applied sciences

MDPI

Article

A Virtual Network Resource Allocation Framework Based on SR-IOV

Zhiyong Ye, Yuanchang Zhong * and Yingying Wei

School of Microelectronics and Communication Engineering, Chongqing University, Chongqing 400044, China; zy.ye@cqu.edu.cn (Z.Y.); 20161202010@cqu.edu.cn (Y.W.)

* Correspondence: zyc@cqu.edu.cn

Received: 9 December 2018; Accepted: 27 December 2018; Published: 2 January 2019

Abstract: The workload of a data center has the characteristics of complexity and requirement variability. However, in reality, the attributes of network workloads are rarely used by resource schedulers. Failure to dynamically schedule network resources according to workload changes inevitably leads to the inability to achieve optimal throughput and performance when allocating network resources. Therefore, there is an urgent need to design a scheduling framework that can be workload-aware and allocate network resources on demand based on network I/O virtualization. However, in the current mainstream I/O virtualization methods, there is no way to provide workload-aware functions while meeting the performance requirements of virtual machines (VMs). Therefore, we propose a method that can dynamically sense the VM workload to allocate network resources on demand, and can ensure the scalability of the VM while improving the performance of the system. We combine the advantages of I/O para-virtualization and SR-IOV technology, and use a limited number of virtual functions (VFs) to ensure the performance of network-intensive VMs, thereby improving the overall network performance of the system. For non-network-intensive VMs, the scalability of the system is guaranteed by using para-virtualized Network Interface Cards (NICs) which are not limited in number. Furthermore, to be able to allocate the corresponding bandwidth according to the VM's network workload, we hierarchically divide the VF's network bandwidth, and dynamically switch between VF and para-virtualized NICs through the active backup strategy of Bonding Drive and ACPI Hotplug technology to ensure the dynamic allocation of VF. Experiments show that the allocation framework can effectively improve system network performance, in which the average request delay can be reduced by more than 26%, and the system bandwidth throughput rate can be improved by about 5%.

Keywords: SR-IOV; I/O virtualization; workload-aware; VM scheduling; network resource allocation; bandwidth division

1. Introduction

With the rapid development of cloud computing technology, more and more network service centers are developing and migrating to cloud platforms. At the same time, the continuous development of cloud computing has also expanded the complexity and diversity of its service objects, which requires cloud computing service providers to improve the quality, security, and price requirements of cloud services. How to use resources in cloud platforms economically and efficiently is a difficult problem. Modern cloud data centers can effectively solve this problem by using virtualized technology to run multiple virtual servers on a physical server by means of data isolation [1,2]. In the current virtualization technology, the virtualization technology of processor and memory is relatively mature, and I/O virtualization technology has always been the focus of research [3–8]. The network as a very important I/O resource in the data center is usually the key to the performance of the entire

system [9,10]. At the same time, due to the scarcity and high competitiveness of network resources in the data center, the reasonable scheduling of network resources and the improvement of bandwidth use of data center networks urgently need to be solved.

Network I/O virtualization is an effective means of solving network contention and improving network performance. However, although the workload of the data center has the characteristics of complexity and requirement variability, in reality, the attributes of the network workload are rarely used by the resource scheduler. Failure to dynamically schedule network resources according to workload changes inevitably leads to inability to achieve optimal throughput and performance when allocating network resources. Designing a scheduling framework that can be workload-aware and allocate network resources on demand based on network I/O virtualization is urgently needed. Therefore, during the running of the application, the resource scheduler must be required to dynamically allocate network resources to the service load on demand.

In general, I/O virtualization models can be divided into two categories: *software-based I/O virtualization* and *hardware-based I/O virtualization*. Software-based I/O virtualization can be further divided into *full virtualization* and *para-virtualization* [1,2]. Full virtualization is the interception and simulation of all I/O requests using pure software [3,11,12], and because all instructions are software-simulated, its performance is often poor. In addition, para-virtualization is to split the traditional I/O driver into two parts, in which the front-end driver located in the privileged domain manages the front-end driver located in the virtual machine (VM) to access I/O, in order to provide a VM with a similar hardware platform environment. The lightweight simulation has greatly improved the performance of the para-virtualized I/O [13], but because the *virtual machine monitor* (VMM) still interferes with the I/O path of the VM, the performance of the para-virtualized I/O will be limited [13]. To further improve I/O performance, hardware-based I/O virtualization bypasses the VMM's intervention on I/O paths. By using *Pass-through* technology to make the VM occupy a single I/O Device or *Single-Root I/O Virtualization* (SR-IOV) technology [14] to enable one or more VM to occupy multiple lightweight PCIe *virtual functions* (VFs) of I/O devices at the same time, hardware-based I/O virtualization achieves I/O performance close to that of physical machines [4,6,15–17].

Among the several mainstream virtualization methods mentioned above, none of them can provide workload-aware functions while meeting the performance requirements of VMs. Software-based virtualization has good network management and scalability, but its performance is limited because the VMM intervenes in the VM's I/O path. Hardware-based virtualization, whether Pass-through or SR-IOV, bypasses the intervention of VMM and can achieve I/O performance close to physical machines. However, it allows the VM to occupy the I/O device separately, which reduces the hardware reusability, and cannot sense the load change of the VM and dynamically allocate the VF.

Therefore, we propose a method that can dynamically sense the VM workload to allocate network resources on demand, and can ensure the scalability of the VM while improving the performance of the system. We combine the advantages of I/O para-virtualization and SR-IOV technology and a limited number of VFs to ensure the performance of network-intensive VMs, thereby improving the overall network performance of the system. For non-network-intensive VMs, the scalability of the system is guaranteed by using para-virtualized *Network Interface Cards* (NICs) which are not limited in number. To be able to allocate the corresponding bandwidth according to the VM's network workload, we hierarchically divide the VF's network bandwidth, and dynamically switch between VF and para-virtualized NICs through the *Bonding Drive* strategy of *ACPI Hotplug* and *ACPI Hotplug* technology to ensure the dynamic allocation of VF.

Based on this idea, this paper will study the allocation framework and strategy of virtual network I/O resources in a virtualized environment. However, the research in this paper is different from the work of Guan et al. [5]. The framework proposed in this paper is especially used to schedule the network I/O of VMs. It does not modify the scheduling strategy of KVM (Kernel-based Virtual Machine), but is only a supplement to the existing scheduling strategy. The virtual network resource scheduling framework proposed in this paper can also be applied to other VMM platforms.

The prototype system of this paper is deployed in KVM, and the VM in KVM is regarded as a *qemu-kvm* process in Linux *operating system* (OS) at runtime, which is scheduled by the process scheduler as with other processes in Linux system. Because the running state of the VM is transparent to the process scheduler, network-intensive VMs and non-network-intensive VMs will be treated equally in scheduling. In this paper, according to the characteristics of load diversity and variability of VMs, combined with the I/O characteristics of the VM, VF is divided into different levels of network bandwidth, and the bandwidth throughput and network delay are optimized to maximize the performance of cloud network services.

There are also many works in the current research that use different means to improve the network performance of data center I/O virtualization [18]. For example, some studies want to reduce network I/O latency by adding new VM scheduling priorities in the scheduler [19], by equilibrium the load of VCPUs at runtime [20], or by dynamically adjusting VCPU time slice of VMs [21]. There are other works, based on different optimization objectives, using game theory or cybernetics to optimize network bandwidth allocation [22,23]. However, these works are mainly aimed at some specific application or do not take into account the optimization of network bandwidth use and network delay at the same time. Therefore, there are still some limitations in the overall improvement of data center performance.

This paper presents a method to dynamically sense the workload of VMs and allocate virtual network resources on demand. The contributions of this paper are as follows:

1. Based on KVM, this paper designs and implements a method that can combine the advantages of para-virtualization and SR-IOV and can dynamically switch VF and para-virtualized NICs when the VM at runtime. In addition, this method is not only limited to KVM, but also can be applied to other virtualization platforms.
2. By hierarchically dividing the network bandwidth of the VF, the scheduler implemented in this paper can dynamically sense the workload of VMs and allocate virtual network resources to the VM on demand.
3. Comprehensive experiments are conducted to evaluate the allocation framework performance and show that the framework can effectively improve system network performance, in which the average request delay can be reduced by more than 26%, and the system bandwidth throughput rate can be improved by about 5%.

