applications, the network performance is not much affected, because CPU-intensive domains use other idle physical cores without harming the I/O-intensive domain. However, when there are more than three domains running CPU-intensive

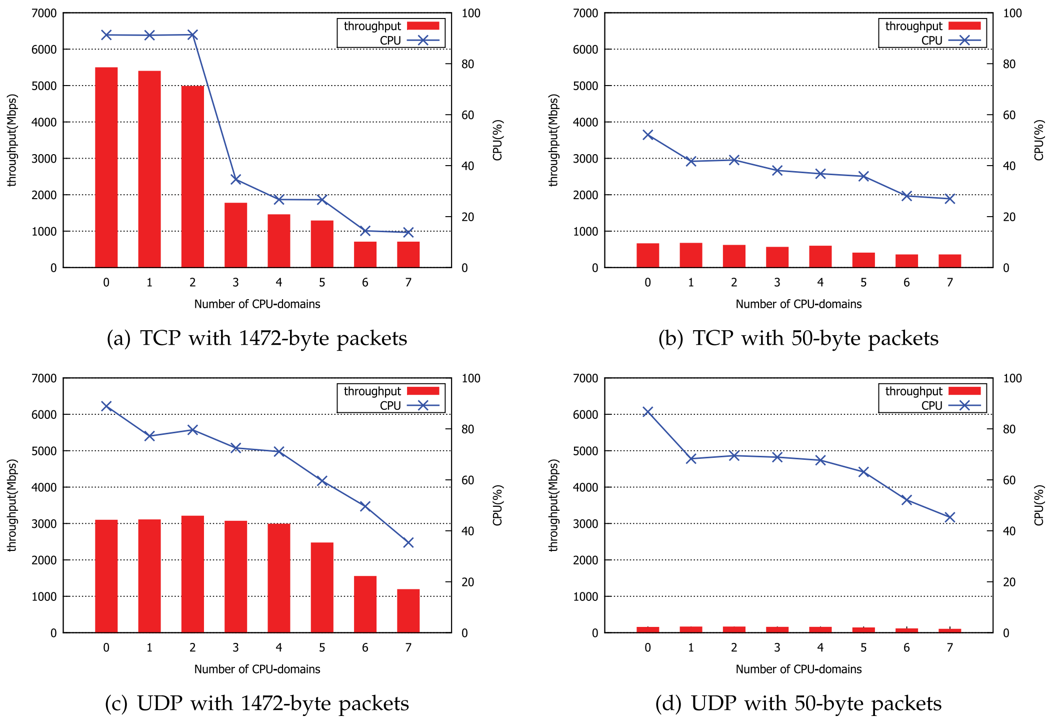


Fig. 3. I/O performance with heavy load.

applications, network I/O performance declines significantly, because the I/O-intensive domain has to compete its CPU resource with CPU-intensive domains and domain0. Comparing Figs. 3a and 3c, the TCP test shows better performance with 0-2 CPU-intensive domains. This is mainly because TCP is a reliable protocol with congestion avoidance and flow control algorithms to control the transmission rate intelligently. While, UDP protocol stack is as not well controlled as the TCP protocol stack, and when the network is congested Ethernet flow control mechanism in Data Link layer is triggered to stop the transmission. This is the reason why UDP achieves lower throughput than TCP. On the other hand, since UDP is more light-weight, UDP throughput drops only when there are more than four CPU-intensive domains, as shown in Fig. 3c, and the throughput performance of UDP degrades slower than TCP, as shown in Fig. 3a.

In the test cases with small packets, the throughput is low, and drops slightly, as shown in Figs. 3b and 3d. That is because there are more arrival packets owing to the small packet size, and the software stack has not enough CPU resources to process. In this case, lots of packets are dropped. As a reliable protocol, TCP also outperforms UDP. In summary, with enough CPU resources, an I/O-intensive domain has sufficient processing capacity to perform network I/O; otherwise, network performance degrades significantly. It is illustrated that the more hosted domains with CPU-intensive workload competing resources, the lower the throughput achieved by the I/O-intensive domain.

3.2 Long Waiting Time

The credit scheduler maintains a list of active domains. The scheduler allocates credits to active domains every time-slice of 30 milliseconds. If an I/O-intensive domain happens to be inactive (no arrival and suspending I/O events)when the scheduler distributes credits, all available resources are distributed to active domains. Without credits, the I/O-intensive domain has to wait until the next scheduling time-slice. Therefore, this waiting time can achieve up to 30 ms. Note that shortening the time-slice will incur a higher overhead from the context switch among VMs, and hurt the whole system performance (especially unfair to CPU-intensive domains).

We also conduct another experiment to measure the VM's scheduling frequency for I/O and CPU operations. Scheduling here means that a domain is active and assigned a credit value when the scheduler distributes credits. The experiment runs eight VMs concurrently with the same weight, where only one VM performs I/O operations, and the other VMs perform CPU-intensive tasks. We measure the number of times that each domain was assigned credits in a three-minute interval (see Fig. 4). Seeing that the I/O-intensive domain is scheduled with less times than the

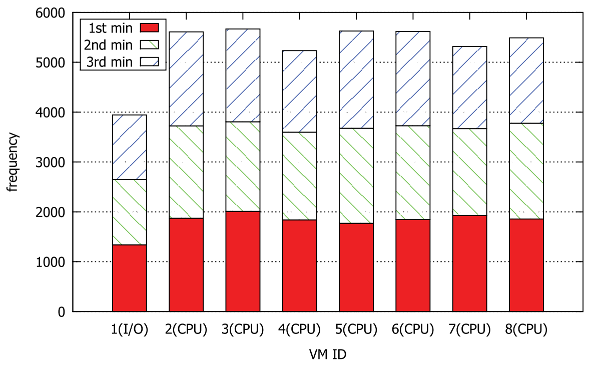


Fig. 4. Scheduling frequency with different types of load.

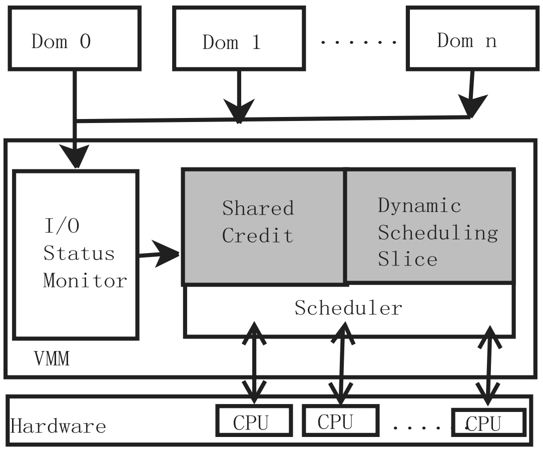


Fig. 5. Scheme of scheduling strategy.

