IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 May 2022, accepted 21 June 2022, date of publication 1 July 2022, date of current version 18 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3187731

RESEARCH ARTICLE

Fine-Grained I/O Traffic Control Middleware for I/O Fairness in Virtualized System

JAEHAK LEE[1], (Member, IEEE), HWAMIN LEE[2], AND HEONCHANG YU[1], (Member, IEEE)
[1]Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea
[2]Department of Medical Informatics, College of Medicine, Korea University, Seoul 02841, South Korea

Corresponding author: Heonchang Yu (yuhc@korea.ac.kr)

This work was supported in part by the Research and Development Program for Forest Science Technology provided by Korea Forest Service (Korea Forestry Promotion Institute) under Project 2022427C10-2224-0801, in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant 2021R1A2C1009290, and in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea Government (MSIT) (Development of an Edge Cloud-Based Vehicle Sharing Platform That Supports User-Specific Automotive Healthcare Services) under Grant 2022-0-00983.

ABSTRACT The development of IT technology in the 21st century has created a new paradigm for real-time, data-intensive user services, such as connected cars, smart factories, and remote health care services. The considerable computational resources required by these services are rendering the cloud increasingly more important. In the cloud server, user services are forced to share physical resources because of the emerging resource competition, thus introducing various types of unpredictable workloads. The core technology of the cloud is a virtualized system, which isolates and shares the powerful physical resources of the server in the form of a virtual machine (VM) to increase resource efficiency. However, the scheduling policy of a virtual CPU (vCPU), which is a logical CPU of a VM, generally schedules the vCPU based on the degree of occupation of the physical CPU (pCPU) without regarding I/O strength; so it brings the unfair I/O performance among VMs in the virtualized systems. The user services performing on the VM are not aware of the user-contention architectures, which sharing of I/O devices, in the virtualized systems; Furthermore, the current virtualized system simply adopts the Linux-based I/O processing process which optimized for user-contention-free architectures. Therefore, the architecture that brings the unfair usage of I/O devices among user services is hardly regarded and has low awareness in current virtualized systems. To overcome this problem, in this study, I-Balancer is presented to provide fair I/O performance among I/O-intensive user services by applying an asynchronous inter-communication control technique for the virtualized system with a high VM density. The main design goal of I-Balancer is to increase the awareness of user-contention architectures in the hypervisor. I-Balancer derives the fine-grained workload and I/O strength for each vCPU during the scheduler and event channel areas. Subsequently, to strengthen fair I/O performance, an I/O traffic control mechanism is implemented to control the inter-domain communication traffic according to the I/O strength of the VMs. Experiments were performed on the fairness of I/O(disk and network) performance on virtualized systems with Xen 4.12 hypervisor adopted based on various performance metrics. The experimental results showed that the virtualization system to which I-Balancer is applied reduces the network and disk I/O performance standard deviation among VMs by up to 71% and 61% respectively compared to the existing virtualization system; and, performance interference and overhead are also confirmed to be negligible.

INDEX TERMS Cloud computing, virtualization, system communication, hypervisor, fairness, middleware, network I/O, disk I/O, traffic control, resource management, Xen.

The associate editor coordinating the review of this manuscript and approving it for publication was Hang Shen.

I. INTRODUCTION

With advances in virtualization technology, many organizations and businesses are driving digital transformation

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

to the cloud [1]. In addition, the network latency between users and servers is decreasing as a result of technological developments in 5G-based network communication, hardware supporting system virtualization, and core cloud server configurations. With the development of IT technology, new paradigms of user service types, such as connected cars, smart factories, and remote healthcare, are generating large amounts of data requiring real-time processing. Research combining these technologies with cloud technology is also being actively performed.

Edge clouds (also known as fog cloud) are emerging rapidly to alleviate problems with existing central clouds [2]–[4]. The central cloud, which processes the user workload by centralizing virtualized servers using powerful resources has limitations, such as network bottlenecks created by increased user services and network latency due to geographic locations. Figure 1 shows the edge cloud architecture. Edge clouds reduce network latency due to geographic location by placing virtualized servers close to users thereby avoiding network bottlenecks and by distributing the user workload to virtualized servers.

In the central cloud area, as shown in Figure 1, user services that require relatively high resources are performed, and in the edge cloud, real-time processing or user services that require relatively fewer resources are performed [5]–[7]. User service types that can maximize the advantages of the edge cloud include connected car platforms, smart factories, and remote healthcare, which need to process massive real-time data. However, the edge cloud has to process a large amount of data generated from users through limited resources compared to that available with the central cloud, and the resulting latency for user services that require real-time processing is fatal. In addition, as the number of user services increases, various types of workloads appear in cloud servers [6]–[8]. Therefore, the complexity of resource management for cloud servers increases and the range of Quality of Service (QoS) satisfaction for user services is also broadened [1], [3]–[5].

The hypervisor (also known as Virtual Machine Monitor, VMM), is a software stack that enables the system virtualization of the servers that make up the cloud infrastructure and is closely related to the QoS guarantee of diverse user services. The vCPU scheduler, which consists of a hypervisor, plays a key role in system resource management. It schedules the virtual CPU (vCPU), which is a logical CPU that consists of a set of contexts generated by the virtual machine (VM) to provide fair physical CPU (pCPU) occupancy between VMs. This means that the performance of the VM is determined by the extent of VM vCPU takeover of the pCPU. The hypervisor's vCPU scheduling policy, which schedules each vCPU based on the degree of pCPU occupancy, provides highly fair CPU performance to CPU-intensive VMs [9]–[11]. However, a vCPU scheduler that does not consider the I/O strength of data-intensive VMs results in the unfair I/O performance of the virtualized system [9]–[13]. In addition, the current virtualized system simply adopts Linux-based I/O procedures,

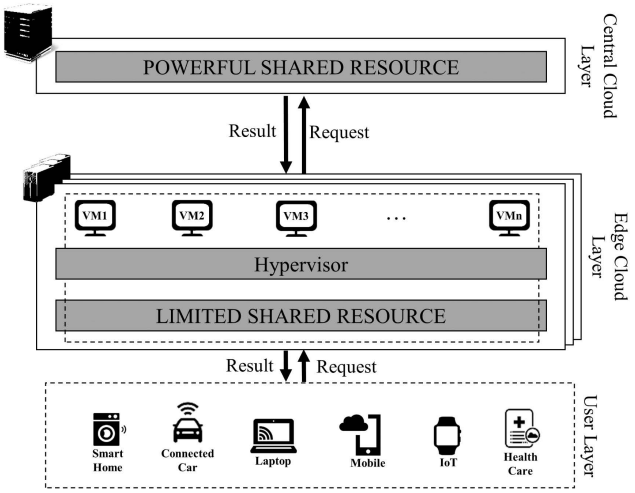

FIGURE 1. Edge cloud architecture.

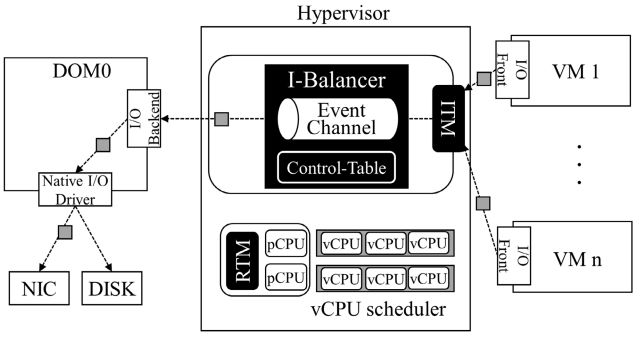

FIGURE 2. The overall structure of I-Balancer.

which leads to a lack of awareness of user-contention architectures, resulting in internal unfair I/O resource usage in virtualized systems [14], [15]. Unfair I/O performance among I/O-intensive VMs can lead to QoS degradation for various user services and result in unexpected SLA violations because the network or disk performance cannot be guaranteed fairly.

In this study, I-Balancer is introduced to mitigate the unfair I/O performance caused by scheduling dependency problems and low awareness of the intensity of the I/O degree among vCPUs of VMs for the same resource configuration in a virtualized system with high VM density. I-Balancer makes the hypervisor aware of the fine-grained I/O strength of VMs and allows the fair sharing of I/O devices among VMs based on the derived I/O strength through an I/O traffic control mechanism on inter-domain communication in the virtualized system.

Figure 2 shows the overall structure of the I-Balancer. I-Balancer places an I/O tracking module (ITM) and a runtime tracking module (RTM) on the hypervisor to derive a fine-grained disk and network I/O strength for vCPUs. And then, if the I-Balancer determines that the degree of I/O between vCPUs is unfairly occurring, it takes the I/O

traffic control mechanism in the event channel based on the derived I/O strength of vCPUs. By doing so, it provides a high chance of I/O device access to VM with a low I/O strength by restricting I/O device access to VM with high I/O strength. This provides an opportunity for vCPUs with lower I/O intensity to occupy more I/O devices and can alleviate unbalanced network and disk I/O performance in virtualized systems.