The remainder of this paper is organized as follows. In the second part, we overview the related work. In addition, then we will introduce the background knowledge of KVM I/O virtualization technology and SR-IOV technology and compare the two technologies. In the third part we will present our virtual network resource scheduling framework and propose our scheduling algorithm based on the hierarchical division of VF network bandwidth. In the fifth part we will experimentally evaluate the performance of our virtual network resource scheduling algorithm and the final conclusions will be given in the sixth part.

2. Related Work

In the current virtualization technology, the virtualization technology of CPU and memory is relatively mature, and I/O virtualization becomes the key to control the performance of the whole system. Therefore, there is much research on I/O virtualization [3–8]. However, in general, the research on I/O virtualization basically includes two aspects, in which the first is the optimization of system I/O performance, including the reduction of network I/O delay [24,25], improving network throughput and the reduction of CPU usage, etc.; the other is management of I/O in VM, such as live migration of VMs [24,26], filtering of network packets [27], etc.

The research on I/O para-virtualization is mainly aimed at improving the performance of system I/O [28]. For example, Kaushik et al. [29] proposed a mechanism to reduce the overhead of network virtualization interfaces when using the I/O virtualization driver domain model. The model modifies

Xen to support multi-queue network interfaces, assigns multi-queues of NICs in the privileged domain to a fixed VM, and uses the packet classification function of NICs to eliminate software overhead when packet demultiplexing and copying. This model moves performance bottlenecks from privileged domains to customer domains, increasing scalability and enabling higher data rates. In addition, Bourguiba et al. [9] proposed a packet aggregation and decomposition mechanism for transporting packets between privileged domains and VMs for the I/O channel bottleneck of privileged domains and VM transport packets. The mechanism can aggregate and decompose data packets at the network layer or the data link layer. The data packets are aggregated into a pack in the privileged domain before being transmitted, and then sent to the destination VM to be decomposed into the original data. Experimental evaluation shows that the mechanism can increase the throughput of the system linearly with the increase of the number of VMs and can dynamically adjust the aggregation mechanism to achieve the best trade-off between network delay and throughput. In addition to improving I/O virtualization performance, there are some studies for para-virtualized network management functions. For example, Pfaff et al. [14] proposed the concept of Open vSwitch, a multi-layer virtual switch for managing VMM platform, and introduced in detail the use of Open vSwitch to optimize the operation of VMM platform and the advanced stream classification and caching technology for saving resources.

There are also many researches on SR-IOV technology [30,31], such as Dong et al. [30] expatiates on the framework design and implementation of SR-IOV technology based on Xen. In addition, Liu et al. [31] made a very detailed evaluation of the 10 Gbps NIC with SR-IOV function in terms of performance metrics such as bandwidth and network latency, CPU use, memory access and VM exit in the KVM virtualization environment. To solve the problem of limited online migration of SR-IOV technology, Dong et al. [15] used the active backup mode of *Bonding Driver* in each VM to switch VF and para-virtualized NICs during online migration. In addition, there is a class of research that wants to combine para-virtualization and SR-IOV, mainly to find a compromise by combining the network manageability of para-virtualization technology with the high I/O performance of SR-IOV technology. For example, Dong et al. [32] elaborated on the optimization of advanced hardware acceleration support in hardware virtualization, and proposed a hybrid virtualization solution HYVI, combining the advantages of software virtualization and hardware virtualization to solve performance and manageability. However, this model is a static allocation method when assigning VF to a VM, so the maximum number of VMs that can be supported is limited by the number of VFs in the SR-IOV network card. In addition, the bandwidth allocation between multiple VFs of the model is determined by the hardware arbitration of the SR-IOV network card and cannot be dynamically adjusted according to the load of the VM. In addition, Zeng et al. [33] also proposed an allocation framework that combines I/O para-virtualization and SR-IOV technology to dynamically sense the load of VMs and assign VFs to network-intensive VMs to improve network performance. However, this model uses weighted averaging method to allocate the bandwidth in VF allocation and does not consider the difference of bandwidth requirements between different loads of VMs.

3. Background Knowledge

3.1. I/O Virtualization in KVM

In the traditional I/O driver model, the device driver is used for the interaction between the OS kernel and the physical device, and the OS calls the device driver in the local environment to control the physical device to work. However, in I/O virtualization technology, because there are multiple VMs sharing the same hardware device, the VMM is required to simulate the virtual network card for use by the VM. For example, in the full virtualization technology, the VMM uses pure software to simulate the behavior of the virtual device. The VMM intercepts all I/O requests from the VM, and then invokes the physical driver to interact with the physical device to complete I/O operation. The traditional way of KVM's I/O virtualization is to use QEMU to simulate I/O devices. As shown in Figure 1a, the *I/O Trap Code* located in the KVM module intercepts the I/O request of the VM and

places it on the *I/O sharing page*, and notifies the QEMU program in user space to obtain the I/O request. Then, the QEMU program calls the *Hardware Simulation Code* to simulate the I/O operation and returns the result to the *I/O sharing page*, which the VM can read in the KVM module. QEMU pure software simulation can simulate a variety of hardware devices and can provide a completely consistent environment for VMs and hardware platforms, so compatibility is better.

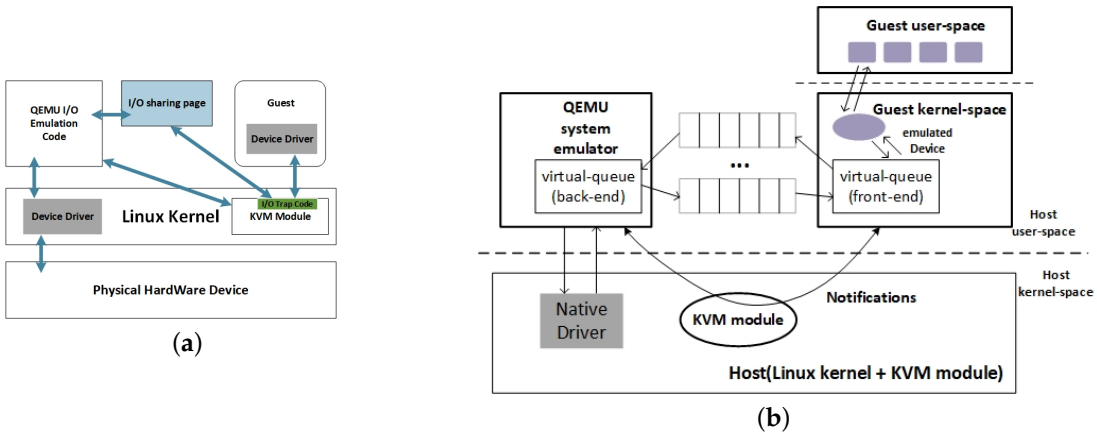

Figure 1. (**a**) I/O virtualization model in QEMU. (**b**) I/O virtualization model in virtio.

The pure software simulation of QEMU which has a poor performance due to its costs needs to intercept all I/O requests of VM. In KVM, the *virtio* para-virtualization framework is generally used to improve I/O performance. The *virtio* para-virtualization framework divides the device driver into two parts by means of front-end and back-end separation. The front-end driver is the driver module of the VM, and the back-end driver is located in the *QEMU* program. In addition, the *virtio* layer and the *virtio-ring* layer are defined between the front-end and back-end drivers for communication between the two. In the para-virtualization framework, only the back-end drivers located in QEMU programs have direct access to hardware, while the front-end drivers in VMs access hardware devices indirectly through the back-end drivers. I/O para-virtualization provides a lightweight analog device to the VM, and all I/O requests of the VM are processed by the QEMU program. Therefore, the *virtio* framework eliminates the process of capture and simulation of I/O request in pure software simulation mode, and the VM can communicate directly with the I/O module of *QEMU*. As shown in Figure 1b, the KVM module notifies the back-end driver of the *QEMU* to retrieve data by interruption and stores it in the virtual queue of the *virtio* layer. Finally, the KVM module notifies the front-end driver of the VM to retrieve data in the *virtual queue*.

