Raccoon: A Novel Network I/O Allocation Framework for Workload-Aware VM Scheduling in Virtual Environments

Lingfang Zeng, *Member, IEEE*, Yang Wang, Xiaopeng Fan, *Member, IEEE*, and Chengzhong Xu, *Fellow, IEEE*

Abstract—We present a network I/O allocation framework, called *Raccoon*, for workload-aware VM scheduling algorithm to facilitate hybrid I/O workloads in virtual environments. *Raccoon* combines the strengths of paravirtual I/O and SR-IOV techniques to not only minimize the network latency, but also optimize the bandwidth utilization for workload-aware VM scheduling. In *Raccoon*, a limited number of VFs in SR-IOV are granted to I/O-intensive VMs while the paravirtual Network Interface Cards (vNICs) are allocated to other non-I/O-intensive VMs as the default resources. With this design, *Raccoon* provides latency reduction and bandwidth guarantee under the premise that I/O-intensive VMs will always be granted the VFs to facilitate their I/O operations. The types of workloads in each VM are identified at runtime by modified *XenMon*. By leveraging the ACPI Hotplug technique, *Raccoon* can adaptively plugin and plugout the SR-IOV VFs upon the changes of VM requirements so that an efficient I/O workload-aware VM scheduling algorithm can be implemented based on the bonding driver technique. The experimental results reveal that *Raccoon* can combine the benefits of para-virtual I/O and SR-IOV techniques to improve the overall performance of virtualized platforms with VMs that have diverse I/O workloads.

Index Terms—Workload aware, VM scheduling, hypervisor, bandwidth allocation, I/O virtualization, SR-IOV, bonding driver, hotplug

1 Introduction

It is well known that the attributes of service workloads are inherently diverse in nature and dynamically varied over time in terms of CPU, memory, and I/O requirements. However, these attributes in practice are often invisible to resource scheduler, who is seldom able to take their advantages for making effective decisions on resource allocation. With the rapid deployment of cloud computing, more and more traditional service workloads are migrating to the cloud environments (e.g., datacenter) in a form of virtual machine (VM) hosted programs running on the same physical host for different goals. Consequently, this unawareness of workload attributes would result in sub-optimal resource utilization and poor performance as well.

Among the datacenter resources, the network bandwidths, even with substantial improvements, are still deemed as precious resources in practice, which are not only scarce, given most service applications are becoming network intensive, but also hard to control, as the network traffic usually fluctuate significantly due to the interference of network intensive workloads. On the other hand, the network latency is also a pragmatic concern since the overhead of network stack is in general hard to reduce. As a result, an effective framework for network I/O allocation is highly desirable for the VM scheduling that is conscious with the workload attributes.

I/O virtualization has been intensively studied in recent years [1], [2], [3], [4], [5], [6], and well accepted as one of effective ways to boost the network performance. With the I/O virtualization, the workload-aware computing could be built in a virtualization environment to bridge the gap between the workload attributes and the scheduler strategies [4].

Nowadays, the para-virtualization [7] and the SR-IOV [8], [9] are two of the most popular I/O virtualization approaches, but each with its own performance issues. The SR-IOV has high hardware requirements as the number of supported virtual devices depends on the hardware configuration. Moreover, it also suffers from the availability of network bandwidth resources as all the virtual devices will share the same finite bandwidth. In contrast, the para-virtualization does not need special hardware supports, and achieves the relatively good performance in a pure software way. However, this way not only elongates the data I/O path, but also entails the driving field (domain) to become an access bottleneck, resulting in poor network performance.

- *L. Zeng is with the Wuhan National Laboratory for Optoelectronics, School of Computer, Huazhong University of Science and Technology (HUST), Wuhan, HuBei 430074, China. E-mail: lfzeng@hust.edu.cn.*
- *Y. Wang and X. Fan are with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: {yang.wang1, xp.fan}@siat.ac.cn.*
- *C. Xu is with the Shenzhen Institutes of Advanced Technology (SIAT), Chinese Academy of Sciences, Shenzhen 518055, China, and the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. E-mail: cz.xu@siat.ac.cn.*

Manuscript received 9 Oct. 2016; revised 16 Jan. 2017; accepted 12 Mar. 2017. Date of publication 21 Mar. 2017; date of current version 9 Aug. 2017. Recommended for acceptance by J. N. Amaral.
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/TPDS.2017.2685386

There exist many studies to improve the I/O virtualization. For example, some researchers consider to reduce the I/O latency by introducing new VM scheduling priority [10], by balancing the workloads of VCPUs [11], [12], or by adjusting the time slice of VCPUs [13], while others, each with different goals, study to optimize the I/O bandwidth allocation either by leveraging bargaining game approach [14] or by exploiting the logistic model under the control-theoretic framework [15]. However, to the best of our knowledge, quite few studies jointly consider the network latency and bandwidth at the same time with respect to the VMs with hybrid workload attributes, especially from the system perspective.

In this paper, we study the network I/O allocation framework for the VM scheduling in virtual environments. To this end, we design a framework, called *Raccoon*, that classifies the VMs into I/O-intensive VMs and non I/O-intensive VMs, and combines the advantages of para-virtual I/O and SR-IOV to implement an I/O workload-aware VM scheduling algorithm. The essence of this design is to fully utilize the features of SR-IOV by prioritizing the I/O-intensive VMs with granted VFs for I/O latency reduction and by limiting the number of VFs to guarantee the sufficient bandwidth allocated to each VF for high network throughput. Also, *Raccoon* allows other non-I/O-intensive VMs to share the same para-virtual *Network Interface Card* (vNIC) to minimize the impact on the network contention.

We prototyped *Raccoon* based on Xen's Credit scheduler. Xen is one of widely deployed virtualization platforms, and its default Credit scheduler suffers from the same weakness in adapting to the workloads with different I/O behaviors. On the other hand, *Raccoon* has minimal impact on the implementation of the Credit scheduler, it does not adjust the credit distribution scheme, which is different from the work of Guan et al. [4], where the credits of each domain is dynamically altered in favor of the I/O-intensive VMs. As such, *Raccoon* is more generic, and powerful enough, not only beneficial to Xen but also useful to other VM scheduling algorithms for handling diverse applications. Our empirical studies show that with *Raccoon*, we can provide flexible and efficient virtual I/O allocations and retain all the benefits of the paravirtual I/O and SR-IOV techniques.

The remainder of the paper is organized as follows. We first overview the SR-IOV techniques and analyze some potential performance issues of Xen to the latency sensitive tasks in Section 2. With these issues in mind, we then introduce the proposed *Raccoon* optimization on the Credit scheduler and the network I/O virtualization in Section 3. The performance evaluation on *Raccoon* is presented in Section 4, and some related work is overviewed in Section 5. Finally, we conclude the paper with some directions for our further work in Section 6.

2 Background Knowledge

Overall, I/O virtualization is typically divided into two kinds of technologies, *software-based I/O virtualization* and *hardware-based I/O virtualization*. The software-based I/O virtualization can be further classified into *full virtualization* and *para-virtualization* [7] while the hardware-based I/O virtualization requires the supports from hardware designs such as *directly assigned equipment* [16] and *Single-Root I/O Virtualization* (SR-IOV) [8], [9].

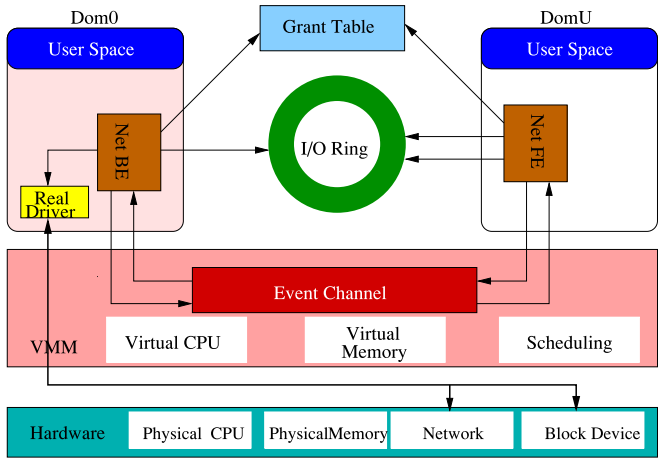

Fig. 1. I/O para-virtualization in Xen.