CPU-intensive domains in every minute, and with the growth of testing time, the gap increases more linearly, which means the I/O-intensive domain is more likely to be removed from the active list.

We summarize the major factors for improving performance of a high-speed network with SR-IOV into two points: 1)enough credits should be provided so that the I/O-intensive domain is able to process large number of packets, when there are I/O-intensive applications; and 2) the number of available credits should be adaptable so that I/O events can be responded to in time.

4 Workload-Aware Credit Scheduler

In this section, an extension to credit scheduler for enhancing the I/O performance will be discussed. The scheduling strategy scheme is shown in Fig. 5, which consists of three modules: *I/O status monitor*; *Shared Credit (SC)*; and *Dynamic Scheduling Slice*.

4.1 I/O Status Monitor

Every time a VM handles a packet, its VF driver invokes the *ixgbevf_clean_tx_irq* or *ixgbevf_clean_rx_irq* function. We count the execution number of these two functions for every VCPU running the workload tasks. Compared with more than 1,000 times a second for I/O tasks, the number for CPU tasks is less than 100, so we choose a threshold value of 500 a second in consideration of practical application attributes. Once the number of I/O events exceeds the threshold, the VM is marked as an I/O-intensive domain, otherwise, the VM is marked as a CPU-intensive domain. Note that this distinction is executed online to count the invocations for the two key functions, and commit the VCPU states every second. Once the VCPU state changes, the VF driver sends a hypercall to notify the scheduling.

With the help of I/O status monitor, a domain is distinguished as an I/O-intensive domain if the I/O operation frequency of one domain reaches the threshold. The scheduler calculates the necessary credits to make I/O-intensive domains take full advantage of CPU resources, then transfers the shared credits from CPU-intensive domains to I/O-intensive domains by Shared Credit scheduling module.

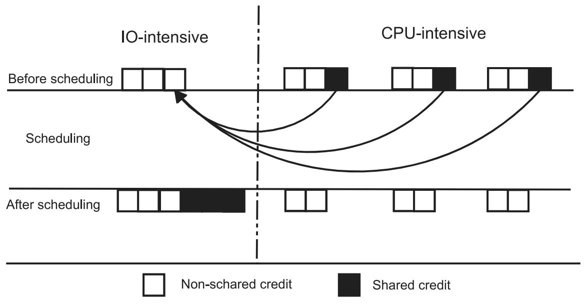


Fig. 6. Illustration of shared credit.

4.2 SC: Shared Credit Scheduling

As mentioned before, high-performance networks have great need of CPU resources for processing a large amount of occasional arrival packets. In order to provide adequate credits to I/O-intensive domains, we propose the concept of Shared Credit scheduling, as illustrated in Fig. 6. Briefly, every domain can set the proportion of credit that it is willing to share with other domains. If the needed credits do not exceed the threshold that CPU-intensive domain is willing to share, I/O-intensive domains acquire the needed credits. Otherwise, they can only acquire the credits up to the threshold that CPU-intensive domains are willing to share. This proportion of shared credit constraint keeps the fairness for the CPU-intensive domains according to their unity consciousness.

For the convenience of our description, we define some variables as follows:

- C_t: The credit value that I/O-intensive domains can leverage to take full advantage of CPU resource.
- S_i: The threshold of credit value that domain i is willing to share.
- T_i: The credit value that is transferred from CPU-intensive domain i.
- T: The total credit value that is transferred from CPU-intensive domains
- C_c: The credit value that one processor core provides.
- V_i: The number of virtual functions that are allocated to domain i.
- C_i: The credit value that scheduler has allocated to domain i.
- N_i: The number of I/O-intensive domains.
- P_i: The proportion of credit value that domain i is willing to share.

The value of C_t is denoted as Equation (1):

$$C_t = \sum_{i=1}^{N_i} (C_c \times V_i - C_i). \tag{1}$$

In this equation, C_c is a constant value of 300 due to the default credit scheduler definition, and each processor core provides 300 credits every 30 milliseconds. As virtual functions cannot run in parallel at present, each virtual function can only make use of one processor core at a time, so one vCPU needs 300 credits to occupy one processor core for time of 30 ms. So C_t is the sum of the credit values needed for each I/O-intensive domain to occupy processor cores.

The threshold of credit that domain i is willing to share with other domains is denoted as Equation (2):

$$S_i = P_i \times C_i. \tag{2}$$

In order to make CPU-intensive domains reserve credits that they are unwilling to share with others, a smaller value between C_t and S_i would be chosen. The credit value that is transferred from domain i is denoted as Equation (3):

$$T_i = min(C_t, S_i). \tag{3}$$

After all, we first calculate the total needed credits for I/O intensive domains. Then we count the total credits provided by CPU-intensive domains to share with I/O-intensive domain. At last, we poll the CPU-intensive domains, and transfer the shared credit to I/O-intensive domains.

Note that each domain has a threshold for credit that can be shared with other domains, which conserves the fairness of their own performance. If the amount of credit needed exceeds the threshold of one domain sharing threshold, only the amount of shared credit is transferred from this CPU-intensive domain. We evaluate in Section 5 how it improves I/O-intensive domains' performance when it can acquire adequate credits from CPU intensive domains. In addition, the fairness for the CPU-intensive domains will also be evaluated when granting their credits.

4.3 ACA: Dynamic Scheduling Slice Adjustment

In addition to sufficient credits, response time should also be short enough. An I/O-intensive task is different from a CPU-intensive task, as a CPU-intensive task can use the CPU at any time, while an I/O-intensive task needs the CPU only after packets arrive. The default credit scheduler in Xen redistributes credits every time-slice (30 ms). Unfortunately, if an I/O-intensive domain is inactive at the time of the allocation of credits, this domain is considered as idle and is removed from the list of active domains. As a result, it can not acquire any credit from allocation until the next time-slice and all the credits are allocated to active domains, even if packets arrive during the current time-slice.