3.2. Single-Root I/O Virtualization

To improve I/O performance, hardware-based virtualization is proposed to eliminate the intervention of VMM on I/O paths. Lightweight emulation has greatly improved the performance of para-virtualized I/O compared to full virtualization, but because in the para-virtualized model the VMM still interferes with the I/O path of the VM, so the performance of para-virtualized I/O is still limited. Unlike the para-virtualization model, DMA operations for SR-IOV devices can be directly accessed by IOMMU or device-embedded TLB without the need for a VMM. Therefore, this bypasses the intervention of the VMM on the VM I/O path, and the VM can directly access the SR-IOV hardware device so that can achieve the performance close to the physical machine.

SR-IOV is an I/O virtualization solution supported by PCIe device hardware and developed by the standardization organization PCI-SIG. As shown in Figure 2, each SR-IOV device has one or more *physical function* (PF), and each PF has one or more VF corresponding to them. Generally speaking, the PF driver runs in the privileged domain (where KVM is a QEMU program) and has the right to access all the resources of the SR-IOV hardware, while the VF driver runs in the non-privileged domain (that is, VMs), and the VF is managed and configured by the PF. VF is a lightweight PCIe device and

has the necessary resources to run independently. A VF is like a traditional PCIe device for a VM, with a unique BDF identification number in the PCI bus, so a VF can be bound to a specified VM. In the SR-IOV NIC device, the bandwidth allocation between multiple VFs is based on the hardware arbitration of the NIC device, and the bandwidth is allocated in a weighted average manner. Therefore, when the total bandwidth is limited, if the number of VFs is too large, the effective bandwidth acquired by each VF will be inversely proportional to the total number of configured VFs, which will also cause damage to the system network performance.

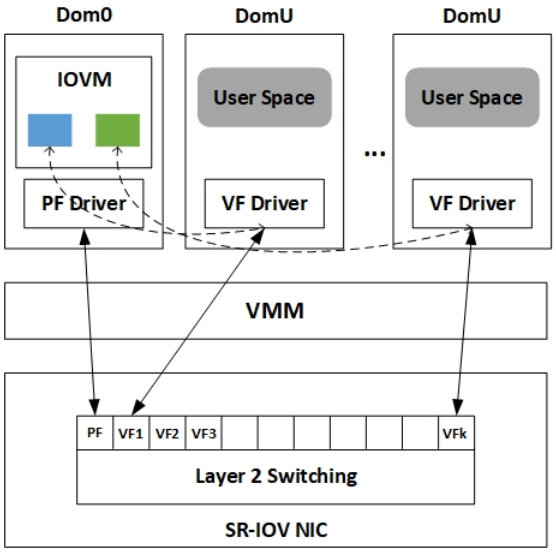

Figure 2. Single-Root I/O Virtualization model.

4. Basic Idea and Implementation

4.1. Basic Ideas

From an abstract point of view, VM can be seen as encapsulating access to a set of physical resources [1]. But if the workload characteristics of each VM is not available when allocating resources, this will lead to a semantic gap between the scheduler and the VM, which will make the operation of the whole system lack coordination. Here, the semantic gap refers to the fact that the network characteristics of the VM cannot be obtained by the VMM, so that the network resources cannot be optimally scheduled. In addition, the VM in KVM is scheduled by the OS as a *qemu-kvm* process [2]. The running attributes of the VM are transparent to the process scheduler, which will inevitably affect the network performance of the network-intensive VM. Therefore, we seek to optimize the network latency and bandwidth throughput of the virtualization platform by combining the para-virtualization technology and the SR-IOV technology to improve the overall network performance of the system.

We observed that even though the VF can greatly reduce the communication delay of the VM, the bandwidth that each VM can use cannot meet the network-intensive VM communication requirements, which is because the effective bandwidth obtained by each VF is inversely proportional to the total number of configured VFs. Therefore, when multiple VMs need network bandwidth at the same time, each VM can only allocate a small amount of bandwidth, which will not meet the performance requirements of the network-intensive VM. To solve this problem, we set the para-virtualized NICs as the default network card for each VM when allocating network resources, while network-intensive VMs work with VFs that are limited in number but have lower network latency. This cannot only ensure that network-intensive VM reduce communication delay and improve network performance, but also make non-network-intensive VM have enough network bandwidth to work, thereby improving the overall use of network bandwidth. In addition, for different network bandwidth requirements of network-intensive VMs, we limit the bandwidth rate of the VF inof the

SR-IOV NIC to achieve the hierarchical division of the bandwidth in the VF network, so that can allocate bandwidth to each VM on demand.

To ensure the performance of the system, it is necessary to ensure that the network-intensive VM always uses VF, and whether a VM is network-intensive or not is decided by the workload of the VM, which involves the VM network card dynamic switching between the para-virtualized NICs and VF. We use the *active backup* mode of *Bonding Driver* technology to divide the network card into master and slave devices and aggregate them into a logical network card. Only one device can work at any time in this mode of *Bonding Driver* technology, and by default the master device works, and the slave device works only when the master device is inactive. In the design, as show in Figure 3, we set the VF as the master device, and the para-virtualized NIC is set as the slave device. When the VM is perceived as network-intensive, the scheduler assigns it a VF, thereby improving the network performance of the network-intensive VM. In addition, when it becomes non-network-intensive, the scheduler uses *Hotplug* technology to remove the VF so that it can work with a para-virtualized NIC. This ensures both the performance of the VM and the connectivity of the network.

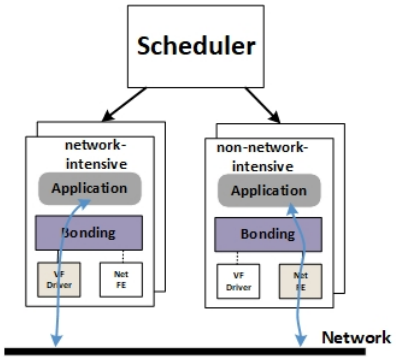

Figure 3. Design according to basic ideas.

As shown in Figure 3, each VM is equipped with two network cards, and these two network cards will switch dynamically according to the workload of the VM. Therefore, when designing the system framework, the first step is to monitor the workload characteristics of each VM, whereby the network density of VMs can be deduced on the fly to direct the strategy of network card switching. In addition, there may be a large number of VMs on the host. Therefore, we need to manage all VMs in two queues, one for storing network-intensive VMs, the other for storing other types of VMs, and the VMs in the two queues will vary according to the load of the VMs.

4.2. System Framework

The SR-IOV-based virtual network resource allocation framework proposed in this paper is based on the workload perception of the VM to schedule and allocate the VF after the bandwidth level is divided. As shown in Figure 4, we divide the system framework into four main parts, namely the *I/O Status Monitor module*, the *Network Status Monitor module*, *VF Schedule module*, and the *Transmit Rate Scheduler module*. Among them, I/O Status Monitor module and Network Status Monitor module are called status monitoring module.

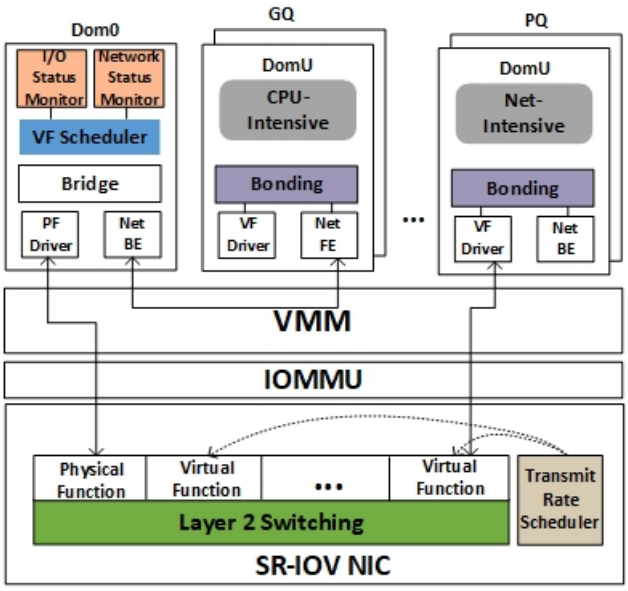

Figure 4. System Framework.