Through the development of I-Balancer, the contributions of this study to solving the I/O performance unfairness problem in virtualized systems are as follows:

1) Unfair I/O performance between VMs was confirmed through an experiment, and the causes were analyzed

2) I-Balancer can derive fine-grained I/O strength of VMs without relying on external or in-VM performance metric monitoring programs

3) I-Balancer does not need to modify the OS kernel of the VM, does not require in-VM component, and is designed to accommodate various vCPU schedulers, so it has a high portability

4) I-Balancer provides fair I/O device access among I/O-intensive VMs with dynamic I/O traffic controlling and does not affect the performance of neighbor-VMs and incurs negligible overhead to the virtualized system

5) Through various I/O performance-related experiments, I-Blanacer showed that the degree of I/O fairness among VMs for network and disk performance was superior to that of the existing virtualized system, and proved that the performance interference to neighbor-VMs and the system overhead were negligible.

The remainder of this paper is organized as follows: In Section II, the concept of a virtualized system is explained to provide the background for the I-Balancer. In Section III, the causes of unfair I/O performance in the virtualized system are analyzed with simple experiments. In Section IV, the overall design and implementation of the I-Balancer are described. In Section V, the I-Balancer's performance is confirmed by performing a comparison experiment between the I-Balancer and the existing virtualized system. In Section VI, related work is discussed, and Section VII discourses the I-Balancer's limitation and future work. And last, VII concludes the paper.

II. BACKGROUND

Xen [16] is a para-virtualization hypervisor based on the split-driver model consisting of a DOM0 (also known as a driver domain) with physical drivers to handle the I/O requests of all guest VMs. DOM0's involvement in the guest VM's I/O procedure simplifies the guest VM kernel code and further enhances the data safety of the virtualized systems. Currently, Xen occupies a high proportion of the data center market along with other hypervisors such as KVM [17], VM-ware [18], and Hyper-V [19]. In this section, the I/O procedure of the virtualized system is briefly discussed with respect to I/O unfairness along with the proposed method.

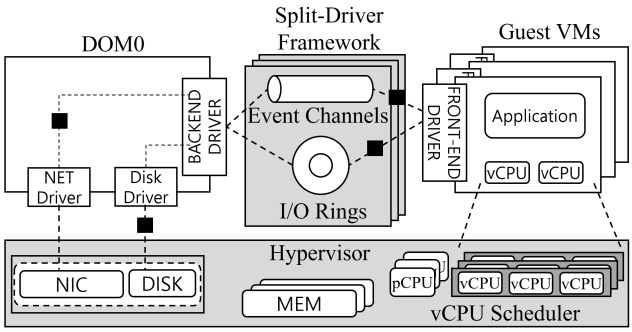

FIGURE 3. Xen architecture.

A. I/O PROCESSING PROCEDURE OF XEN

Virtualized systems based on Xen or kernel-based virtual machines (KVM) handle asynchronous networks and disk I/O of VMs that run user services by adopting the split-driver architecture. Figure 3 represents the virtualized system architecture of Xen. DOM0 has a physical network/disk driver that handles all I/O requests from guest VMs that have front-end drivers, which are logical networks and disk drivers. Therefore, practically all I/O of guest VMs is performed through DOM0, and the guest VM simply performs logical I/O.

Xen with a split-driver architecture applies a shared ring buffer and event channel to handle asynchronous I/O requests from guest VMs [16]. The shared I/O ring buffer uses the zero-copy mechanism [23], [24] to transfer I/O data through shared memory communication based on the physical address page frame mapping between DOM0 and the guest VM. The transfer is managed by a grant table such that the VM can only access the shared ring buffer memory space allocated to it. The event channel implements a notification mechanism for asynchronous I/O requests and VM responses in Xen.

The overall I/O handling process [10], [11], [20]–[22] in Xen is as follows: When an I/O request is raised on the guest VM, the I/O request event is sent to the back-end driver of DOM0 by the front-end driver of the guest VM. Then, when the vCPU of DOM0 is occupying on the pCPU, DOM0 checks the event channel bound to the guest VM to see if there is a pending I/O request event from the guest VM. If a pending event exists, DOM0 processes the corresponding data of the I/O request using an interrupt handler. The physical I/O device (e.g., NIC, SDD, or HDD) processes the routed I/O request of the guest VM by DOM0 and returns the I/O response for that I/O request to the physical I/O driver of DOM0.

The back-end driver of DOM0 then sends the data of the I/O response event to the front-end driver of the guest VM, which raises the I/O response event through the event channel. Subsequently, when the vCPU of the guest VM occupies the pCPU, the front-end driver checks whether a pending I/O response event exists in the event channel bound to DOM0. If it exists, the guest VM performs an interrupt

service routine (ISR) based on the corresponding data of I/O response.

B. ASYNCHRONOUS I/O NOTIFICATION EVENTS

In the virtualized system with the para-virtualization hypervisor, the event channel, which is an asynchronous notification mechanism for the I/O of VM, plays the role of an interrupt in the native system [25], [26]. The main types of events in the event channel in Xen consist of the inter-domain and Inter-Processor Interrupt(IPI) events. Inter-domain event is used for asynchronous bidirectional I/O data exchange on shared ring buffers between VMs and is generated by virtualized devices. The IPI event is used for communication among the vCPUs of the VM. In addition, there are interrupt request (IRQ) events that DOM0 uses to communicate with hardware devices by mapping real IRQs to event channels; VIRQ events are used to communicate with virtual devices.

C. XEN's SCHEDULING INTERFACE

Xen provides an abstract scheduling interface (where in xen/common/schedule.c) can accommodate multiple vCPU schedulers [16], [27]–[29]. This abstract scheduling interface can adopt all vCPU schedulers (Credit1, Credit2, RTDS in Xen 4.12), and each vCPU scheduler must include a pre-common pointer-based scheduling function (e.g., *do_schedule*(), *sleep*(), *wake*(), *yield*(), etc.). If a new vCPU scheduler is designed, information related to the scheduler must be added to the schedule.c file so that Xen can recognize the vCPU scheduler. This abstract scheduling interface of Xen provides scalability and flexibility for vCPU scheduling policy selection of the virtualized system.

The Credit1 scheduler [30] adopts a priority-based round-robin scheduling policy to schedule vCPUs according to BOOST, UNDER, and OVER priorities. Credit, which refers to the pCPU occupancy time of vCPU, is consumed every 10 ms. If the credit is less than or equal to 0, vCPU has OVER priority, and if it is greater than 0, it has UNDER priority. The BOOST priority is the highest priority for the VM performing a fast I/O response. When the VM receives an I/O event from the event channel, the vCPU in an idle state of VM acquires BOOST priority. If the vCPU occupying the pCPU has OVER or UNDER priority, it preempts the pCPU for 10 ms. The BOOST mechanism improves I/O responsiveness to enhance the I/O performance of data-intensive VMs; however, the same BOOST priority between vCPUs causes an I/O performance imbalance [10], [31]. In addition, the round-robin scheduling policy defines the limits on the I/O performance improvement of the Credit1 scheduler [13], [32].

After extensive study, the Credit2 scheduler [33] was adopted as the default scheduler for Xen in 2019 to avoid the problems of the Credit1 scheduler and improve the I/O performance of the VM. The Credit2 scheduler was designed with the goal of improving the performance of VMs with mixed workloads in terms of lowering I/O latency. The Credit2 scheduler removes the three priorities of the Credit1 scheduler and schedules first the vCPU with the most remaining credit. If the credit amount of the next vCPU to be scheduled is less than or equal to 0, the Credit2 scheduler invokes *reset_credit*() to reallocate the same amount of credit to all vCPUs in the run-queue. With the Credit2 scheduler, the I/O-intensive vCPU has frequent idle states and consumes relatively fewer credits than the CPU-intensive vCPU; therefore, pCPU occupancy is guaranteed over CPU-intensive vCPU because of the high scheduling priority, thereby reducing the I/O latency in the scheduling delay of the run-queue.

III. MOTIVATION

As the need for cloud servers increases, the complexity of system resource management also increases because of the increased number of user services that generate data-intensive workloads, such as video streaming, connected cars, IoT, and healthcare. However, if fair I/O performance between VMs is not guaranteed because of competition for I/O resources in the central or edge cloud, which consists of a set of servers virtualized system adopted, or dependency on the hypervisor's resource management policy, then user services not only cannot guarantee Quality of Service (QoS), but can also fail, which would be a very critical problem. In this section, the unfair I/O performance between VMs is confirmed in a virtualized system through a simple experiment, and the causes are then analyzed.