Full I/O virtualization is a hardware-independent solution that provides a full I/O platform illusion, and thus has better scalability. However, compared to its native environment, it suffers from a great loss of performance because virtual machine monitor (VMM) has to intercept and simulate *all* the I/O operations [2], [17], [18]. As opposed to the full virtualization, the para-virtualization provides a similar virtualized platform via a modified lightweight simulation [19], [20]. As a consequence, the its performance is usually better than that of the full virtualization. To further improve the performance, the hardware-based virtualization usually bypasses the VMM to achieve the desired "bare-metal" performance [1], [3], [21], [22], [23]. It mainly has *Passthrough* and *SR-IOV* modes, which allow the I/O device is either exclusively assigned to a VM or multiplexed as a set of virtual devices, each granted to a separate VM for direct access without VMM intervention.

2.1 I/O Para-Virtualization in Xen

Dom0 in Xen is a privileged domain that can function as a driver domain (if it hosts the device drivers) to perform the I/O operations on behalf of the unprivileged guest domains (i.e., DomUs) on the same host. In order to transit and receive the data packets, the DomUs cooperate with the Dom0 via two ring buffers as shown in Fig. 1, which are implemented in Xen based on *grant table* and *event channel*.

The grant table is a mechanism to share or transfer memory pages between domains. With grant table, one domain can inform the hypervisor of granting another domain to access its own memory pages. Instead, the hypervisor can keep the grants in a grant reference (GR) table and then pass the grant reference onto the other domain and signals this I/O request via the event channel with the target domain by pending a virtual interrupt. The target domain will check its event channel when it is scheduled and deliver the pending event by invoking the corresponding interrupt handler. In particular, for a packet transmission, the interrupt handler will forward the request to the native device driver in Dom0. Of course, the latency between pending and delivering the event depends on the underlying VM scheduling algorithms. From the figure, the native device driver can communicate with the network card to complete the packet transmission, which could be also virtualized, say by exploiting the following technology.

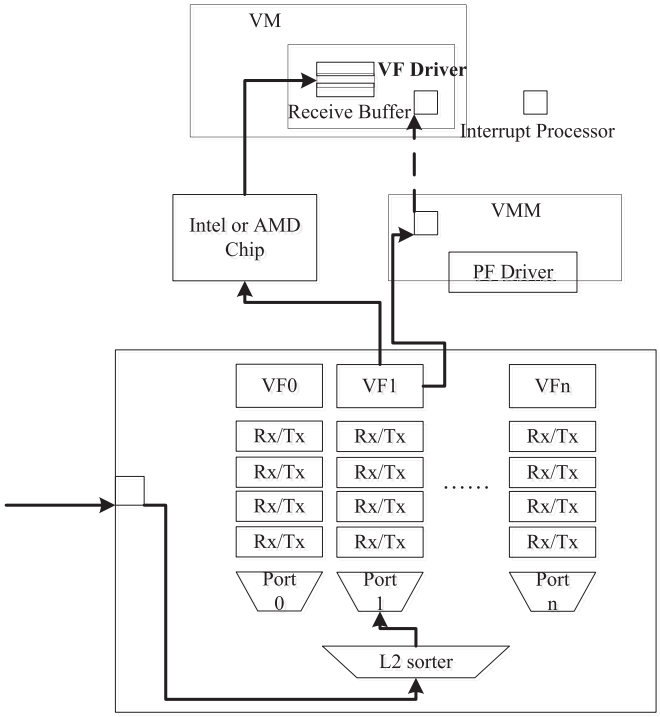

Fig. 2. The SR-IOV workflow to handle an incoming data packet.

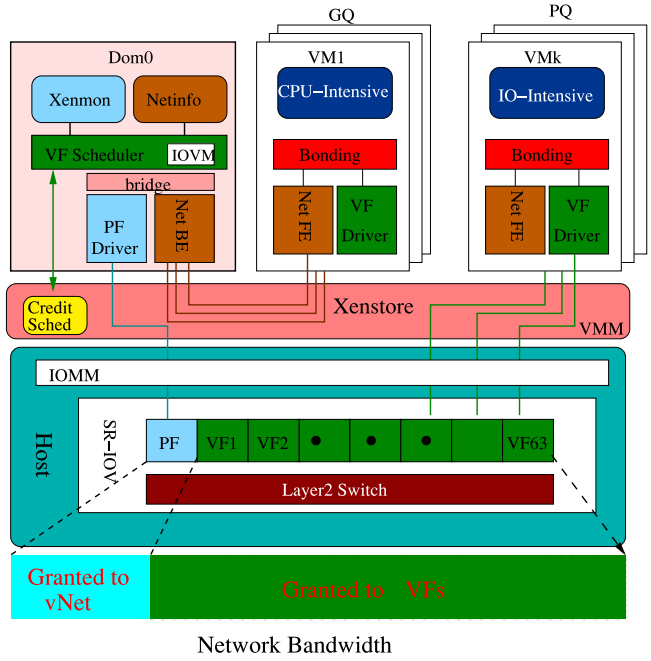

Fig. 3. Raccoon framework.

2.2 Single-Root I/O Virtualization

Unlike the I/O para-virtualization, which heavily relies on the VMM supports, the SR-IOV eliminates the VMM interventions to substantially improve the VM's I/O performance. The SR-IOV device contains a *Physical Function* (PF) and one or more VF resources. The PF is a complete PCIe device that contains extended functionality for the configuration and management of SR-IOV resources, while the VFs are lightweight PCIe devices that possess the necessary resources for data transmissions (e.g., sending and receiving registers). The PF driver is usually dedicated to the virtual machine monitor or runs inside Dom0 in Xen with a high privilege level. It can directly access all hardware resources, configure and manage all of the VFs, including setting the number of VFs, starting or stopping the VFs, and so on. In contrast, the VF drivers run in the VMs and are VMM agnostic. With the hardware supports such as Intel VT-d technology, the VF driver can directly accesses the granted VFs without the intervention from VMM.

Fig. 2 depicts how an incoming Ethernet packet is handled by SR-IOV. When a data packet arrives at the Ethernet NIC, it is first sent to the receive queue of the target VF via the Layer2 sorter. Then, the DMA operation is initiated where the target memory address, defined by the VF driver within the VM, is remapped to a physical host address (i.e., the recipient's buffer) for the incoming packet. As soon as the data transfer is completed, an interrupt is fired by the Ethernet NIC, indicating a packet has arrived. The VMM captures the interrupt and informs the target VM by sending a virtual interrupt. Finally, when the target VM is scheduled to execute with the virtual interrupt, it reads the packet from its local buffer via its VF driver and complete the entire data receiving process. The process of sending data is similar to this process, but in reverse order of the events.

3 Design and Implementation

3.1 Basic Ideas

Raccoon is designed to improve the I/O-network performance by optimizing the utilization of I/O NIC resources to boost the performance of workload-aware VM scheduling. Our design is motivated by the observation that even though leveraging the VFs could dramatically reduce the network latency, the bandwidth available to each VF may not be sufficient to fully exert its advantages since the available network bandwidths to each single VF are inversely proportional to the number of configured VFs. As such, when multiple VMs are active at same time, demanding the bandwidths simultaneously, each associated VF will be allocated less bandwidths, which could fail to guarantee the performance of those I/O-intensive VMs.

To address this issue, *Raccoon* combines the virtualization approaches in both worlds to provide bandwidth guarantee under the premise that I/O-intensive VMs will always be granted the VFs to facilitate their I/O operations. In other words, by limiting the number of VFs, sufficient bandwidths could be reserved for each VF granted to those network-intensive VMs, while the remaining bandwidths are allocated to the vNet shared by other non-network I/O intensive VMs (more than one VFs can be granted to a VM, but a VF is only allowed to assign to one VM each time). With this strategy, *Raccoon* can optimize the bandwidth utilization and boost the performance of the workload-aware VM scheduling as well.

3.2 Framework Overview

As shown in Fig. 3, *Raccoon* is composed of three major parts: improved *XenMon*, *Netinfo*, and VF *Scheduler*, and all the virtual machines are classified into two categories for effective managements, *Priority Queue* (PQ) for *NetIO* workloads and *General Queue* (GQ) for others, including both *DiskIO* workloads and *CPUInten* workloads.