To ensure quick response for I/O, we keep a domain participating in the credit distribution even though it is inactive, as long as it is labelled as an I/O-intensive domain. We modified the credit scheduler into two steps: credits are allocated to each domain first, then each domain distributes to its vCPUs. Before allocating credits to domains, we count the number of active vCPUs and idle I/O vCPUs, then calculate the proportion of active vCPUs in these vCPUs. Finally the total credits to distribute are the product of total credits and this calculated proportion. The method is named as Adjustable Credit Allocation (ACA).

For our theoretical analysis, we define some variables as follows:

- C_a: Total credits after adjusting.
- C_b: Total credits before adjusting.
- μ: Adjustment factor.
- N_{ai}: Number of active I/O-intensive vCPUs.
- N_{ac}: Number of active CPU-intensive vCPUs.
- N_{ii}: Number of inactive I/O-intensive vCPUs.
- N_p: Number of available processor cores.

The adjustment of total available credit is denoted as Equation (4):

$$C_a = C_b \times \mu. \tag{4}$$

In this Equation, C_b is a constant value that is obtained by Equation (5):

$$C_b = C_c \times N_p. \tag{5}$$

The meaning of C_c is introduced in Section 4.2. The μ is denoted by Equation (6):

$$\mu = \frac{N_{ai} + N_{ac}}{N_{ii} + N_{ai} + N_{ac}}. \tag{6}$$

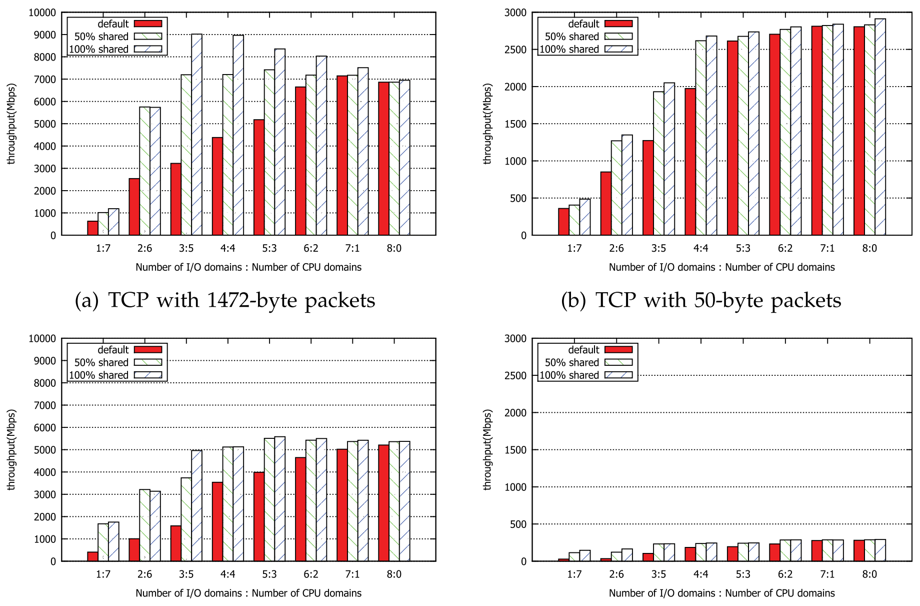
Note that the parameters in Equation (6) are all available because: I/O status monitor module, introduced in Section 4.1, can distinct I/O-intensive and CPU-intensive domains; and Xen platform can detect active/inactive domains as introduced in Section 2.1. In this way, we retain credits for I/O-intensive domains even though they are inactive at credit allocation time. The total credits is adjusted by μ, which is smaller than one. The fewer the active domains, the shorter the waiting time. Notice that the ACA method makes the I/O-intensive domain keep the possibility of be scheduled even if it is idle at the credit allocation time, but it will not really occupy the physical CPU if no packet arrives in this credit time-slice. Instead, the time-slice length is shortened. Moreover, when an I/O-intensive VM is legitimately not active, the I/O Status Monitor will be distinguished and labelled it not to be an I/O-intensive domain automatically as introduced in Section 4.1, so the overall performance would not always be affected. Therefore, the I/O-intensive domains are classified in runtime and CPU-intensive domains are treated without losing fairness, which will be evaluated later.

5 Performance Evaluation

We now present a performance evaluation of the different optimizations described in the previous sections. We first evaluate the throughput of SC and ACA independently as well as their combination effect, and then evaluate the response time and fairness.

5.1 Experimental Set-Up

The hardware platform of our test has two computer servers with the same hardware configuration to test network performance: one is used to transmit data packets; and the other receives them. Each has a quad-core Xeon E5506 with 32 KB L1 I-Cache, 32 KB L1 D-Cache, 256 KB L2-Cache and 4 MB L3-Cache. Physical memory is 16 GB DDRIII-1333. A Seagate 136 GB IDE hard disk and Intel 82599 10 Gigabit Ethernet controller are used. We run Xen-4.2.0 hypervisor in our experiment, domainU was allocated 512 MB memory and ran the CentOS-5.5 Linux distribution with the Linux-2.6.18 kernel. All of the domainUs were allocated one VCPU, one VF, and 8 GB disk space as their virtual file system. The VCPUs of the driver domain are pinned to physical processor core 0 to eliminate the interference of domain0. VCPUs of domainUs use the other three physical cores.



(a) TCP with 1472-byte packets (b) TCP with 50-byte packets

(c) UDP with 1472-byte packets (d) UDP with 50-byte packets

Fig. 7. Throughput test result with different shared ratios.

Two types of workloads are involved: Sysbench benchmark is used to produce the CPU-intensive workload to fully consume a domain's processor resources; and Netperf is used as an I/O-intensive workload to measure the maximum streaming throughput. The Netperf sever runs on the VM, and the Netperf client runs on a native Linux of another physical machine.

5.2 Evaluation of Throughput

5.2.1 Evaluation of SC Method

To show the network performance improvement with shared credit, a total of eight guest domains with different workloads are configured. We increase the number of I/O-intensive domains from one to eight, while the other domains execute CPU-intensive operations. Every domain could set the proportion of credit that may be shared, and we measured the bandwidth with zero, 50 and 100 percent sharing.