In the host, there is a *Network Status Monitor module* running as a daemon to collect network data running by the VM. In addition to the network data of the VM, in order to better distinguish between I/O-intensive and CPU-intensive VMs, we also need to obtain data about the running state of the VM, which is in the *I/O Status Monitor module*, including CPU use, VM running time and VM waiting time and so on. The *Status Monitor module* continuously sends data to the *VF Schedule module* according to the set period, and the *VF Schedule module* uses the collected data to perform VF scheduling allocation. In addition, owing to the workload of VMs is different, the network resources required for each VM are different. Therefore, the *Transmit Rate Scheduler module* is used to set a maximum bandwidth for each VF in a hierarchical method, so that *VF Schedule module* can allocate VFs with different bandwidth sizes for different loads of VMs.

For better management, we put all the VMs into two queues, *Priority Queue* (PQ) and *General Queue* (GQ), where PQ is a *red-black tree* and hosts network-intensive VMs, while GQ hosts other types of VMs, including disk I/O-intensive and CPU-intensive VMs. The dynamic allocation of VMs is based on temporal locality, that is, network-intensive VMs are also likely to be network-intensive VMs in the next sampling period. Using the data information collected by the *Status Monitor module*, the *VF Schedule module* can always allocate VF resources to network-intensive VMs. The *Transmit Rate Scheduler module* limits the bandwidth of the VF and hierarchically divides it. The VM can always obtain the desired bandwidth resources as needed, thereby maximizing the use of hardware resources and improving the overall performance of the system.

4.3. NIC Resources

As discussed earlier, we will combine the advantages of SR-IOV and para-virtualization to achieve overall network performance, for which we need to manage VF and para-virtualized NICs. We will allocate NIC resources to the VM according to the type of VM, and we put the VM into different queues for management purposes. Each VM is configured with two network cards. The para-virtualized NIC is automatically assigned to each VM when it creates a VM through Dom0, while VF can only dynamically schedule allocation to network-intensive VMs. By default, each VM is configured with only a para-virtualized NIC, and only when the *VF Schedule module* determines that a VM is a network-intensive VM, a VF is assigned to improve its performance. When the *VF Schedule module* senses that the VM is no longer a network-intensive VM, its VF resources will be deprived and can be allocated to other network-intensive VMs. Dynamic allocation of VF resources can make

network-intensive VMs have better network performance, thereby improving the overall network performance of the system. However, to achieve this goal, while considering the live migration of VMs when using SR-IOV devices, there are two problems to be solved: (1) how to allocate and remove VF; (2) how to make VF and para-virtualized NICs can be switched dynamically.

VF allocation and removal: For the problem of how to allocate and remove VF, PCIe device Hotplug technology can dynamically allocate or remove VF devices without affecting the operation of the VM. There are three Hotplug mechanisms for PCIe devices: *ACPI Hotplug*, *SHPC* (Standard Hotplug Controller) and *Vendor-specific*. Because the ACPI method is relatively simple and easy to define, we use the ACPI mechanism to perform Hotplug of PCIe devices for VMs.

The workflow of the virtual ACPI Hotplug is shown in Figure 5. Once the control panel sends a hot-removal command for the PCIe device, the Hotplug console receives the command and then modifies the device to be removed by modifying the corresponding bit in the GPE bitmap and triggers an SCI (System Control Interrupt) interrupt. When the ACPI driver in the VM detects the SCI interrupt signal, it will clear the corresponding binary bit in the GPE bitmap and query the PCIe device to be removed in the Hotplug controller and send it to Linux. At the same time, Linux will stop using the device and uninstall the device driver. In addition, the ACPI driver will perform the ACPI control operation *_EJ0* to turn off the PCIe device in the VM while using *_STA* to verify that the device was successfully removed. The process of adding a PCIe device is similar to the process of device removal, except that the ACPI control operation *_STA* is not used.

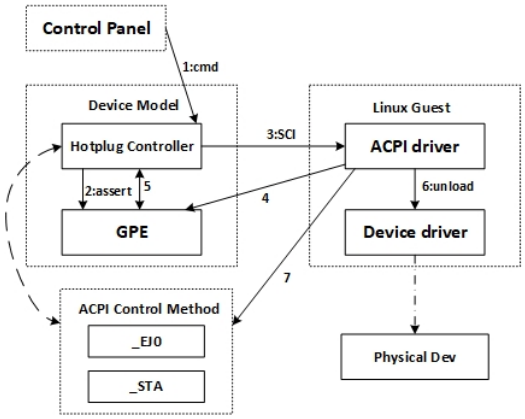

Figure 5. Virtual ACPI Hotplug workflow.

Dynamic switching of network cards: For the problem of how to make dynamic switching between VF and para-virtualized NICs, we use the active backup mode of *Bonding Driver* to set VF as the master device, and the para-virtualized NIC is set as the slave device. By default, the VM is not assigned VF, that is, the primary device is inactive. At this time, the VM works with the para-virtualized NIC, and the VM is assigned VF only when the VM becomes network-intensive. At this time, the master device is activated, so the VM automatically switches to VF to work.

4.4. Status Monitor Module

The *Status Monitor module* is responsible for monitoring and collecting data such as the network and the workload characteristics status of each VM and sending data to the *VF schedule module* for scheduling decision during each sampling period. The sampling frequency is set based on minimizing the performance loss caused by frequent read and write files.

Our goal is to allocate VF to network-intensive VMs. If we only collect the I/O characteristics of VMs, we can only distinguish between I/O-intensive VMs, but we cannot know whether the VM is network I/O-intensive or otherwise I/O-intensive (e.g., disk I/O-intensive). On the contrary, if only the network data of VMs are collected, the network-intensive VMs can only be distinguished, and it is impossible to get whether the VMs are pure network-intensive or hybrid-intensive (for example, both

CPU and network-intensive). We tend to assign VF to pure network-intensive VMs, so *Status Monitor module* needs to acquire the I/O characteristics and network data of VMs.

4.4.1. I/O Status Monitor Module

To be able to distinguish between I/O-intensive VMs, we should first obtain the I/O status information of the VM. We observed that I/O-intensive VMs have similar properties to I/O-intensive processes in Linux [33], that is, the CPU usage time of I/O-intensive VMs is always shorter, but it takes longer to wait for some I/O events to block. To this end, we use the virtualization management tool *Libvirt* to obtain the resource usage of each VM, and use these data to determine the I/O-intensive state of the VM.

Libvirt is a widely used open source virtualization management and monitoring tool that can be used to manage almost all major virtualization platforms including KVM, Xen, etc., and *Libvirt* provides an interface for acquiring VMM data. The VM in KVM is a *qemu-kvm* process of the OS at runtime and is scheduled by the process scheduler like other processes of the OS. CPU-intensive VMs require a large amount of CPU for calculations, while I/O-intensive VMs always use less CPU time because of the long wait times for waiting for certain I/O events. Therefore, we determine the type of VM by calculating the CPU usage of each VM. The CPU usage of the I/O-intensive VM will be much smaller than that of the CPU-intensive VM in the same sampling period.

The library function provided in *Libvirt* cannot directly obtain the CPU use of the VM, but the CPU time difference cpu_time_diff that the VM runs during the sampling period can be obtained through the virDomainGetInfo interface, and the CPU use is calculated by dividing the actual CPU running time of the host. which is:

$$cpu\_time\_diff = cpuTime_{now} - cpuTime_{ago} \quad (1)$$

$$cpu\_usage = 100 \times cpu\_time\_diff/(time \times nr\_cores \times 10^9) \quad (2)$$

In (1), $cpuTime_{now}$ is the total running time of the VM at the current moment, and $cpuTime_{ago}$ is the running time of the VM before the sampling period. In (2), $time$ is the sampling period and nr_cores is the number of server CPU cores. The unit of $cpuTime_{now}$ and $cpuTime_{ago}$ is nanoseconds, and the unit of $time$ is seconds, so the denominator is multiplied by 10^9 when calculating cpu_usage. We define the I/O factor that determines the I/O density of the VM:

$$io\_degree = 100 \times (1 - cpu\_usage) \quad (3)$$

The greater the I/O density of the VM, the larger the I/O factor io_degree. Therefore, the *I/O Status Monitor module* can determine the I/O-intensive state of the VM through io_degree.