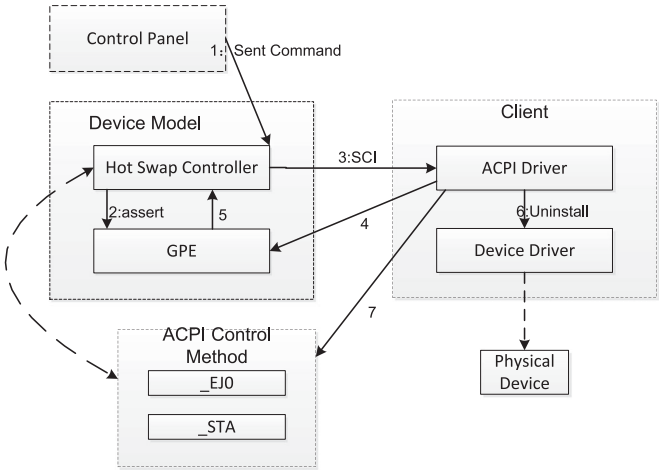

Fig. 4. Virtual ACPI hotplug workflow.

The improved *XenMon* is used to obtain the real-time information regarding the running VMs, including `gotten_time`, `blocked_time` and `waiting_time` as well, while the *Netinfo* is exploited to collect the traffic information sent to or received from each virtual machine. In the design, the improved *XenMon* and the *Netinfo* are collectively referred to as *Assessment Component* (AC) whereby the *VF Scheduler* leverages the gathered information to optimize all the resource utilization by depriving the non-I/O-intensive VMs of their VF resources and re-grant them to the I/O-intensive VMs.

The dynamic resource allocation in *Raccoon* is based on the temporal locality, namely an I/O-intensive VM will be most likely an IO-intensive VM in the next period of time. By exploiting the information from AC, the *VF scheduler* could always allocate the VF resources to the I/O-intensive VMs, leading to a better use of the hardware. Compared to the para-virtualization techniques, this approach can improve the overall performance of the VMs through the enhancements of those I/O-intensive VMs. Furthermore, since *Raccoon* uses the combination of para-virtualization and SR-IOV, it can reduce the pressure on the para-virtualized driver domain, which would also extend the system scalability.

3.3 NIC Resources and Queues

The VF resources are allocated to each VM by *Raccoon* according to its type. The VMs with different types are put into different queues for easy managements and scheduling. This section describes the NIC resources and various designed queues in the virtual machines.

3.3.1 NIC Resources

As discussed, the I/O virtualization in *Raccoon* is divided into two types, the *para-virtualized vNIC* and the *SR-IOV*. *Raccoon* combines the advantages of both methods to facilitate the VM scheduling with hybrid I/O workloads. As shown in Fig. 3, each VM possesses two kinds of resources, vNIC and VFs, and the *Dom0* in particular exploits the network card as a bridge to create the VMs, each having a vNIC as the default resource. Whenever the scheduler identifies an I/O-intensive VM, *VF Scheduler* will grant a VF to the VM in order to improve its I/O performance. As such, with dynamic scheduling of the VFs, *Raccoon* can allow the I/O-intensive VMs to have better performance, which in turn improve the overall system performance. However, to reach this goal, there are two problems to be addressed for each VM, which are the main concerns of this paper: (1) the method for dynamic VF (de)allocation, and (2) the network card switching between vNIC and VFs.

Dynamic VF (de)allocation: In order to minimize the adverse impact on the running VMs with dynamically (de) allocated VF resources, the hotplug PCIe devices are adopted for those VMs. Xen initially did not support the dynamic VF allocations, rather, the VF allocation is static in the sense that it is only accomplished by modifying the configuration file when the VM is created. With the development of virtualization technology, such as *ACPI Hotplug*, *SHPC* (Standard HotPlug Controller), and *Vendor-specific*, the hotplug PCIe problem for VMs has been largely resolved. Currently, all of the above hotplug standards are supported by Linux 2.6 onwards.

Among the current technologies, the *ACPI Hotplug*, as shown in Fig. 4, is amenable to being emulated in device model, and used in our design. When an end-user initiates a hot removal order for the direct device (1), the hotplug controller first quickly updates its status, asserts the General Purpose Event (GPE) bit (2),[1] and then initiates an System Control Interrupt (SCI) interrupt (3). Upon receiving the SCI interrupt, the ACPI driver in the Linux VM clears the corresponding bit in GPE (4), and in the meantime, it is set to query the hotplug controller for the information regarding the device that needs to be removed (5) (dashed arrowed line), the device information is then forwarded to the Linux VM, which, in turn, stops the device and uninstalls the device driver (6). Finally, the ACPI driver executes the control operation `_EJ0` to turn off the PCI device and issues `_STA` to verify whether the device is removed successfully or not (7). As for the hot adds in ACPI, the workflow process is similar to this one, except that the operation `_EJ0` is not executed.

Card Switching: In our design, VM is equipped with a vNIC as the default card when it is created, and the VFs need to be allocated to the VMs with I/O-intensive workloads. On the other hand, with *Raccoon*, the VMs should adapt to their I/O behaviors for performance optimization. As such, some VMs previously with heavy I/O operations could become I/O light weighted again as the computation proceeds. In this case, the allocated VFs to these VMs may be deallocated and granted to other VMs whose I/O operations tend to be increased. To attain network connectivity, it is desired for *Raccoon* to have a capability that allows one VM to immediately switch to its vNIC after the hot removal of VF as discussed above, and another VM to accordingly switch from its vNIC to the granted VF for better performance.

To address this issue, *Raccoon* takes advantage of the *bonding driver* technique that can aggregate multiple network interfaces into a single, logical "bonded" one with the same MAC address. The logical interface can be configured for different purposes by setting its modes as *active-backup*, *load balancing*, or others. In *Raccoon*, we leverage the

1. GPE is a bitmap, in which, according to the hardware design, each bit is connected to a different hardware event value, which corresponds to a different event notification.

active-backup mode for automatic switch, which is often used to tolerate faults in network cards. In this mode, only one slave in the bond is active as a primary slave at any time, and a different slave becomes active if and only if the active slave fails. Fig. 3 shows a virtual machine has a bonding driver that aggregates two slave interfaces into a logical interface, one is VF, a physical NIC, as the primary, and the other is vNIC as the secondary. Normally, the bond would prefer the VF as the primary slave interface and automatically perform the following adaptive actions in response to hotplug events: (1) When the VF is deallocated from the VM, the vNIC become active immediately and in charge of the in/out traffic of the VM. (2) With the vNIC, the VM maintains network connectivity during the adaptive process. (3) Whenever a VF is granted to VM, the VF is immediately used as the active slave with high throughput.

3.3.2 Two Queues

Raccoon maintains two queues, *PQ* and *GQ*, in modified Credit scheduler for the VMs whereby the VF scheduler can manage the VF allocation. *PQ* is a red-black tree, designed for the VMs with *NetIO* workloads while *GQ* is for others, including those VMs with *DiskIO* workloads and *CPUInten* workloads. The purpose of this classification is to allow *Raccoon* to quickly recognize the nature of each VM whereby the VF scheduler can efficiently allocate the VF resources according to the I/O-intensity of each VM. *Raccoon* puts each VM into one of the queues, depending on its workload nature, and allocates the vNIC and/or VF resources to it.

In this paper, we focus on the VM scheduling algorithms for hybrid VMs in both *PQ* and *GQ* to maximize the overall performance of the VM scheduling. To this end, in regards of the VMs in *PQ*, they must have high network workloads and as well clear requirements for the I/O bandwidths that are aggregatedly contributed by all the VFs lined in the queue. Since the network bandwidths are shared by the configured VFs in the SR-IOV NIC (e.g., the Intel 82,599 can support up to 63 VFs), *Raccoon* can guarantee sufficient bandwidths for each VM in *PQ* by limiting the number of VFs in the queue. The number of VFs is set according to the VM workloads and the bandwidths that are available to uses. Unlike the VMs in *PQ*, who occupy the VF resources until they are removed, the VMs in *GQ* would use the vNIC card for their input/output traffic, the vNIC card is supported by the PF or other VFs via the corresponding drivers in Dom0, instead.

Raccoon is initialized as follows to handle the two queues: all the created VMs are first put into *GQ*. Then, based on the configurations, the VMs with certain I/O bandwidth requirements are gradually moved to *PQ* and granted the VF resources. The PF and/or the remaining VFs if they are available after the *PQ* allocation are granted to the vNIC for the VMs in *GQ* based on their creation orders. As designed, *Raccoon* is adaptive to the changes of VMs' nature.