The evaluated bandwidth results are shown in Fig. 7. The network I/O performance is improved by providing shared credits in comparison with the default credit scheduler. As shown in Fig. 7, the more credits are shared, the higher the network I/O performance can be achieved. I/O-intensive domains can acquire adequate credits at every time-slice when there are few such domains. With more I/O-intensive domains, the required credits increase. When there are three or more VMs executing I/O operations concurrently, it needs so many extra credits to perform I/O so that the improvement is more obvious. However, if there are not enough shared credits to use, the performance degrades. Note that when there are some guest domains carrying out I/O-intensive operations, they have to compete with CPU-intensive domains for processor resources, so the total bandwidth of I/O-intensive domains is low. Once I/O-intensive domains have acquired enough credits to achieve 10 Gbps, providing more credit value would be useless. The extreme case is when all eight guest domains are performing I/O operations, which leads to similar performance as the default credit scheduler.

Notice from Fig. 7 that the network performance is poor when there is no domain willing to share credit, and I/O performance is improved with more shared credit. Although the performance of CPU-intensive domains is influenced because of provided extra credit, they never go beyond the permitted ratio, since some credits are reserved for computing except that one CPU-intensive domain is willing to share all of its credits. The fairness for the CPU-intensive domains will be evaluated later.

5.2.2 Evaluation of ACA Method

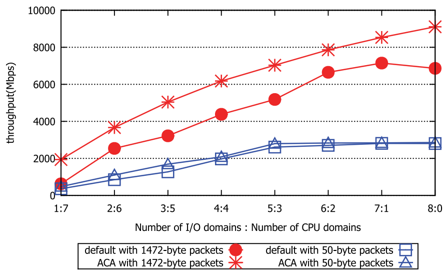
In order to evaluate the improvement from ACA method, we perform a similar experiment by running eight guest domains with different workloads.

We first evaluate the performance by solving the challenge introduced in Section 3.2. Table 1 shows the scheduling times in three minutes for CPU-intensive domains and I/O intensive domains, where default stands for the default credit scheduler in Xen. It is seen that an I/O-intensive domain is scheduled much less often than a CPU-intensive domain by default, while ACA increases the scheduling frequency of the I/O-intensive domain to around the same frequency (~5.9 k in three minutes)as a CPU-intensive domain.

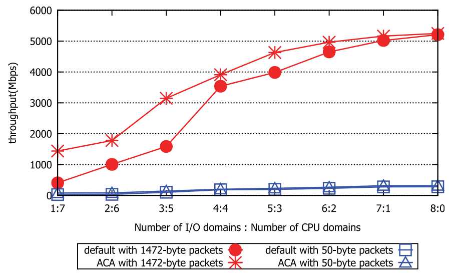
Fig. 8 illustrates the throughput performance. It is seen that ACA clearly improves the I/O network performance

TABLE 1
Comparison of Scheduling Frequency (3 min)

optimization	I/O	CPU
default	3944	5535
ACA	5914	5970



(a) TCP Performance



(b) UDP Performance

Fig. 8. Throughput test result with agile credit allocation.

in comparison with the default credit scheduler when the system is running many CPU-intensive applications. Performance of small packets is also improved, although it is not very obvious, owing to the reason discussed in Section 3.1. When there is only one I/O-intensive domain and the other seven are CPU-intensive, ACA improves the bandwidth to 2.2 Gbps, which is much better than the default credit scheduler (only 0.67 Gbps). By comparing Fig. 7 with Fig. 8, we can see that when there is a small number of I/O-intensive domains, credit may become a major performance bottleneck, which explains the reason why shared credit has better performance than agile credit allocation in these cases. However, once I/O-intensive domains already have sufficient credits, the scheduling slice becomes the new bottleneck instead of the amount of credit. Consequently, ACA outperforms SC when many guest domains execute I/O operations.

5.2.3 *Combination of Optimizations*

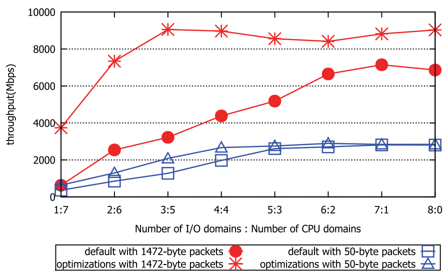
Both SC and ACA methods have been evaluated separately in previous sections. This section evaluates their combined effect. We again run eight guest domains, and vary the number of I/O-intensive domains from one to eight. We compare the default credit scheduler with our modified scheduler, when setting the shared ratio of SC to 100 percent. The results are presented in Fig. 9. Comparing Fig. 9 with Figs. 7 and 8, it is seen that combination of two optimization methods can achieve higher performance than both individual ones.

When there is only one guest domain performing I/O, while the other seven execute CPU-intensive tasks, it is illustrated in Fig. 9a that the combined optimizations greatly improve the TCP bandwidth with large packets. Bandwidth is 3.7 Gbps, reaching the goal of taking full advantage of a high-performance network when the system's load is high. It achieves full bandwidth (about 10 Gbps line-rate) when there are three I/O-intensive domains. In the default case, as the number of I/O domains increases, the bandwidth also increases at first, then begins to decrease because of the high interrupt overhead when there are too many I/O domains. This phenomenon is not obvious for small packets, because the number of interrupts has reached the upper limit. After optimization, the improvement of throughput is obvious. The performance with small packets is shown in Fig. 9. The throughput for small packets is much less than for large packets but also achieves significant improvement when there are less than four I/O-intensive domains among the eight. This is because each interrupt processes less data with small packets than with big packets. Note that TCP performs better than UDP because TCP is a reliable protocol and has better flow control. Fig. 9 shows that UDP has lower bandwidth than TCP, but can still be improved with our optimization.

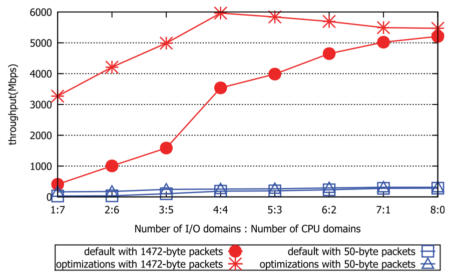
5.3 Evaluation of Response Time

Computer users are sensitive to the response time, and a long response time will seriously affect the work efficiency and user experience. As discussed in Section 3.2, the long waiting time problem can be improved by our proposed methods. Here, we measured the response time changes with Netperf, and the average response time can be converted into the number of completed transactions per second, where a transaction means a round of request/response interaction between in a client/server structure.

