Sova: A Software-Defined Autonomic Framework for Virtual Network Allocations

Zhiyong Ye, Yang Wang, Shuibing He, Chengzhong Xu, *Fellow, IEEE*, and Xian-He Sun, *Fellow, IEEE*

Abstract—With the rise of network virtualization, the workloads deployed on data center are dramatically changed to support diverse service-oriented applications, which are in general characterized by the time-bounded service response that in turn puts great burden on the data-center networks. Although there have been numerous techniques proposed to optimize the virtual network allocation in data center, the research on coordinating them in a flexible and effective way to autonomically adapt to the workloads for service time reduction is few and far between. To address these issues, in this article we propose *Sova*, an autonomic framework that can combine the virtual dynamic SR-IOV (DSR-IOV) and the virtual machine live migration (VLM) for virtual network allocations in data centers. DSR-IOV is a SR-IOV-based virtual network allocation technology, but its operation scope is very limited to a single physical machine, which could lead to the local hotspot issue in the course of computation and communication, likely increasing the service response time. In contrast, VLM is an often-used virtualization technique to optimize global network traffic via VM migration. *Sova* exploits the software-defined approach to combine these two technologies with reducing the service response time as a goal. To realize the autonomic coordination, the architecture of *Sova* is designed based on the MAPE-K loop in autonomic computing. With this design, *Sova* can adaptively optimize the network allocation between different services by coordinating DSR-IOV and VLM in autonomic way, depending on the resource usages of physical servers and the network characteristics of VMs. To this end, *Sova* needs to monitor the network traffic as well as the workload characteristics in the cluster, whereby the network properties are derived on the fly to direct the coordination between these two technologies. Our experiments show that *Sova* can exploit the advantages of both techniques to match and even beat the better performance of each individual technology by adapting to the VM workload changes.

Index Terms—Virtual machine migration, dynamic SR-IOV, software-defined approach, autonomic computing, MAPE-K loop, network allocation

1 Introduction

As numerous applications are being migrated to the cloud, the workloads in data centers tend to exhibit more diverse characteristics in terms of execution behavior and resource usage. Among these workloads, particularly relevant is those supporting service-oriented applications, say search service, game service, etc, which are typically featured by the time-bounded service response that in turn puts great burden on the data-center networks. Although it has been greatly studied in recent years, the network allocation for service time reduction in data centers is still hard to fulfill the ever-increasing requirements on it, especially when these time-bounded services are becoming data-driven and widely deployed to service people's daily life.

To address this issue, many studies have been conducted to optimize the network allocations for improving the quality of service (QoS) in data centers [1], [2]. Some typical results are those characterized by the virtualization technology for flexible and cost-effective resource usages, each with its own advantages and disadvantages [3], [4], [5]. For example, *Dynamic SR-IOV* (DSR-IOV) [5] achieves the network performance by carefully sharing network bandwidths among the VMs via *para-virtual Network Interface Card* (vNIC) and SR-IOV virtual functions (VFs) [2], [6]. DSR-IOV can adaptively switch between the vNIC and the SR-IOV VFs for each virtual machine (VM), according to its workload characteristics at runtime, allowing the I/O-intensive VMs to have more network resources. As a result, it is particularly amenable to those I/O-intensive VM workloads to optimize their response time by reducing the network latency. Although its merits are compelling, DSR-IOV is only a local optimization technique, limiting its application scope to single physical server, and thus, incapable of re-engineering the network traffic to mitigate the hotspot issue across the cluster.[1]

Another example is VM live migration (VLM), which is also a well-studied technique to improve the performance

- *Zhiyong Ye and Yang Wang are with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: {zy.ye, yang.wang1}@siat.ac.cn.*
- *Shuibing He is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: heshuibing@zju.edu.cn.*
- *Chengzhong Xu is with the State Key Laboratory of IoT for Smart City, Faculty of Science and Technology, University of Macau, Macau 999078, China. E-mail: czxu@um.edu.mo.*
- *Xian-He Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. E-mail: sun@iit.edu.*

Manuscript received 25 June 2019; revised 21 July 2020; accepted 21 July 2020. Date of publication 28 July 2020; date of current version 10 Aug. 2020. (Corresponding author: Yang Wang.)
Recommended for acceptance by R. Tolosana.
Digital Object Identifier no. 10.1109/TPDS.2020.3012146

[1] Informally, a hotspot is occurred if the aggregate CPU or network utilization on the physical server exceeds a set threshold.

of virtual network [4], [7] by moving a running VM between different physical machines without disconnecting the clients. With the VLM, the network traffic in virtualized data center can be well engineered to remove the hotspots through carefully remapping or re-shuffling the running VMs to a cluster of physical machines [7], which in turn improves the QoS of the VMs in terms of response time. However, VLM is a fairly expensive operation as it always incurs bulk-data transfer, and much worse, service disruption. On the other hand, as a global optimization technique, VLM lacks the fine tuning ability to provide the network-intensive VMs with more local network resources.

Although the values of the proposed techniques, like DSR-IOV and VLM, have been well evaluated, the research on coordinating them in a flexible and effective way to autonomically adapt to the workloads for service time reduction is still few and far between in current literature.

Given the diversity of services encapsulated into different VMs, the workloads of VMs tend to be dynamically changed and would be severely interfered with each other as well [8], [9], rendering the hotspots easy to occur in the cluster. On the other hand, the network traffic between the VMs is mixed with control messages and data messages, both are highly varied and desired to have different transferring requirements as time goes by [10], leading to the network optimization hard to make for QoS improvements. As a result, no single technique, neither DSR-IOV nor VLM alone, is likely to fit all cases of the network performance to improve the QoS of the VMs, and thus combining them is a viable way to exert each own strength for various network problems. However, the combination is challenging as these two techniques are orthogonal, which needs to recognize different access scenarios, locally and globally, to coordinate them in an effective way that can bring to bear on each one.

To address these issues, we propose *Sova*, an autonomic framework for virtual network performance in data centers, which orchestrates DSR-IOV and VLM in an autonomic way to improve the quality of VM services in different scenarios. With *Sova*, one can reduce the service response time not only by optimizing the network allocations locally but also by getting rid of the hotspot issue across the cluster. As such, it is particularly beneficial to the virtual services such as the three-tier applications deployed in the same cluster of physical machines where the *fine grained* communications between VMs are frequently involved and the computational loads are relatively heavy when the request volume is high.

In order to have a more general perspective both in the methodology and in the mechanisms and techniques to be applied, we design *Sova* as an autonomic framework, which is built around the MAPE-K loop—an often used architecture in autonomic computing [11], [12]—to endow *Sova* with the autonomic abilities to coordinate DSR-IOV and VLM for adaptation to the dynamic changes of workloads with minimizing the service response time as a goal.

In particular, we follow the idea of IOFlow [13] to combine DSR-IOV and VLM in a *software-defined way* in which the *control plane* of the network operation is decoupled from the *data plane* used to access the network card. The control plane in *Sova* as the *Effector* in MAPE-K loop is designed to control how each VM is locally allocated either a SR-IOV VF or the vNIC by performing DSR-IOV, or globally moved to a target machine as an adaptation to its workload changes by executing the informed migration instructions issued from a *controller*–the centralized *Autonomic Manager*, which is devised to maintain and analyze the global hotspot information across the cluster with an attempt to make migration plan selectively sent to each physical server. To this end, *Sova* needs to monitor the network traffic as well as the workload characteristics in the cluster, whereby the network access patterns can be derived on the fly.

With this design, the autonomic manager can simplify its work by only determining the source and target migration servers while leaving the freedom to the pair of servers to select the migrated VMs. Similarly, each physical server can focus squarely on its local DSR-IOV, remaining unaware of the global migration decisions to limit its performance cost.

We implemented *Sova* as a prototype based on Xen4.9 and evaluated its performance in different scenarios by comparing with each individual technique, DSR-IOV and VLM. Although each individual technique has been intensively studied, *Sova*, to the best of our knowledge, is the first attempt that tries to combine them as a holistic approach to the network resource allocations in diverse situations. Our experimental results show that *Sova* can exert the advantages of both techniques to match and even beat the better performance of each individual technology by adapting to the VM workload changes.

The remainder of this paper is organized as follows. We first introduce some background knowledge on DSR-IOV and analyze the challenges when integrating it with VLM in Section 2. With these challenges in mind, we then describe the design of *Sova* in Section 3 and its implementation in Section 4. We make the performance evaluation of *Sova* in Section 5, followed by reviewing some related work for comparison studies in Section 6. Finally, we remark and conclude the paper in the last section.

2 Background and Motivations

In this section, we first introduce some background knowledge regarding the used techniques of DSR-IOV and VLM, and then describe the MAPE-K loop that is often-used in autonomic computing to implement self-adaptive software. Finally, we discuss the challenges behind *Sova* for adaptive virtual network allocations.

2.1 Dynamic SR-IOV

Single-root I/O virtualization (SR-IOV) is a widely deployed I/O virtualization technology to eliminate the hypervisor's intervention from the VM I/O paths via hardware supports [2], [6]. The SR-IOV device contains one or more *physical functions* (PFs) with full functionality of PCIe, and each PF module has one or more *virtual functions* (VFs) which are "lightweight" PCIe functions. The PF has access to all resources of hardware in the SR-IOV device, while the VF contains only the resources necessary for data transfer, such as the transmit and receive registers and interrupt registers. All VFs are managed and configured by the PF, and each VF can be assigned to a VM as a standard PCIe device for efficient direct access.

Since the bandwidth allocation between multiple VFs is based on hardware arbitration, the theoretical bandwidth that a single VF can obtain is the average bandwidth among

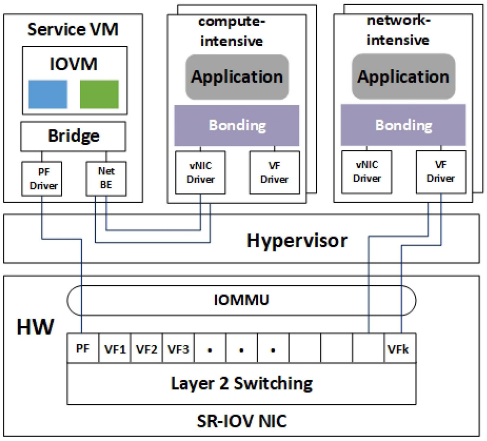


Fig. 1. Dynamic SR-IOV architecture.

the VFs. Consequently, when the number of VFs is large, the bandwidth obtained by each individual VF will decrease accordingly. As a result, the network-intensive VMs with SR-IOV capable device driver are not guaranteed to have sufficient bandwidths to improve their performance. To maximize the potentials of SR-IOV, we have to carefully allocate the network bandwidths among the competitive VMs, either with granted SR-IOV VFs or not.