3.4 Improved XenMon

XenMon is a performance analysis tool designed to report a variety of metrics that present how the time is spent by each domain in Xen. The group of metrics mainly includes *CPU usage*, *blocked time*, and *waiting time*. *CPU usage* specifies the percentage of time that a domain runs on CPU, *blocked_time* reflects the percentage of time of a domain that is blocked on some I/O event, and *waiting time* shows the percentage of time of a domain that is waiting in the run queue to be scheduled for CPU. *Raccoon* exploits these three main metrics for our research goal as they completely cover how time is spent by a domain in Xen.

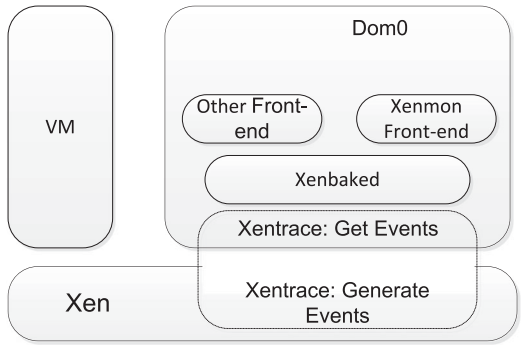

Fig. 5. XenMon composition.

XenMon consists of three inter-connected parts to gather these metrics, namely *xentrace*, *xenbaked*, and *xenmon* as shown in Fig. 5. The xentrace is a lightweight event generation module, which can raise events at any control point in the hypervisor and associate the events with some attributes. Instead, the xenbaked is a user-space process that can catch and process the events produced by the xentrace to extract useful information. This information allows us to derive how the time is spent by each domain, which could be displayed and logged by the xenmon at the front-end.

Since *XenMon* is designed for real-time display, it cannot profile VM executions over a period of time. Also, *XenMon* can only gather the performance data from the first CPU, which is not sufficient for our requirements. To address these issues, we modified *XenMon* to collect *all* the CPU data, and store it in a file for subsequent profiling purposes. To improve the efficiency of the profiling, the modified *XenMon* is configured to access VM information with sampling techniques, whose frequency is set based on the rule that the loss of performance caused by the frequent read and write operations is minimized. The modified *XenMon* samples the performance data from a cache (a shared memory region) over time, which may result in profiling redundant data in the *XenMon*. We address this issue by periodically cleaning the cache.

The modified *XenMon* leverages the xenbaked to extract `gotten_time` and `blocked_time` information for the analysis on the types of VMs. The `gotten_time` specifies a percentage of time that a VM uses CPU over a period of time while the `blocked_time` indicates the percentage of time a VM is blocked on I/O events.

An I/O-intensive VM bears some similarity with an I/O-intensive process in Linux, that is, the CPU usage time is usually short, and the waiting time for I/O event is always long. We can analyze a VM type according to its `gotten_time` and the `blocked_time`. In particular, for a VM with CPU-intensive workloads, its ratio of `blocked_time/gotten_time` is much smaller than that of a VM with I/O-intensive workloads

$$DI_2 = \frac{\sum_{i=0}^{NUM} blocked\_time_i / gotten\_time_i}{Num}. \quad (1)$$

```
struct net_stat {
        const char *dev;
        int up;
        long long last_read_recv, last_read_trans; //last read total num
        long long recv, trans; //real total num
        double recv_speed, trans_speed;
};
```

Fig. 6. net_stat structure.

With this consideration, we can define the *degree of I/O intensity* (DI_2) in Eq. (1) to judge the type of each VM, which is an average ratio across all the counts Num. The DI_2 value of an I/O-intensive VM is typically greater than that of a CPU-intensive VM. As such, it reflects the trend of a virtual machine over a period of time to be either I/O-intensive or CPU-intensive. *Raccoon* takes advantage of this value to manage the VMs in the two queues.

3.5 Netinfo Module

Although the use of improved *XenMon*, in conjunction with the concept of DI_2, can recognize the virtual machines with I/O intensive workloads, it is unable to distinguish between the VMs with a large volume of disk access operations and the VMs with a large amount of network transmissions since both kinds of VMs have similar DI_2 values. In order to accurately allocate the VF resources (only available to network traffic), we need to recognize the network I/O-intensive virtual machines. For the sake of easy presentation, hereinafter, we use the I/O-intensive and the network I/O intensive interchangeably.

Raccoon develops a new *Netinfo* module to collect the network traffic information of each virtual machine by periodically reading file `/proc/net/dev`, which contains the information regarding each configured network interface. It keeps the collected information in a designed data structure `net_stat` as shown in Fig. 6. In this structure, `last_read_trans` and `last_read_recv` record the total number of packets that have been sent and received, respectively, before the last sampling. The total number of sent and received packets so far are accumulated from the file in respective `trans` and `recv` fields, which are deducted by `last_read_trans` and `last_read_recv`, respectively, to get the sent and received traffic volumes for the current sampling cycle. After obtaining the sampling results, *Netinfo* places them into the xenstore in Xen whereby the improved scheduler can facilitate its VM scheduling. This is accomplished by determining the type of each VM based on a concept of *Intensive_factor* (IF), which is defined as follows:

$$IF = \alpha * Traffic\_freq + (1 - \alpha) * Traffic\_avg, \qquad (2)$$

where *Traffic_freq*=(*Traffic_num*)/*Total*, and *Traffic_avg*= ($\sum_{i=1}^{Total}$*Traffic*)/*Total*(*Total* is the total number of samples and *Traffic_num* indicates the number of samples that have network transmissions), and α is a combination parameter between 0 and 1, which is used to balance the weights of *Traffic_freq* and *Traffic_avg*in definition (2). *Traffic_freq* reflects how frequently a VM makes network transmissions while *Traffic_avg* represents the weighted average network traffic across the *Total* samples. The frequency and the average size of traffic-flow reflect how the VM is busy on the networks.

3.6 VF Scheduler

The *VF Scheduler* is designed as a module in Dom0, which is invoked by communicating with the Credit scheduler to configure the VFs via the *SR-IOV Manager* (IOVM). Given the two metrics of DI_2 (in the last sub-section) and *IF*, the *VF Scheduler* can combine them to judge the type of each VM in the sense that it can not only distinguish between I/O-sparse VMs and I/O intensive VMs, but also further discriminate those VMs with network intensive workloads from the later ones.

More specifically, by comparing the DI_2 value with a threshold value set according to the environments, *Raccoon* can recognize the I/O-sparse virtual machines in liner time. In particular, when the DI_2 of a VM in *PQ* is lower than that of certain VM in *GQ*, *Raccoon* will move the VM from *PQ* to *GQ*, and in the meantime, the *VF Scheduler* determines whether the granted VFs are deprived or not, depending on how they are used (from *XenMon* and *Netinfo*). This is achieved by using the dynamic VF (de)allocation, and the network card switching between vNIC and VFs as presented in Section 3.3.1.

Unlike DI_2, the value of *IF* reflects how a virtual machine is busy on its network interfaces. A VM with network I/O-intensive workloads would in general have a large *IF*, while a VM with low network-I/O workloads (even if is disk I/O-intensive) will have a small *IF*. By comparing the *IFs*, the type of the VMs can be correctly identified, and the *IFs* of the VMs are sorted in descending order for the VF allocation. In this way, *Raccoon* would ensure that the I/O-intensive VMs would always have the VF resources for the desired bandwidth within $O(\log n)$ time to improve the overall performance, where n is the number of I/O-intensive VMs in *PQ*.

Remarks: Compared to hybrid VMs, we intend to allocate the resources to I/O-intensive VMs. If we only use "IF", we then cannot distinguish the VMs with heavy I/O workloads from those hybrid VMs that are heavy in both CPU and I/O workloads. Instead, if we use DI_2, we can reach this goal. Therefore, we combine both metrics to allocate the VFs resources.

4 Performance Evaluations

In this section, we evaluate the performance of *Raccoon* via intensive empirical studies. We first introduce the experimental setup, and then, based on some standard network benchmarks, assess *Raccoon* with respect to its optimization on the network bandwidth throughput and network latency. Finally, we measure the impact of *Raccoon* on the I/O performance of database storage system in terms of the OLTP response latency and the OLTP IOPS throughput, and study the dynamic scheduling behaviors regarding the workloads with different hybrid characteristics. Our numerical results showed that the proposed *Raccoon* can remarkably improve the overall performance of virtualization platforms with hybrid I/O workload natures.