To measure the I/O performance benefits owing to our scheduler, we run several experiments to compare the client response time with those on default Xen. As in the previous experiments, eight guest domains with different workloads

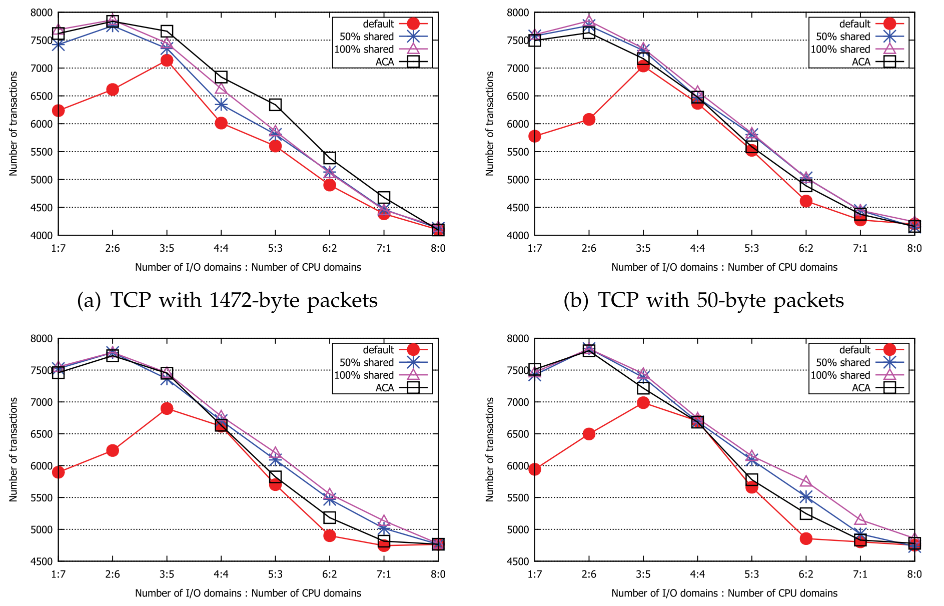


(a) TCP Performance



(b) UDP Performance

Fig. 9. Throughput test result with combination of optimizations.



(a) TCP with 1472-byte packets

(b) TCP with 50-byte packets

(c) UDP with 1472-byte packets

(d) UDP with 50-byte packets

Fig. 10. Response time test result with different shared ratios.

are configured. The total number of guest domains is always eight, while the ratio between the I/O-intensive domains and CPU-intensive domains number varies. The response time test results are shown in Fig. 10. Test results demonstrate that the network I/O response time is improved up to 22 percent after optimization. When there are a few I/O-intensive domains performing I/O, performance improvements are most obvious, because the I/O-intensive domains acquire extra credits from the CPU-intensive domains. However, with an increase in the number of I/O-intensive domain, the effect becomes weaker. The reason for this is that CPU-intensive domains can provide fewer extra credits with the reduction in their numbers.

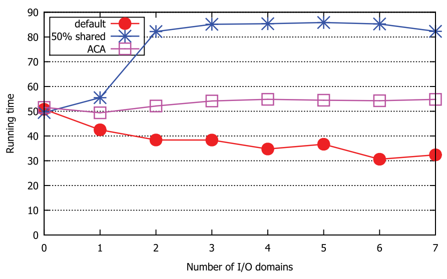
From Fig. 10, we can see that the transaction rate increases at first, then declines. This is because when the number of CPU-intensive domains decreases, the systems load is reduced, and therefore I/O response time is improved at first. It is also shown that shared credit and ACA can achieve almost the maximum transaction (8,000) constrained by interrupt throttle rate (ITR) since our efforts on resolving the problem stated in Section 3.2. Furthermore, when there are many I/O-intensive domains that the overhead of interrupts cannot be ignored any more, then the number of completed transactions rate begin to decrease. The default credit scheduling algorithm has the same performance trend in Fig. 10, since it adds a BOOST state to improve the response time, and a virtual machine in the BOOST state has the highest priority. When an event is received by the event channel of an idle virtual machine, it will enter the BOOST state, and be scheduled immediately because of the high priority of that state. The BOOST mechanism works well for a small number of I/O-intensive domains, but if many virtual machines perform I/O operations at the same time, they all will be boosted and its effectiveness is reduced.

The interrupt throttle rate represents the largest number of interrupts that is generated by controller per second. Notice that ITR in SR-IOV is configurable. By configuring the number of interrupts permitted into the network adapter, the system is able to control the number of interrupts per second. Increase of the ITR will reduce the delay time at the expense of higher CPU usage, because SR-IOV can process more packets in the same period of time. A decrease in the ITR indicates that the adapter can only send up to the number of ITR per second even if there are more packets which need to be processed. This will reduce the interrupt load, but also reduce the CPU usage when the network load is high. However, it will enlarge the waiting time, because the packet processing will slow down. The default value of ITR in the VF driver is set to 8,000. Consequently, as can be seen from Fig. 10, the number of completed transactions per second can not go beyond 8,000. Fig. 10 demonstrates that the completed transaction numbers for large and small packets are not very different. The system deals with a similar number of packets per second, so the number of completed transactions per second is similar. The difference is in the throughput, since large packets give higher throughput, with the same number of packets.

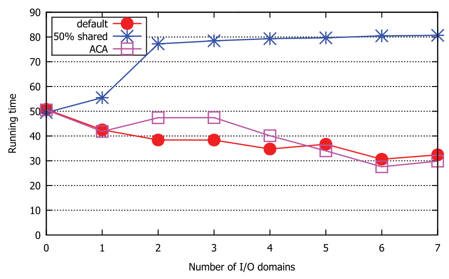
5.4 Evaluation of Fairness

Fairness means whether our proposed methods have a serious impact on CPU-intensive domain performance when taking care of I/O performance. Shared credit and ACA are collaborative scheduling between VMs, where intuitively I/O-intensive domains have more privileges than CPU-intensive domains, which may reduce the performance of CPU-intensive domains. In this section, we design an experiment to evaluate if our extended credit scheduler can still guarantee the performance of a CPU-intensive domain in comparison with the default credit scheduler.