A. SIMPLE EXPERIMENT FOR I/O PERFORMANCE UNFAIRNESS IN INTER-COMMUNICATION

The unfairness of the I/O performance in the virtualized system, was confirmed in an experimental environment (see Table 3 of section V) with one target-system (acting as a server) and host-system (six Client-VMs are running and acting as a client). For the resource configuration of VMs, six vCPUs and 8 GB of memory were allocated for DOM0. For Client-VM, four vCPUs and 2 GB of memory were allocated. To avoid performance interference from the sharing of pCPUs, all vCPUs of DOM0 were pinned to pCPU0-1, Client-VMs to pCPU2-7. Thus, a total of 24 vCPUs on Client-VMs shared six pCPUs. To generate a network I/O-intensive workload, the iperf tool was used (see Table 3 of Section V), and the Client-VM sent TCP/IP streaming data of size 1024 KB to all target-system for 60 s. At the same time, to configure a mixed workload, the stress tool (see Table 3 of Section V) was used to generate a CPU workload with four threads in all Client-VMs, thereby causing performance interference between I/O-intensive and CPU-intensive processes.

Figure 4(a) represents the virtualized system in which the Credit1 scheduler is adopted and Figure 4(b) represents the virtualized system in which the Credit2 scheduler is adopted. The x-axis represents each round for a total of 15 experiments, and the y-axis represents the network bandwidth of the Client-VM. For Figure 4(a) and (b), it can be seen that each Client-VM displays unfair network performance and that Credit2 scheduler provides a relatively fairer network

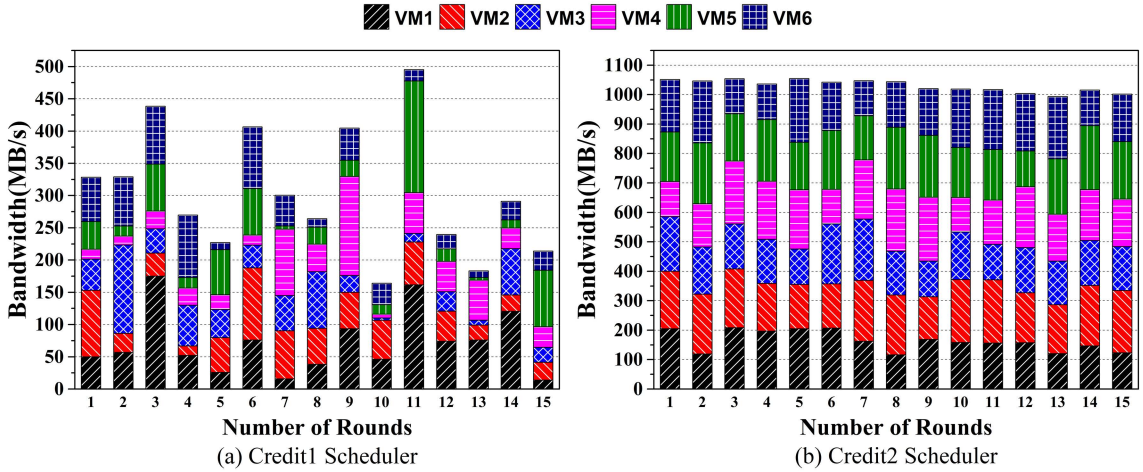

FIGURE 4. Results of simple experiments on network I/O performance for among VMs.

TABLE 1. Average-based numerical analysis for a number of 15 rounds in Figure 4.

vCPU Scheduler	Bandwidth(MB/s)	Max-Avg(MB/s)	Min-Avg(MB/s)	Min-Max Gap(MB/s)
Credit1 scheduler	50.61	108.66	13.89	94.77
Credit2 scheduler	171.60	211.55	119.74	91.81

performance for Client-VM than Credit1 scheduler, but still has an unfair performance.

The bandwidth in Table 1 refers to Figure 4 and is the average of the bandwidths determined by each vCPU scheduler. Max-Avg is the highest average bandwidth among Client-VMs for the 15 rounds, and Min-Avg is the lowest average bandwidth among Client-VMs for 15 rounds. Finally, the min-max gap is the average difference between Max-Avg and Min-Avg. From the experimental results, it can be seen that Max-Avg and Min-Avg are significantly different for the two schedulers. In particular, for Min-Max gap, the performance of the Credit1 scheduler and the Credit2 scheduler were 94.77 MB/s and 91.81 MB/s, respectively, which shows that the network performance is not fair among the Client-VMs. The reasons for unfair I/O performance among VMs in virtualized systems is explained in the following sub-section.

B. SCHEDULING DEPENDENCY PROBLEM

In the virtualized system, vCPUs are scheduled based on pCPU occupancy to provide fair runtime for VMs. The vCPU schedulers directly track the accumulated runtime of vCPUs to recognize and return overrunning vCPUs to schedule the next vCPU, providing relatively fair CPU performance among CPU-intensive VMs. However, because the I/O-intensive vCPUs also need to occupy the pCPU to handle I/O-intensive workloads and the number of pCPUs is limited, there is a restriction to the ability to ensure fair I/O performance among the I/O-intensive VMs. For example, if the scheduling policy of the vCPU scheduler makes it more likely that the vCPU of a particular VM will occupy the pCPU, then the vCPU of another VM is less likely to occupy the pCPU because pCPU is a limited resource. If this continues, I/O performance becomes unfair in virtualized systems that can handle I/O workloads, as there is less opportunity for I/O intensive VMs to occupy the pCPU.

1) LACK OF I/O WORKLOAD INTENSITY AWARENESS

The Credit1 and Credit2 schedulers increase the I/O responsiveness of the VM by providing an opportunity to preempt the pCPU when I/O occurs on the VM. The Credit1 scheduler increases the I/O performance of the VM by prioritizing I/O-intensive VMs to BOOST priority, thereby ensuring pCPU preemption to handle immediate I/O workloads. KVM's completely Fair Scheduler (CFS) and Xen's Credit2 scheduler first schedule vCPUs that consume less time slices (more remained time slices in CFS, more remaining credits in credit2 scheduler). At this time, the I/O-intensive vCPU consumes relatively fewer time slices (credits) because it has more frequent sleep states than the CPU-intensive vCPU, such that the I/O performance of the VM can be increased by guaranteeing the pCPU occupancy. The BOOST mechanism improves the I/O performance of VMs on the Credit1 scheduler, but when I/O resource contention occurs between VMs, multiple vCPUs in the run-queue may frequently have BOOST priority. This situation raises the multi-boost problem for the virtualized system, resulting in unfair I/O performance of VMs by causing competition for pCPU preemption as the role of the BOOST priority is lost [9]–[11], [35].

In addition, in the case of the aforementioned vCPU schedulers, a vCPU with a mixed workload, which simultaneously processes CPU- and I/O-intensive workload, consumes a relatively large number of time slices because of handling

CPU-intensive workload needs the more time slice (credit) consuming. In this case, I/O-intensive workloads of vCPUs with mixed workloads will not guarantee pCPU occupation because of the increasingly smaller difference in the remaining time slice from neighboring vCPUs of the run-queue, resulting in scheduling latency that causes I/O delays [9], [11], [13], [14].

Furthermore, as vCPUs of VMs with various workloads are deployed in the run-queue, vCPUs with I/O-intensive workloads have inconsistent scheduling latencies and degrees of pCPU occupancy. When I/O-intensive vCPUs are placed in the run-queue, the I/O performance is inconsistent because the degree of pCPU occupancy varies with the workload trends of the neighboring vCPUs. The placement state of vCPUs with different workloads affects the workload tendencies of the run-queue itself; hence, the I/O guarantee degree varies depending on which run-queue the I/O-intensive vCPUs are placed. This leads to unfair I/O performance between VMs when there is I/O resource contention [12], [14], [20], [26]. Such situations cause a scheduling dependency problem leading to unpredictable and inconsistent I/O performance of VMs depending on the workload tendency of neighboring vCPUs in the virtualized system with a high VM density.

As a result, the aforementioned vCPU schedulers focus only on increasing the I/O performance of the VM by increasing I/O responsiveness, and do not consider the fair utilization of I/O devices among I/O-intensive vCPUs. More specifically, the current hypervisor does not consider the I/O intensity for I/O-intensive vCPUs and does not perform corresponding I/O controls; therefore, I/O devices are allocated unevenly between I/O-intensive VMs in virtualized systems. In the 21st century IT infrastructure, cloud users are increasing as user services with various workloads are performed on cloud servers, and virtualized systems encounter unpredictable workload types and intensities. This implies the need to perform fine-grained resource management by strictly tracking the types of user services running on the VM.