4.1 Experimental Setup

Our experimental setup is a cluster that consists of three hosts connected via a Brocade 800 switch. `Host1` is used as

TABLE 1
Hardware and Software Configuration

CPU	Intel Xeon E5540
RAM	DDR ECC RG 64 GB/16GB
NIC	Intel 82,599 10 Gbps
VMM	XEN-4.2.2
Dom0 OS	CentOS 5.5 (Linux-3.6.3 SMP)
VM OS	CentOS 6.0
VM RAM	1 GB
VM NIC	Intel 82,599 VF

the shared storage device, which is a Dell R720 machine running Linux 2.6.32 SMP kernels. `Host2` and `Host3` have a similar configuration shown in[2] Table 1 where `Host2` has 64 GB RAM and `Host3` has `16 GB`. Both hosts mount the storage in `Host1` over the iSCSI protocol.

In our experiments, `Host2` always takes initiative to send requests, and `Host3` is the target. Although the scale of the physical cluster is small, it still satisfies our requirements since our concern is the performance of VM scheduling, and the number of VMs on each machine can be varied from `1` to `63`, each could be installed CPU-intensive, I/O-intensive or hybrid workloads. We use different tools to mimic each kind of workloads, for example, we use *Lookbusy* [24] for CPU-intensive workloads, *Netperf* [25] for I/O-intensive workloads, and the combination of *Lookbusy* and *Netperf* for hybrid workloads. Addtionally, to further validate our findings, we also employ *ApacheBench* [26], *Httperf* [27] as well as a database tool as benchmarks in our experiments. Given the relatively large number of VFs supported by each port, in all the experiments we only use one port in the Intel 82,599 network card (a dual-port SR-IOV) to test the benchmarks.

4.2 Network Throughput

We first evaluate the improved bandwidth performance using an often-used tool *Netperf*, and show that *Raccoon* can not only enhance the scalability but also guarantee the performance of I/O-intensive virtual machines. The *Netperf* is one of the popular tools for the evaluation of network performance. It is adopted for each VM running on `Host2` to open a TCP connection with the remote `Host3` to carry out the bandwidth experiments. The results of *Netperf* are shown in Fig. 7, the number of virtual machines is varied from `1` to `63` and the bandwidth is measured in Gpbs. The results show that with *Raccoon*, the bandwidth can be maximized to a stable level, which is constantly better than its default curve, moreover, it even increases up to `90` percent, from `5799.92` Mbps to `9329.78` Mbps, when there are `63` virtual machines. These improvements can be clearly attributed to the design of *Raccoon* where the network I/O-intensive VMs are always prioritized to facilitate the network performance by using granted VFs.

As for the throughput behavior of the *Before Opt*, it fluctuates with the number of VMs being increased, and has a tendency to gradually enlarge the gap with respect to the *After Opt*. Not surprisingly, as the number of VMs is increased, the network-intensive workloads would bring in high contention to the vNet resources, which, together with the overhead of scheduling and interrupt processing, could dramatically compromise the total throughput of the system, more worse, also incur the fluctuation, an uncertain bandwidth utilization.

2. It is configured as 2-socket of 2.53 GHZ, 4-cores-per-socket, and 2-threads-per-core

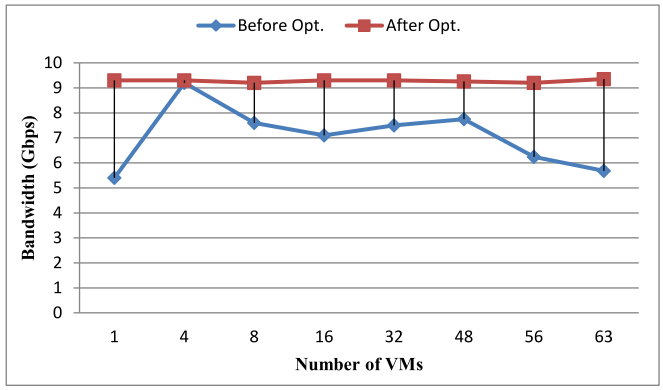

Fig. 7. Netperf bandwidth.

To further validate our findings, we also leverage ApacheBench, another tool attached to the *Apache*, to evaluate the bandwidth of HTTP server when multiple concurrent requests are made at the same time. In this experiment, the *Apache* server is installed in a remote virtual machine on `Host3`, and the *ApacheBench* is running as a client on a local virtual machine on `Host2` to repeatedly send requests from two concurrent threads to the HTTP server for accessing web pages in size of `7KB`. Fig. 8 shows the results of the *ApacheBench*, from which we can see that as with the case in the *Netperf*, compared to the default case, the bandwidth is consistently improved, even nearly `100` percent from `114.088` to `192.523` Mbps when `56` VMs are available. By following the same arguments, these results confirm our previous conclusions that *Raccoon* can enhance the scalability while attaining the performance of I/O-intensive virtual machines.

Unlike the case in *Netperf*, for the bandwidth of HTTP server, since it is not that network intensive, compared to the used Netperf benchmarks, the throughput performance of the HTTP server is highly predictable.

4.3 Network Latency

4.3.1 *Httperf Benchmark*

Next, we evaluate the impact of *Raccoon* on the network latency of web server. To this end, we follow the configuration in the last experiment, and use Httperf as the

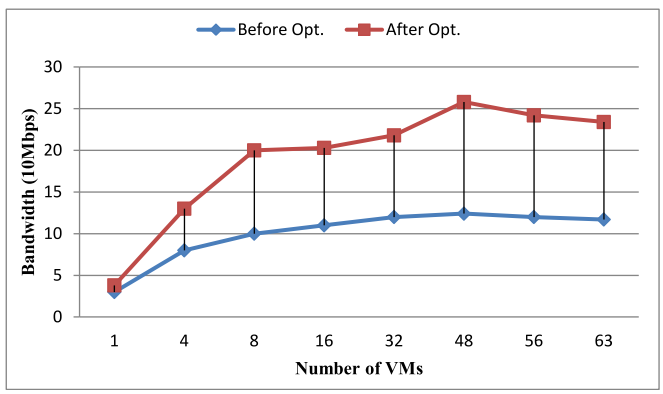

Fig. 8. ApacheBench bandwidth.

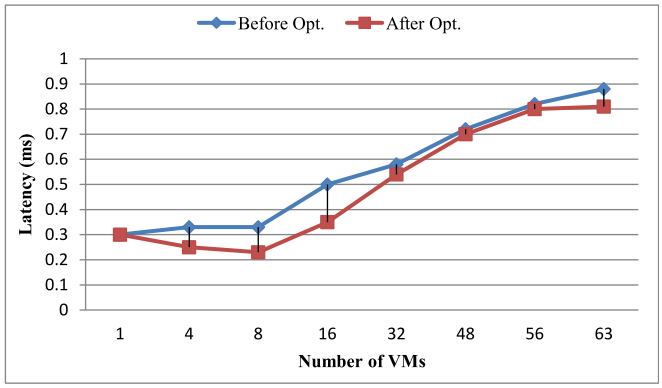

Fig. 9. Latency comparisons under different number of VMs with a fixed request rate of 50/s.

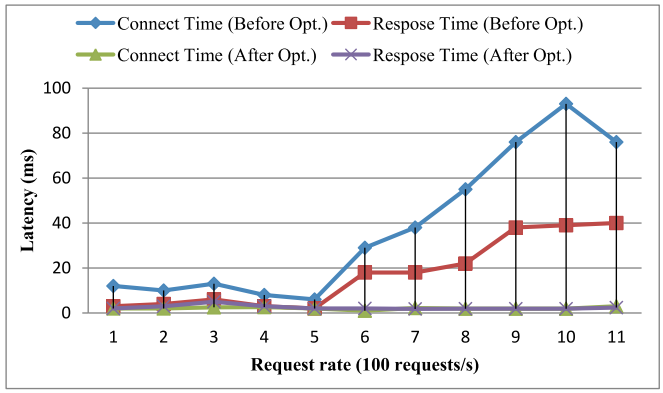

Fig. 10. Comparison of server performance in terms of connection and response times.

benchmark since it can generate various HTTP workloads to measure the server performance.

In the first experiment, we enable the *Httperf* to generate 50 requests between the client and the server within 1 second for the latency measurements over different number of VMs. Fig. 9 shows the average latency of *Raccoon* is consistently better than the defaults, up to 8.7 percent improved, compared to the value in default case when there are 63 VMs.