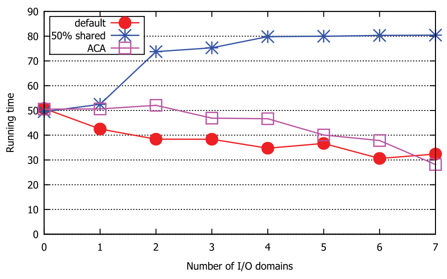
In order to measure the fairness for the CPU-intensive domains, we choose Sysbench as the computational performance benchmark. Sysbench is a modular, cross-platform and multi-threaded benchmark tool, for evaluating OS



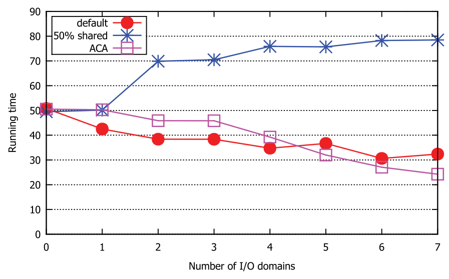
(a) TCP with 1472-byte packets



(b) TCP with 50-byte packets



(c) UDP with 1472-byte packets



(d) UDP with 50-byte packets

Fig. 11. Evaluation of fairness.

parameters that are important for a system running a database under intensive load. In the CPU benchmark, each request calculates a prime number up to a value specified by the tester, which is configured as 10,000 here. In the experiments, there are in total eight hosted domains, and we vary the number of I/O-intensive domains from one to seven. The CPU-intensive domain runs a CPU benchchmark, and we set the shared ratio as 50 percent instead of 100 percent, because 100 percent will transfer all the credits of CPU-intensive domains, so the result will make no sense. I/O-intensive domains run a different networking workload, with TCP (or UDP) streaming of large (or small) packet size.

The test result is demonstrated in Fig. 11, where the Y-axis represents the task running time for completing the given prime number calculation, which illustrate the effect of the CPU-intensive domain's performance (the shorter the running time, the higher performance). Since traditional credit scheduler treats I/O-intensive and CPU-intensive domains equally, we use it as the basic standard. When there is no I/O-intensive domain, the running times of the three methods are almost the same (the leftmost points in Fig. 11). As the number of I/O-intensive domains increases, task completion time becomes shorter and shorter with the default credit scheduler.

Here, the fairness we desired is that performance of CPU-intensive domains remains stable along with the increase number of I/O-intensive domains. While the shorter running time of the default credit scheduler is obtained at the cost of the degradation of the I/O-intensive domain throughput (as illustrated in Figs. 7, 8 and 9, where the default credit scheduler always has the lowest throughput). It is a result of the problem stated in Section 3.2, that an increasing proportion of I/O and CPU tasks may introduce the I/O-intensive domain occasionally to be idle, so consequently, CPU-intensive domains are more scheduled and can complete CPU tasks within a shorter time. ACA keeps an I/O-intensive domain participating in the credit distribution even though it is idle, so I/O-intensive domains are optimized to have the same opportunity to be scheduled as CPU-intensive domains. This is the reason that ACA keeps task completion time the most constant even though the ratio of I/O-intensive/CPU-intensive domains varies, and therefore ACA obtains better fairness, as shown in Fig. 11a, for TCP streaming with large packet size. For TCP streaming test with small packet size, shown in Fig. 11b, ACA has very similar performance as the default credit scheduler. This is because that there are much more I/O executions (arrival packets) for streaming with the small packet size, and the challenges stated in Section 3.2 are more prominent. Therefore, ACA is more reasonable to keep the I/O-intensive domain participating in credit distribution, even in an idle state, and its advantages in performance are more evident when there are more than five I/O-intensive domains.

Under the shared credit scheduler, CPU-intensive domains provide shared credits with I/O-intensive domains. When the sharing ratio is 50 percent, the running time of CPU tasks increases at first because half of their credits can be transferred to I/O-intensive domains. Then, the task completion time becomes stable when reaching the shared credit limit. This configurable ratio of the shared credit makes the CPU-intensive domains able to guarantee themselves with the unshared credits. This proves the performance fairness for CPU-intensive domain after granting all the allowed shared credit ratio, and the I/O-intensive domain cannot affect the CPU-intensive domain further.

In addition, it can be found from Figs. 11b and 11d that the ACA scheduler achieves the lowest running time when there are more than five I/O-intensive domains for TCP and UDP streaming with the small packet. This is very effective for the CPU-intensive domain since ACA can reduce the total allocated credit amount if I/O domains are idle, so as to decrease the scheduling slice. The reason is that ACA credit adjustment algorithm reduces the total credit amount and shorten the time-slice length, as described in Equations (4)-(6), where $\mu < 1$. Furthermore, a reduced total credit amount for I/O-intensive domain means a shorter execution time for all I/O-intensive domains. Therefore, the CPU-intensive domain can improve the task completion time in this case.

In summary, ACA and shared credit methods can both increase the network throughput, and their combination can further improvement this. ACA and SC can obviously improve the system response time when there are fewer I/O-intensive domains. At the same time, ACA can keep the system fairness effectively when there are more I/O-intensive domains.

6 Related Work

This section gives an overview of previous research on network I/O virtualization and VMM scheduling.

In a virtualized environment, I/O operations are handled in different ways by various VM technologies [19], [20], [14], [21]. However, the overhead of I/O virtualization is huge. Menon showed that in Xen, network throughput is only one-fifth of the native Linux for processor-bound networking workloads. The situation in VMware is even worse; native Linux's I/O performance is six times more than a virtual machine. In order to improve the performance of I/O operations, various optimizations have been proposed [22], [23], [24], [25].

6.1 Network I/O Virtualization

Network I/O performance is an important issue in a virtualization system, and many related works focused on the efficient I/O virtualization and performance enhancement. Direct I/O is an efficient I/O virtualization mechanism [26], [27]. VMDq, a component of Intel virtualization technology for connectivity, optimizes the processing of VM data traffic to improve CPU utilization and bandwidth.XenLoop [28] is an inter-VM loopback channel that enables direct network traffic between two VMs in the same physical machine without the intervention of a third software component such as Domain0. Palacious VMM [29] hooks I/O operations between device drivers on a guest OS and physical hardware, and a VPIO model is proposed to direct assignment I/O via the passthrough with little hypersivor interaction. ELI [13], [30] further enhance the network performance by introducing an exit-less interrupt mechanism, which allows a physical interrupt to be injected into the VM directly, without the intervention of the VMM.

6.2 VMM Scheduling for I/O Performance

Scheduling in VMM also has a close relationship with I/O performance [31], [32]. VMM schedulers focus typically on fair scheduling, while ignoring I/O task performance. Lots of research has been devoted to improving I/O performance.