As with SR-IOV, the I/O para-virtualization is another widely used network virtualization approach, which shares the network bandwidths among the VMs via vNIC—a virtual NIC based on the physical NIC. However, unlike the SR-IOV as described, the I/O para-virtualization, i.e., vNIC, optimizes the VM network performance in a pure software way, leading to a long data I/O path and as well a hotspot driver domain. Thus, it, though simple and cheap, suffers from the degraded network performance.

Dynamic SR-IOV (DSR-IOV) is proposed to exploit the advantages of both the vNIC and the SR-IOV in *Raccoon* – a network I/O allocation algorithm for VM scheduling in virtual environments [5] – to improve the network performance of hybrid VMs as shown in Fig. 1, each with different kinds of workloads, either compute-intensive or network-intensive. The basic idea of DSR-IOV is to leverage the *bonding driver* technique [14] that combines the vNIC and the SR-IOV to distribute the network bandwidths bias towards those network-intensive VMs, which in turn improves the overall network resource utilization. To this end, DSR-IOV first derives the workload nature of each VM via some monitor mechanism (e.g., XenMon in Xen), and then figures out the bandwidth distribution between the VMs according to their workloads. In particular, each of the network-intensive VMs is allocated a SR-IOV VF, which can be accessed directly without the interference from the hypervisor, while other compute-intensive VMs will share the vNIC, which is allocated a fixed quota of the network bandwidths.

Since the VM workloads are dynamically changed over time, DSR-IOV is also required to switch between the vNIC and the SR-IOV VFs in an adaptive way by allocating and deallocating the VFs among the VMs at runtime. Although the DSR-IOV is amenable to the network-intensive VMs for latency reduction, it leaves the hotspot issue untouched as it lacks the ability to remap the VMs to different hosts and re-engineer the network traffic.

2.2 VM Live Migration

VM live migration is a relatively mature technology often used to remap physical resources to virtual servers by moving all states and data of running VMs across different physical machines. Clark *et al.* [4] proposed and implemented a *pre-copy* approach that can accomplish an efficient VM live migration based on Xen [15] in several steps. First, the VLM copies all the volatile states of the VM from the source machine to the destination. During this process, the service may generate new dirty pages, which will be iteratively copied to the destination to keep the memory consistent. Next is the *stop-and-copy* phase where the source VM is shut down and a small number of non-synchronized memory pages are then copied to the destination. Lastly, the migrated VM is restarted at the destination to resume the service.

The pre-copy has become the predominant approach for VM live migration, supported by various VM Monitors such as Xen [16], VMWare [17], and KVM [18], whereby a trade-off between *service downtime* and *total migration time* can be well made to adapt to different situations [19]. Clearly, this migration strategy dose not fit nicely to memory intensive applications, where the VM has large page change rates (relative to the available bandwidth). In this case, *post-copy* live migration algorithm could be more friendly [20].

By virtue of its flexibility, VLM has been proposed to handle workload dynamics for different requirements in data centers [21], [22], [23]. In all the cases, the migration strategy regarding when to start migrating VMs and where to migrate the VMs is very important for the effectiveness of VLM. As such it is also an attractive research topic [24], [25], [26]. For example, Wood *et al.* [7] present *Sandpiper* – a framework designed to monitor, detect and get rid of hotspots across the data-center cluster. The essence of this framework is the proposed *Black-box and Gray-box* strategies, which can make migration decision by either simply observing each VM from the outside or investigating each VM from its inside.

The idea of migration strategy as well as its implementation adopted by *Sova* are largely borrowed from the Greybox strategy in [7] with a customization to our combination requirements.

2.3 Motivation Challenges

In this paper, we propose to coordinate DSR-IOV and VLM in an autonomic way to revolving around the QoS required by the production environments. To reason about this coordination, we made an experiment where a client VM leverages *Httperf* [27] benchmark to make a sequence of requests at rate of $2500/sec$ to a server VM for different sizes of data blocks. Depending on whether or not the two VMs are co-located, we compared the server's response times in different configurations as shown in Fig. 2. One can observe that for both co-located and non-colocated VMs, the response times under DSR-IOV are consistently better than those in default (using vNIC) as the data size increases over 128 KB, demonstrating the value of DSR-IOV. Unfortunately, DSR-IOV cannot effectively optimize the scenario when two communicating VMs are co-located on the same host as in this case a hotspot resulted from the network I/O contentions (via Dom0 in

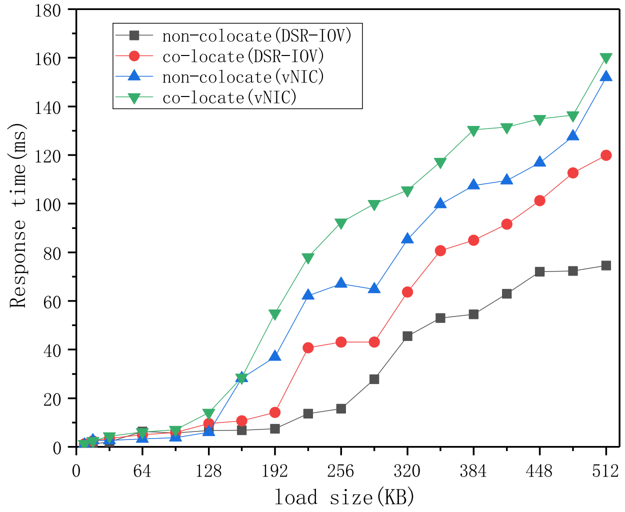


Fig. 2. Response time comparisons between different configurations. A client VM makes a sequence of requests at rate of $2500/sec$ to a VM server for different block sizes.

particular) could be incurred, motivating the integration of VLM to migrate out one VM for the QoS improvements.[2]

However, there are many challenges in making the coordination between DSR-IOV and VLM in terms of mechanism and strategy. In this research, we are particularly motivated by the following challenges to design our autonomic framework.

1) Combination of DSR-IOV and VLM. By the nature, DSR-IOV and VLM are two orthogonal techniques, one (DSR-IOV) primarily targets the local network optimization while the other (VLM) is a global optimization method. As such, they can work independently with each other, and it would be very hard to exert their respective strength if an effective combination approach between them is absent. DSR-IOV and VLM are combined in our framework, which needs to address two major challenges. Since DSR-IOV and VLM are implemented with different technologies, we not only have to analyze and deconstruct them but also need to synthesize them to re-construct a *mechanism* that can unify their control and support the dynamic switch between them with minimum overhead.

2) Autonomic Coordination. In addition to the combination mechanism, we also desire a generic framework that allows the combination to work in an autonomic way to improve the QoS of the VM services. As such, a closed control-loop to drive the combination with informed feedback is highly expected. To this end, an effective design for adaptive switch *strategy* between these two techniques is indispensable. Of course, this is highly dependent on the availability of local and global workload knowledge of VMs. However, given the diversity of data services in data centers, the workloads in VMs are not only very difficult to gather with minimum overhead, but also fairly hard to predict in accuracy. Given the two integrated adaptive methods, we need a more fined prediction model that can distinguish the cases amenable to either DSR-IOV or VLM. We have to address all these challenges in the first place to design our autonomic framework.

3 Sova Design

In this section, we present the design of *Sova*, the autonomic framework that combines the advantages of DSR-IOV and VLM for the network allocation optimization with the improvements of the QoS of VMs as the goal. We first introduce our design principles and related techniques, and then describe the *Sova* framework in more details with focus squarely on how the two technologies are fruitfully combined to make an autonomic coordination in a closed loop.

3.1 Design Principles

Sova is designed to coordinate DSR-IOV and VLM in an autonomic way, where the DSR-IOV is implemented locally to optimize the network allocation to each VM and the VLM is managed in a centralized fashion to remove the potential hotspots across the cluster. The rationale behind this design is that we need not only to well control the VM migration process but also to substantially reduce the DSR-IOV overhead otherwise incurred by the centralized control. To this end, we divide the design of *Sova* into two parts as follows, which corresponds to the solutions to the challenges we identified in the last section.

1) Software-Defined Combination. With the design principles in mind, we are inspired by IOFlow [13] to exploit the idea of software-defined technique to combine DSR-IOV and VLM where the DSR-IOV is implemented as a local process in each hypervisor to serve its VM communication and the VLM is well controlled via a separate network component (Sova controller) that disassociates the decision process (control plane) with the migrating process (data plane). The combination of the two techniques can not only re-balance the network traffic and remove the hotspots across the cluster but also prioritize those network-intensive VMs in a much finer way.

2) Autonomic Coordination. Based on the combination of DSR-IOV and VLM, we further design *Sova* as a generic framework that allows the combination to work in an autonomic way to improve the QoS of the VM services. To this end, *Sova* is built around the MAPE-K loop—a typical architecture in autonomic computing to implement adaptive software [11], [12]. The rationale behind this choice lies in the fact that the MAPE-K loop is not solely well suited to the autonomic control of *Sova* but also aligned with the software-defined architecture to coordinate the DSR-IOV and VLM for reduced service response time. So, we require the prediction model mentioned above be accurate, flexible, and scalable enough, so that it can effectively select whichever the better operation adaptive to the workload changes.

3.2 Overview of Framework Architecture

By following the design principles, we design *Sova* as a software-defined autonomic framework that exploits the MAPE-K loop to coordinate DSR-IOV and VLM for virtual network allocations with improved QoS of VM services as a goal. The architecture of *Sova* framework is shown in Fig. 3, where the cooperative components are designed according

[2] While this result is counter-intuitive, our further investigation demonstrated that the virtual network architecture of Xen creates a bottleneck in local-to-local communications which is not present in local-to-remote communications.