2) LACK OF AWARENESS OF THE I/O PROCEDURE OF THE VIRTUALIZED SYSTEM

Tasks generated by the user service are perceived by the guest VM as being processed by their OS scheduler; however, in reality, they are abstracted into the vCPU logical unit of tasks and processed through the vCPU scheduler of the hypervisor. In a native system, a specific user program has a system environment that can be prioritized based on user preferences. However, in virtualized systems where multiple users' programs with no clear priorities share physical resources, total network and disk I/O bandwidth are limited; Furthermore, DOM0, which performs the actual I/O of guest VMs through a split-driver architecture, is simply applied with the I/O procedure of the existing Linux kernel-based optimized for the native system.

As a result, all I/O requests from guest VMs are handled on a first-in-first-out (FIFO) basis, similar to existing native systems [15], [34]–[39]. In a virtualized system, I/O-intensive workloads of the VM can be processed only when the vCPU occupies the pCPU, similar to CPU-intensive workloads. Therefore, a vCPU with a high degree of pCPU occupancy has a higher probability of handling I/O-intensive workloads. Naturally, vCPUs with a high degree of pCPU occupancy raise more I/O request events to the backend driver of DOM0. Subsequently, all I/O requests are processed based on FIFO in DOM0, thereby occupying I/O devices more frequently than VMs with less pCPU occupancy. If this situation persists, guest VMs accounting for a relatively large portion of the total I/O bandwidth of a virtualized system with limited resources will appear, resulting in unfair disk or network I/O performance.

In the native system, the OS scheduler first schedules the I/O intensive process of a program that the user wants to handle and then simply returns the I/O result to that program. However, in virtualized systems, the I/O-intensive workloads of guest VMs are not handled based on user priorities as in native systems, but are treated as dependent on vCPU scheduling policies based on the degree of pCPU occupancy. If I/O resource contention in the virtualized system intensifies, more I/O requests from vCPUs with high pCPU occupancy guarantees to the DOM0 will emerge. If this situation persists, FIFO-based I/O request processing in DOM0 will make it difficult to ensure fair occupation of network and disk devices among VMs.

IV. DESIGN AND IMPLEMENTATION

A. I-BALANCER: CHALLENGES AND GOALS

The native system has a user-contention-free architecture that can immediately process I/O requests from users' preferred programs without contention for I/O resources. The threads that make up the user process have a relatively high scheduling priority because of real-time interaction with the user and can handle I/O-intensive workloads immediately. This user-contention-free architecture only needs to process I/O-intensive workloads from a single user, and the I/O workload generated by a program is the workload that the user wants to process first. When I/O requests generated from user programs are handled in the kernel thread through interrupts, the set of I/O requests is loaded into the queue of I/O devices based on FIFO. Therefore, in the native system, the user can handle a set of I/O requests that reflect the user's I/O priorities among programs.

However, a virtualized system has a user-contention architecture in which multiple VMs compete to obtain I/O devices by handling I/O-intensive workloads. As explained in Section III, in a virtualized system, unfair I/O performance between VMs is caused by the scheduling dependency problem and lack of awareness of I/O strength. As the OS of the guest VM applies the I/O philosophy of the existing Linux kernel, it is assumed that the pCPU will be preempted within a short scheduling latency to handle the I/O-related interrupt execution procedure. This is because the VM's OS perceives the vCPUs, which are logical cores consisting of the VM's

task set, as physical CPUs, so it has strong confidence that the vCPUs will always remain in an active state. However, all I/O-related workload generated by the VM is also mapped to the vCPU, which is the logical core, and scheduled by the vCPU scheduler, thus competing for pCPU along with vCPUs of other VMs handling different workloads.

As a result, I/O-intensive vCPUs face scheduling dependency problems caused by the inconsistent scheduling latency of neighboring vCPUs handling various workloads. Thus, the asynchronous I/O requests generated by guest VMs are loaded unfairly into the event channel mechanism. The I/O request event from the guest VM, which is the signal to call the ISR to DOM0, is loaded into the event queue of the backend driver (network or disk) on DOM0, based on FIFO. Thus, the actual ISR is executed when the vCPU of DOM0 occupies the pCPU, according to the order of the I/O request events loaded into the event queue of the backend driver in DOM0. Due to the scheduling dependency problem, if the vCPU of a particular guest VM that generates I/O requests has higher pCPU usage than other guest VMs, the rate of I/O request events to DOM0 increases because it has a relatively high probability of processing the I/O workload. As a result, the I/O request event of a specific VM is loaded into the backend driver at a relatively high rate; therefore, DOM0 performs an ISR for a relatively large number of that VM's I/O requests. As a result, the probability of occupying physical I/O devices (SSD, HDD, or NIC) increases, and guest VMs with low pCPU occupation have a lower chance of occupying the I/O device because fewer I/O request events are loaded, resulting in low I/O performance.

The I/O procedure of the existing virtualized system, in which the hypervisor does not recognize the I/O degree of each guest VM, causes unfair I/O performance in the virtualized system with split-driver architecture design. To mitigate unfair I/O performance in the virtualized system, the hypervisor must raise awareness of user-contention architectures by controlling the I/O degree between VMs. Thus, I-Balancer was designed to mask the scheduler dependency problem and provide the hypervisor with a strong awareness of the user-contention architecture for I/O procedure. The I-Balancer adopts a fine-grained I/O strength-tracking technique and I/O traffic control mechanism based on inter-domain communication for fair I/O resource sharing among guest VMs with different types of work and intensity.

B. I-BALANCER: OVERVIEW

Credit1 and Credit2 schedulers, which are general-purpose schedulers applied to Xen-based virtualized systems, each have different vCPU scheduling policies; hence, the I/O performance guarantee mechanism is also different. The Credit1 scheduler guarantees pCPU preemption by giving BOOST priority to the network I/O-intensive vCPUs to ensure I/O performance [9], [10], [26], [30]. By contrast, the Credit2 scheduler ensures I/O performance by enabling pCPU preemption by leveraging the fact that I/O-intensive vCPUs have less credit consumption because of frequent sleep states, and the amount of remaining credit is normally more than that of CPU-intensive vCPUs [33], [40], [41]. Each of these vCPU schedulers manages the I/O performance of the VMs by recognizing the coarse-grained degree of I/O for vCPUs through the I/O guarantee policy. However, these I/O management mechanisms based on the vCPU scheduling policy focus only on improving inbound-based I/O performance and do not ensure fair I/O device usage in virtualized systems.

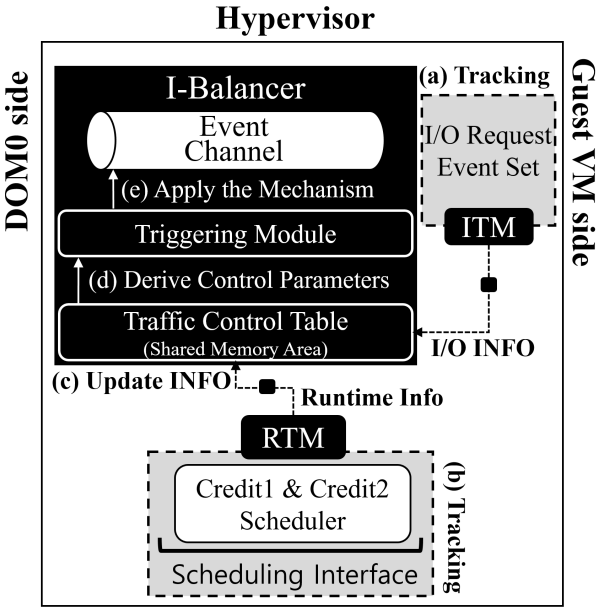

FIGURE 5. The overall procedure of I-Balancer.

I-Balancer was introduced to mitigate the unfair I/O performance of virtualized systems caused by scheduling dependency problems and low awareness of the intensity of the I/O degree among vCPUs. In Figure 5, which represents the overall procedure of I-Balancer based on Figure 2, the ITM tracks the inter-domain communication flow between vCPUs in the event channel area **(a)**, and RTM tracks the pCPU occupancy time of each vCPU in the scheduling area **(b)**. They transmit the tracked vCPU metric information based on I-Balancer's specific structure which is to take the I/O metric through shared memory communication **(c)**, allowing I-Balancer to build its own traffic control table to derive the relative I/O intensity for the vCPU. And I-Balancer derives control parameters from the traffic control table where the received vCPU metric information is stored and managed by itself and derives the relative I/O strength between vCPUs **(d)**. Finally, when the I/O strength gap between the I/O-intensive vCPUs exceeds the specific threshold related to I/O unfairness, I-Balancer enforces an I/O traffic control mechanism on the event channel to control the I/O device usage of vCPUs with high I/O strength **(e)**. This provides an opportunity for vCPUs with lower I/O intensity to occupy more I/O devices and can alleviate unbalanced network and disk I/O performance in virtualized systems.