To validate these observations, by using the same configuration, we then evaluate the latencies of requests in terms of the connection and response time. Here, the connection time is calculated as the elapsed time that a computer is logged on to a remote server, as through an Internet Service Provider (ISP), and the response time is defined as the total service time of the requests, which is the aggregation of client time, network time, and server time.

We fixed the number of VMs as four and vary the request rates from 100/s to 1100/s.[3] Fig. 10 shows the latencies of connection and response time for the *Apache* VM to serve the requests at various rates. From this figure, one can observe that *Raccoon* is very effective to reduce both the connection time and the response time of the requests. More specifically, according to the figure, when the request rate is less than 900/s, *Raccoon* can reduce about 60 to 80 percent of the connection and response time, and this value is further increased up to 90 percent when the request rate is greater than 900/s. The rationale behind this observation is that the I/O request time is not uniformly distributed, rather, it is concentrated with a high probability in a short period of time after some VMs are adaptively scheduled and granted the VFs by *Raccoon*. This demonstrates that, to reduce the delay of I/O responses, it is better to shorten the time slice of VM scheduling so that the execution opportunities for each VM could be increased.

4.3.2 Ping Round-Trip Delay

Unlike the last experiment, which focuses on the mixed workloads in web server accesses, in this experiment, we measure the *round-trip delay* (RTD) of *Ping* to evaluate the network latency for small requests. The RTD is defined as the time required for a *Ping* packet to travel from a specific source to a specific destination and back again.

3. Since the HTTP service is a mixed workload, the *Lookbusy* is no longer run in the the target VM.

To measure the *RTD*, we first create three VMs, each with two virtual CPUs. Among these VMs, two of them run the *Lookbusy tasks*, while the remaining one runs on a remote *netserver*, hosting an Apache server and a Mysql sever. In Fig. 11, we compare the *RTD* performance before and after applying *Raccoon*. From the figure, we can find that most of the delays are very small (less than 100 ms in our experiment), and as a result, the performance and stability is better than those of the default algorithm. We attribute this phenomenon to *Raccoon* that allows more I/O requests to be handled by caching for the *RTD* reductions.

4.4 Database Performance

In this section, we leverage *Oracle ORION* (Oracle I/O Calibration Tool) tool to evaluate the performance of *Raccoon* on database storage systems, which are I/O-intensive, and different from our previous experiments that are mostly from a web server point of view. As opposed to the previous benchmarks, the ORION tool can generate diverse database I/O workloads, including OLT and data warehouse, which, in conjunction with the web server results, can help us gain insight into the impact of *Raccoon* on the overall service systems.

The experimental setups is similar to those for disk I/O testing, but with an important difference that the target VM runs the *ORION*, instead of *dd* command, to measure the database I/O performance, represented by *latency* and *IOPS*. Our experimental results are shown in Figs. 12 and 13, from which we can observe that, with *Raccoon*, under various load stress levels, the latency of OLTP is reduced 10 percent on average and the *IOPS* throughput is improved 10 percent

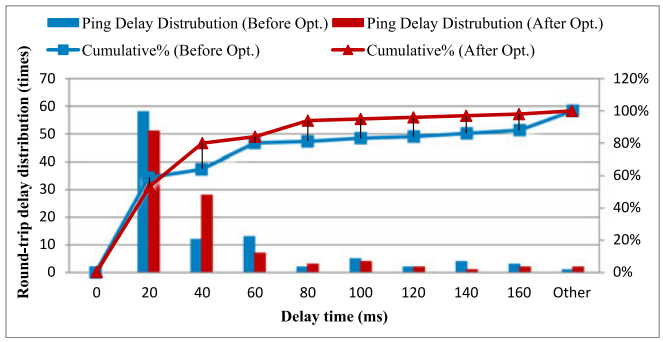

Fig. 11. Ping round-trip delay distribution.

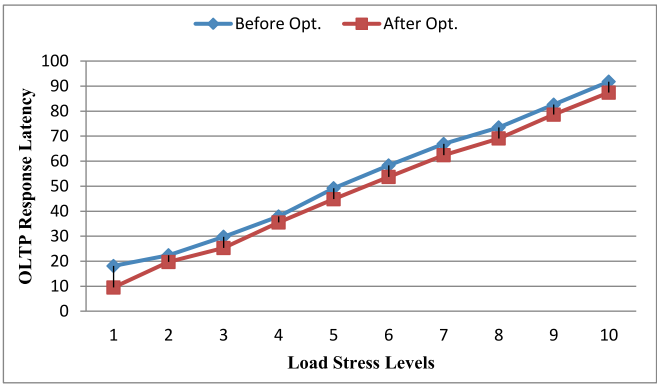

Fig. 12. Comparison of OLTP latency.

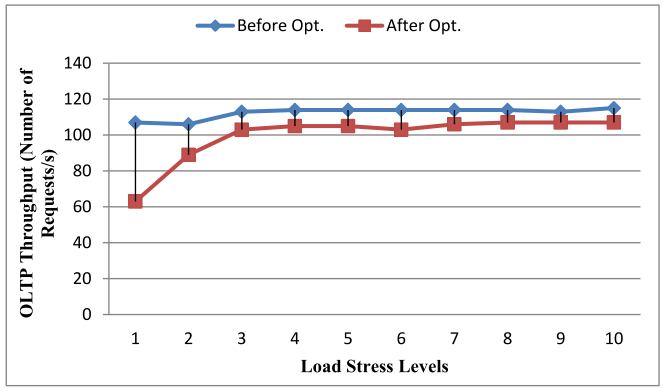

Fig. 13. Comparison of OLTP IOPS.

in most cases, compared to the default scheduler. However, the improvements in this test, compared with the web server results, are not that pronounced. This is because the portion of CPU computation in most OLTP applications is relatively large (i.e., resulting in CPU-intensive VMs), which compromise the effectiveness of *Raccoon* to a certain extent.

4.5 Dynamic Hybrid Workloads

On the contrary to the previous studies, which more or less focus on the VMs with specific and relatively static workloads, in this section, we evaluate the performance of *Raccoon* under dynamic hybrid workloads, which may give us a more holistic view of *Raccoon*. The attributes of the workloads in terms of CPU-intensive, I/O-intensive as well as mixed-type virtual machines are summarized in Table 2. According to the table, there are totally 16 VMs, among which VM16 is the reference VM, which is idle initially, and the other VMs, VM1-VM15, are evenly divided into three groups. The VMs in each group are sorted in ascending order of the VMID and run benchmark programs of *Lookbusy*, *Netperf* or both in 20, 40, 60, 80 and 100 percent of its total active time, respectively.

From Table 2, we can see that the DI_2 value of CPU-intensive VM is usually less than that of I/O-intensive VM, and moreover, the longer the CPU-intensive benchmarks run, the smaller the DI_2 value is. For the hybrid VMs, those with high CPU-intensive workloads in general have equally small DI_2 values. As such, by using the DI_2 values, *Raccoon* can tell the CPU-intensive VMs and the I/O-sparse VMs from the hybrid VMs. In contrast, as far as the *IF* is concerned, the I/O-intensive VMs usually have large values of *IF*, say VM10 in the table.

In order to verify that *Raccoon* can distinguish the I/O-intensive VMs and dynamically grant the VFs to them, we equip the test server with 8 VFs, and allow the scheduler to be triggered every 20 minutes. Furthermore, we set *TRAFFIC_FREQ* to 80, that is, the VMs spending more than 80 percent of time on network transmission are deemed as being with a very I/O *IF*, whose calculation is weighted by the parameter α set by 0.8 in our experiments.