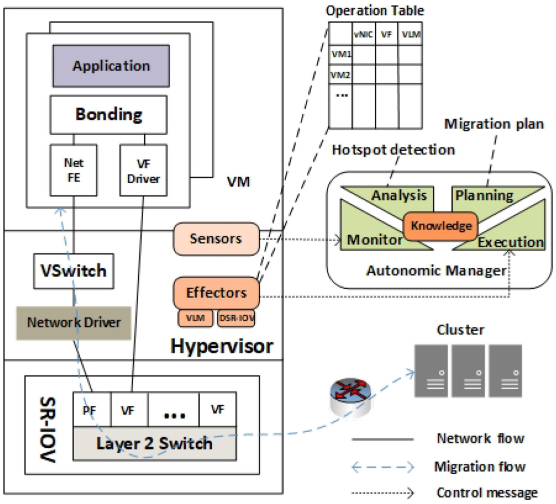


Fig. 3. Sova framework architecture.

to the MAPE-K model. Specifically, there are four components—*Monitor*, *Analysis*, *Planning* and *Execution*—that join together with a shared *Knowledge-base* to construct an *Autonomic Manager* who is coupled with the managed cluster—a collection of machines as well as their hosted VMs to endow them with autonomic behaviour.

Sensors, often called probes or gauges, collect the information about the managed cluster, which may include the response time to client requests, network and disk usage, CPU and memory utilization. Based on the gathered information, *Effectors* either perform the DSR-IOV to optimize local network allocation or carry out the VM migrations across the managed cluster to balance the VM workload distribution.

The data collected by the sensors allows the *Autonomic Manager* to *monitor* the managed cluster and *execute* the migration instructions through *Effectors*. The *Autonomic Manager* is a software component that ideally can be configured by human administrators for high-level goals, say QoS of VM services in our case, and exploits the monitored data from the sensors and internal *knowledge* to *plan* and execute the VM migrations. The internal knowledge of the framework is a topological model of the managed cluster.

Next, we describe the *Sova* framework from the perspectives of its managed cluster internals and autonomic manager with special attention on how the software-defined method is used to combine DSR-IOV and VLM, and then the autonomic coordination in more details.

3.3 Managed Clusters

We design a *Sensor* and an *Effector* local to each hypervisor where the *Sensor* is a software component to collect the data from each VM and the *Effector* is a software-defined component playing a dual role of a control plane and data plane in the framework for the network allocations. First, by using a prediction model for VM workload behaviors, the *Effector* implements the function of DSR-IOV in hypervisor that can prioritize the network-intensive VMs to have more network resources. Second, it provides a mechanism that not only translates the migration policy issued by *Autonomic Manager* into the VM-specific operations (control plane), but also coordinates the DSR-IOV and VLM to further adapt to the network changes (data plane).

3.3.1 Runtime Information Collection

We exploit the monitoring mechanism in hypervisor (say XenMon and `/proc`) to design a *Sensor* for the collection of the data from each VM by sampling, and then share the sampling data with the *Effector* in the hypervisor for the DSR-IOV and also send them to the *Autonomic Manager* for VM migration at the end of each sampling period. The sampling frequency is set with an attempt to minimize the overhead due to frequent read and write files. Specifically, there are three kinds of information gathered by the sensor.

1) *VM Workloads:*It can be derived that network-intensive VMs often have some common properties, such as, their CPU times are usually short, and their waiting time for network events are relatively long [5], [28], [29]. Therefore, the workload characteristics of the VMs, in terms of *CPU usage*, *blocked time* and *waiting time*, should be obtained in order to make a distinguish between the network-intensive VMs.
2) *Network Status:* To accurately assign VFs to VMs with intensive communication pattern, it is necessary to obtain the network information of the VM. In our design, the sensor acquires the network data of each VM through a gray-box approach [7], which leverages an installed lightweight monitoring daemon to gather the OS-level statistics. The total sent (or received) traffic minus the last sent (or received) traffic is the total traffic transmitted during this sampling period.
3) *Host Statistics:* The sensor also tracks the total resource usages from each host by aggregating the monitored data from all the resident VMs. As with gathering the network data from each VM, the sensor also gathers the memory usage through a monitoring daemon. These data are sent to the *Autonomic Manager* to detect if hotspot has occurred and to make migration decision.

3.3.2 Local DSR-IOV Optimization

The purpose of DSR-IOV is to minimize the network latency for those VMs exhibiting intensive network patterns so that their QoS can be improved. It configures two NICs for each VM via the *bonding driver* technique. One is vNIC automatically assigned by the hypervisor when a VM is created, and the other is SR-IOV VF whose (de)allocation is determined by the *Effector*. By default, each VM is configured with only the vNIC and when the VM becomes network intensive, the DSR-IOV would allocate a SR-IOV VF to the VM to improve its performance. On the contrary, if the VM is no longer network-intensive, its allocated VF will be deprived and granted to other network-intensive VMs. Dynamic allocation of the SR-IOV VFs can enable network-intensive VMs to have better network performance, thereby improving the overall network performance of the system.

As in [5], the *Effector* maintains two queues, *Priority Queue* (PQ) and *General Queue* (GQ), for more efficient accommodation of the hosted VMs. The PQ is designed for network-intensive VMs while the GQ for the others, including those VMs with disk I/O-intensive and CPU-intensive workloads. The purpose of this classification is allowing the *Effector* to quickly identify the types of VMs and efficiently allocate the SR-IOV VFs. As the bandwidth allocation across multiple VFs is based on hardware arbitration and the weighted average of VFs in the SR-IOV NIC, the VFs not only need to be allocated to those network-intensive VMs, but also desire to be granted sufficient bandwidth. This can be achieved by limiting the number of the VFs (also the length of the GQ) that share the finite network bandwidth.

Since the I/O-intensive VMs typically have shorter CPU time, while waiting for some I/O events in a long-time block state, we enable the *Effector* to combine I/O factor (*IF*) and network factor (*NF*) as in [5] to determine the network intensity of VM where

$$IF = \frac{\sum_0^{num} blocked\_time/gotten\_time}{num}, \tag{1}$$

which represents a statistical average of the ratio $blocked\_time/gotten\_time$, and num is the number of samplings. Given the features of I/O-intensive VMs, its IF value will be much larger than that of CPU-intensive VMs.

Although I/O factor IF can distinguish I/O-intensive VMs, but it cannot judge whether the VM is disk I/O-intensive or network I/O-intensive, so network factor (NF) is used to determine the network intensity of VM

$$NF = \varepsilon * Traffic\_freq + (1 - \varepsilon) * Traffic\_avg, \tag{2}$$

where $Traffic\_freq$ indicates the busyness of the network while $Traffic\_avg$ reflects the weighted average of the total traffic of VM network data, and parameter ε is used to balance the weights between the two terms. The higher the network intensity of VM, the larger network factor NF is

$$Traffic\_freq = \frac{traffic\_num}{num} \tag{3}$$

$$Traffic\_avg = \frac{\sum_0^{num} \frac{traffic - min}{max - min}}{traffic\_num}, \tag{4}$$

where $traffic\_num$ is the number of samples that have network transmissions in all samples, and $traffic$ is the total number of transmitted and received packets in one sampling period, and $\min(\max)$ is the smallest (largest) number of the packets in the sample.

By following the definitions of IF and NF, the DSR-IOV can determine the network intensity of each VM by first using IF to classify VMs into either I/O-intensive or non-I/O-intensive classes, and then exploiting NF to distinguish those network-intensive VMs from the I/O-intensive VMs as in [5].

3.3.3 Software-Defined Combination

The combination of DSR-IOV and VLM is achieved via a *Operation Table* maintained by the *Effector* in each host as shown in Fig. 3. The *Operation Table* is used to record whether each VM in the host uses the DSR-IOV or the VLM operations. To reflect a host (or VM) that may become intensive in one or more aspects in CPU, memory, and network, we borrow the idea from [26] to define a new metric for the *Effector* that captures the sheer volume of CPU, memory, and network to measure the hotness of host (or VM)

$$volume = \sqrt{\sum_{i=1}^{m} (\frac{r_i}{\overline{r}} - 1)^2}, \tag{5}$$

where r_i is the utilization of the ith resource corresponding to the host (or VM), and $\overline{r}$ is the average utilization of all m resources in the physical server.

As the VM migration process also takes up a certain amount of network bandwidth, it will affect the performance of the application service if the migration consumes too much bandwidth. On the other hand, when selecting a VM to migrate, the *Effector* needs to consider not only the heavy load, but also the memory size of the VM. Therefore, the *Effector* defines the VM *migration decision factor* $MF = volume/size$, where $size$ is the memory footprint of the VM. The migration algorithm sorts MF in descending order, and selects the VM with the largest MF to migrate. This allows the largest $volume$ (i.e, load) to be transmitted per unit byte, which has proven to be the minimum migration overhead [7], [30].

Given the consideration above, we populate the *Operation Table* with a coordinated scheduling algorithm as shown in Algorithm 1.