Algorithm 1 I-Balancer: ITM Procedure

tCPU: Represents a structure that contains I/O metric information on the vCPU that generated I/O events and it is managed by ITM and RTM of the I-Balancer
NOW: Current system time based on nanoseconds returned from function *NOW*()

```
Some I/O event is exposed on Event Channel
if the type of I/O event is inter-domain communication then
    j ← lchn->notify_vcpu_id
    k ← ld->domain_id
    if rd->domain_id == 0 and tCPU_j^k.state == NON-MASKING then
        if tCPU_j^k ∉ TRACK-POOL then
            Insert tCPU_j^k in TRACK-POOL
        end if
        tCPU_j^k.eventCount++
        tCPU_j^k.eventInterval ← NOW - tCPU_j^k.lastSendTime
        tCPU_j^k.lastSendTime ← NOW
        tCPU_j^k.eventFrequency ← WEMA(tCPU_j^k)
    end if
end if
```

C. I-BALANCER: vCPU I/O TRACKING module(ITM)

I-Balancer tracks inter-domain communication information generated by guest VMs by placing the ITM in the event channel area to monitor the asynchronous notification mechanism in para-virtualization hypervisors such as Xen or KVM based on a split-driver architecture. To provide the hypervisor with fine-grained awareness of the I/O degree for guest VM, it is important that the ITM design allows the detailed derivation of I/O metrics from the event channel status information collected per I/O-intensive vCPU, not per VM. Here, the I/O metrics tracked by the ITM include the channel port number, VM ID, and vCPU ID for each sender and receiver that performs inter-domain communication, I/O event time generation, etc.

In Xen's split-driver architecture, the guest VM sends I/O data to DOM0, which handles the actual I/O requests of all guest VMs. At this point, the guest VM takes an asynchronous notification mechanism to DOM0 by sending an inter-domain event for handling its I/O, so, the destination VM for the inter-domain event to requesting network or disk I/O is also DOM0 (Hereafter, the inter-domain event sent to DOM0 is referred to as an I/O event). ITM uses this to form a TRACK-POOL for vCPUs with DOM0 as the destination VM of the I/O event (in Xen, the VM ID of DOM0 is assigned as '0'), to extract the I/O metric information. The key behavior of ITM is to derive the I/O event frequency metric of vCPUs included in TRACK-POOL and deliver it to the I-Balancer through shared memory communication. The I/O event frequency, determined by leveraging a simple weighted exponential moving average (WEMA) formula per vCPU, reflects recent variability in I/O strength.

Algorithm 1 shows the ITM process in the event-channel area. The 'ld' and 'rd' are structures of the domain that contain information on VM and are managed by Xen. For 'ld', it represents the VM to generate the I/O event to handle the I/O request. The 'rd' represents the VM that forms an event channel with 'ld' to receive the I/O event. The 'lchn' represents the structure to take the event channel mechanism and is also managed by Xen. The ITM tracks the event channel status of vCPUs with a destination VM ID of '0' for vCPUs performing inter-domain communication. If the vCPU does not belong to the TRACK-POOL, I-Balancer simply adds that vCPU to the TRACK-POOL. In this case, the ITM tracks only the vCPU with a nonmasking state and does not track the vCPU with a masking state. The reason for this is explained in next Section D of IV, where the operation of RTM is described. The ITM then applies a WEMA equation to the frequency of an I/O event for the corresponding vCPU, thereby reflecting the recent trend in the degree of I/O generation in each vCPU of the TRACK-POOL.

D. I-BALANCER: vCPU I/O RUNTIME TRACKING MODULE (RTM)

As the I/O-intensive vCPU has a greater chance of taking up more pCPU, the opportunity to perform I/O through DOM0 increases, thereby increasing the opportunity for I/O device occupancy. However, I/O-intensive vCPUs are affected by the neighboring frequency of vCPUs and the time of pCPU occupancy. Regarding the run-queue placement for I/O-intensive vCPUs, the degree of pCPU contention for I/O handling is relatively higher for run-queues with a higher density of I/O-intensive vCPUs. In addition, vCPUs with both I/O-intensive and CPU-intensive workloads have a higher credit burnout rate than vCPUs with only I/O workloads, which can lower I/O performance by reducing the chance of pCPU preoccupation. The scheduling dependency problem leads to unfair I/O performance between vCPUs with similar I/O levels in virtualized systems.

I-Balancer adopts RTM to eliminate scheduling dependency problems as much as possible and to follow the vCPU I/O in more detail. RTM is performed in an abstract scheduling interface area that can adopt multiple vCPU schedulers provided by Xen to enable the tracking of the pure and virtual runtime of each vCPU included in the TRACK-POOL of ITM, independently. The reason for tracking the virtual runtime of vCPUs is to determine the extent of the I/O activity that occurs until the vCPU runs out of the default time slice (30ms for Credit1 scheduler and 10ms for Credit2 scheduler). Thus, I-Balancer derives an approximate percentage of I/O activity during the default time slice of the vCPU scheduler. The reason I-Balancer updates the I/O information for only vCPUs that occupied the pCPU as much as the default time slice of vCPU scheduler is that each vCPU in the TRACK-POOL has a various credit burnout rate. It is difficult to determine the degree of I/O request occurrence of vCPUs belonging to the TRACK-POOL because of changes in workload and timing differences in time slice reallocation.

The recently reallocated vCPUs have a relatively high amount of credit with respect to neighboring vCPUs, pCPU preemption is guaranteed, which naturally leads to higher levels of I/O requests in DOM0; vCPUs with mixed workloads burn their time slices faster and therefore have less

Algorithm 2 I-Balancer: RTM Procedure

prev: Represents vCPUs before context switch by $do_schedule()$. it's structure is managed by Xen
tCPU:Same on Algorithm 1

```
j ← prev->domain->domain_id
k ← prev->vcpu_id
entryTime ← prev->run_state_entry_time
if tCPU_j^k.State == NON-MASKING and tCPU ∈ TRACK-POOL then
    tCPU_j^k.pure-runTime ← tCPU_j^k.pure-runTime + (NOW - entryTime)
    if tCPU_j^k.lastEventCount ≤ tCPU_j^k.eventCount then
        tCPU_j^k.Virtual-RunTime ← tCPU_j^k.Virtual-RunTime + (NOW - entryTime)
        tCPU_j^k.eventCount ← tCPU_j^k.lastEventCount
    end if
end if
if tCPU_j^k.pure-runTime ≥ DEFAULT TIME SLICE then
    tCPU_j^k.state ← MASKING
end if
Do do_schedule()
Do context_switch() or continue_running()
```

chance of occupying pCPUs than vCPUs that handle only I/O workloads. With vCPUs guaranteed different levels of pCPU occupancy as a result of the scheduling dependency problem, it is not appropriate to update the traffic control table based on the synchronization time of all vCPUs in the TRACK-POOL. Therefore, considering the each vCPUs in the TRACK-POOL, I-Balancer updates I/O information only for vCPUs with ran as much as deault time slice through ITM and RTM. The traffic control table is referred to by the I-Balancer to derive the control parameters for I/O traffic control mechanisms of I/O-intensive vCPUs.

Algorithm 2 shows the operation of RTM. The 'prev' is the vCPU structure of Xen's abstract scheduling interface and refers to the vCPU which is occupying the pCPU before scheduling occurs. This vCPU structure map the vCPU runtime information managed by the Credit1 scheduler or Credit2 scheduler. RTM derives the runtime of prev based on the current system time (by leveraging a function NOW() provided by Xen) by using the member(*s_time_t run_state_entry_time* in xen/include/sched.h) of the 'prev' structure, which stores the time the vCPU enters the running state. RTM tracks pure runtime and virtual runtime if that vCPU belongs to TRACK-POOL. In this case, if the runtime of the vCPU is meet the default time slice, the RTM changes the vCPU to a MASKING STATE. The reason for I-Balancer taking the MASKING STATE is to extract the I/O degree of each vCPU independently based only on the default time slice of the vCPU scheduler by excluding the scheduling dependency problem as much as possible. So, vCPU with masking states are excluded from tracking vCPU information in RTM and ITM until I-Balancer updates the information in the traffic control table of that vCPU.

E. I-BALANCER: I/O TRAFFIC CONTROL MECHANISM

I-Balancer derives the control parameters from the traffic control table that manages the network and disk-intensive vCPU of I/O information obtained from ITM and RTM. Table 2 represents the control parameters using in I/O Traffic Control Mechanism. I-Balancer measures the I/O strength of each vCPU in TRACK-POOL, based on the control parameters of the vCPUs in the TRACK-POOL. If the I/O strength is unfair among the vCPUs in the TRACK-POOL, the I/O traffic control mechanism is activated to provide fair I/O strength to the I/O-intensive vCPUs. The detailed procedure for the I/O traffic control mechanism of the I-Balancer is introduced as follows.