4.4.2. Network Status Monitor Module

Our goal is to assign VFs to network-intensive VMs. I/O factor io_degree can only distinguish I/O-intensive VMs, but it is impossible to determine whether the VMs are disk I/O-intensive or network I/O-intensive. Therefore, in order to allocate VF accurately, we also need to determine the network state of the VM.

The black box technology is used to determine the I/O density of VMs, that is, to obtain the status information of each VM through the virtualization management tool *Libvirt* in Dom0. However, we cannot use black box technology to obtain the network information of VM. Because we use *Bonding Driver* to bind VF and para-virtualized NIC together in DomU, and VF bypasses the intervention of VMM. Therefore, we can only get the data of DomU para-virtualized NIC in Dom0. We use gray-box technology to obtain the network status information of the VM and send the data of each VM to Dom0 through *virtio-serial* communication.

Virtio-serial is a para-virtualized universal bus based on the *virtio* framework. When using *virtio-serial* to communicate between a VM and a host, there is no requirement for network bandwidth, we built two *virtio-serial* channels for each VM when acquiring VM network data, namely the *virtio data channel* and the *virtio control channel*. The daemon running in the VM asynchronously listens to the control channel. When the host sends an instruction to the VM, the VM completes the response. To save resources, the data channel is always closed, only when the control channel listens for the corresponding request will the data channel be opened, when the data is transferred will be closed again.

The pseudo file system */proc* is an interface for Linux to dynamically manipulate kernel information. It is another important way for Linux kernel space to exchange data with user space outside of system calls. The *Network Status Monitor module* reads the Linux kernel data (i.e., reads the file */proc/net/dev*) to get the VM's network status accurately. The total flow of sending (or receiving) minus the total flow of sending (or receiving) last time is the total flow of sending (or receiving) during this sampling period. We define network factors that determine the density of VM networks:

$$net\_degree = \varepsilon \times data\_avg + (1 - \varepsilon) \times traffic\_freq \tag{4}$$

where:

$$data\_avg = (\sum_{0}^{num} traffic\_data)/num \tag{5}$$

$$traffic\_freq = traffic\_num/num \tag{6}$$

Among them, $traffic\_data$ represents the total size of the transmitted and received data packets in one sampling period, num represents the total number of samples, $traffic\_num$ denotes the number of samples of network transmission in all samples, $data\_avg$ reflects the weighted average of the total traffic of the VM network data, $traffic\_freq$ responding to the busyness of the network, parameters ε are used in Equation (4) to balance the weight between the two.

4.5. VF Bandwidth Division

To limit the network rate of VF, modern SR-IOV network cards (such as Intel 82599 network cards) have added the ability to control the transmission rate on their hardware. They can set a maximum transmission rate independently for each VF without affecting the transmission rate of other VFs or PFs. However, VF has no permission to set the transmission rate itself and can only be set and modified by PF with global operation permission. SR-IOV network card provides Transmit Rate Scheduler (TRS) function to limit the number of each transmission queue in order to limit the transmission rate of VF.

As shown in Figure 6, it is the flow chart of the SR-IOV network card receiving packets. When a network packet is received by a network card, it is placed in the receiving queue of the corresponding VM by the Layer 2 Sorter according to the MAC address and VLAN of the packet, and then the DMA maps the target DMA address (i.e., the memory address of the VM) to the physical host address. Once the DMA operation is completed, the data will be transferred to the memory address of the VM. At this time, the network card will trigger the interrupt to inform the VMM that the packet has arrived, and the VMM will tell the specific VM to read the data from the transmission queue according to the interrupt vector table to complete the operation of receiving the packet. The process of sending network data by SR-IOV network card is similar to that of receiving. The receiving/sending queue of VF will directly affect the transmission rate of the VM. Therefore, the transmission rate of VF can be limited by limiting the receiving/sending queue.

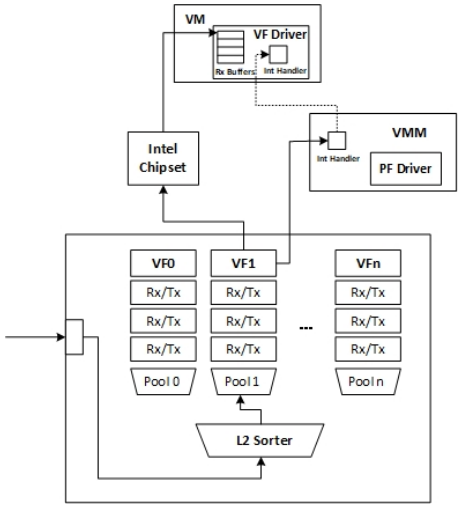

Figure 6. The process of receiving data packets by SR-IOV NIC.

To allocate the corresponding size of bandwidth according to the network requirements of the network-intensive VM, we use the method of hierarchical division when limiting the transmission rate of the VF, that is, the maximum bandwidth of each VF is divided according to the step-up method:

$$\begin{cases} VF_n = VF_0 + (n-1) \times \delta \\ \sum_{i=0}^{0} VF_i \leq B \end{cases} \tag{7}$$

In (7), VF_i is the maximum bandwidth of the i-th VF, δ is the size of the step, and B is the total bandwidth of the network card. The size of δ should be set according to the size of the network difference between the data center VM. If δ setting is too small, the network-intensive VM with high bandwidth requirements cannot obtain sufficient bandwidth to affect the performance of the system. If it is too large, the VM will have a surplus when using the bandwidth, so that the system cannot achieve the maximum network throughput.

4.6. VF Scheduler

The VF Scheduler module is located in the privileged domain Dom0 and is responsible for the global scheduling of the VF, and all VMs are placed in the two queues of PQ and GQ. The purpose of classifying VMs is to be able to quickly identify the type of VM and efficiently allocate VF resources dynamically. Among them, PQ is a *red-black tree*, which stores network-intensive VMs, while GQ stores other types of VMs, including disk I/O-intensive and CPU-intensive VMs. To improve the network performance of network-intensive VMs and improve the overall network performance of the system, VMs in PQ will always use VF, while VMs in GQ work with para-virtualized NICs. Since the effective bandwidth acquired by each VF is inversely proportional to the total number of configured VFs, by limiting the number of VFs, it is ensured that there is sufficient bandwidth allocated for non-network-intensive VM. The number of VFs is set according to the specific environment, such as the number of VMs, the size of available bandwidth, and so on.

To ensure the network performance of the system, the purpose of VF scheduling is to make the network-intensive VM always use VF, so the VM in the PQ should be dynamically adjusted. Initially all VMs are put into the GQ, and then the VM is inserted into the PQ according to the network density. When the VM network density in the PQ is lower than that of the VM in GQ, the VM in the PQ is moved to the GQ and the VF is deprived, and the VM with the most network density in the GQ is selected to be inserted into the PQ and allocate the remaining VFs. The VF and para-virtualized NIC switching during VF dynamic allocation uses the technique of Section 3.

We combine I/O factor *io_degree* and network factor *net_degree* to determine the degree of network-intensive VMs. For hybrid VMs (that is, VMs with dense network and CPU), we prefer to assign VF to network-intensive VMs, but we cannot distinguish the two by using network factor *io_degree* alone. For this reason, we first use I/O factor *io_degree* to classify VMs into I/O-intensive and non-I/O-intensive VMs, and then use network factor *io_degree* to distinguish network-intensive VMs from I/O-intensive VMs. Specifically, letting each VM's I/O factor *io_degree* be compared to a threshold (according to a specific environment setting) can divide the VM into I/O-intensive and non-I/O-intensive in a linear time. Then I/O-intensive VMs are sorted according to network factor *io_degree* from large to small, and the network-intensive VMs are judged according to the number of the remaining number of assignable VFs, and the VFs are allocated according to bandwidth requirement of the VM. In this way, in order to improve the overall network performance of the system, the network-intensive VM in the PQ dynamically allocates the VF with a time complexity of $O(\log n)$, where n is the number of network-intensive VMs in the PQ.