Algorithm 1. Combination of DSR-IOV and VLM

```
Require: $migration$ signal from Autonomic Manager;
$r = < r_1, \dots r_m >, size = < size_1, \dots, size_n >$ from Sensor;
the data to calculate $IF$ and $NF$ for each VM from Sensor;
Ensure: Operation[$VM_i$]=VLM or DSR-IOV, $i = 1, \dots, n$;
Calculates $IF_i$ and $NF_i$ for $VM_i, i = 1, \dots, n$;
Calculates $volume$ according to Eq. (5);
if ($migration.flag = True$) then
    for (each $VM_i$) do
        $MF_i \leftarrow volume/size_i$;
    end for
    $VM_k \leftarrow$ getMax($MF$);
    Operation[$VM_k$] $\leftarrow$ migration.targethost;
end if
// $q$ and $p$: size of minheap ( $VM's\ numbers > q > p$ )
$minIOHeap \leftarrow$ MinHeap( $q$ );
$minNetHeap \leftarrow$ MinHeap( $p$ );
for (each $VM_i$) do
    if (Operation[$VM_i$] = NULL) then
        $minIOHeap$.insert( $VM_i$, $IF_i$ );
    end if
end for
for ($VM_j$ in $minIOHeap$) do
    $minNetHeap$.insert($VM_j$, $NF_j$ );
end for
for (each $VM_i$ in $minNetHeap$) do
    Operation[$VM_i$] $\leftarrow$ VF;
end for
for (each $VM_j$ not in $minNetHeap$) do
    if (Operation[$VM_j$] = NULL) then
        Operation[$VM_j$] $\leftarrow$ vNIC;
    end if
end for
```

In this algorithm, the *Effector* first receives the migration signal $flag$ from the *Autonomic Manager*, the utilization of the ith resource r_i corresponding to the host (or VM) and the memory footprint $size$ of each VM from the *Sensor*. For each hosted VM, its $volume$ value and IF and NF are calculated (Line 1-2). Afterwards, if *flag* is true, the algorithm calculates MF for each VM, and then selects the VM with the largest MF value in the host as the VLM operation (Line 3-9). For other VMs, the algorithm uses a *heap tree* to prioritize the network-intensive VMs (Line 11-12). Specifically, the I/O-intensive VMs are obtained by using the $minIOHeap$, and then the $minNetHeap$ is used to obtain the network-intensive VMs from them (Line 13-20) and selects and marks them as using the VF operations in descending order (Line 21-23), while leaving the non-network-intensive VMs labeled as using the vNIC operations (Line 24-27). Based on Algorithm 1, each VM in the host is marked by a unique operation, either DSR-IOV (assigned VF/vNet) or VLM, which is performed by the *Effector* via a synchronized background daemon process. Note that in this process as with [26], only a single VM migration is performed for the overall QoS improvement. This design is reasonable because the hotspot could be removed after a single migration with minimum cost, or otherwise, it could be removed in the next decision run.

When the migration is triggered, the *Effector* would migrate the VM with the largest MF to the physical server with the smallest $volume$ (that is, the least loaded server) every time it is scheduled, until the migration is no longer triggered. The server with the smallest $volume$ is globally determined by the *Planner* in the *Autonomic Manager*.

3.4 Autonomic Manager

The managed cluster carries out local network optimization through DSR-IOV and delegates its plan of VM migration via the *Effector* to the *Autonomic Manager*, who is crafted to perform the global network optimization by leveraging the VLM to re-engineer the network traffic. The central manager has two main functions, *hotspot detection* and *migration planning*, which are accomplished by *Analysis* and *Planner* components, respectively, based on the gathered cluster-wide information from *Monitor* and the network topology from *Knowledge*.

3.4.1 Monitor and Knowledge

Since the availability of local and global workload knowledge of VMs determines the quality of the framework in exploiting the network resources for QoS improvements, we design *Monitor* to monitor and collect resource usages from the *Sensor* in each host and synthesize the gathered data as the state information for the host, which, together with a topology graph of the cluster, is maintained in *Knowledge*.

3.4.2 Autonomic Coordination

The manager detects the hotspots across the cluster, and predicts the future trends of workloads in each physical server based on the proposed prediction model. As a result, the manager should have global visibility to make its migration decision as shown in Algorithm 2 to command each individual *Effector* to translate the migration plan into VM-specific operations. In the following, we describe the algorithm based on the components of MAPE-loop to specify its functionality.

Algorithm 2. Autonomic Coordination

```
Require: $r =< r_1, \dots r_m >$ from Monitor;
$volume =< volume_1, \dots, volume_{p-1} >$ from Knowledge;
Ensure: migration[$Host_i$] $\leftarrow < True$ or $False, Host_j >$ , $i = 1, \dots, n, j \neq i$;
1:  Calculates $volume_p$ for $Host_i, i = 1, \dots, n$;
2:  for (each $Host_i$) do
3:    migration[$Host_i$] $\leftarrow < False, NULL >$ ;
4:    if ($k$ out of $p$ $volume$ values are greater than $\alpha$) then
5:      $\hat{v}olume_{p+1} \leftarrow \mu + \phi(volume_p - \mu) + \delta$;
6:      if ($\hat{v}olume_{p+1} > \alpha$) then
7:        $Host_j \leftarrow$ getMinNotInDestination($volume_p$);
8:        if (($volume_p[Host_j] < \alpha) and (Host_j \neq Host_i$)) then
9:          migration[$Host_i$] $\leftarrow < True, Host_j >$ ;
10:       end if
11:     end if
12:   end if
13:   send the migration[$Host_i$] signal to the $Effector_i$;
14: end for
```

Hotspot Detection. We leverage the threshold detection as the basic strategy for VLM, which means when the overall workloads of the physical server exceeds a certain threshold, the server is deemed to be overloaded and qualified to trigger the migration. However, in order to have stable spikes and avoid the unnecessary triggered migration, as with [7], a hotspot is flagged only when the threshold is continuously exceeded for a while. More specifically, if there are at least k reports exceeding the threshold out of any $n-1$ reports in time series (gathered from all the monitors), and the next report is also predicted to exceed the threshold, then a migration is triggered (Line 4). The values of n and k will directly affect the network performance, making the decision to trigger the migration either aggressive or conservative.

The *Analysis* component of the manager uses an auto-regressive family [7], [31] of predictors to predict the future data (Line 5-6). A pth order auto-regressive model, denoted by $AR(p)$, means that the p prior observations in conjunction with other statistics of the time series to predict the value of the next moment. For instance, for $AR(1)$, consider the time series $\{x_1, x_2, \dots, x_p\}$, and predict the value of the $(p+1)$th interval

$$\hat{x}_{p+1} = \mu + \phi(x_p - \mu) + \delta, \tag{6}$$

where μ is the mean value of the time series, parameter ϕ is used to capture the changes of the time series, and δ is the white Gaussian noise determined by the degree of fluctuations of the current sequence.

Migration Plan. The migration plan is made by the *Planner* component to get rid of the detected hotspot where only the overloaded host is informed to offload some workloads to underloaded hosts via the migration signal *flag*, leaving the actual migrated VM selection to the corresponding *Effector* itself (Line 13). Note that in order to ensure that the destination server also has sufficient resources to host the VM, the *Planner* need to calculate the heaviness of the destination server to discover the so-called *cold spot* as with [26], [32] before each migration. If the destination server load itself is heavy already (one or more resource usage exceeds threshold α), the *Planner* will not conduct a migration to it (Line7-10). With this design, we can grant much more freedom to

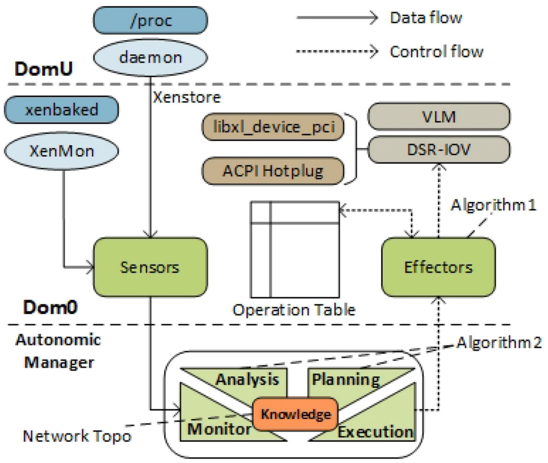


Fig. 4. Sova implementation architecture.

the *Effector*, who can make its own coordination between DSR-IOV and VLM (select the migrated VM) in reaction to the setting changes.

Clearly, a simple yet effective way is to migrate the most heavily loaded VMs to the least loaded server while minimizing the amount of data transferred during the migration process. As such, we can follow the same arguments in Section 3.3.3 to select the host with the least workloads as the destination host.

4 Implementation

We prototyped the *Sova* framework based on Xen4.9 running in a cluster in which all the VM storage is mounted to the same shared storage server through the iSCSI protocol so that it is not necessary to move the disk state when a VM is migrated. The live migration mechanism adopted by Xen is the *pre-copy* method [4], where the *pure stop-and-copy* and *pure demand-migration* mechanism are combined to minimize the downtime and the total migration time of the VM by iteratively copying memory pages.

The overall architecture of the *Sova* implementation is shown in Fig. 4 where the implementation of each component of the MAPE-K loop as well as the control and data flows are depicted. The *Sensor* residing in `Dom0` obtains the workload characteristics of the VM through *XenMon* [33], which is a performance analysis tool designed to capture the resource usage of each domain in Xen. The *XenMon* reports a variety of metrics for each VM, such as *CPU usage*, *blocked time* and *waiting time*. But as a real-time analysis tool, the *XenMon* cannot profile the VM executions over a period of time, instead only one CPU data is obtained at a time, so the sampled data is not accurate. Therefore, we modified the *XenMon* to collect the data from all the CPUs, and stored each sampled data into a log file for subsequent processing, and solved the problem of the data redundancy by periodically cleaning up the cache (i.e., shared memory) in *xenbaked*.

The *Sensor* gathers the network traffic information and the memory usage of each VM by periodically reading interface file `/proc`. Specifically, a daemon located in `DomU` obtains the historical total traffic of the VM by reading the information of the aggregated network card in file `/proc/net/dev`. Then, the size of the memory being used by the VM can be obtained by reading the file `/proc/meminfo`. Finally, the daemon sends the VM's network traffic data and memory information to the *Sensor* located in `Dom0` through `Xenstore`. Totally, the *Sensor* contains approximately 900+ lines of Python and C codes.

The *Effector* also runs in `Dom0`, which completes the functionality of the DSR-IOV and the combination of DSR-IOV and VLM by running Algorithm 1. The *Sensor* collects and reports the measurements once every `10` seconds, which is also the time interval for updating *Operation Table* and scheduling the SR-IOV VFs in the *Effector*. When dynamically allocating and removing the VFs, the *Effector* exploits *ACPI Hotplug* technology [34] to minimize the adverse effects on running VMs, and adopts the `libxl_device_pci` family of functions to manage the VFs as shown in Fig. 4. In order to facilitate the scheduling, the *Effector* also needs to record if there are VM creation, destruction, shutdown and other events by modifying function `create_domain()`, `destroy_domain()` and `shutdown_domain()` and others in `xen-4.9.1/tools/xl/xl_vmcontrol.c` file. In total, the *Effector* includes 1300+ lines of C source codes.