TABLE 2. Control parameters.

Control Parameters	
EN	Number of I/O Events of vCPU
EI	I/O event frequency of vCPU based on WEMA
VR	Segmented virtual runtime of vCPU that is generating I/O events
PR	Segmented pure runtime of vCPU

1) FIRST STAGE: TRIGGERING

In a virtualized system based on the Xen architecture, the vCPU scheduler and event channel operate in areas independent of each other. In the I-Balancer, the I/O traffic control mechanism is triggered when unfair I/O performance is detected based on the control parameters derived from the traffic control table reflecting the shared memory communication from ITM and RTM. I-Balance applies Equation 1 to each vCPU in the TRACK-POOL to determine if it is practically I/O intensive.

$$vCPU_{type} = \begin{cases} CPU, & if \frac{VR_{sum}}{PR_{sum}} \times 100 < \alpha, \\ I/O, & if \frac{VR_{sum}}{PR_{sum}} \times 100 \geq \alpha \end{cases} \quad (1)$$

In Equation 1, about $vCPU$ in the TRACK-POOL, VR_{sum} represents the total virtual runtime, and PR_{sum} represents the pure runtime of $vCPU$. Therefore, Equation 1 derives the I/O degree of $vCPU$ based on the virtual runtime over pure runtime ratio of $vCPU$; α is a constant for determining the workload type of the vCPU. If the VR_{sum}/PR_{sum} ratio is less than α, the I-Balancer concludes that the vCPU is CPU-intensive, and if the ratio is greater than α, the vCPU is determined to be I/O-intensive. In this study, the value of the constant α in Equation 1 was set to 30, which is the optimal value derived through our empirical experiments.

Figure 6 shows the procedure for determining the vCPU workload type for the I-Balancer. In Figure 6, the RTM gathers the segmented virtual runtime for the vCPUs in the vCPU scheduler based on the I/O event generation time (black box) tracked by ITM. The vCPU's pure runtime until meets the amount of default time slice is divided into a CPU-intensive workload handling time (gray box) and an I/O-intensive workload handling time (scratch box). The (1), (2), and (3) in Figure 6 represent cases in which the type of vCPU workload is determined through Equation 1. The (3) represents the type of vCPU workload based on the virtual runtime when processing a mixed workload. The state on the

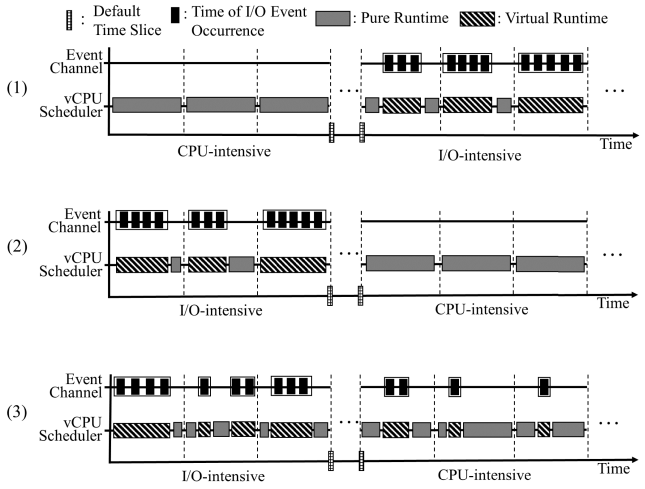

FIGURE 6. vCPU workload type determination state based on Equation 1.

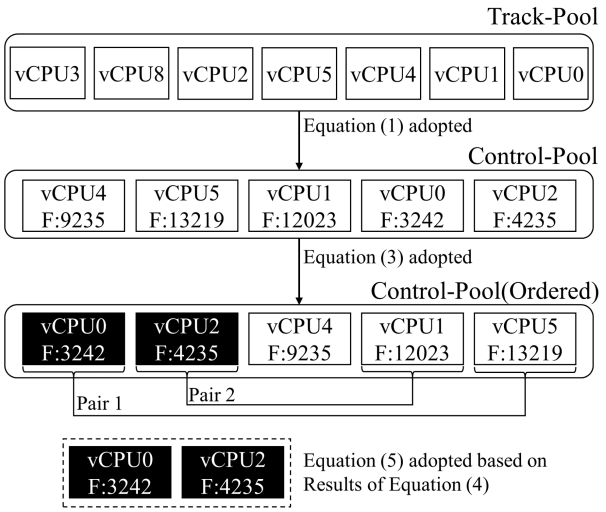

FIGURE 7. I-Balancer operation procedure up to second stage.

left of (3) is recognized as vCPU processing of I/O-intensive workload because the virtual runtime ratio satisfies the condition that the result of Equation 1 is I/O, and the right shows the opposite situation. As a result, the I-Balancer can quickly determine the vCPU workload type corresponding to the change in the workload characteristics through virtual runtime tracking.

In the event channel area, the I/O event traffic of each vCPU can be sufficiently classified for I/O-intensive vCPUs. However, because I/O events can be generated only by the pCPU occupancy of a vCPU, so defining workload type of vCPU only in event channel area has no awareness of the scheduling dependency problem, such as the workload impact between neighbor-vCPUs and the timing difference in credit reallocation of each vCPU. Based on the virtual runtime to pure runtime ratio of vCPUs, the I-Balancer can derive the actual degree of I/O workload handling of vCPUs, reflecting the scheduling dependency problem. After determining the I/O-intensive vCPUs through Equation 1, the I-Balancer forms a CONTROL-POOL for the I/O-intensive vCPUs.

$$Triggering() = \begin{cases} 1, & if \ \frac{\sum_{i=1}^{n} |V_i^{EN} - \overline{V^{EN}}|}{\overline{V^{EN}}} \times 100 > \beta, \\ 0, & if \ \frac{\sum_{i=1}^{n} |V_i^{EN} - \overline{V^{EN}}|}{\overline{V^{EN}}} \times 100 \leq \beta \end{cases} \quad (2)$$

- n is number of vCPUs in TRACK-POOL

I-Balancer then checks whether there is unfair I/O performance in the virtualized system by adopting Equation 2 for each vCPU in the CONTROL-POOL. Subsequently, the I-Balancer calculates the deviation percentage for the number of I/O events of all vCPUs in the CONTROL-POOL, and if the deviation percentage is greater than β, the I/O traffic control mechanism is performed.

$$Period(ms) = DEAFAULT\ TIMESLICE \times n \quad (3)$$

- n is number of vCPUs in TRACK-POOL

The I-balancer triggering operation cycle is given by Equation 3. The default time slice of the vCPU scheduler (Credit1 is 30 ms, Credit2 is 10 ms) is multiplied by the number of I/O-intensive vCPUs that make up the TRACK-POOL. This is because both Credit1 and Credit2 schedulers have a scheduling policy that prefers to give a more pCPU occupancy to the I/O-intensive vCPUs than CPU-intensive vCPUs. Therefore, it is sufficient to understand the overall degree of I/O traffic of the virtualized system. If Equation 2 is satisfied in the triggering module of the I-Balancer, the vCPU sorting and pairing stage follows next.

2) SECOND STAGE: SORTING AND PAIRING

In this step, a more detailed I/O control mechanism is applied by considering the relative I/O strength between the vCPUs as much as possible. In split-driver architectures, vCPUs with lower I/O strength generally have a relatively longer I/O event occurrence period and fewer I/O events than vCPUs with higher I/O strength. Based on this, the I-Balancer performs vCPU pairing for the vCPUs in the CONTROL-POOL by considering the relative difference in I/O strength. Figure 7 shows the procedure of an I/O traffic control mechanism from the first to the third stage of I-Balancer. As shown in Figure 7, I-Balancer lists the vCPUs in the order of high WEMA-based I/O event frequency and maps them starting with the vCPUs with the lowest WEMA-based I/O event frequency. The reason for performing vCPU mapping is to apply a detailed I/O delay time by differentiating the various I/O strengths of vCPUs formed by vCPU scheduling latency, which is one of the vCPU scheduling dependency problems. After performing the vCPU pair configuration, I-Balancer initiates the next step, the I/O delay time calculation.