Fig. 14 depicts the states of the virtual machines along the time line where the workloads of the virtual machine are changed surrounding the time points of interest at 0 and 20. In the graph, the x-axis is the execution time, and the y-axis indicates what percentage of the execution time is used by the benchmark programs, either *Netperf* represented by the dashed lines or *Lookbusy* by the solid lines. According to the graph, VM1 has been running the *Netperf* with the time percentage of 35 percent. Before time point 0, each of VM1-VM8 is granted a VF, and from time 0 to 20, VM1 has the shortest running time on *Netperf* among these VMs. There is a scheduling occurred at time 20 because in VM9, the running time of *Netperf* is 60 percent and the *IF* is 79027.4, while for VM1, the I/O-intensity of its workloads, compared with those of other VMs (VM2-VM8), is lower in terms of the proportion of I/O time and *IF* (58319.6 is the minimum), VM9 will be granted the VF that is deprived from VM1. Later, in the moment of 40, there are 3 VFs whose owners are changed, that is, the VFs in VM4 and VM9 are removed as their workloads are changed from I/O-intensive to CPU-intensive, and their *IFs* are 0. Since the time

TABLE 2
CPU-Intensive, I/O-Intensive and Mixed Property Values

DomID	1	2	3	4	5	6	7	8
Traffic_freq	0	0	0	0	0	19	34	56
DI_2	3,291.28	9,251.81	921.862	29.9489	34.582	7,384.23	5,628.71	5,396.86
Traffic_avg	0	0	0	0	0	187,965	326,492	517,295
IF	0	0	0	0	0	37,385.3	65,827.9	110,838.2
DomID	9	10	11	12	13	14	15	16
Traffic_freq	78	98	17	39	59	79	96	0
DI_2	3,918.63	4,385.75	3,723.57	4,491.839	689.572	285.9402	389.3761	3,971.32
Traffic_avg	692,703	892,601	201,876	418,351	581,063	802,873	895,520	0
IF	158,362	170,329	24,032	80,256.6	118,932	138,549.4	192,650	0

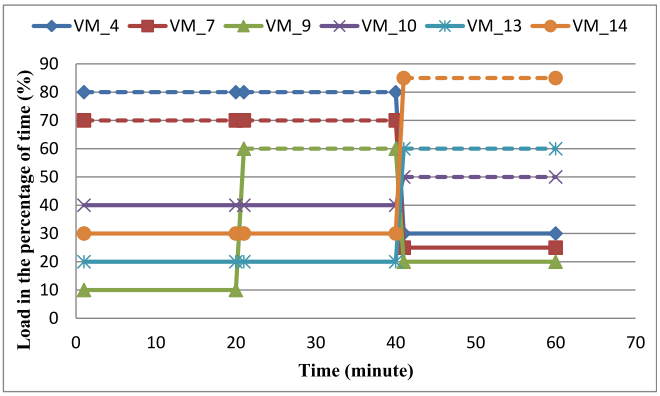

Fig. 14. Virtual machine load changes over time.

proportion of VM7 on the benchmark is decreased from 60 to 25 percent and its I/O-intensity is low, the VF in VM7 is also deprived (its *IF* is 16,037.4). As a consequence, VM10, VM13, and VM14, due to their relatively long-lasting, high I/O-intensive workloads after time 20, will obtain the VFs, their *IFs* are 79205.8,128303.2, and 90251.4, respectively. We can see from these results that *Raccoon* is always able to dynamically grant the VF resources to those VMs with a high degree of I/O-intensive workloads.

5 Related Work

There exist lots of work on optimization of VM schduling in virtual environments. Rao et al. [11] present a *Bias Random VCPU Migration* (BRM) algorithm, which is a NUMA awareness for the virtual machine scheduling to minimize the system-wide uncore penalty. Xu et al. describe *vSlicer* [13] and *vTurbo* [28], *vSlicer* is a hypervisor-level technique that schedules each latency-sensitive VM more frequently but with a smaller micro time slice while *vTurbo* involves a VM scheduling policy that enforces fair sharing between VMs. Unlike the previous studies, Zhuravlev et al. [29] discuss the issue of different classification schemes for scheduling algorithms in multicore processors. The scheduling algorithms they implemented aim to allocate VMs among the cores so that the miss rates could be spread evenly. Although these studies are more or less effective to the VM scheduling, they are not squarely designed with focus on the I/O efficiency of VMs.

Improving VM scheduler in general and Credit scheduler in particular for multicore processors to assure the timely response of the VMs with diverse I/O workloads has long been an active research topic [1], [10], [30], [31], [32], [33], [34]. HarÉl et al. [34] design *ELVIS*, a low-overhead and scalable I/O interposition mechanism for paravirtual I/O Systems. To support the I/O intensive workloads, Landau et al. [17] propose an alternative virtualization model for multicore systems, and Kim et al. [35] introduce a communication-driven scheduling that controls time-sharing in response to interprocessor interrupts between VCPUs. Most recently, Zeng et al. [12], [36] present *XCollopt*, which performs a collection of novel optimizations that coordinate the Credit scheduler and underlying I/O virtualization to improve the performance of I/O-latency sensitive applications on multicores. Although these methods are I/O effective in some way, they are mostly software based, and have inherent limitation to further improve the I/O performance for VMs. As such, in practice network I/O is still an obstacle for those workloads with heavy network traffic, which precludes the use of virtualization in many applications.

There has been a wide range of research to exploit the hardware-based virtualization technologies for I/O performance optimization. For example, Gordon et al. [37] propose an exitless paravirtual I/O model that enables the guests and the hypervisor running on distinct cores to exchange exitless notifications instead of the costly exit-based notifications. In contrast, Dong et al. [9], [38] describe the design and implementation of SR-IOV NIC in Xen while Zhai et al. [39] propose a virtual hotplug technology, in conjunction with the bonding driver, to solve the live migration issues for the SR-IOV based VMs. A similar idea was also adopted by Radhika et al. [40] in design of *FasTrak* that leverages the software-defined network (SDN) technology to enable express lanes for those rule-specified traffic flows in multi-tenant data centers. The design of *Raccoon* is informed by the above works, but with a distinct feature to achieve a workload-aware VM scheduling by dynamically distributing the bandwidth resources among the hybrid VMs.

Recently, Guan et al. [4] leverage the SR-IOV to design a workload-aware Credit scheduler for improving network I/O performance in virtualization environments. The key idea of their design is allowing the Credit scheduler to adjust the total number of Credits, and grant those I/O-intensive domains with extra credits that CPU-intensive domains are willing to share for accelerating the I/O responsiveness. This design can be viewed as CPU-oriented since the total CPU time is re-distributed upon the changes of credit allocation. In contrast, *Raccoon* is VF resource-oriented as it adopts an adaptive strategy to dynamically allocate VF resources among the hybrid workloads to maximize the overall I/O efficiency, instead of changing the credit distribution. As a result, *Raccoon* has minimal impact on the implementation of the Credit scheduler, and thus, more generic to be integrated with other VM schedulers for handling diverse applications. Of course, *Raccoon* could be further improved with the ideas from [4], given their orthogonal designs.

6 Conclusions

In this paper, We presented a framework, called *Raccoon*, to optimize the network I/O allocation for workload-aware VM scheduling algorithm in virtual environments. *Raccoon* incorporates the strengths of paravirtual I/O and SR-IOV techniques to enable the workload-aware assignments of two types of virtual I/O resources so that the I/O-intensive VMs will always be granted the VFs as "express lanes" to facilitate their I/O operations. To this goal, the paravirtual Network Interface Cards (vNICs), as the default resources, are allocated to every VM while the SR-IOV VFs are only granted to those I/O-intensive VMs. The VF (de)allocation implemented by ACPI Hotplug is dynamically adaptive to the changes of VMs' workload natures for overall performance improvements. To achieve this goal, *Raccoon* leverages the modified *XenMon* to identify the workload types of each VM in real time, and adopts the bonding driver technique to implement an efficient I/O workload-aware VM

scheduling algorithm. Our experimental results revealed that *Raccoon* can retain all the benefits of para-virtual I/O and SR-IOV techniques to improve the overall system performance for hybrid workloads.

Acknowledgments

This work was sponsored in part by the China National Basic Research Program (973 Program, No. 2015CB352400), the National Natural Science Foundation of China (NSFC) under grant No. 61472153, No. 61672513, and No. 61572377, the National Science Foundation of Hubei Province (2015CFB192), Science and Technology Planning Project of Guangdong Province (2015B010129011, 2016A030313183), and Shenzhen Overseas High-Caliber Personnel Innovation Funds (KQCX20140521115045446). Additionally, we would also like to express our gratitude to Ju Zhang, Yu Li, Yuqi Zuo and Tao Guo for their contributions to this research. Yang Wang is the corresponding author of this paper.