The *Autonomic Manager* is simply implemented as a daemon that runs on a control node of the cluster. It first acts the role of the *Monitor* to listen to the *Sensor* from each hypervisor for periodic usage reports and then uses these statistics to detect the hotspots and make the migration plan via Algorithm 2, which are the functions of the *Analysis* and *Planner* components played by the manager. Currently the migration is triggered when at least 3 out of the 5 most recent observations and the next predicted value exceeds a threshold. Totally, the *Autonomic Manager* comprises 700+ lines of C source codes.

5 Performance Evaluation

Sova is evaluated based on our cluster that consists of 43 physical servers inter-connected over 10 Gigabit Ethernet, among which 7 servers are installed Intel 82,599 network cards (a dual-port SR-IOV). One of the 7 servers is used as a shared storage server, while all the others run Linux 4.4.16 and Xen 4.9.1 and are equipped with 64 GB RAM. One node in the cluster is designated as the control node that runs *Autonomic Manager*, while each of the rest hosts one or more VMs, and runs both *Sensor* and *Effector* in `Dom0`.

As the goal of *Sova* is designed to improve the QoS of VM services by carefully allocating the network resources and removing the hotspots across the cluster, we measure its performance by comparing the service response time and bandwidth utilization.

5.1 DSR-IOV Effectiveness

Our first experiment demonstrates the effectiveness of DSR-IOV under different and dynamic workloads. Each VM could be installed CPU-intensive, network-intensive and/or hybrid workloads, which are mimicked by different benchmarks, i.e., *Lookbusy* [35] for the CPU-intensive workloads, *Netperf* [36] for the network-intensive workloads, and the combination of the two for the hybrid workloads. We run `15` VMs simultaneously on the same physical server, `VM1`-`VM15`, and the VMs are divided into three groups, each with `5` VMs, characterized by its hosted workloads.

TABLE 1
CPU-Intensive, Network-Intensive and Hybrid Workloads Property Values

CPU-intensive(Lookbusy)					
VMID	VM1	VM2	VM3	VM4	VM5
Time	20%	40%	60%	80%	100%
IF	9.22	1.06	0.46	0.24	1.01
NF	0	0	0	0	0
Network-intensive(Netperf)					
VMID	VM6	VM7	VM8	VM9	VM10
Time	20%	40%	60%	80%	100%
IF	213.78	103.39	66.08	47.06	47.54
NF	0.056	0.237	0.285	0.743	0.952
Hybrid(Lookbusy & Netperf)					
VMID	VM11	VM12	VM13	VM14	VM15
Time	20%	40%	60%	80%	100%
IF	11.47	5.52	2.26	0.37	0.46
NF	0.081	0.139	0.226	0.945	0.936

The characteristics of *Sova* when running different workloads are shown in Table 1 where each workload runs in different percentage $10\% - 100\%$ of 10-minute test time. We can observe from the table that the IF values of CPU-intensive VMs are always smaller than those of network-intensive VMs, and moreover, the longer the *Lookbusy* runs, the smaller the IF value is. For the VMs hosting hybrid workloads, as expected those with high CPU-intensive workloads often have equally small IF values. Besides, the higher the network intensity of the VM, the greater the NF value is, this observation also meets our expectation. Therefore, by using the IF value, *Sova* can successfully exclude the CPU-intensive VMs and those non-network-intensive VMs among the hybrid VMs. Consequently, the network-intensive VMs can be selected based on their NF values for network optimization.

Next, we verified whether *Sova* can discriminate the network-intensive VMs and dynamically allocate SR-IOV VFs to those network-intensive VMs. To this end, we ran 8 VMs on the physical server equipped with SR-IOV that has 4 VFs and let the workloads of VMs change every 10 minutes to see how the VFs are allocated among the VMs. The workloads of VM1-VM3 remain unchanged and simulate the network-intensive, the CPU-intensive and the hybrid workloads, respectively, while the workloads of other VMs change over time.

TABLE 3
VM's Workloads at Different Phases, Memory Allocations, and Initial Home Machines

VMID	Stage1	Stage2	Stage3	RAM(MB)	HM
VM1	50%	40%	40%	512	1
VM2	30%	70%	70%	512	1
VM3	30%	40%	30%	512	2
VM4	30%	30%	20%	512	2
VM5	20%	40%	40%	1024	3

Table 2 shows the trend of VM workloads over time and the distribution of VFs among VMs. The time instance of interest are 10 and 20, because the workload of the VM changes only at those two instances. Prior to time instance 10, VM1, VM6, VM7, and VM8 have the highest network intensity (i.e., the NF value is largest) and are assigned VFs. And then, there is a VF scheduling at the time instance 10, as VM7 and VM8 become CPU-intensive, their VFs are removed and re-assigned to VM3 and VM5, both are highly network-intensive. Later, in the moment of 20, there is a VF whose owner changed, that is, the VF of VM3 is removed and re-assigned to VM4, since the network intensity of VM4 is increased to align with VM3, but its IF value is much smaller than that of VM3. This is because for the hybrid VMs, *Sova* is biased towards the allocation of VFs to those network-intensive VMs. In summary, *Sova* is always able to dynamically grant VFs to the VMs with high network intensity.

5.2 Migration Effectiveness

We demonstrated the migration effectiveness of *Sova* by testing its migration hotspot detection and migration strategies. To this end, we constantly made service requests to the server and overload the server repeatedly to see how the VMs migrate between physical servers. In the experiment we used three physical servers and five VMs with memory allocations as shown in Table 3. We used *Netperf* to generate different workloads on different VMs in three stages, each generating a hotspot at different physical servers. The time percentages for each VM to run *Netperf* at different stages (again 10 minutes each stage), together with its home machine (HM), are shown in Table 3.

Fig. 5 shows a time series diagram of the system response process when the migration is triggered by running the workloads on different VMs. In the first stage, as the network

TABLE 2
VM Workload Changes Over Time in Minutes

Time(m)	State	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8
0-10	netperf	50%	0	30%	20%	40%	60%	80%	100%
	lookbusy	0	50%	30%	0	0	0	0	0
	NIC	VF	vNIC	vNIC	vNIV	vNIC	VF	VF	VF
10-20	netperf	50%	0	30%	20%	40%	60%	0	0
	lookbusy	0	50%	30%	0	0	0	20%	40%
	NIC	VF	vNIC	VF	vNIC	VF	VF	vNIC	vNIC
20-30	netperf	50%	0	30%	30%	50%	70%	0	0
	lookbusy	0	50%	30%	0	0	0	20%	40%
	NIC	VF	vNIC	vNIC	VF	VF	VF	vNIC	vNIC

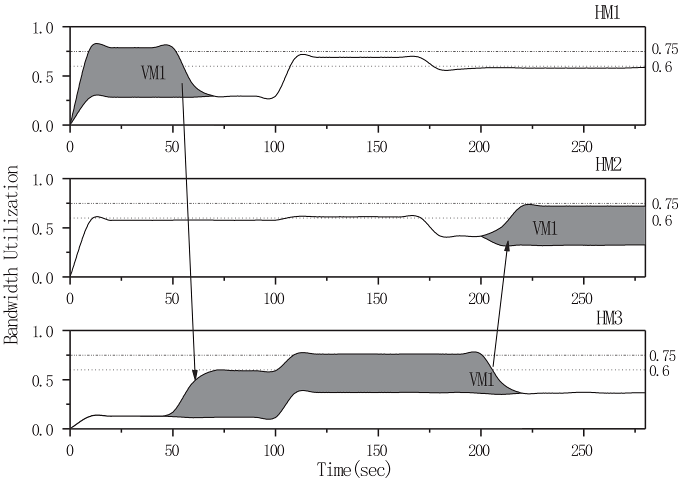


Fig. 5. Process of migrating VM across different physical servers to remove the hotspot where the shaded areas represent the migrating VM.

bandwidth utilization of HM1 continues to exceed the threshold, the system detects a hotspot at $t = 50$ seconds. Then, the *Effector* calculates the migration factor MF across all the VMs co-reside at HM1 and selects the candidate for migration in descending orders of MF. Since VM1 has the largest MF value, it is chosen as a candidate. In addition, the *Autonomic Manager* calculates the load status of the physical servers in the cluster and selects the physical server with the smallest $volume$ value as the migration destination. Here, HM3 has the smallest $volume$ value and enough space to accommodate VM1, so VM1 is migrated here to eliminate hotspot. This represents an ideal case for migration algorithms: if possible, we choose the most loaded VM from the overloaded physical server to migrate to the server with enough free resources.

In the second stage, HM3 becomes overloaded due to the load of VM5 increases. As the load of HM3 continues to exceed the threshold, the system generates a hotspot at $t = 150$ seconds, but no VM migration occurs. This verifies that in order to ensure that the destination server also has sufficient resources to receive the VM, the *Effector* should calculate the heaviness of the destination server before each migration. If the destination server itself were overloaded, the *Effector* would terminate the migration operation.

In the last stage, the load of the VMs is reduced in HM2, which has sufficient resource capacity to receive the migrated VM now. Therefore, the *Effector* triggers the migration to eliminate the hotspot in HM3 at $t = 200$ seconds. However, unlike the first case where the candidate VMs had identical memory footprints, VM1 has only half of the memory of VM5 in the HM3, but their network loads are almost the same, so VM1 is selected for migration. This shows that by selecting the VM with lower memory footprint, *Sova* can maximize the reduction in the load per byte of data transferred.

5.3 Effectiveness of Combination

Next, we demonstrated the effectiveness of *Sova* to combine DSR-IOV and VLM. To this end, we used two physical servers equipped with a SR-IOV with 2 VFs and 5 VMs with the same memory footprint to test the system's response time when the network loads change. VM1-VM4 are initially placed in HM1, while VM5 is placed in HM2. The network loads on VM1 steadily increase during the experiment, while the others remain constant. Because the changes of the VM workloads mainly occur in HM1, we focused on the system response time of HM1. And as before, the VMs use *Netperf* to simulate the changes of the network traffic.

