Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4708195, 10 pages
http://dx.doi.org/10.1155/2016/4708195

Research Article

ANCS: Achieving QoS through Dynamic Allocation of Network Resources in Virtualized Clouds

Cheol-Ho Hong, Kyungwoon Lee, Hyunchan Park, and Chuck Yoo

Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea

Correspondence should be addressed to Chuck Yoo; chuckyoo@os.korea.ac.kr

Received 27 February 2016; Accepted 19 May 2016

Academic Editor: Zhihui Du

To meet the various requirements of cloud computing users, research on guaranteeing Quality of Service (QoS) is gaining widespread attention in the field of cloud computing. However, as cloud computing platforms adopt virtualization as an enabling technology, it becomes challenging to distribute system resources to each user according to the diverse requirements. Although ample research has been conducted in order to meet QoS requirements, the proposed solutions lack simultaneous support for multiple policies, degrade the aggregated throughput of network resources, and incur CPU overhead. In this paper, we propose a new mechanism, called ANCS (*Advanced Network Credit Scheduler*), to guarantee QoS through dynamic allocation of network resources in virtualization. To meet the various network demands of cloud users, ANCS aims to concurrently provide multiple performance policies; these include weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. In addition, ANCS develops an efficient work-conserving scheduling method for maximizing network resource utilization. Finally, ANCS can achieve low CPU overhead via its lightweight design, which is important for practical deployment.

1. Introduction

The use of cloud computing technology is rapidly increasing, because it can provide computing resources to remote users efficiently in terms of time and cost. By using cloud platforms, cloud users can run diverse applications without considering the arrangement of the hardware platforms. In addition, because of the agile and elastic traits of cloud computing, the amount of computing resources can be adjusted to reflect the requirements of each user. Therefore, cloud computing is rapidly being disseminated for general-purpose, network and database server, and high-performance computing applications [1].

System virtualization [2] is an enabling technology of cloud computing. A hypervisor or virtual machine monitor (VMM) provides cloud users an illusion of running their own operating system (OS) on a physical platform. Using a hypervisor in the basement of the software layer, cloud vendors can realize IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Service as a Service), which are core services of cloud computing. Recent representative hypervisor titles for cloud computing include KVM [3], Xen [4], and VMware ESXi [5].

To satisfy the diverse requirements of cloud users, cloud computing providers have recently allowed users to select the Quality of Service (QoS) of each resource. As a result, users can specify the needs appropriate to their programs and pay fees according to the QoS level. However, it is challenging for cloud providers to support stable QoS for applications in each virtual machine (VM) because of interference between cloud users [6]. In such environments, users can experience unpredictable performance and performance degradation. Therefore, research on achieving performance isolation during resource sharing is gaining significant attention [6–10].

In particular, guaranteeing network performance is widely studied because network performance influences the Quality of Experience (QoE) that defines the satisfaction level of each user. Most previous research efforts [11–14] focused on methods to dynamically adjust network performance. However, the proposed methods have limitations in that (1) they can support only one type of policy among several viable ones and (2) they incur high CPU overhead by adopting feedback control, which frequently monitors the

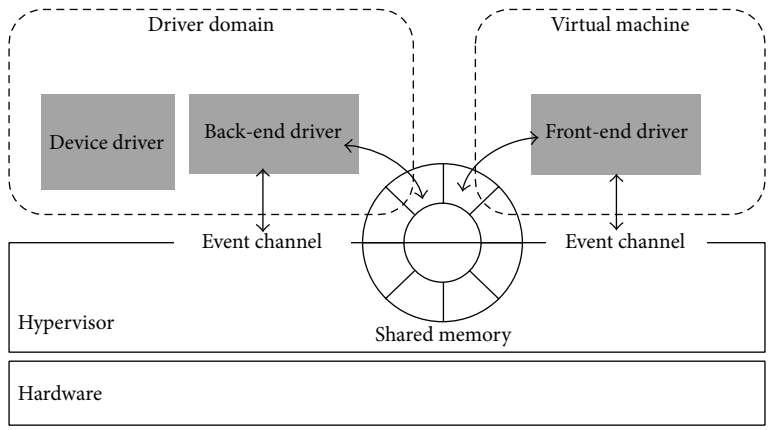

Figure 1: Xen I/O architecture.

actual usage of the network and compensates insufficient resources. Therefore, a new approach is required to meet the demands of the users while significantly reducing overhead.

In this paper, we propose ANCS (*Advanced Network Credit Scheduler*), which can dynamically provide network performance according to the requirements of each user of a virtualized system. Compared to solutions proposed in the previous research, ANCS can support several network performance control policies simultaneously; these include weight-based proportional sharing, minimum bandwidth guarantees, and maximum bandwidth guarantees. In addition, at all times ANCS provides a work-conserving scheme [15] that distributes the unused resources of any VM to other VMs, which maximizes total network utilization. Finally, ANCS incurs low CPU overhead through its lightweight design.

The main contributions of this paper related to previous studies are as follows:

(i) We deliver the details of ANCS that can satisfy diverse performance demands through multiple performance policies in virtualization. Each cloud computing tenant has different performance requirements. In order to satisfy the needs, ANCS provides three performance policies: weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. We elaborate on how to guarantee desired performance according to each demand.