3) THIRD STAGE: CALCULATING I/O DELAY TIME

Before describing this step, an experiment was performed to derive the optimal I/O event delay processing time that affects

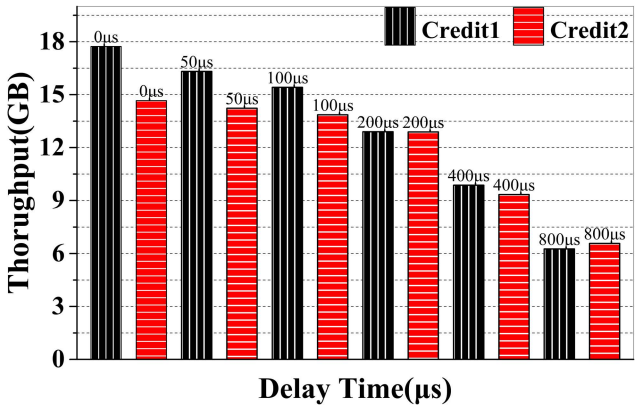

FIGURE 8. Experimental results for adopting delay time.

the I/O degree of the vCPU such that the I/O requests of the VM could be processed evenly. In the experimental environment, a delay processing time of $50\mu s$ to $800\mu s$ was applied whenever an I/O event for an I/O request occurred in each of the six VMs by modifying the event channel area source code. All VMs, including DOM0, were allocated a total of four vCPUs; DOM0 was pinned to pCPU0, and the server-VM acting as a server was pinned to pCPU7; All vCPUs of VM1-6 were pinned in order from pCPU1 to pCPU6 to avoid the vCPU scheduling dependency problem. Experiments were conducted in the virtualized system to which Credit1 and Credit2 schedulers were applied, and the VM1-6 environment generated I/O events through iperf for server-VM. The experimental results are shown in Figure 8, and it is confirmed that a delay time of $400\mu s$ for the inter-domain communication for each vCPU scheduler affects the VM I/O traffic of Xen's split driver architecture. Thus, a fixed constant of $200\mu s$ was used in Equation 5, which calculates the I/O event delay time of each vCPU, to limit the period difference range of each vCPUs for the I/O events. This constant $200\mu s$ can prevent VM with high I/O strength from dramatically degrading as a result of the I/O traffic control mechanism that provide more I/O device access for VMs with low I/O strength.

Based on the order of vCPU pairing in the second stage, the I-Balancer designates a vCPU with high I/O strength as the target vCPU and a vCPU with low I/O strength as a non-target vCPU. The target vCPU is a vCPU subject to I/O delay processing in controlling the I/O traffic, and the non-target vCPU is referenced in deriving a relative delay processing time for target vCPUs and is excluded from delay processing. Equations 4 and 5 represent the procedure of I-Balancer, determining the I/O delay processing time through the target and nontarget groups.

$$DRV(\mu s) = \frac{\sum_{i=1}^{n/2} V_i^{EI}}{n/2} \times \frac{\sum_{i=n/2+1}^{n} V_i^{EI}}{n/2} \tag{4}$$

- n is number of vCPUs in CONTROL-POOL

For Equation 4, first, the I-Balancer forms target and non-target vCPU groups to obtain the delay reference value

Algorithm 3 I-Balancer: I/O Control Mechanism

tCPU: Same as *tCPU* in Algorithm 1, 2
mCPU: A I-Balancer's vCPU structure used to derive control parameters
n: Number of vCPUs in the CONTROL-POOL

```
set j with the value of lchn->notify_vcpu_id
set k with the value of ld->domain_id
if Equation (3) is satisfied then
    for tCPU_i ∈ TRACK-POOL do
        if tCPU_i.state == MASKING then
            Copy mCPU_i with tCPU_i
            Initializes the tCPU_i's value of all members of the structure to
            the value of '0'.
            tCPU_i.state ← NON-MASKING
        end if
        mCPU_i.type ← Equation 1
        if mCPU_i.type == IO then
            Insert mCPU_i into CONTROL-POOL
        end if
    end for
    if the value of Equation 2 is FALSE then
        for mCPU_i ∈ CONTROL-POOL do
            mCPU_i.isTarget ← FALSE
        end for
        goto OUT
    end if
end if
Perform the sorting and pairing on the vCPUs in the CONTROL-POOL

DRV ← Equation 4
while i ≦ n/2 do
    mCPU_i.delayTime ← Equation 5 based on value of DRV
    i++
end while
OUT:
if rd->domian_id == 0 and mCPU_j^k.isTarget == TRUE then
    Adopt the delay time with the value of mCPU_j^k.delayTime
end if
Send network or disk I/O event to destination VM
```

(DRV), which is the average I/O event period difference for the two groups. This reflects the relative difference for the WEMA-based event period on the general I/O strength of the nontarget and target vCPU groups. Through the sorting and paring of the second stage, target vCPUs can be represented as V_1 to $V_{n/2}$ and nontarget vCPUs as $V_{n/2+1}$ to V_n. After deriving the DRV, I-Balancer applies Equation 5 to derive the I/O delay time relative to the I/O strength for each vCPU of the target group, whereby the relative I/O delay time is derived by multiplying the DRV by the decrease rate for the relative I/O event period of the vCPU of the non-target group.

$$vCPU_{DealyTime}(\mu s) = 200\mu s \times \frac{V_{n-i+1}^{EI} - V_i^{EI}}{DRV} \tag{5}$$

- n is number of vCPUs in CONTROL-POOL

4) LAST STAGE: ADOPTION, AND TERMINATION

In the last stage, I-Balancer applies I/O delay processing based on the time value of Equation 5 whenever the vCPU of the target group (Black boxes in Figure 7) generates an I/O event to DOM0 during inter-domain communication. When I-Balancer confirms that the triggering condition is not

TABLE 3. Experiment environment.

Host System Environment	
CPU	Intel core i9-9900 @ 3.6 GHZ, 8 cores without no hyper-threading technic
RAM	DDR4 32GB
OS	Ubuntu 18.04 LTS, including All guest VMs
Hypervisor	Xen 4.12
Guest VM	4 vCPUs with 2048MB of RAM
DOM0	6 vCPUs with 8096MB of RAM, pinned to pCPU0 to pCPU1
Target System Environment	
CPU	Intel core i9-9900 @ 3.6 GHZ, 8 cores without no hyper-threading technic
RAM	DDR4 32GB
OS	Ubuntu 18.04 LTS

satisfied through Equation 2, the I-Balancer recognizes that there is only a weak difference in I/O strength between vCPUs performing inter-domain communication and stops the I/O traffic control mechanism. Here, the I/O delay processing time of all vCPUs of the target group is initialized to '0' so that delay processing is no longer performed. Algorithm 3 shows the overall procedure of the I/O traffic control mechanism of the I-Balancer.

V. EXPERIMENT

In this section, the extent to which I-Balancer alleviates the fairness of I/O performance by controlling the inter-domain communication of virtualized systems through I/O traffic control mechanism the among VMs is demonstrated through comparative experiments on an existing virtualized system (existing system) as well as a virtualized system with I-Balancer (I-Balancer). In addition, the degree of overhead caused by the I/O traffic control mechanism of I-Balancer is compared with that of the existing system. To confirm that I-Balancer is not dependent on a specific scheduler, an experiment was conducted by dividing the virtualized system to apply Xen's two general-purpose schedulers, Credit1 and Credit2. In all test scenarios, existing systems are classified into Credit1 scheduler-based virtualized system (Credit1) and Credit2 scheduler-based virtualized system (Credit2). Virtualized systems with I-Balancer are classified as Credit1 scheduler-based virtualized system (Credit1-IB) and Credit2 scheduler-based virtualized system (Credit2-IB).

A prototype I-Balancer was designed and developed based on the Xen 4.12 based para-virtualized system, and Table 3 outlines the overall experimental environment and configuration. For the resource configuration of the VM, vCPUs were pinned to a total of two pCPUs with pCPU0-1 for the six vCPUs allocated to DOM0 to avoid performance interference caused by the pCPU preemption contention and four vCPUs allocated to all guest VMs, and the remaining vCPUs were pinned to pCPU2-7 [5], [15], [40], [47], [48]. All vCPUs can perform vCPU migration for load-balancing against a pinned pCPU pool.

Before confirming and analyzing the experimental results, the definitions used in the experiment are explained. First, the workload types are classified into three categories: I/O, CPU, and mixed. The vCPU or VM that handles I/O-intensive workloads is represented as a network or disk-intensive vCPU and VM. A vCPU and VM that handle CPU-intensive workloads are referred to as CPU-intensive vCPU and VM. Mixed-intensive vCPU or VM means that vCPU and VM concurrently handle I/O -and CPU-intensive workloads. The experimental environment was configured as two experimental environments according to the characteristics of the VM's workload: the first is defined as an environment in which one type of workload occurs (union environment) and the second is the environment in which two types of workloads occur (mixed environment).

TABLE 4. Benchmark tools.

Benchmark Tool	
stress [42]	Used for generating CPU workload on Mixed-intensive vCPU
iperf [43]	Used for generating network workload
dd [44]	Used for generating disk workload
sysbench [45]	Used for generating CPU workload to aware of CPU overhead
hping3 [46]	Used to measure network RTT for each VM

The overall experimental test scenario is as follows. The first experiment confirmed the effect of node-to-node communication on the actual network latency in the virtualized system. The second experiment compared the degree of fairness guarantees for I/O performance by generating the network- and disk-intensive workloads. In the third experiment, the degree to which Equation 2 in Section IV was satisfied was checked in detail to find out if I-Balancer's I/O traffic control was well applied. Finally, in the fourth experiment, the degree of overhead and performance interference that can occur when applying I/O traffic control mechanisms of the I-Balancer to virtualized systems was explored. Table 4 describes the benchmark tools used to generate I/O- and CPU-intensive workloads in the experimental evaluation.