References

[1] A. Gordon, et al., "ELI: Bare-metal performance for I/O virtualization," in *Proc. 17th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2012, pp. 411–422.
[2] M. Ben-Yehuda , M. Factor, E. Rom, A. Traeger, E. Borovik, and B.-A. Yassour, "Adding advanced storage controller functionality via low-overhead virtualization," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 1–8.
[3] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. D. Panda, "SR-IOV support for virtualization on InfiniBand clusters: Early experience," in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, May 2013, pp. 385–392.
[4] H. Guan, R. Ma, and J. Li, "Workload-aware credit scheduler for improving network I/O performance in virtualization environment," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 130–142, Apr.–Jun. 2014.
[5] J. Pfefferle, P. Stuedi, A. Trivedi, B. Metzler, I. Koltsidas, and T. R. Gross, "A hybrid I/O virtualization framework for RDMA-capable network interfaces," in *Proc. 11th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2015, pp. 17–30.
[6] F.-F. Zhou, R.-H. Ma, J. Li, L.-X. Chen, W.-D. Qiu, and H.-B. Guan, "Optimizations for high performance network virtualization," *J. Comput. Sci. Technol.*, vol. 31, no. 1, pp. 107–116, 2016.
[7] P. Barham, et al., "Xen and the art of virtualization," in *Proc. 19th ACM Symp. Operating Syst. Principles*, 2003, pp. 164–177.
[8] PCI I/O virtualization. PCI-SIG. 2013. [Online]. Available: http://www.pcisig.com/specifications/iov/
[9] 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.
[10] 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.*, Mar. 2008, pp. 1–10.
[11] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu, "Optimizing virtual machine scheduling in NUMA multicore systems," in *Proc. 19th Int. Symp. High Perform. Comput. Archit.*, Feb. 2013, pp. 306–317.
[12] L. Zeng, Y. Wang, W. Shi, and D. Feng, "An improved Xen credit scheduler for I/O latency-sensitive applications on multicores," in *Proc. Int. Conf. Cloud Comput. Big Data*, 2013, pp. 267–274.
[13] C. Xu, S. Gamage, P. N. Rao, A. Kangarlou, and R. R. Kompella, "vSlicer: Latency-aware virtual machine scheduling via differentiated-frequency CPU slicing," in *Proc. 21st Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2012, pp. 3–14.
[14] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.
[15] J. Guo, et al., "On efficient bandwidth allocation for traffic variability in datacenters," in *Proc. IEEE INFOCOM*, 2014, pp. 1572–1580.
[16] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2010, pp. 1–12.
[17] A. Landau, M. Ben-Yehuday , and A. Gordon, "SplitX: Split guest/hypervisor execution on multi-core," in *Proc. 1st USENIX Workshop I/O Virtualization*, 2011, pp. 1–7.
[18] Y. Dong, et al., "Improving virtualization performance and scalability with advanced hardware accelerations," in *Proc. IEEE Int. Symp. Workload Characterization*, 2010, pp. 1–10.
[19] R. Russell, "Virtio: Towards a de-facto standard for virtual I/O devices," *SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 95–103, Jul. 2008.
[20] A. A. Sani, K. Boos, S. Qin, and L. Zhong, "I/O paravirtualization at the device file boundary," in *Proc. 19th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2014, pp. 319–332.
[21] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, "Towards high-quality I/O virtualization," in *Proc. Israeli Exp. Syst. Conf.*, 2009, Art. no. 12.
[22] J. Suzuki, Y. Hidaka, J. Higuchi, T. Baba, N. Kami, and T. Yoshikawa, "Multi-root share of single-root I/O virtualization (SR-IOV) compliant PCI express device," in *Proc. IEEE 18th Annu. Symp. High Perform. Interconnects*, 2010, pp. 25–31.
[23] AMD-V. 2013. [Online]. Available: http://sites.amd.com/cn/business/it-solutions/virtualization/Pages/amd-% v.aspx
[24] lookbusy–a synthetic load generator, Open Source Development Labs., 2016. [Online]. Available: http://www.devin.com/lookbusy/
[25] Netperf. 2016. [Online]. Available: http://www.netperf.org/netperf
[26] 2016. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html
[27] 2016. [Online]. Available: http://www.labs.hpe.com/research/linux/httperf
[28] C. Xu, S. Gamage, H. Lu, R. Kompella, and D. Xu, "vTurbo: Accelerating virtual machine I/O processing using designated turbo-sliced core," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2013, pp. 243–254.
[29] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. 15th Int. Conf. Archit. Support Program. Languages Operating Syst.*, Mar. 2010, pp. 129–142.
[30] 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.*, Mar. 2009, pp. 101–110.
[31] A. Burtsev, K. Srinivasan, P. Radhakrishnan, L. N. Bairavasundaram, K. Voruganti, and G. R. Goodson, "Fido: Fast inter-virtual-machine communication for enterprise appliances," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 1–14.
[32] 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.*, Sep. 2010, pp. 287–296.
[33] 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.
[34] N. HarÉl, A. Gordon, and A. Landau, "Efficient and scalable paravirtual I/O system," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2013, pp. 231–242.
[35] H. Kim, S. Kim, J. Jeong, J. Lee, and S. Maeng, "Demand-based coordinated scheduling for SMP VMs," in *Proc. 18th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2013, pp. 369–380.
[36] L. Zeng, Y. Wang, D. Feng, and K. Kent, "XCollOpts: A novel improvement of network virtualization in Xen for I/O-latency sensitive applications on multicores," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 2, pp. 163–175, Jun. 2015.
[37] A. Gordon, N. Har'El, A. Landau, M. Ben-Yehuda , and A. Traeger, "Towards exitless and efficient paravirtual I/O," in *Proc. 5th Annu. Int. Syst. Storage Conf.*, 2012, pp. 10:1–10:6.
[38] Y. Dong, Z. Yu, and G. Rose, "SR-IOV networking in Xen: Architecture, design and implementation," in *Proc. 1st Conf. I/O Virtualization*, 2008, pp. 10–10.
[39] E. Zhai, G. D. Cummings, and Y. Dong, "Live migration with pass-through device for Linux VM," in *Proc. Ottawa Linux Symp.*, 2008, pp. 261–268.
[40] R. N. Mysore, G. Porter, and A. Vahdat, "Fastrak: Enabling express lanes in multi-tenant data centers," in *Proc. 9th ACM Conf. Emerging Netw. Experiments Technol.*, 2013, pp. 139–150.

Lingfang Zeng received the BS degree in applied computer from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2000, the MS degree in applied computer from China University of Geoscience, China, in 2003, and the PhD degree in computer architecture, from HUST, in 2006. He was research fellow for more than four years in the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, during 2007-2008 and 2010-2013. He is currently with Wuhan National Lab for Optoelectronics, and School of Computer, HUST, as an associate professor. He publishes more than 40 papers in major journals and conferences. He is a member of the IEEE.

Yang Wang received the BSc degree in applied mathematics from Ocean University of China, in 1989, and the MSc computer science from Carleton University, in 2001, and the PhD degree in computer science from the University of Alberta, Canada, in 2008. He is currently in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, as a professor. His research interests include cloud computing, big data analytics, and Java virtual machine on multicores. He is an Alberta Industry R&D Associate (2009-2011), and a Canadian Fulbright Scholar (2014-2015).

Xiaopeng Fan received the BE and ME degrees in computer science from Xidian University, in 2001 and 2004, respectively, and the PhD degree in computer science from Hong Kong Polytechnic University, in 2010. He is currently an associate professor in the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests include big data analytics, mobile cloud computing, and software engineering. His recent research has focused on big data analytics in urban computing. He has published more than 40 papers in conferences and journals, and served as a TPC member for several conferences. He is a member of the IEEE and the ACM.

Chengzhong Xu received the PhD degree from the University of Hong Kong, in 1993. He is currently a professor of the Department of Electrical and Computer Engineering, Wayne State University. He also holds an adjunct appointment in the Shenzhen Institute of Advanced Technology, Chinese Academy of Science as the director of the Institute of Advanced Computing and Data Engineering. His research interests include parallel and distributed systems and cloud computing. He has published more than 200 papers in journals and conferences with more than 7,000 citations. He was a Best Paper Nominee of the 2013 IEEE High Performance Computer Architecture (HPCA), the 2013 ACM High Performance Distributed Computing (HPDC), and the 2005 International Conference on Parallel Processing (ICPP). He served or serves on a number of journal editorial boards, including the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Cloud Computing*, the *Journal of Parallel and Distributed Computing*, and the *China Science: Information Sciences*. He was a recipient of the Faculty Research Award, Career Development Chair Award, and the Presidents Award for Excellence in Teaching of WSU. He is a fellow of the IEEE due to contributions in resource management in parallel and distributed systems. For more information, visit http://www.ece.eng.wayne.edu/ czxu.