(ii) We develop an efficient work-conserving method for utilizing network resources. In a non-work-conserving method, when a VM cannot fully utilize an allocated resource, the unused amount is wasted, resulting in decreased overall network utilization. To maximize the utilization, ANCS develops an efficient work-conserving method to guarantee high utilization regardless of the resource usage of each VM.

(iii) We design ANCS to use minimal CPU resources when managing the network performance of VMs. We do not utilize packet inspection or packet queuing to reduce overhead in the QoS control.

The remainder of this paper is structured as follows: In Section 2, we explain the background of virtualization and the methods it uses to distribute network resources. Section 3 elaborates on the design of ANCS. Section 4 shows the performance evaluation results. Section 5 explains related work. Finally, we present our conclusions in Section 6.

2. Background and Motivation

We select Xen [4] for implementing ANCS because it is an open-source hypervisor and also supports a simple round robin scheduling method. We can regard this scheduling method as a baseline for our experiment and intend to improve the scheduling method in terms of supporting weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation.

2.1. System Virtualization and Xen. System virtualization allows several OSs to be consolidated on a single physical machine. At the bottom of the system software stacks, the hypervisor multiplexes all system resources, including processors, memory, and I/O devices; it exports the virtualized resources in the form of a VM. Therefore, each VM can provide a complete system environment (composed of virtual CPUs, virtual memory, and virtual devices) to each guest OS.

Xen [4] is an open-source hypervisor that adopts a paravirtualization technique to minimize virtualization cost. The paravirtualization technique enables communication between the hypervisor and a guest OS by providing a communication channel, which is referred to as a hypercall. Hypercalls are analogous to system calls between an OS and a user application. Hypercalls request Xen to execute sensitive instructions on behalf of the guest OS. They include processor state update operations such as memory management unit (MMU) updates and physical interrupt masking operations, which guest OSs cannot execute directly. To contain hypercalls, the source code of a guest OS must be modified to facilitate paravirtualization.

2.2. The I/O Model of Xen. To process I/O requests from guest OSs, Xen adopts the back-end driver in domain 0, which is a driver domain, and the front-end driver in the guest OS [16], as depicted in Figure 1. The back-end driver delivers

I/O requests using the shared memory from the front-end drivers to the actual device driver in domain 0. Afterward, the back-end driver conveys the responses to each front-end driver by notifying them via virtual interrupts. The front-end driver behaves as an actual device driver in the guest OS. It receives I/O requests from the guest OS and delivers them to the back-end driver. It also brings the processed result to the guest OS. For network devices, the front-end network driver creates virtual network interfaces called *vif*s. The virtual network interfaces behave as the actual network devices in the guest OS. In systems that have several VMs, the back-end network driver processes the requests of several virtual network interfaces in a round-robin manner without a proportional sharing policy. Therefore, each competing guest OS will have the same network performance, which cannot satisfy the different requirements.

3. Design of ANCS

In this section, we present the details of ANCS, an advanced network credit scheduler that dynamically allocates network resources to VMs for guaranteeing the network performance of VMs under various scenarios. First, we describe the design goals of ANCS that aim to guarantee QoS for network virtualization. Next, we elaborate on the coarse-grained algorithm of ANCS using its flow chart. Finally, we explain the fine-grained subalgorithms of ANCS for achieving diverse performance goals.

3.1. Design Goals. The goals of ANCS that aim to implement a network scheduler for guaranteeing QoS in virtualization are as follows.

(i) Multiple Performance Policies. ANCS aims to meet the network demands of each VM, which are described by multiple performance policies. Recent cloud computing users tend to run diverse network applications on their VMs. These applications have different performance demands, especially in terms of networks; minimum sustainable bandwidth is an example of such a requirement. In order to meet the various needs, ANCS provides the following three performance policies: weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. ANCS aims to satisfy the desired performance of cloud users by allowing them to select an appropriate performance policy among the three policies, according to their demands.

(ii) High Network Resource Utilization. ANCS endeavors to efficiently utilize the total network resources of a system by applying a work-conserving method. In a non-work-conserving environment, when a VM receives network resources in proportion to its weight and does not consume its allocated amount, the unused network resources then become wasted, resulting in decreased total network utilization. In order to enhance resource management efficiency, ANCS adopts a work-conserving method, which yields unused network resources to other VMs that have pending network requests. This maximizes the resource utilization of the entire system regardless of the usage of each VM.

(iii) Low CPU Overhead. ANCS focuses on minimizing additional CPU overhead in achieving QoS for networks. Other research efforts that aimed to guarantee QoS failed to reduce CPU overhead, because of the complexity of the proposed algorithms [12, 14, 17]. These studies adopted either packet inspection or packet queuing for throttling the sending packet rate per network interface, and therefore this mechanism demands nonnegligible computation resources. Unfortunately, this causes total network performance to deteriorate. ANCS adopts a credit-based simple accounting algorithm that incurs only minimal overhead, which prevents performance degradation during the QoS control.

Although ANCS is implemented in the back-end driver of the Xen hypervisor, ANCS can be applied to other hypervisors that use a paravirtual model where the back-end and the front-end cooperate with each other to deal with network packets. In a paravirtual model, each hypervisor processes packets in a similar manner. For example, the back-end receives/sends packets from/to the front-end by using shared memory composed of descriptor rings. Also, after receiving and sending packets, the back-end delivers virtual interrupts to the front-end in order to notify that the packets have been processed. Because of this architectural similarity, we believe that ANCS can still satisfy the performance requirements in other hypervisors.