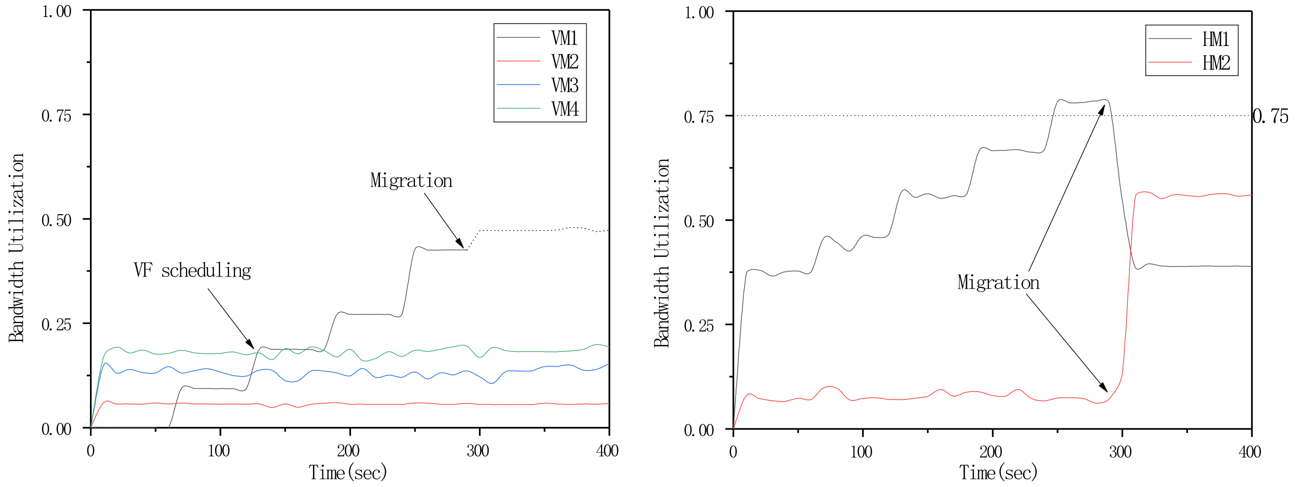
Fig. 6 shows how *Sova* uses either the DSR-IOV or the VLM to handle the network-intensive workloads. As shown in Fig. 6a, the VFs are allocated to VM3 and VM4 initially as they have the highest network intensity. However, with the increase of VM1 workload, a VF scheduling occurs at the time point $t = 140$ seconds. Since *VM1* has the highest network intensity at this time, the VF of VM3 are removed and re-assigned to *VM1* to improve its network performance.

As the network loads of VM1 continue to increase, as shown in Fig. 6b, the network of HM1 is overwhelmed and the effects of using the DSR-IOV to optimize the network performance is not significant at this time. Therefore, at $t = 290$ seconds, *Sova* detects the occurrence of a hotspot in HM1 and triggers the VM migration. Because the migration factor MF of VM1 is the largest, *Sova* picks it up as a migration candidate and revokes its granted VF. Later, VM1 is migrated to HM2, which has a large number of network resources. Through migration, the heavy network state in HM1 is alleviated, and the free resources in HM2 are utilized to improve the overall network performance.

5.4 Overall Performance

Now we are conducting experiments to evaluate the overall performance of *Sova* by comparing it with DSR-IOV and VLM in particular in terms of service response time. As these two compared techniques are orthogonal, they are relatively independent and, as building blocks, can be tested separately under the control of *Sova*. Since both DSR-IOV and VLM are derived from *Raccoon* [5] and *Sandpiper* [7], respectively, as described in Section 2, they can represent some existing technologies for comparisons.

To this end, we deployed 4 physical servers and 24 VMs. HM1 and HM2 run 12 VMs (VM1-VM12) as servers, while the other 12 VMs act as the clients in HM3 and HM4 and every time all the client VMs access the corresponding server VMs at the same time. *Apache* servers are installed on the server-side VMs, and the client-side VMs use *Httperf* to continuously send requests for accessing web pages in different sizes. The initial placements of the server-side VMs and the



(a) Changes of VM workloads in HM1.

(b) Changes of physical server workloads.

Fig. 6. VF scheduling and migration to handle the network-intensive workloads.

web page sizes in the VM are shown in Table 4. We gradually increased the request rates of *Httperf* in the client VM from 1,000 requests/s to 2,600 requests/s and initiated a total of 20,000 TCP connections when testing each request rate, and on each connection, 200 HTTP calls are performed (a call includes sending a request and receiving a response). We recorded the response time which includes response time and transfer time for the server-side VMs to serve the requests at various rates.

Fig. 7 shows the average response time of the system when different optimization strategies are used. As seen from the figure, DSR-IOV and VLM have a certain degree of optimization effects compared with the default strategy. Moreover, we can also see from the figure that *Sova* can combine the advantages of both DSR-IOV and VLM to match and even beat the better performance of each individual technology.

When the request rate is less than $2,300/s$, the DSR-IOV can optimize the network performance because it can dynamically allocate VFs to network-intensive VMs to reduce the network latency. However, as the request rate continues to increase, the server-side network is overwhelmed and the effects of using the DSR-IOV to optimize network performance is not beneficial. Compared with the DSR-IOV, the performance of the VLM is relatively better since it as a global network traffic optimization technique can balance the network loads on the server side. Fig. 8 illustrates the changes in the bandwidth utilization of the server-side machines when either the default strategy or *Sova* is used. As shown in Fig. 8a, the network loads of the default strategy in `HM1` is quickly overwhelming (when rate = $1,300/s$), which results in serious network performance impairments. Then *Sova* is used, as shown in Fig. 8b, `VM8` in `HM1` is migrated to `HM2` at rate = $1,200/s$, which has relatively large un-used network resources, to alleviate the network loads in `HM1`.

TABLE 4
Server-Side VM's Workloads and Initial Home Machines

VMID	VM1	VM2	VM3	VM4	VM5	VM6
load(KB)	8	8	32	32	128	128
Start HM	1	1	1	1	1	1
VMID	VM7	VM8	VM9	VM10	VM11	VM12
load(KB)	256	256	8	32	128	256
Start HM	1	1	2	2	2	2

Fig. 9 shows the distribution of service response times with respect to different load sizes when the request rates change. The VLM is amenable to the minimization of network traffic in the *coarse grained* communication case that a large amount of data tend to communicate between the local and remote VMs. As a result, it reduces the possibility of hotspot impact. As shown in Fig. 9b, the response time of the coarse-grained load (128 and 256 KB) in the VLM is significantly better than that of the default strategy shown in Fig. 9a. In contrast, the DSR-IOV can effectively handle the *fine grained* communication situation (8 and 32 KB) and reduce the network delay of fine-grained communication mode without obvious impact on the coarse-grained communication mode, as shown in Fig. 9c. In comparison, Fig. 9d shows that *Sova* has the best optimization effect

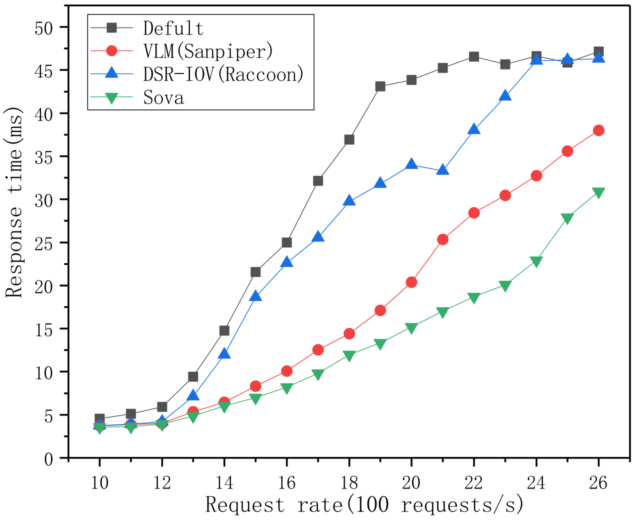
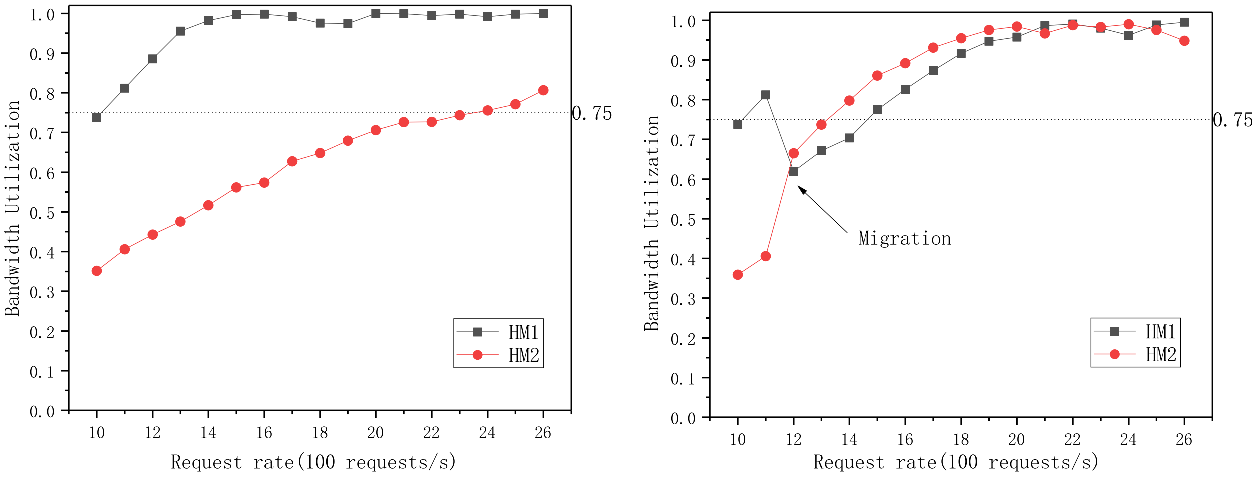