5. Evaluation

In this section, we will use standard test tools to evaluate the network performance of the SR-IOV-based virtual resource allocation framework, and gradually increase the number of VMs to test system performance in terms of bandwidth throughput and latency. To simulate different load types and load changes of VMs, we introduced different types of benchmarks to simulate CPU-intensive VMs and network-intensive VMs. In addition, based on this, we compared the default scheduling strategy of KVM and the network performance based on SR-IOV virtual resource allocation framework.

5.1. Experimental Setup

We used two DELL R720 servers and one Brocade 80 switch to test and the two servers connected over 10 Gigabit Ethernet. As shown in Table 1, the two servers have basically the same hardware configuration. One server runs the server program by running Apache 2.4.37, and the other server simulates the client to initiate a service request to the server. Although the cluster size is relatively small, our goal is to test the performance of VM scheduling. Our experimental environment can still meet the requirements of our system network performance when testing VM load changes.

Table 1. Server configuration.

CPU	Intel(R) Xeon(R) E5-2630 v4 @ 2.20GHz
RAM	DDR4 ECC RG 64GB
NIC	Intel 82599 10Gbps
VMM	KVM 1.5.3/QEMU 1.5.3
Dom0 OS	CentOS 7.5 (Linux 3.10.0 SMP)
VM OS	CentOS 6.9 (Linux 2.6.32 SMP)
VM RAM	512M
VM NIC	Intel 82599 VF

In the process of testing, each time the client initiates a request to the server, and the number of VMs in each server can be adjusted constantly. Each VM can run different types of load, including CPU-intensive load, network-intensive load, and hybrid load. We use *Lookbusy* [34] to simulate CPU-intensive loads, *Netperf* [35] to simulate network-intensive loads, and a combination of *Lookbusy* and *Netperf* to simulate hybrid loads. To ensure fairness, we used only one port of the Intel 82599 network card (dual port network card) for testing.

5.2. Dynamic Scheduling

First, we test the attribute characteristics of the SR-IOV-based virtual resource allocation framework under dynamic hybrid load. We run 15 VMs on the same host, *VM1–VM15*, and the VMs are divided into three categories, each with 5 VMs. As shown in Table 2, The VMs in each

group simulate network-intensive (VM1–VM5), CPU-intensive (VM6–VM10), and mixed-load VMs (VM11–VM15), respectively, and the proportion of time in which the benchmarks are run in the order of increasing VM IDs in each group is 20%, 40%, 60%, 80%,100%.

Table 2 shows the differences that the system exhibits when running at different loads. From the table we can see that the *io_degree* value of CPU-intensive VM is always lower than that of network-intensive VM, and the longer the benchmark running time of the CPU-intensive VM, the lower its *io_degree* value. In addition, for a mixed-load VM, its *io_degree* value is also larger than the CPU-intensive VM. By judging the value of *io_degree*, we can judge the network-intensive VM and the mixed-load VM with a large network density. In addition, as shown in the table, the longer the *Netperf* runtime of a network-intensive VM, the greater its *net_degree* value. Therefore, combining the *net_degree* value of the VM, the VM with high network density can be judged.

To test whether our proposed allocation framework can distinguish between network-intensive VMs and dynamically assign VF to network-intensive VMs, we run 8 VMs on the same host and configure the number of VFs to be 4. In addition, the VF Scheduler is triggered for every 10 s and the parameter δ is set to 0.7 when *net_degree* is calculated. Table 3 shows how the load on each VM changes over time. Among them, the proportion of time that *VM1* runs *Netperf* has been kept at 55%, the proportion of time that *VM2* runs *Lookbusy* is 40%, and the proportion of time that *VM3* runs both *Netperf* and *Lookbusy* is 30%. The load of other VMs will change with time, so we can test the allocation status of VF when the load changes.

Table 2. CPU-intensive, network-intensive and mixed-load property values.

VM_id	**VM1**	**VM2**	**VM3**	**VM4**	**VM5**
io_degree	91.45	90.95	92.43	94.07	93.98
net_degree	27,703	52,832	104,386	146,989	162,072
VM_id	**VM6**	**VM7**	**VM8**	**VM9**	**VM10**
io_degree	81.43	59.97	45.54	23.61	7.82
net_degree	0	0	0	0	0
VM_id	**VM11**	**VM12**	**VM13**	**VM14**	**VM15**
io_degree	86.23	61.98	45.77	25.09	11.73
net_degree	30,283	69,032	95,619	138,566	175,117

Table 3. The load of VMs changes over time.

Time	**0 (min)**			**10 (min)**			**20 (min)**			**30 (min)**		
state	net	cpu	result	net	cpu	result	net	cpu	result	net	cpu	result
VM1	55%	0	VF	55%	0	VF	55%	0	VF	55%	0	VF
VM2	0	40%		0	40%		0	40%		0	40%	
VM3	30%	30%		30%	30%	VF	30%	30%		30%	30%	
VM4	20%	0		20%	0		30%	0	VF	30%	0	VF
VM5	40%	0		40%	0	VF	40%	0	VF	40%	0	VF
VM6	60%	0	VF	60%	0	VF	60%	0	VF	60%	0	VF
VM7	80%	0	VF	0	20%		0	20%		20%	20%	
VM8	100%	0	VF	0	40%		0	40%		40%	40%	

Experiments show that before time point 10, because *VM1*, *VM6*, *VM7* and *VM8* have the largest *net_degree* value, these four VMs are assigned VF. There is a scheduling occurred at time 10 because *VM7* and *VM8* became CPU-intensive VMs, and their VFs were removed and assigned to *VM3* and *VM5*. Later, in the moment of 20, there is a VF who owners are changed as the network density of *VM4* is increased and its *net_degree* is greater than the value of *VM3*. Finally, at time 30, no VF removal and allocation occurs, because although the network density of *VM8* is increased, it is also a CPU-intensive

VM whose *io_degree* value is much larger than *VM4*. As can be seen from the results of the scheduling, VF is always dynamically allocated to VMs with high network density.

5.3. Network Latency

The following is a performance evaluation of the SR-IOV virtual network resource allocation framework. First, we evaluate the web server network latency performance. We use the benchmarking tool *Httperf* [36] to generate different types of HTTP payloads to test the server performance of the system. In this test, the Apache service was run in the server-side VM, while the *Httperf* load in the client-side VM repeatedly requested a 10 KB Web page to the server. We have gradually increased the number of VMs, and 100 requests between the server and the client are generated by *Httperf* within one second to measure the delay of the system request response time. Here, the system's request response time is the entire time when the client initiates a request and gets a response on the server.

Figure 7a shows a comparison of network delays before and after optimization when the request rate is fixed at 100/s. It can be seen from the figure that the request delay using the SR-IOV-based virtual resource allocation framework is smaller than the default scheduling strategy of KVM. When the number of VMs is greater than or equal to 32, the optimized system average request delay is reduced by more than 26%. In particular, when the number of VMs is 48, the optimized average request latency is reduced by 94% compared to the default scheduling policy.