3.2. ANCS Algorithm. In basic terms, ANCS is based on the proportional share scheduling mechanism that allocates network resources to each virtual interface in proportion to its weight. Each virtual interface, called *vif*, is owned by its VM and behaves like an actual network interface within the VM. ANCS operates in the driver domain of the Xen hypervisor, particularly in the back-end driver that plays the role of the communication channel between the hardware device driver and VMs, as depicted in Figure 2. ANCS processes the network operation requests of VMs in a round-robin manner, checking whether a *vif* has sufficient resource allocation to possess the network resources. For this purpose, ANCS uses the credit concept [18, 19] to represent the amount of resource allocation. While network resources are being used, every *vif* consumes its credit according to the requested size; the credit value of a *vif* is recharged regularly in proportion to its weight. If the credit value that a *vif* has is less than the requested size, ANCS does not process its requests. The *vif* must then wait for the next credit value to be allocated. Therefore, the amount of credit a *vif* has determines the network bandwidth of the corresponding VM. The amount of credits allocated is calculated by the configured performance policy of each VM.

Figure 3 depicts an overall flow chart of the ANCS algorithm. In this section, we elaborate on the coarse-grained algorithm of ANCS using the flow chart. We introduce variables used in the ANCS algorithm and provide a general description of them. Then, we explain the fine-grained subalgorithms of ANCS. The detailed operations of ANCS will be described in Sections 3.3 and 3.4.

The accounting function of ANCS runs every 30 ms and allocates network resources to *vif*s on a physical machine in order. The value of 30 ms is selected to decrease the

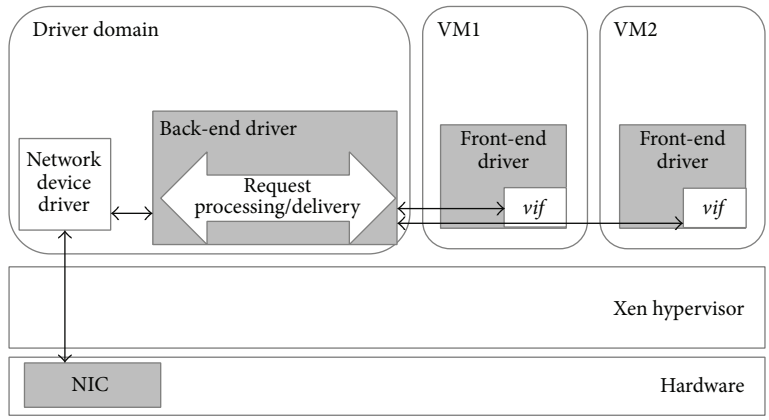

Figure 2: Virtualized system architecture with ANCS.

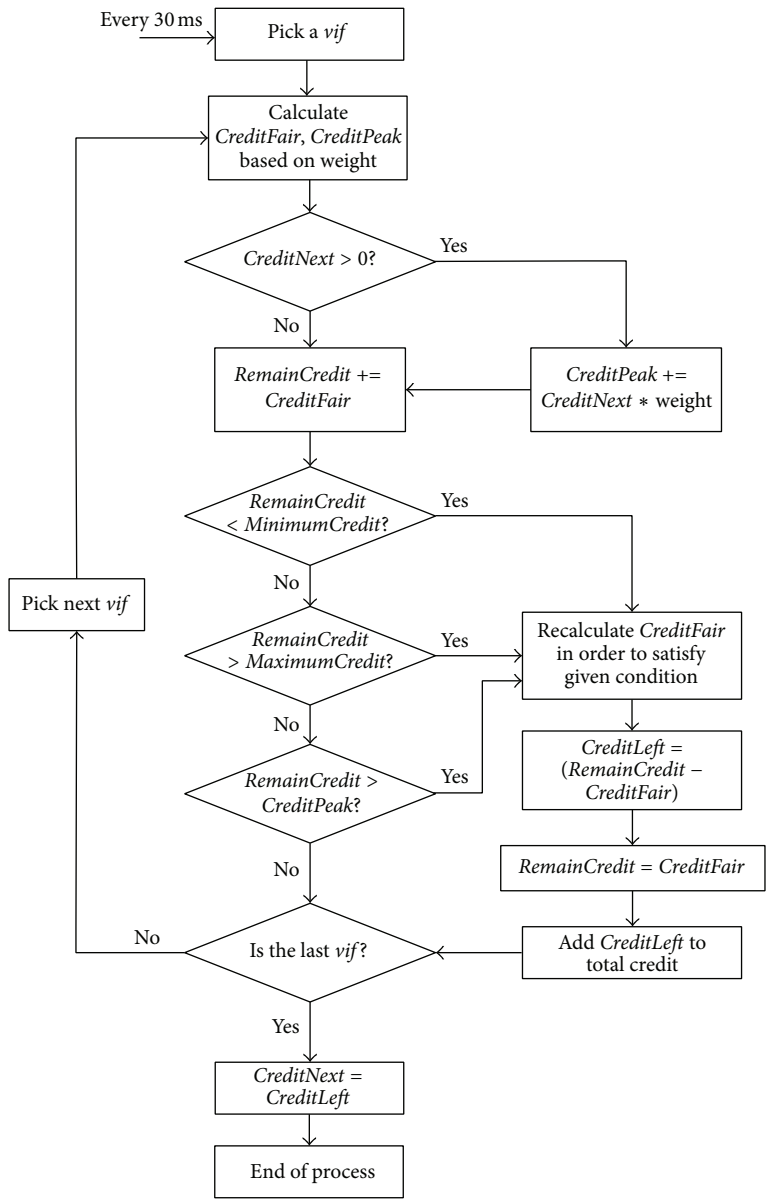

Figure 3: ANCS scheduling algorithm.

overhead of periodically running the accounting function. Please note that regardless of this period each virtual interface can execute if it has sufficient credit values to process its network packets. Therefore, the 30 ms period does not affect the network latency of each virtual interface. The following procedure is performed per *vif*.