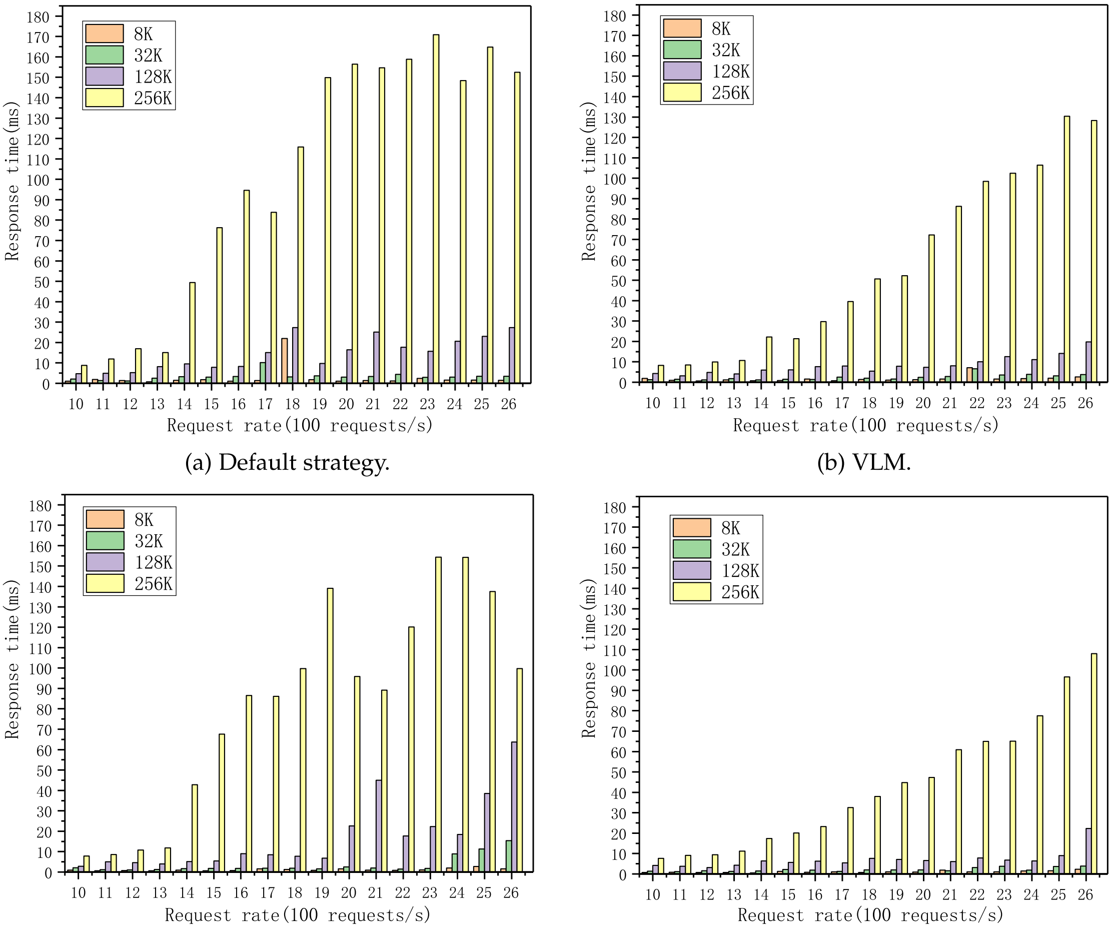
Fig. 7. Performance comparison of different optimizations.



(a) Using default strategy

(b) Using Sova

Fig. 8. The bandwidth utilization changes of server-side physical servers under different strategies.



(a) Default strategy.

(b) VLM.

(c) DSR-IOV.

(d) Sova.

Fig. 9. The response time distribution of different loads in different sizes when the request rate changes.

given its combination of the advantages of DSR-IOV and VLM.

5.5 System Overhead

In this section, we measured the system overhead of *Sova* whose CPU and network overhead is dependent on the number of HMs and VMs in the data center.

Network Overhead. In *Sova*, the network overhead mainly occurs in the data exchange between the *Sensors* and the *Autonomic Manager* and the control commands initiated by the *Manager* to the *Effectors* across cluster. Host statistics and usage reports are sent from each *Sensor* to the *Autonomic Manager* every measurement interval (10 seconds by default). Table 5 a shows the breakdown of network overhead in

TABLE 5
System Overheads for HM

Log type	Bytes
Host statistics	224
commands	80
Total	304

(a) Network overhead.

Num. of VMs	Overhead
1	1.12%
4	1.2%
8	1.24%
16	1.36%
32	1.4%

(b) CPU overhead

physical servers when running 12 concurrent VMs. Since each physical server uses only 304 bytes of network overhead every measurement interval, which could be negligible for 10 Gbs networks.

CPU Overhead. To evaluate the CPU overhead, we compared the performance of the CPU benchmark with and without *Sova* optimization. Table 5 b shows the CPU overhead incurred during running multiple VMs on a single physical server at the same time. We compared the major overhead coming from *XenMon* [33], which appears to have extra $1-2\%$ CPU overhead. This demonstrates the availability of our approach.

6 Related Work

Given the inherent difficulties, improving the QoS of VM-based services in data center with response time reduction as a goal generally needs holistic approaches that can take advantages of various techniques to optimize the virtual network allocation and the computational workload scheduling. However, most of existing techniques work in piecemeal fashion, either focusing on the networking [5], [37], [38] or delving into the compute resources [32], short of the notion of the combination to exert respective strengths to address both the local and global computation issues.

Networking. There are tremendous studies on network optimization to improve the QoS of VM-based services in virtual environments [10], [39], [40], [41], [42]. For example, Kaushik *et al.* [43] proposed a hardware-supported method to reduce the overhead of driver domain in Xen so that a multi-queue network interface could be exploited for network performance. In contrast, Bourguiba *et al.* [37] presented an aggregation-based mechanism to facilitate the transfers of packets from driver domain to user domains with overcoming the network performance bottleneck as the goal.

Unlike the foregoing studies, which focus on the packet processing, other studies optimize the virtualization model itself [39], [44]. Gordon *et al.* [39] designed *ELI* (Exit-Less Interrupts) to remove the hypervisor from the interrupt processing path and transfer the physical interrupts directly to its VM for response time reduction. Agesen *et al.* [44] identified instruction clusters that would usually result in multiple exits and translated them together with an attempt to reduce the frequency of VM exits. In contrast, Guan *et al.* [10] presented a workload-aware scheduler that limits the total number of credits and allocates more credits to I/O-intensive VMs to improve bandwidth and reduce response time. Although these efforts can more or less improve the QoS of the VM-based services, they are largely limited to a single physical server, which could cripple the performance when hotspots occur.

Compute Resources. Dynamic VM placements for load balancing in the data center via VLM is a well-studied approach to optimizing the compute resource utilization for the QoS improvement of VM-based services [45], [46], [47], [48]. Wood *et al.* [7] proposed *Sanpiper*, a system that can automatically monitor and detect hotspots, using black-box and gray-box strategies to guide the dynamic remapping of VMs to physical servers, so that the hotspots in the system can be eliminated. Xiao *et al.* [26] adopted a similar idea to prevent hotspots in the system effectively while saving energy by minimizing the number of used servers as a goal.

As opposed to the previous studies, which focus squarely on local optimization in clusters, Hermenier *et al.* [49] proposed *Entropy* that exploits the constraint programming to perform global optimizations and takes the migration costs into account to further improve the remapping effects. Although the dynamic provisioning of virtual services in clusters can effectively improve the QoS to a certain degree, it lacks the ability to fine-tune the resource allocation for the optimal QoS.

Holistic Methods. In contrast to the aforementioned works, which are piecemeal per se, the proposed *Sova* is a holistic method that combines the advantages of DSR-IOV and VLM to improve the QoS of VM-based services in data center. Of course, the combination idea is not new and it can be found in or achievable from some literature [32], [50]. For example, Giurgiu *et al.* [32] proposed a concept of *cold spot* with an aim at addressing the defects that most existing methods experience by integrating all factors together and making the problem of virtual infrastructure placement effective and manageable. However, the resulted placement is static in nature, not adaptive to the dynamic environment changes. As for the combination of DSR-IOV and VLM, one naive solution is the software-based switch such as *Open vSwitch* [50], which could enable not only the state tracking at the per-flow level but also the VM migration if there is any detected hotspot. However, this technique requires the hypervisor to remain inline to bridge the traffic between VMs and the outside world, which is different from using SR-IOV in *Sova* that can bypass the hypervisor with respect to the network operations.

In design, *Sova* has two notable features, compared with the existing methods that are built on top of similar technologies [13], [51], [52], [53], [54]. First, it leverages the software-defined method to combine DSR-IOV and VLM as with IOFlow [13], which is a software-defined storage architecture that enables end-to-end I/O policies in data center, and second, it exploits the MAPE-K loop to coordinate these two operations in an autonomic way to self-adapt to the environment changes, which also bears some similarities to the designs of both [51], [54] where the MAPE-K loop is applied either to the autonomic management of cloud infrastructure [51] or to the service-based cloud application itself when combined with reinforcement learning algorithm [54]. *Sova* unifies these two technologies for response time reduction to improve the QoS of VM-based service, making it distinct from the existing works. A similar autonomic framework design with the same performance goal, yet for data streams is the work presented by Tolosana-Calasanz *et al.* [53]. However, it is a feedback-control and queuing theory-based controller to elastically provision VMs for the goal, different from the MAPE-K loop adopted by *Sova*.

7 Conclusion

In this paper, we presented *Sova*, an autonomic framework to combine the strengths of DSR-IOV and VLM to improve the QoS of VM services by optimize the network allocations. On the one hand, the DSR-IOV can improve the network performance of network-intensive VMs by granting more network resources, and on the other hand, the VLM operations can complement the DSR-IOV to cope with the hotspot issues by re-engineering the network traffic.

Sova is designed as a generic framework by following the model of MAPE-K loop in autonomic computing to centralize the control intelligence in a separate network component (Autonomic Manager) through a software-defined method. With *Sova*, the migrating process of VMs (data plane) is disassociated from the decision process (control plane). Moreover, the controlled VLM is also adaptively coordinated with the locally performed DSR-IOV, enabling a holistic approach to the network allocations. We prototyped *Sova* based on Xen4.9 and conducted experiments to show that *Sova* can combine the advantages of both DSR-IOV and VLM with acceptable overhead to match and even beat the better QoS of each individual technology by adapting to the VM workload changes.