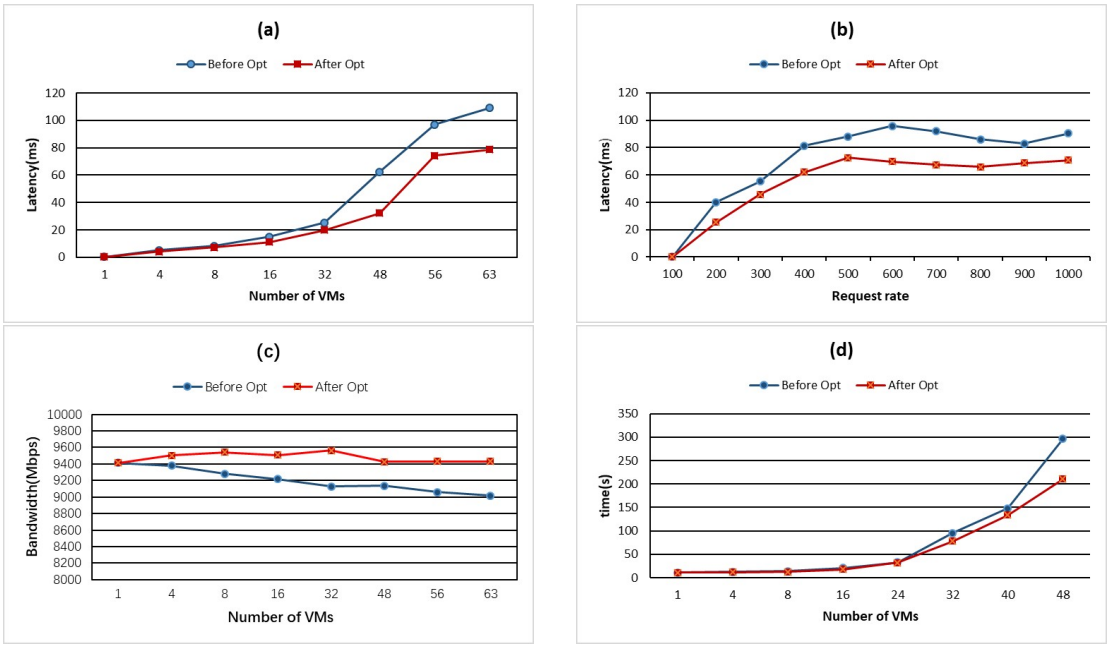

Figure 7. (**a**) Comparison of network delay when the request rate is fixed. (**b**) Comparison of network delay when the number of VMs is fixed. (**c**) Comparison of bandwidth throughput. (**d**) Comparison of file transfer time.

To further verify our experimental results, we fixed the number of VMs to 32, gradually increasing the client request rate from 100/s to 1000/s and measured the response latency of client requests to the server. Figure 7b shows a comparison of network latency between before and after optimization when the number of VMs is fixed to 32. It can be seen from the graph that using SR-IOV-based virtual resource allocation framework can effectively reduce the system request response time. When the client request rate is greater than 100/s, the optimization can reduce the network delay by 21% to 58% before the optimization.

The reason the SR-IOV-based virtual resource allocation framework can effectively reduce the request response time of the system is that SR-IOV technology can reduce I/O request path of the VM. Moreover, the I/O requests of each VM are not uniformly distributed during the running of the

system but are centrally distributed in a short period of time after the VM is scheduled. At this time, the VM will be perceived by the VF Scheduler module and VF is allocated to network-intensive VMs to ensure their network performance. This indicates that in order to reduce the latency of I/O response of the VM, the time slice scheduled by the VM should be reduced, which can increase the scheduling opportunity of each VM.

5.4. Bandwidth Throughput

Next, we use the benchmark *Netperf* to test the improvement of system bandwidth throughput based on SR-IOV virtual network resource allocation framework. *Netperf* is a common network performance testing tool based on TCP or UDP transmission. In this test, we open a remote TCP connection to the server in each VM on the client side. Figure 7c shows a comparison of bandwidth throughput before and after system optimization when the number of VMs increases from 1 to 63. From the experimental results, we can see that the bandwidth throughput of the virtual resource allocation framework based on SR-IOV can always be at the maximum and keep in a relatively stable range, while the bandwidth throughput of the system before optimization is gradually decreasing with the increase of the number of VMs. It can be seen from the graph that the optimized bandwidth throughput curve will continue to be better than that before optimization. Compared with the default scheduling strategy, the virtual resource allocation framework based on SR-IOV can maximize the bandwidth throughput by 5%, that is, when the number of VMs is 63, it will increase from 9018.04 Mbps to 9428.92 Mbps.

When using the default scheduling strategy of KVM, it is expected that the bandwidth throughput of the system decreases as the number of VMs increases. As the number of VMs increases, VMs will compete highly for network resources, and the overhead of scheduling and interrupt processing will lead to the decrease of system bandwidth throughput, even the fluctuation of system bandwidth throughput and other unpredictable situations. When the number of system VMs increases, the use of SR-IOV-based virtual resource allocation framework can always ensure that network-intensive VMs get sufficient bandwidth resources. Despite the slight overhead of scheduling and interrupt processing, the bandwidth throughput of the system can always be at its maximum and remain within a relatively stable range.

5.5. File Transfer Time

In the end, we tested the performance of the file transfer system under mixed load conditions. We use the command scp for remote file copying in Linux to test how long the VM transfers files. In this test, we gradually increase the number of VMs, of which 70% run the *Lookbusy* program, allowing all VMs to copy local 1G files to the server at the same time, measuring the average transfer time of the VM. Figure 7d shows a comparison of the average file transfer time before and after the system optimization when the number of VMs is gradually increased from 1 to 48. From the experimental results, it can be seen that using SR-IOV-based virtual resource allocation framework can effectively reduce the average file transfer time of VMs, with the increase of the number of VMs, the average transfer time after optimization is less than before optimization. For example, when the number of VMs is 48, the average transmission time before optimization is 295.95 s, and after optimization is 210.81 s.

The reason SR-IOV-based virtual resource allocation framework can reduce the average file transfer time of VMs is similar to the principle of system network delay reduction. The demand of VMs for network resources is concentrated on a period of time after being scheduled, when the VMs are perceived as network-intensive and will be assigned VF to ensure their network performance.

6. Conclusions

This paper proposes a virtual network resource allocation method that can combine the advantages of I/O para-virtualization and SR-IOV technology. It can dynamically sense the load changes of VMs, use the hierarchical division method to limit the VF bandwidth, and allocate different levels of VF according to the bandwidth requirements of different workloads of VMs. In addition, by limiting the number of VFs to ensure the performance of network-intensive VMs, it can improve the overall network performance of the system. The load awareness of the VM is obtained by reading the pseudo file */proc* in VM and sending it to the VF Scheduler module of the privileged domain by means of *virtio-serial* communication in KVM. The dynamic scheduling of VF adopts ACPI Hotplug technology and the active backup mode of *Bonding Driver* technology to meet the dynamic switching of VF and para-virtualized NICs when the VM load changes. Our experiments show that our proposed virtual network resource allocation framework can effectively improve the overall network performance of the system.

Our prototype system is implemented based on KVM and our comprehensive evaluation shows that the virtual network resource allocation framework proposed in this paper is effective since VF is always dynamically allocated to VMs with high network density. In addition, the allocation framework also can effectively improve system network performance, in which the average request delay can be reduced by more than 26%, and the system bandwidth throughput rate can be improved by about 5%.

Author Contributions: Z.Y. and Y.Z. conceived and designed the experiments; Y.W. performed the experiments; Z.Y. and Y.W. analyzed the data; Z.Y. wrote the original draft, Y.Z. and Y.W. edited and reviewed the paper, Y.Z. supervised this work.