First, the two variables required to obtain the fair share of each *vif* are calculated: *CreditFair* and *CreditPeak*. *CreditFair* is determined depending on the weight of a *vif*. *CreditFair* represents the fair share of network resources for a *vif* according to its weight. Then, *CreditPeak* is calculated. *CreditPeak* indicates the amount of available resources for a *vif* based on the assumption that the *vif* dominates the network resource when other *vif*s are idle. *CreditPeak* is used to maintain the total amount of credits in the system in order to distribute the proper amount of credits to each *vif*.

Second, the procedure to support work-conserving and various performance policies is performed. ANCS checks whether *CreditNext* is greater than zero. *CreditNext* is accumulated in the previous scheduling period when some *vif*s did not fully use their credit values. A positive *CreditNext* value indicates that there were unused credits in the previous scheduling period. This value is added to *CreditPeak* in order to distribute unused credits in the current scheduling period. We will explain this in more detail in Section 3.4. Then, for proportional sharing, ANCS adds *CreditFair* to *RemainCredit*, which is the current credit value of the *vif*. In order to support minimum bandwidth reservation and maximum bandwidth limitation, ANCS determines whether the credits allocated to the *vif* satisfy the configured performance policy, which will be explained in depth in Section 3.3.

Finally, if the current *vif* is the last *vif* in the system, the ANCS process for a single scheduling period is finished. If there are additional *vif*s to which credits can be allocated, ANCS selects the next *vif* and iterates the credit calculation. If a VM does not consume its allocated credits in the current period, ANCS adds the remaining credit value to *CreditNext* to give other VMs a chance to consume unused credits.

3.3. Multiple Performance Policies. For the diverse performance requirements of cloud computing users, ANCS provides multiple performance policies, including weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. Weight-based proportional sharing is a base policy that proportionally differentiates the amount of allocated resources based on the weight of each VM. Minimum bandwidth reservation guarantees that the quantitative network performance of a VM is always greater than the configured value, *MinimumCredit*, as shown in Figure 3. When the amount of credits allocated to a *vif* becomes less than *MinimumCredit*, ANCS adjusts *CreditFair* to satisfy the minimum bandwidth; the *CreditFair* value of the *vif* is set to *MinimumCredit* in order to support minimum bandwidth. When supporting minimum bandwidth, the aggregated amounts of minimum bandwidth requests from all *vif*s should not exceed the total throughput of the physical device. By using the minimum bandwidth reservation, ANCS can prevent performance degradation of a specific VM and guarantee sustainable minimum bandwidth. On the other hand, maximum bandwidth limitation prevents aggressive resource consumption by a specific VM. If the resources allocated to a *vif* exceed the *MaximumCredit* value, ANCS reclaims surplus resources from the *vif* for distribution to other VMs.

The administrator can apply an appropriate performance policy to each VM according to the performance demands of each user. The designated policy can be dynamically changed during runtime. In addition, multiple performance policies can be applied in a conjunctive form. For instance, when a user requires a specific boundary of network performance (e.g., larger than 200 Mbps and less than 500 Mbps), both the reservation and limitation requirements can be applied by specifying the minimum and maximum bandwidth values. Moreover, ANCS can satisfy the different performance demands of multiple applications in a single VM, because the designated performance policy is based on a *vif*, not a VM. Therefore, a VM with several *vif*s can establish various network performance requirements by assigning a different performance policy to each *vif*.

3.4. Support for Work-Conserving. ANCS supports work-conserving, in which a *vif* is idled when it does not have network requests or responses. Work-conserving is a crucial factor in cloud computing; it allows resources to be allocated efficiently and system utilization to be maximized. For work-conserving, ANCS modifies the amount of allocated credits according to fluctuations in network usage. When a VM does not fully use its credits, ANCS gives other VMs a chance to receive unused credits by distributing additional credits to them.

When a *vif* is idled for a certain amount of time, the *RemainCredit* value of the *vif* can exceed *CreditPeak*, which denotes the maximum credit value a *vif* can have. Then, the credits of this *vif* must be distributed to other *vif*s. For this purpose, *CreditLeft*, which is obtained by subtracting *CreditFair* from *RemainCredit*, is added to the total credit value. This increases the credits that can be allocated to the next *vif*s. If a *vif* exceeds its *CreditPeak*, ANCS resets the credit value of the *vif* to *CreditFair*, to give it a credit value according to its weight. By distributing idle resources to active VMs, ANCS can efficiently utilize network resources of the entire system.