Cherkasova et al. compared three kinds of Xen scheduler: BVT, SEDF and the credit scheduler [33], [3]. Three relatively simple system benchmarks were used to show that the scheduler parameters can impact the I/O performance drastically. Furthermore, they proposed SEDF-DC [18] to distribute the CPU usage into the guest domains that trigger I/O operations to an IDD.

Govindan et al. identified a key shortcoming in existing virtual machine monitors and developed a new communication-aware CPU scheduling algorithm for the Xen VMM [34]. Their scheduling mechanism achieves high performance on a network-intensive server, but does not tackle block I/O tasks.

Ram et al. proposed two optimizations to reduce the CPU overhead of network I/O virtualization [9]. First, Xen was modified to support a multi-queue network interface, to remove the software overhead of demultiplexing and copying. Second, a grant reuse mechanism was used to reduce the overhead of memory protection. These two mechanisms remove the bottleneck of I/O from the driver domain with additional enhancement, such as Large Receive Offload (LRO), software pre-fetching and half-page buffers.

Liao et al. proposed a cache-aware scheduler to improve network virtualization [35]. They developed two optimizations for the VMM scheduler. The first uses cache-aware scheduling to reduce inter-domain communication cost, and the second steals scheduler credits to favour I/O VCPUs in the driver domain. They also proposed two optimizations to improve packet processing in the driver domain. First, they designed a simple bridge for more efficient switching of packets. Second they developed a patch to make transmit queue length in the driver domain configurable.

Chen et al. proposed dynamic switching frequency to schedule pinned domains in the Xen VMM [36]. Their extended scheduler greatly improved the performance of communication-intensive and I/O-intensive applications in pinned domains.

Hu et al. proposed an I/O scheduling model based on multi-core dynamic partitioning [37]. The processor cores are divided into three subsets: a driver core to run the driver domain; fast-tick cores to speed up I/O response; and general cores to perform normal computation tasks. This model can improve the performance and stability of I/O VMs on a multi-core platform.

Since the scheduler is unaware of the task type in a VM, I/O tasks cannot be scheduled in time [38]. Although the BOOST mechanism reduces I/O latency efficiently, when a VM runs both I/O and CPU tasks, a CPU task may exhaust the time slice and put the VCPU into OVER status. As a result, an I/O task may not be boosted in time. Kim et al. proposed task-aware virtual machine scheduling to improve the performance of I/O-bound tasks [39]. They used gray-box knowledge to infer the I/O-boundness of guest-level tasks and correlate incoming events with I/O-bound tasks. So an I/O-bound task is scheduled to handle its incoming events promptly.

Dong et al. proposed an efficient interrupt coalescing method for network I/O virtualization, and a virtual receive side scaling scheme to leverage multi-core processors effectively [40], [41], [14]. Their proposed optimizations improve network I/O virtualization performance significantly, and tackle the performance challenges effectively.

Unlike previous work, our research is SR-IOV oriented, so the driver domain is no longer the bottleneck of I/O. Adequate CPU resources and timely scheduling are also important factors on a high-performance network. Our optimizations improve bandwidth, even when the system load is very high.

7 Conclusion and Future Work

This paper focused on I/O performance improvement in the SR-IOV environment for cloud computing architecture. We indicated two reasons for unsatisfying network performance: one is that an I/O-intensive domain normally lacks credits to perform I/O, and the other is that an I/O-intensive domain is more likely to be inactive at credit allocation time, and may wait longer to handle inputs. To solve these problems, we propose two optimizations of Xen's credit scheduler. The first optimization is shared scheduling, which provides extra credits to I/O-intensive domains from a pool of shared credits. The second is agile credit allocation, which can detect the occasional inactive I/O domain and reserve its credits. ACA adequately adjusts the total number of credits to shorten the waiting time for an I/O domain. The simulation results demonstrate that either optimization is effective alone, and a combination of the two gives even more significant improvement. The experiment results demonstrate the effectiveness in both bandwidth and response time, while keeping the fairness for CPU-intensive domains. Our solutions make virtualization more suitable for a wide variety of applications in cloud computing, such as fluent audio playing. The future implementation is expected to be more friendly to heterogeneous workloads, and more sophisticated testing scenarios can be carried out.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 61272100, 61202374), 863 Program of China (No. 2012AA010905), International Cooperation Program of Shanghai (No. 1251070610), the Ministry of Education Major Project (No. 313035), and NRF Singapore under its CREATE Program. Jian Li is the corresponding author.

References

[1] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *ACM SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, Apr. 2006.

[2] J.-W. Jang, M. Jeon, H.-S. Kim, H. Jo, J.-S. Kim, and S. Maeng, "Energy reduction in consolidated servers through memory-aware virtual machine scheduling," *IEEE Trans. Comput.*, vol. 60, no. 4, pp. 552–564, Apr. 2011.

[3] L. Cherkasova and D. Gupta, "When virtual is harder than real: Resource allocation challenges in virtual machine based IT environments," HP Labs., Palo Alto, CA, USA, Tech. Rep. HPL-2007-25, 2007.

[4] T. C. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *Proc. IEEE Conf. e-Business Eng.*, 2009, pp. 281–286.

[5] F. Hao, T. V. Lakshma, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," in *Proc. IEEE Conf. e-Business Eng.*, 2009, pp. 281–286.

[6] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the XEN virtual machine environment," in *Proc. 1st ACM/USENIX Int. Conf. Virtual Execution Environ.*, 2005, pp. 13–23.

[7] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in XEN," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2006, pp. 2–2.

[8] L. Cherkasova and R. Gardner, "Measuring CPU overhead for I/O processing in the XEN virtual machine monitor," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2005, pp. 24–24.

[9] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner, "Achieving 10 GB/s using safe and transparent network interface virtualization," in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2009, pp. 61–70.

[10] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the gap between software and hardware techniques for I/O virtualization," in *Proc. USENIX Annu. Tech. Conf. Annu. Tech. Conf.*, 2008, pp. 29–42.

[11] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, "Towards high-quality I/O virtualization," in *Proc. SYSTOR 2009: The Israeli Exp. Syst. Conf.*, 2009, pp. 12:1–12:8.

[12] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in *Proc. 16th Int. Symp. High Perform. Comput. Archit.*, 2010, pp. 1–10.

[13] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "Eli: Bare-metal performance for I/O virtualization," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 411–422, Mar. 2012.