Funding: This work was supported by the National Natural Science Foundation of China (No. 2012CB215202), the Fundamental Research Funds for the Central Universities of China (Nos. 106112016CDJXZ168815, 106112017CDJZRPY0101), and the Scientific and Technological Project of Chongqing (No. cstc2017shmsA40003).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. Xen and the art of virtualization. *ACM SIGOPS Oper. Syst. Rev.* **2003**, *37*, 164–177. [CrossRef]
2. Kivity, A.; Kamay, Y.; Laor, D.; Lublin, U.; Liguori, A. Kvm: The Linux virtual machine monitor. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 28 June–1 July 2007; Volume 1, pp. 225–230.
3. Ben-Yehuda, M.; Factor, M.; Rom, E.; Traeger, A.; Borovik, E.; Yassour, B.A. Adding advanced storage controller functionality via low-overhead virtualization. In Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12), San Jose, CA, USA, 14–17 February 2012; Volume 12, p. 15.
4. Gordon, A.; Amit, N.; Har'El, N.; Ben-Yehuda, M.; Landau, A.; Schuster, A.; Tsafrir, D. ELI: Bare-metal performance for I/O virtualization. *ACM SIGPLAN Not.* **2012**, *47*, 411–422.
5. Guan, H.; Ma, R.; Li, J. Workload-aware credit scheduler for improving network I/O performance in virtualization environment. *IEEE Trans. Cloud Comput.* **2014**, *2*, 130–142. [CrossRef]
6. Jose, J.; Li, M.; Lu, X.; Kandalla, K.C.; Arnold, M.D.; Panda, D.K. SR-IOV support for virtualization on infiniband clusters: Early experience. In Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Delft, The Netherlands, 13–16 May 2013; pp. 385–392.
7. Pfefferle, J.; Stuedi, P.; Trivedi, A.; Metzler, B.; Koltsidas, I.; Gross, T.R. A hybrid I/O virtualization framework for RDMA-capable network interfaces. *ACM SIGPLAN Not.* **2015**, *50*, 17–30. [CrossRef]
8. Zhou, F.F.; Ma, R.H.; Li, J.; Chen, L.X.; Qiu, W.D.; Guan, H.B. Optimizations for high performance network virtualization. *J. Comput. Sci. Technol.* **2016**, *31*, 107–116. [CrossRef]
9. Bourguiba, M.; Haddadou, K.; El Korbi, I.; Pujolle, G. Improving network I/O virtualization for cloud computing. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 673–681. [CrossRef]

10. Gao, P.X.; Narayan, A.; Karandikar, S.; Carreira, J.; Han, S.; Agarwal, R.; Ratnasamy, S.; Shenker, S. Network Requirements for Resource Disaggregation. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, OSDI, Savannah, GA, USA, 2–4 November 2016; Volume 16, pp. 249–264.
11. Landau, A.; Ben-Yehuda, M.; Gordon, A. SplitX: Split Guest/Hypervisor Execution on Multi-Core. In *WIOV*; USENIX Association: Berkeley, CA, USA, 2011.
12. Dong, Y.; Zheng, X.; Zhang, X.; Dai, J.; Li, J.; Li, X.; Zhai, G.; Guan, H. Improving virtualization performance and scalability with advanced hardware accelerations. In Proceedings of the 2010 IEEE International Symposium on Workload Characterization (IISWC 2010), Atlanta, GA, USA, 2–4 December 2010; pp. 1–10.
13. Har'El, N.; Gordon, A.; Landau, A.; Ben-Yehuda, M.; Traeger, A.; Ladelsky, R. Efficient and Scalable Paravirtual I/O System. In Proceedings of the USENIX Annual Technical Conference, San Jose, CA, USA, 26–28 June 2013; Volume 26, pp. 231–242.
14. Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; et al. The design and implementation of open vswitch. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, CA, USA, 4–6 May 2015; pp. 117–130.
15. Dong, Y.; Yang, X.; Li, J.; Liao, G.; Tian, K.; Guan, H. High performance network virtualization with SR-IOV. *J. Parallel Distrib. Comput.* **2012**, *72*, 1471–1480. [CrossRef]
16. Dong, Y.; Dai, J.; Huang, Z.; Guan, H.; Tian, K.; Jiang, Y. Towards high-quality I/O virtualization. In Proceedings of the SYSTOR 2009: The Israeli Experimental Systems Conference, Haifa, Israel, 4–6 May 2009; ACM: New York, NY, USA, 2009; p. 12.
17. Suzuki, J.; Hidaka, Y.; Higuchi, J.; Baba, T.; Kami, N.; Yoshikawa, T. Multi-root share of single-root I/O virtualization (SR-IOV) compliant PCI Express device. In Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects, Mountain View, CA, USA, 18–20 August 2010; pp. 25–31.
18. Chen, L.; Li, B.; Li, B. Allocating bandwidth in datacenter networks: A survey. *J. Comput. Sci. Technol.* **2014**, *29*, 910–917. [CrossRef]
19. Ongaro, D.; Cox, A.L.; Rixner, S. Scheduling I/O in virtual machine monitors. In Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Seattle, WA, USA, 5–7 March 2008; ACM: New York, NY, USA, 2008; pp. 1–10.
20. Rao, J.; Wang, K.; Zhou, X.; Xu, C.Z. Optimizing virtual machine scheduling in NUMA multicore systems. In Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013), Shenzhen, China, 23–27 February 2013; pp. 306–317.
21. Xu, C.; Gamage, S.; Rao, P.N.; Kangarlou, A.; Kompella, R.R.; Xu, D. vSlicer: Latency-aware virtual machine scheduling via differentiated-frequency CPU slicing. In Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, Delft, The Netherlands, 18–22 June 2012; ACM: New York, NY, USA, 2012; pp. 3–14.
22. Guo, J.; Liu, F.; Lui, J.C.; Jin, H. Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach. *IEEE/ACM Trans. Netw.* **2016**, *24*, 873–886. [CrossRef]
23. Guo, J.; Liu, F.; Huang, X.; Lui, J.C.; Hu, M.; Gao, Q.; Jin, H. On efficient bandwidth allocation for traffic variability in datacenters. In Proceedings of the EEE Conference on Computer Communications (INFOCOM), Toronto, ON, Canada, 27 April–2 May 2014; pp. 1572–1580.
24. Huang, Z.; Ma, R.; Li, J.; Chang, Z.; Guan, H. Adaptive and scalable optimizations for high performance SR-IOV. In Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER), Beijing, China, 24–28 September 2012; pp. 459–467.
25. Tian, K.; Dong, Y.; Mi, X.; Guan, H. sEBP: Event based polling for efficient I/O virtualization. In Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER), Beijing, China, 24–28 September 2012; pp. 135–143.
26. Hines, M.R.; Deshpande, U.; Gopalan, K. Post-copy live migration of virtual machines. *ACM SIGOPS Oper. Syst. Rev.* **2009**, *43*, 14–26. [CrossRef]
27. Martins, J.; Ahmed, M.; Raiciu, C.; Olteanu, V.; Honda, M.; Bifulco, R.; Huici, F. ClickOS and the art of network function virtualization. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, Seattle, WA, USA, 2–4 April 2014; USENIX Association: Berkeley, CA, USA, 2014; pp. 459–473.

28. Shafer, J. I/O virtualization bottlenecks in cloud computing today. In Proceedings of the 2nd Conference on I/O virtualization, Pittsburgh, PA, USA, 13 March 2010; USENIX Association: Berkeley, CA, USA, 2010; p. 5.
29. Ram, K.K.; Santos, J.R.; Turner, Y.; Cox, A.L.; Rixner, S. Achieving 10 Gb/s using safe and transparent network interface virtualization. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Washington, DC, USA, 11–13 March 2009; ACM: New York, NY, USA, 2009; pp. 61–70.
30. Dong, Y.; Yu, Z.; Rose, G. SR-IOV Networking in Xen: Architecture, Design and Implementation. In Proceedings of the Workshop on I/O Virtualization, San Diego, CA, USA, 10–11 December 2008; Volume 2.
31. Liu, J. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Atlanta, GA, USA, 19–23 April 2010; pp. 1–12.
32. Dong, Y.; Zhang, X.; Dai, J.; Guan, H. HYVI: A hybrid virtualization solution balancing performance and manageability. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2332–2341. [CrossRef]
33. Zeng, L.; Wang, Y.; Fan, X.; Xu, C. Raccoon: A Novel Network I/O Allocation Framework for Workload-Aware VM Scheduling in Virtual Environments. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 2651–2662. [CrossRef]
34. Carraway, D. Lookbusy—A synthetic load generator. *Look Busy Accessed August* **2013**, *18*, 2017.
35. Jones, R. Welcome to the Netperf homepage. *Retriev. March* **2000**, *30*, 2010.
36. Mosberger, D.; Jin, T. Httperf—A tool for measuring web server performance. *ACM SIGMETRICS Perform. Eval. Rev.* **1998**, *26*, 31–37. [CrossRef]