4. Evaluation

For evaluation, we implement ANCS in the driver domain, domain 0, of the Xen hypervisor. The Linux version of domain 0 is 3.10.55, and the version of Xen is 4.2.1. The experiments are conducted on two identical physical servers, each of which has an Intel i7 Quad core 3770 CPU running at 3.5 GHz, 16 GB of RAM, and a Gigabit Ethernet card. One of them runs the Xen hypervisor with our modified driver domain, and the other one runs vanilla Linux 3.10.11 working as an evaluation machine. Because ANCS does not require any modification on guest OSs, we run unmodified vanilla Linux 3.2.1 for each VM. Iperf [20] benchmark is used to measure the network performance of each VM. In addition,

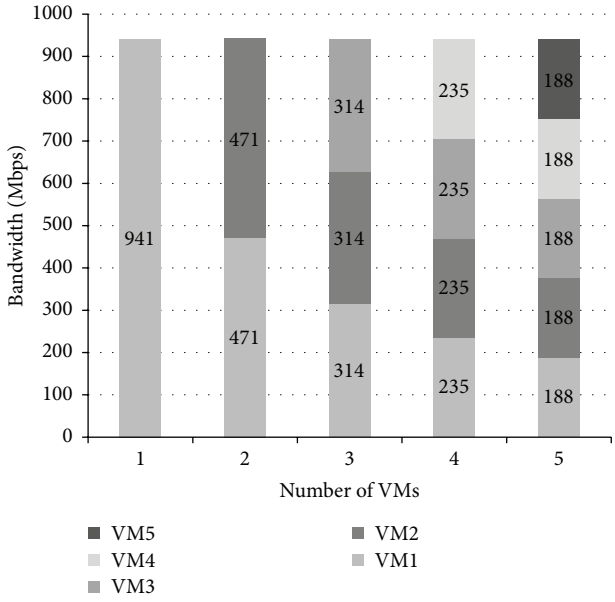

Figure 4: Bandwidth of VMs without ANCS.

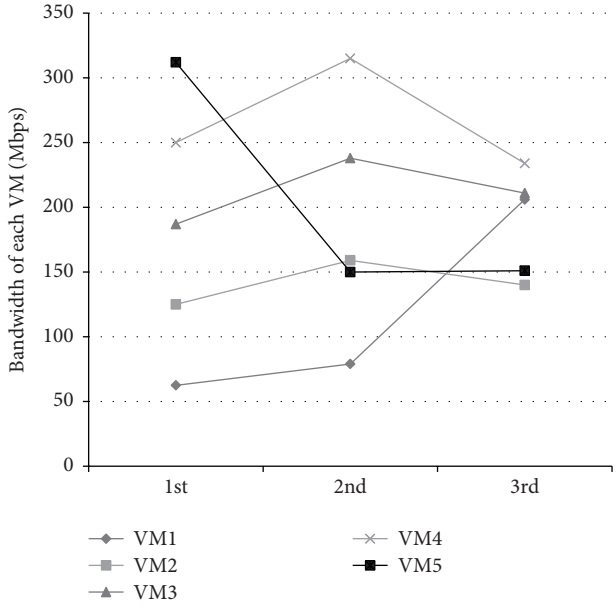

Figure 5: Bandwidth of VMs with different performance policy settings.

we use Xentop to measure the CPU utilization of the driver domain, which is described in Section 4.4.

4.1. Base Performance. Before we show the evaluation results of ANCS, we produce the base performance for comparison. For this purpose, we measure the network performance of default VMs on the Xen hypervisor without ANCS. We ran Iperf while launching up to five VMs one by one. As depicted in Figure 4, the network performance of each VM on our system is always even. When there is only one VM, the VM (VM1) consumes all network resources, achieving 941 Mbps bandwidth. As the number of VMs increases, the system's network resources are equally shared among VMs. This occurs because the unmodified driver of the Xen hypervisor processes the requests of several virtual network interfaces in a round-robin manner with the same weight. Therefore, each guest OS has the same network performance.

4.2. Multiple Policies. To demonstrate that ANCS guarantees the network performance of VMs under various scenarios, we change the performance policy during runtime and observe that the change is reflected properly in the system. When the system is started, we set the weights of five VMs for weight-based proportional sharing as {VM1 = 1/15, VM2 = 2/15, VM3 = 3/15, VM4 = 4/15, VM5 = 5/15}; thus, the ratio of the weights is 1 : 2 : 3 : 4 : 5. After measuring the performance of each VM, we additionally set the maximum bandwidth limitation policy to VM5. The bandwidth of VM5 is set to 150 Mbps while the weight-based proportional sharing is maintained. Finally, the minimum bandwidth reservation is set to VM1 with a value of 200 Mbps. Similarly, the previous configuration applied to other VMs is maintained.

The results of the experiment are shown in Figure 5. At the beginning of the experiment, ANCS allocates credits to *vif*s according to their weights. Therefore, the network performance of each VM is differentiated proportionally, as expected. When the maximum bandwidth limitation policy is applied to VM5 in the second phase, VM5 shows the bandwidth limited to 150 Mbps. As the credit value allocated to VM5 decreases compared to the first phase, the network performance of other VMs increases naturally. Moreover, the increment in the bandwidth of other VMs is proportional to their weights. During the last phase, the bandwidth of VM1 increases to 200 Mbps because of the reservation policy, while the bandwidth of VM5 is preserved as 150 Mbps, the same as it was in the second phase. Then, VM2, VM3, and VM4 receive less credit value compared to the second phase, and therefore their performance decreases.

Based on the above experiment, ANCS shows that it guarantees the network performance of VMs under various scenarios that combine the policies explained in Section 3.3. The scenarios include all three performance policies provided by ANCS. Our evaluation demonstrates that ANCS is capable of handling various performance requirements in cloud computing.