[14] H. Guan, Y. Dong, R. Ma, D. Xu, Y. Zhang, and J. Li, "Performance enhancement for network I/O virtualization with efficient interrupt coalescing and virtual receive-side scaling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1118–1128, Jun. 2013.

[15] VMware, "The cpu scheduler in vmware vsphere® 5.1," http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf, USA, Tech. White Paper, Jan. 2013.

[16] T. Newhouse and J. Pasquale, "Alps: An application-level proportional-share scheduler," in *Proc.IEEE Int.Symp.HighPerform. Comput*,2006,pp.279–290.

[17] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the xen virtual machine monitor," in *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, 2004, pp. 1–1.

[18] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in Xen," in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*, 2006, pp. 342–362.

[19] C. Waldspurger and M. Rosenblum, "I/O virtualization," *Commun. ACM*, vol. 55, no. 1, pp. 66–73, Jan. 2012.

[20] H. Guan, Y. Dong, K. Tian, and J. Li, "SR-IOV based network interrupt-free virtualization with event based polling," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2596–2609, Dec. 2013.

[21] X. Liao, K. Chen, and H. Jin, "Adaptidc: Adaptive inter-domain communication in virtualized environments," *Comput. Electr. Eng.*, vol. 39, no. 7, pp. 2332–2341, Oct. 2013.

[22] J. Liu, W. Huang, B. Abali, and D. K. Panda, "High performance VMM-bypass I/O in virtual machines," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, Berkeley, CA, USA, 2006, pp. 3–3.

[23] K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, and S. Pope, "Getting 10 GB/s from Xen: Safe and fast device access from unprivileged domains," in *Proc. Conf. Parallel Process.*, Berlin, Germany, 2008, pp. 224–233.

[24] D. Guo, G. Liao, and L. Bhuyan, "Performance characterization and cache-aware core scheduling in a virtualized multi-core server under 10gbe," in *Proc. IEEE Int. Symp. Workload Characterization*, Oct. 2009, pp. 168–177.

[25] M. Kesavan, A. Gavrilovska, and K. Schwan, "Differential virtual time (DVT): Rethinking I/O service differentiation for virtual machines," in *Proc. 1st ACM Symp. Cloud Comput.*, New York, NY, USA, 2010, pp. 27–38.

[26] H. Raj and K. Schwan, "High performance and scalable I/O virtualization via self-virtualized devices," in *Proc. 16th Int. Symp. High Perform. Distrib. Comput.*, New York, NY, USA, 2007, pp. 179–188.

[27] P. Willmann, S. Rixner, and A. L. Cox, "Protection strategies for direct access to virtualized I/O devices," in *Proc. USENIX Annu. Tech. Conf. Annu. Tech. Conf.*, Berkeley, CA, USA, 2008, pp. 15–28.

[28] J. Wang, K. Wright, and K. Gopalan, "Xenloop: A transparent high performance inter-vm network loopback," in *Proc. ACM Int. Symp. High Perform. Parrallel Distrib. Comput.*, 2008, pp. 109–118.

[29] L. Xia, J. Lange, and P. Dinda, "Towards virtual passthrough I/O on commodity devices," in *Proc. First Conf. I/O Virtualization*, ser. WIOV'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–1.

[30] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "Eli: Bare-metal performance for I/O virtualization," in *Proc. 17th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, New York, NY, USA, 2012, pp. 411–422.

[31] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly virtual machines: Leveraging a feedback-control model for application adaptation," in *Proc. 1st ACM/USENIX Int. Conf. Virtual Execution Environ.*, 2004, pp. 2–12.

[32] D. Li, X. Liao, H. Jin, B. Zhou, and Q. Zhang, "A new disk I/O model of virtualized cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1129–1138, Jun. 2013.

[33] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, Sep. 2007.

[34] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: Communication-aware CPU scheduling for consolidated Xen-based hosting platforms," in *Proc. 3rd Int. Conf. Virtual Execution Environ.*, 2007, pp. 126–136.

[35] G. Liao, D. Guo, L. Bhuyan, and S. R. King, "Software techniques to improve virtualized I/O performance on multi-core systems," in *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, New York, NY, USA, 2008, pp. 161–170.

[36] H. Chen, H. Jin, K. Hu, and J. Huang, "Dynamic switching-frequency scaling: Scheduling pinned domain in Xen VMM," in *Proc. 39th Int. Conf. Parallel Process.*, 2010, pp. 287–296.

[37] Y. Hu, X. Long, J. Zhang, J. He, and L. Xia, "I/O scheduling model of virtual machine based on multi-core dynamic partitioning," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 142–154.

[38] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *Proc. 4th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2008, pp. 1–10.

[39] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for I/O performance," in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual execution Environ.*, 2009, pp. 101–110.

[40] Y. Dong, D. Xu, Y. Zhang, and G. Liao, "Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2011, pp. 26–34.

[41] K. Salah, "Integrated performance evaluating criterion for selecting between interrupt coalescing and normal interruption," *Int. J. High Perform. Comput. Netw.*, vol. 3, no. 5/6, pp. 434–445, Mar. 2005.

Haibing Guan received the PhD degree in computer science from the Tongji University, China, in 1999. He is currently a professor with the Faculty of Computer Science, Shanghai Jiao Tong University, China. He is a member of CCF. His current research interests include, but are not limited to, computer architecture, compiling, virtualization, and hardware/software co-design. He is a member of the IEEE.

Ruhui Ma received the BS and MS degrees from the School of Information and Engineering from Jiangnan University, China, in 2006 and 2008, respectively. He received the PhD degree in computer science from Shanghai Jiao Tong University, China, in 2011. His main research interests are in virtual machines, computer architecture, and compiling.

Jian Li received the BS degree in electronics and information technology from TianJin University, China, in 2001, the MS degree in telecommunication and computer science from the University of Henri Poincare, France, in 2003, and the PhD degree in computer science from the Institute National Polytechnique de Lorraine (INPL), Nancy, France, in 2007. He is an associate professor in the School of Software at Shanghai Jiao Tong University. He has worked as a postdoctoral researcher at the University of Toronto and as an associated researcher at McGill University in 2007 and 2008, respectively. His research interests include real-time scheduling theory, Cyber-Physical system, real-time communication, network protocol design and quality of service, real-time computing and embedded system. He is a member of the IEEE.