Acknowledgments

The authors would like to thank the anonymous reviewers for their invaluable feedback. This work was supported in part by National Key R&D Program of China (No. 2018YFB1004804), National Natural Science Foundation of China (61672513) and also in part by Science and Technology Planning Project of Guangdong Province (No. 2019B010137002), Shenzhen Oversea High-Caliber Personnel Innovation Funds (KQCX20170331161854), and Shenzhen Basic Research Program (JCYJ20170818153016513).

References

[1] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance study on 10 BbE NICs with SR-IOV support,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2010, pp. 1–12.

[2] PCI-SIG, “PCI I/O Virtualization,” 2013. [Online]. Available: http://www.pcisig.com/ specifications/iov/

[3] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, “High performance network virtualization with SR-IOV,” *J. Parallel Distrib. Comput.*, vol. 72, no. 11, pp. 1471–1480, 2012.

[4] C. Clark *et al.*, “Live migration of virtual machines,” in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implementation*, 2005, pp. 273–286.

[5] L. Zeng, Y. Wang, X. Fan, and C. Xu, “Raccoon: A novel network I/O allocation framework for workload-aware VM scheduling in virtual environments,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2651–2662, Sep. 2017.

[6] Y. Dong, Z. Yu, and G. Rose, “SR-IOV networking in Xen: Architecture, design and implementation,” in *Proc. 1st Conf. I/O Virtualization*, 2008, Art. no. 10.

[7] T. Wood *et al.*, “Black-box and gray-box strategies for virtual machine migration,” in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2007, pp. 17–17.

[8] X. Bu, J. Rao, and C.-Z. Xu, “Interference and locality-aware task scheduling for MapReduce applications in virtual clusters,” in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2013, pp. 227–238.

[9] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, “Improving MapReduce performance in heterogeneous environments,” in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, 2008, Art. no. 7.

[10] 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, Second Quarter 2014.

[11] M. Parashar and S. Hariri, “Autonomic computing: An overview,” in *Proc. Int. Workshop Unconventional Program. Paradigms*, 2004, pp. 257–269.

[12] M. C. Huebscher and J. A. McCann, “A survey of autonomic computing-degrees, models, and applications,” *ACM Comput. Surv.*, vol. 40, no. 3, 2008, Art. no. 7.

[13] E. Thereska *et al.*, “IOFlow: A software-defined storage architecture,” in *Proc. 24th ACM Symp. Operating Syst. Princ.*, 2013, pp. 182–196.

[14] T. Davis, W. Tarreau, C. Gavrilov, C. N. Tindel, J. Girouard, and J. Vosburgh, “Linux ethernet bonding driver howto,” Linux Channel Bonding project, 2011. [Online]. Available: http://sourceforge.net/projects/bonding/

[15] P. Barham *et al.*, “Xen and the art of virtualization,” *ACM SIGOPS Operating Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[16] Xen, “Xen Project,” 2020. [Online]. Available: https://xenproject.org/

[17] VMware, 2020. [Online]. Available: https://www.vmware.com

[18] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “KVM: The Linux virtual machine monitor,” in *Proc. Linux Symp.*, 2007, pp. 225–230.

[19] N. Tziritas, T. Loukopoulos, S. U. Khan, C. Xu, and A. Y. Zomaya, “Online live VM migration algorithms to minimize total migration time and downtime,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2019, pp. 406–417.

[20] M. R. Hines, U. Deshpande, and K. Gopalan, “Post-copy live migration of virtual machines,” *ACM SIGOPS Operating Syst. Rev.*, vol. 43, no. 3, pp. 14–26, 2009.

[21] A. Zhou, S. Wang, X. Ma, and S. S. Yau, “Towards service composition aware virtual machine migration approach in the cloud,” *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2962128.

[22] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, “Learn-as-you-go with Megh: Efficient live migration of virtual machines,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1786–1801, Aug. 2019.

[23] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, “Power-aware and performance-guaranteed virtual machine placement in the cloud,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1385–1400, Jun. 2018.

[24] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, “Live migration of multiple virtual machines with resource reservation in cloud computing environments,” in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, 2011, pp. 267–274.

[25] J. Hu, J. Gu, G. Sun, and T. Zhao, “A scheduling strategy on load balancing of virtual machine resources in cloud computing environment,” in *Proc. 3rd Int. Symp. Parallel Archit. Algorithms Program.*, 2010, pp. 89–96.

[26] Z. Xiao, W. Song, and Q. Chen, “Dynamic resource allocation using virtual machines for cloud computing environment,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.

[27] D. Mosberger and T. Jin, “httperf—A tool for measuring web server performance,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, 1998.

[28] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, “Performance analysis of network I/O workloads in virtualized data centers,” *IEEE Trans. Services Comput.*, vol. 6, no. 1, pp. 48–63, First Quarter 2013.

[29] U. Vallamsetty, P. Mohapatra, R. Iyer, and K. Kant, “Improving cache performance of network intensive workloads,” in *Proc. Int. Conf. Parallel Process.*, 2001, pp. 87–94.

[30] V. Sundaram, T. Wood, and P. Shenoy, “Efficient data migration in self-managing storage systems,” in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2006, pp. 297–300.

[31] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2015.

[32] I. Giurgiu, C. Castillo, A. Tantawi, and M. Steinder, “Enabling efficient placement of virtual infrastructures in the cloud,” in *Proc. ACM/IFIP/USENIX Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, 2012, pp. 332–353.

[33] D. Gupta, R. Gardner, and L. Cherkasova, “XenMon: QoS monitoring and performance profiling tool,” Hewlett-Packard Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2005–187, 2005, pp. 1–13.

[34] L. Brown, “ACPI in Linux,” in *Linux Symp.*, vol. 51, pp. 51–67, 2005.

[35] D. Carraway, “Lookbusy—A synthetic load generator,” *Look Busy*. Accessed: Aug., vol. 18, 2013, Art. no. 2017.

[36] “Netperf,” 2020. [Online]. Available: https://hewlettpackard.github.io/netperf

[37] M. Bourguiba, K. Haddadou, I. El Korbi, and G. Pujolle, “Improving network I/O virtualization for cloud computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 673–681, Mar. 2014.

[38] J. Li *et al.*, “When I/O interrupt becomes system bottleneck: Efficiency and scalability enhancement for SR-IOV network virtualization,” *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1183–1196, Fourth Quarter 2019.

[39] A. Gordon *et al.*, “ELI: Bare-metal performance for I/O virtualization,” in *Proc. 17th Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2012, pp. 411–422.

[40] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “SR-IOV support for virtualization on InfiniBand clusters: Early experience,” in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2013, pp. 385–392.

[41] 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,” *ACM SIGPLAN Notices*, vol. 50, no. 7, pp. 17–30, 2015.

[42] 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.

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

[44] O. Agesen, J. Mattson, R. Rugina, and J. Sheldon, “Software techniques for avoiding hardware virtualization exits,” in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 373–385.

[45] N. Bobroff, A. Kochut, and K. Beaty, “Dynamic placement of virtual machines for managing SLA violations,” in *Proc. 10th IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2007, pp. 119–128.

[46] B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” *J. Netw. Syst. Manage.*, vol. 23, no. 3, pp. 567–619, 2015.

[47] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “CloudScale: Elastic resource scaling for multi-tenant cloud systems,” in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, pp. 1–14.

[48] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, “Dynamic resource management using virtual machine migrations,” *IEEE Commun. Mag.*, vol. 50, no. 9, pp. 34–40, Sep. 2012.

[49] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, “Entropy: A consolidation manager for clusters,” in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virt. Execution Environ.*, 2009, pp. 41–50.

[50] B. Pfaff *et al.*, “The design and implementation of open vSwitch,” in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 117–130.

[51] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic, “Revealing the MAPE loop for the autonomic management of cloud infrastructures,” in *Proc. IEEE Symp. Comput. Commun.*, 2011, pp. 147–152.

[52] M. Liu *et al.*, “ACIC: Automatic cloud I/O configurator for HPC applications,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2013, pp. 1–12.

[53] R. Tolosana-Calasanz, J. Diaz-Montes, O. F. Rana, and M. Parashar, “Feedback-control & queueing theory-based resource management for streaming applications,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1061–1075, Apr. 2017.

[54] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, “An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach,” *Future Gener. Comput. Syst.*, vol. 78, pp. 191–210, 2018.

Zhiyong Ye received the BSc degree in communication engineering from Nanchang University, Nanchang, China, in 2016, and the MS degree in electronics and communications engineering from Chongqing University, Chongqing, China, in 2019. He was an intern with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences from 2017 to 2019, where he worked on network virtualization technology. He is currently working with Baidu, Shanghai as a software engineer. He is interested in system software and virtualization technology in cloud computing.

Yang Wang received the BSc degree in applied mathematics from the Ocean University of China, Qingdao, China, in 1989, the MSc degree in computer science from Carleton University, Ottawa, Canada, in 2001, and the PhD degree in computer science from the University of Alberta, Edmonton, Canada, in 2008. He currently works with the 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).

Shuibing He received the PhD degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2009. He worked with Wuhan University, China, as an associate professor from 2015 to 2018 and a professor with the College of Computer Science and Technology, Zhejiang University, China, afterwards. His current research areas include parallel I/O system, file and storage system, high-performance computing, and distributed computing.

Chengzhong Xu (Fellow, IEEE) received the PhD degree from the University of Hong Kong, Hong Kong, in 1993. He is currently the dean of Faculty of Science and Technology, University of Macau, China, and the director of the Institute of Advanced Computing and Data Engineering, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests include parallel and distributed systems and cloud computing. He has published more than 200 papers in journals and conferences. He serves on a number of journal editorial boards, including the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*, and *China Science Information Sciences*.

Xian-He Sun (Fellow, IEEE) received the BS degree in mathematics from Beijing Normal University, Beijing, China, in 1982, and the MS and PhD degrees in computer science from Michigan State University, East Lansing, Michigan, in 1987 and 1990, respectively. He is a distinguished professor with the Department of Computer Science, Illinois Institute of Technology (IIT), Chicago. He is the director of the Scalable Computing Software Laboratory, IIT, and is a guest faculty with the Mathematics and Computer Science Division, Argonne National Laboratory. His research interests include parallel and distributed processing, memory and I/O systems, software systems, and performance evaluation and optimization.