4.3. Support for Work-Conserving. In cloud computing, efficient resource management is crucial for reducing maintenance costs. For better efficiency in resource management, ANCS maximizes resource utilization even if the number of VMs and their performance policies are changed. We evaluated the efficiency of ANCS in resource management by measuring aggregated bandwidth in different environments.

First, Figure 6 provides the bandwidth results of the VMs on a physical server. We configured only one policy (weight-based proportional sharing) on all VMs. In addition, we maintained the weight of each VM for weight-based proportional sharing as {VM1 = 1/15, VM2 = 2/15, VM3 = 1/5, VM4 = 4/15, VM5 = 1/3}, throughout the remaining experiments.

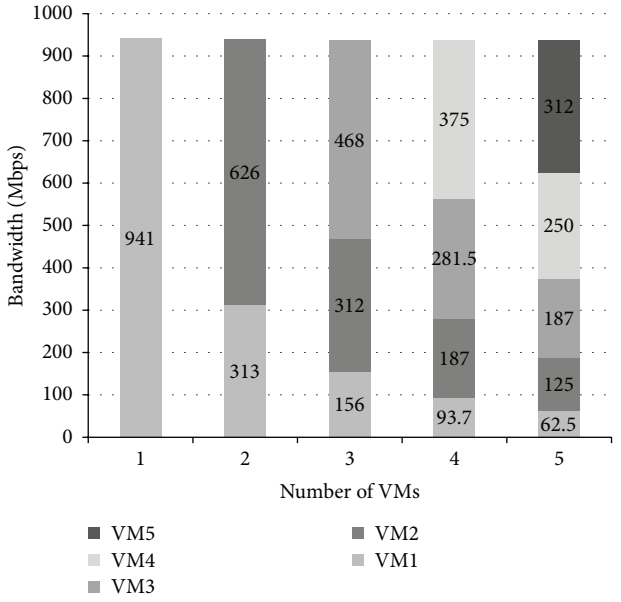

Figure 6: Aggregated bandwidth of VMs with ANCS (only weight-based proportional sharing is applied).

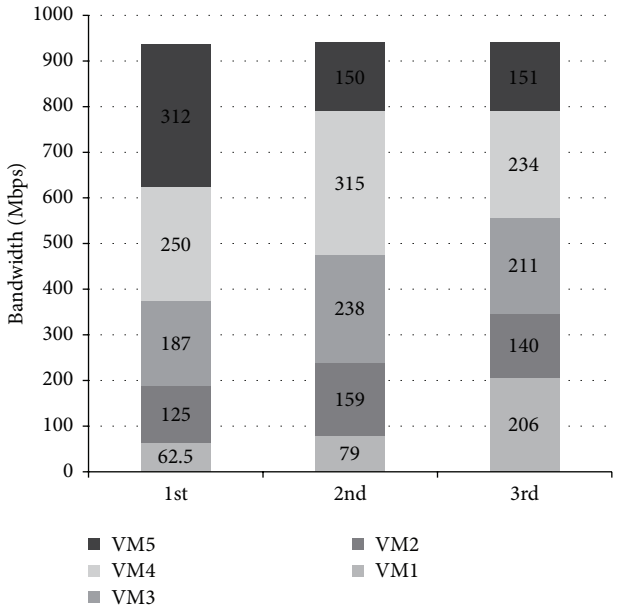

Figure 7: Aggregated bandwidth of VMs with ANCS.

We then launched the VMs sequentially (VM1 through VM5) and measured the network performance of each VM. Evaluation results show that the network performance of each VM is proportional to the weight. While the network performance of each VM changes according to the weight, the aggregated bandwidth maintains 940 Mbps. We find that ANCS fully supports work-conserving scheduling by achieving the maximum aggregated bandwidth, independent of the number of VMs on the physical server.

Next, Figure 7 shows the aggregated bandwidth in the three experiment scenarios described in Section 4.2. Through this experiment, we aimed to show that ANCS maximizes resource utilization even if the performance policy of each VM is changed. In the first phase, only weight-based proportional sharing is applied to all VMs. In the second phase, the maximum limitation policy is applied to VM5 while maintaining the weight-based proportional sharing of the other VMs. Finally, the minimum bandwidth reservation is set to VM1. In the final phase, all of the performance policies that ANCS provides are applied on the system: reservation to VM1 (200 Mbps), limitation to VM5 (150 Mbps), and weight-based proportional sharing to all VMs. As depicted in Figure 7, the aggregated bandwidth always achieves the maximum bandwidth, 940 Mbps, in all the experiment scenarios. It is clear that the work-conserving property of ANCS is not affected by the required performance policy of each VM, or by the number of VMs.

4.4. CPU Overhead. In cloud computing, it is important to decrease CPU overhead in terms of energy efficiency when applying a new policy to the system. To illustrate the CPU overhead caused by ANCS, we measured the CPU utilization in the driver domain while sequentially generating VMs one by one. Figure 8 shows the CPU utilization in

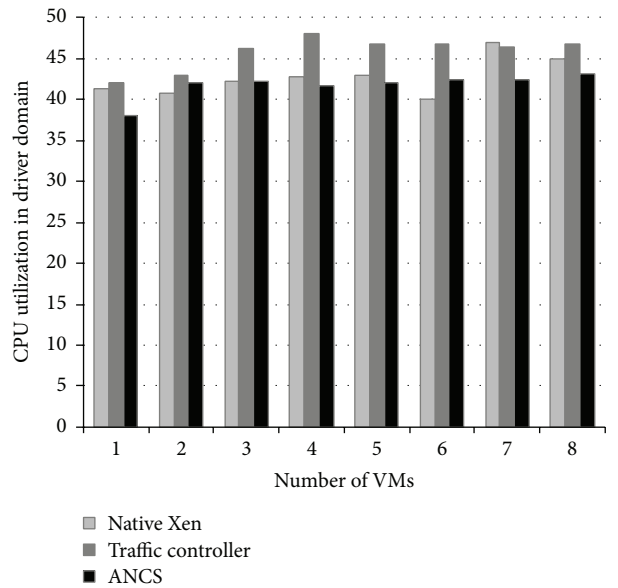

Figure 8: CPU utilization in the driver domain when the number of VMs is increased from one to eight.

three cases: Native Xen, Linux traffic controller, and ANCS. Both Native Xen and the traffic controller are based on the Xen hypervisor; they run unmodified Linux for driver domains. In the experiment with Native Xen, the network performance of each VM is equal because the default setting of Xen is fair sharing. For the traffic controller experiment, we utilize the Linux traffic controller in the driver domain to differentiate the network performance of VMs, similar to ANCS. The traffic controller does not support work-conserving and only provides network performance policies with fixed performance values; this is different from ANCS,

Table 1: Latency values in ms measured by a ping program when the number of VMs is increased from one to five.

	Number of VMs				
	1	2	3	4	5
Native Xen	0.30	0.34	0.25	0.30	0.32
Traffic controller	0.30	0.30	0.30	0.32	0.32
ANCS	0.24	0.28	0.23	0.23	0.34

Table 2: Latency values in ms measured by Netperf when the number of VMs is increased from one to five.

	Number of VMs				
	1	2	3	4	5
Native Xen	0.12	0.13	0.12	0.12	0.12
Traffic controller	0.12	0.12	0.12	0.13	0.13
ANCS	0.12	0.12	0.12	0.12	0.12

which dynamically adjusts the network performance of VMs. As depicted in Figure 8, the traffic controller demands non-negligible computation resources as it adopts packet queuing for throttling the sending packet rate. As ANCS adopts a credit-based simple accounting algorithm that incurs only minimal overhead, ANCS shows low CPU utilization in the driver domain, which is comparable to the two different techniques.

4.5. Network Latency. In this section, we evaluate network latency in ANCS and compare the results with Native Xen and the Linux traffic controller in the same experiment setup described in Section 4.4. For the performance policy, we configure the weight of each VM as one to provide the same bandwidth to VMs, which prevents different resource allocation from affecting network latency. We use both a ping program and Netperf [21] with TCP_RR to quantify the latency of UDP and TCP pings, respectively. The ping program sends 64-byte UDP packets to a client whereas Netperf with TCP_RR sends 1-byte TCP packets to a client. For these experiments, we respectively execute both programs between a client and a VM for 60 seconds and take the average value, because network latency fluctuates on each measurement. Tables 1 and 2 show the latency values (in ms) of each method measured by the ping program and Netperf, respectively, when the number of VMs is increased from one to five. In both results, ANCS shows lower or similar latency compared to Native Xen and the traffic controller. As explained in Section 3.2, although the accounting function of ANCS runs every 30 ms, virtual interfaces with sufficient credit values can run at certain points regardless of this period. Therefore, the 30 ms period does not affect the network latency of each virtual interface.

5. Related Work

As cloud computing based on virtualization becomes prevalent and the requirements of each user are diversified, techniques for improving QoS are increasingly studied. In particular, network devices are considered to offer unstable QoS, because sharing of the devices involves another scheduling layer (such as the scheduler in the back-end driver).

vSuit [11] adopts a feedback controller in order to analyze variations in network performance and to react to such deviations by adjusting network resources to each guest OS. Using this mechanism, vSuit can offer maximum bandwidth reservation and limitation for each VM. Furthermore, CPU overhead is shown to be under 1%. However, this research does not support weight-based proportional sharing, and the experiments are performed in 100 Mbps environment.

DMVL [12] provides a technique to distribute the network bandwidth to each VM in a stable and fair manner. For this purpose, it separates the logical data path and the request I/O queue belonging to each VM. In addition, it records the allocated and consumed amounts of network bandwidth for each VM by using shared memory. This information is used to adjust the amount of network bandwidth allocated in the next iteration. The limitation of this research is that this technique incurs nonnegligible CPU overhead in domain 0, up to 7%.

PSD [13] differentiates the network performance of each VM based on each virtual network interface. It schedules each virtual interface by means of a leaky bucket controller and a time slot-based resource allocator, which distribute the resource in proportion to a different ratio. However, this approach cannot provide absolute bandwidth assignment.

The Linux traffic controller [14, 17] is a traditional approach to adjusting the network bandwidth of each network application. It classifies each packet according to its header information and configures different bandwidths according to each classification. In the paravirtualized Xen, the traffic controller in domain 0 can be used to perform network performance differentiation by classifying the packets based on the IP address of each VM. In this case, severe CPU utilization occurs (up to 12%) when the number of VMs increases, as shown in Figure 8.

Recently, Silo [22] proposes to consider both VM placement and end host pacing to guarantee message latency in cloud computing. In Silo, message latency between two VMs is determined by the capacity of network switches connecting two virtual machines. End host pacing controls the transmission rate in a server by utilizing hierarchical token bucket scheduling. Even though Silo guarantees quantitative network latency in cloud computing, it only covers communication between two specific VMs. When a VM starts new connection with another VM or migrates to a different server, Silo needs to recalculate guaranteed latency.

Cerebro [23] predicts bounds on the response time performance of web APIs exported by applications hosted in a PaaS cloud. However, Cerebro only predicts response time of cloud applications and does not involve network scheduling in the case of violation of predicted response time. Moreover, Cerebro incurs computational overheads since it adopts a static analysis to predict the response time.

As shown in Table 3, the solutions proposed to enhance network QoS either provide only a single policy or incur high CPU overhead through the use of high-frequency feedback controllers. ANCS can support various policies, including proportional sharing and minimum/maximum network bandwidth reservation. It also maximizes the total

Table 3: Comparison of virtual network scheduling methods.

	ANCS	vSuit	DMVL	PSD	Traffic controller	Silo	Cerebro
Work-conserving	o	o	o	x	o	x	x
Proportional sharing	o	x	o	o	o	x	x
Minimum and maximum reservation	o	o	x	x	x	o	x
CPU overhead	Low	Low	High	N/A	High	Low	High

network bandwidth by supporting work-conserving while reducing CPU overhead as much as possible.

6. Conclusion

In this paper we propose ANCS, which dynamically enhances network performance to meet the various requirements of users in cloud computing environments. Compared to solutions proposed in previous research efforts, ANCS can provide several network performance control policies, including weight-based proportional sharing, minimum bandwidth guarantees, and maximum bandwidth guarantees, while achieving low CPU overhead.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea Government (MSIP) (no. R0126-16-1066, (SW StarLab) Next Generation Cloud Infra-Software toward the Guarantee of Performance and Security SLA) and Grant no. B0126-16-1046 (Research of Network Virtualization Platform and Service for SDN 2.0 Realization). This work was also supported by Korea University Grant.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, Elsevier, New York, NY, USA, 2005.

[3] I. Habib, "Virtualization with KVM," *Linux Journal*, vol. 2008, no. 166, article 8, 2008.

[4] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[5] C. Chaubal, "The architecture of VMware ESXi," VMware White Paper, 2008.

[6] X. Pu, L. Liu, Y. Mei et al., "Who is your neighbor: net I/O performance interference in virtualized clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 314–329, 2013.

[7] H. Park, S. Yoo, C.-H. Hong, and C. Yoo, "Storage SLA guarantee with novel SSD I/O scheduler in virtualized data centers," *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[8] J. Hwang, C. Hong, and H. Suh, "Dynamic inbound rate adjustment scheme for virtualized cloud data centers," *IEICE Transactions on Information and Systems*, vol. E99.D, no. 3, pp. 760–762, 2016.

[9] N. Jain and J. Lakshmi, "PriDyn: enabling differentiated I/O services in cloud using dynamic priorities," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 212–224, 2015.

[10] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: plan when you can," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 407–420, ACM, 2015.

[11] F. Dan, W. Xiaojing, Z. Wei, T. Wei, and L. Jingning, "VSuit: QoS-oriented scheduler in network virtualization," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '12)*, pp. 423–428, Fukuoka, Japan, March 2012.

[12] H. Tan, L. Huang, Z. He, Y. Lu, and X. He, "DMVL: an I/O bandwidth dynamic allocation method for virtual networks," *Journal of Network and Computer Applications*, vol. 39, no. 1, pp. 104–116, 2014.

[13] S. Lee, H. Kim, J. Ahn, K. Sung, and J. Park, "Provisioning service differentiation for virtualized network devices," in *Proceedings of the the International Conference on Networking and Services (ICNS '11)*, pp. 152–156, 2011.

[14] W. Xiaojing, Y. Wei, W. Haowei, D. Linjie, and Z. Chi, "Evaluation of traffic control in virtual environment," in *Proceedings of the 11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES '12)*, pp. 332–335, Guilin, China, October 2012.

[15] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[16] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on Demand IT Infrastructure (OASIS '04)*, pp. 3–7, October 2004.

[17] W. Almesberger, "Linux network traffic control-implementation overview," in *Proceedings of the 5th Annual Linux Expo LCA-CONF- 1999-012*, pp. 153–164, Raleigh, NC, USA, 1999.

[18] K. Mathew, P. Kulkarni, and V. Apte, "Network bandwidth configuration tool for Xen virtual machines," in *Proceedings of the 2nd International Conference on COMmunication Systems and NETworks (COMSNETS '10)*, pp. 1–2, IEEE, Bangalore, India, January 2010.

[19] C.-H. Hong, Y.-P. Kim, H. Park, and C. Yoo, "Synchronization support for parallel applications in virtualized clouds," *The Journal of Supercomputing*, 2015.

[20] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," http://software.es.net/iperf/.

[21] R. Jones, *NetPerf: A Network Performance Benchmark*, Information Networks Division, Hewlett-Packard Company, 1996.

[22] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: predictable message latency in the cloud," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 435–448, ACM, 2015.

[23] H. Jayathilaka, C. Krintz, and R. Wolski, "Response time service level agreements for cloud-hosted web applications," in *Proceedings of the 6th ACM Symposium on Cloud Computing*, pp. 315–328, ACM, Kohala Coast, Hawaii, August 2015.