A. I-BALANCER EFFECTS ON NETWORK LATENCY

In this experiment, the extent to which I-Balancer affects the network latency of the virtualized system is confirmed through the network latency distribution of VMs for the existing system and I-Balancer. The experimental method measures the RTT of each VM from the host system to the target system using the Linux benchmark tool 'hping3' with all VMs in the host system sending one packet every $100\mu s$ to the target system through TCP/IP communication for a total of 25000 pings during the experiment. To construct a mixed environment in which performance interference occurs in network I/O, each VM was set to generate CPU-intensive workloads using a "stress" benchmark tool. All vCPUs of DOM0 were pinned to pCPU0 and pCPU1, and all vCPUs of eight guest VMs were pinned to pCPU2-7, so a total of 32 vCPUs shared six pCPUs. The experimental results are shown in Figure 9.

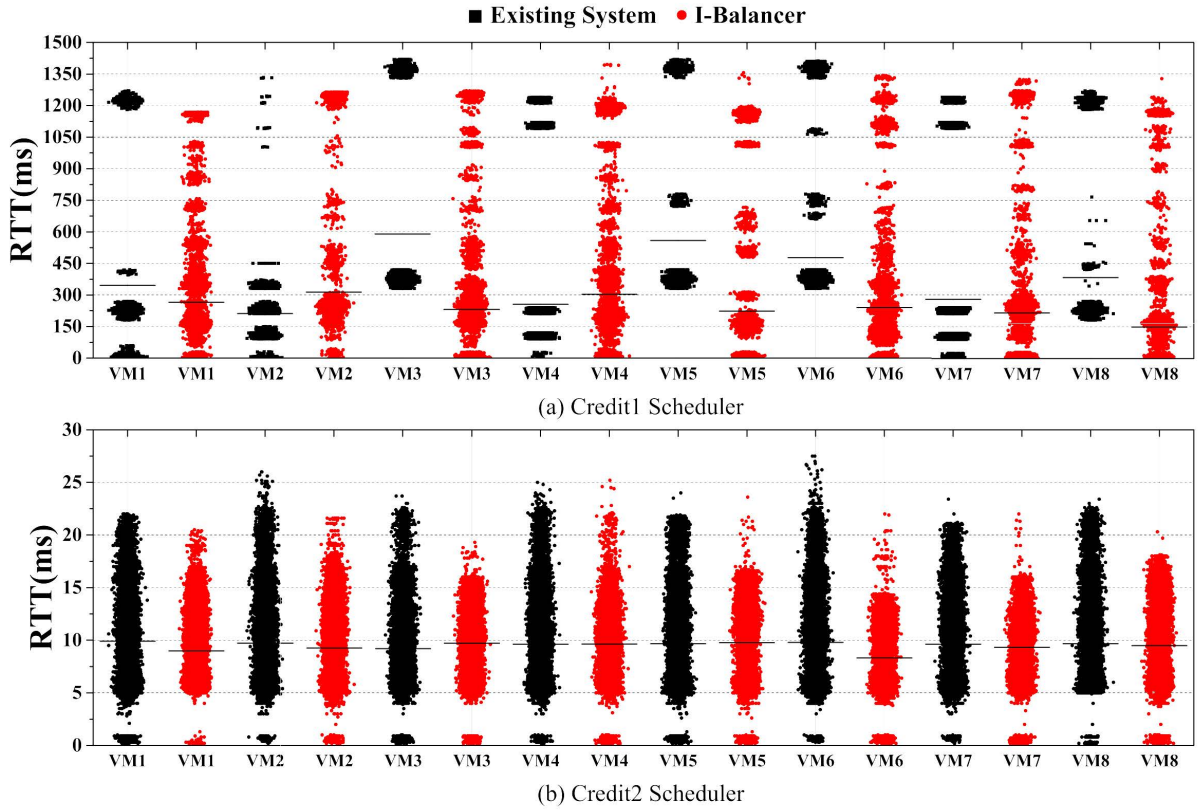

FIGURE 9. Experimental results for network latency measurement.

TABLE 5. Numerical analysis table based on Figure 9.

Mixed Environment	Mean(ms)	SD	MAD	Median(ms)	≥500ms	≥30ms
Credit1	387.82	327.08	233.87	268.47	13.34	-
Credit1-IB	242.80	250.87	142.75	194.17	7.46	-
Credit2	9.64	4.04	3.40	9.42	-	0.64
Credit2-IB	9.31	3.22	2.78	9.35	-	0.28

In Figures 9(a) and 9(b), the RTT distribution for 20000 out of 25,000 pings is shown, excluding 10% of pings after the start of the experiment and 10% of pings before the end of the experiment. Also, in 9(b), only the RTT distribution between 0 ms and 30 ms is expressed since in general, the ping RTT of the Credit2 scheduler is compared to the Credit1 scheduler only within this time interval. This is because, in the case of the Credit2 scheduler in the existing system and the I-balancer, the network latency exceeds 30ms only for approximately 0.5% of 20000 pings, and is therefore negligible. In both figures, the black line in the RTT graph represents the average RTT. Table 5 shows the mean, standard deviation (SD), mean absolute deviation (MAD), and median for Figure 9.

In Figure 9(a), it is observed that the degree of RTT distribution of the Credit1 scheduler is sporadic compared to that of the Credit2 scheduler. In Credit1, the overall RTT is distributed in the specific range of 0 ms and 1500 ms. This can be attributed to the loss of BOOST priority related to the multi-boost problem, which is a chronic problem of the Credit1 scheduler because of the relatively long time slice allocation of vCPU. As a result, certain VMs monopolize BOOST priority through frequent pCPU occupation to ensure pCPU preemption, thus generating more network I/O requests to DOM0 than other VMs. A VM that monopolizes the BOOST priority has a high chance of occupying the NIC because the loading ratio of network I/O request events to the network backend driver of DOM0 is high. A VM with a relatively low boost is not guaranteed to occupy the pCPU, resulting in low network I/O request event generation and lower NIC usage. Regarding the limited network bandwidth of the virtualized system, a situation in which more network bandwidth is allocated to a specific VM repeatedly occurs; therefore, a deviation occurs in the RTT value between VMs.

The results for Credit1-IB confirm that RTT is more evenly distributed than Credit1. The average RTT of Credit1 is 387 ms, which is approximately 140 ms longer than that

of Credit1-IB, and the SD of Credit1-IB is decreased by 23.5% compared to Credit1. For MAD, it is observed that Credit1 is 268 ms, Credit1-IB is 142 ms, and Credit1-IB is reduced by about 47%. In addition, it is observed that Credit1-IB decreased by approximately 44% compared to Credit1 when the RTT ratio was 500 ms or over. In Figure 9(b), Credit2-IB has a lower overall RTT than Credit2, and Credit2-IB decreased by 20.29% and 20.58% in terms of SD and MAD, respectively, compared with Credit2. In addition, it can be seen that Credit2-IB results are approximately 56% lower than those of Credit2 for an RTT ratio of 30 ms or over.

The experimental results confirm that the overall network latency distribution of the I-Balancer is closer to the average than that of the existing system. In Figure 9(a), Credit1-IB is relatively fair in RTT compared to Credit1 as delay processing is applied through the I/O traffic control mechanism to vCPUs that generate network I/O requests for VMs with high pCPU occupancy opportunities because of the multi-boost issue. Furthermore, I-Balancer increases the opportunity network I/O device occupancy for non-target vCPUs by effectively controlling the occurrence of network I/O events based on the optimal time of delay processing using Equations 4 and 5. In addition, Table 5 shows that there is no significant difference in the overall average RTT for the existing system and I-Balancer, not resulting in considerable overhead in the network performance of virtualized systems. The I-Balancer updates and maintains I/O information related to the inter-communication of network-intensive vCPU in the traffic control table whenever the vCPU's time slice is reallocated. Therefore, based on the latest traffic control table for network-intensive vCPUs, I-Balancer monitors I/O traffic control only on vCPUs with high network resource occupancy, not all network-intensive vCPUs, so unnecessary I/O delay processing can be prevented. Moreover, if the I-Balancer determines that the network intensity of all VMs is similar through the triggering module, it quickly stops performing traffic control mechanisms to avoid unnecessary I/O delay processing. Through this experiment, it is confirmed that I-Balancer can reduce the deviation of network latency from network-intensive VMs compared to existing systems.

B. I-BALANCER IMPACT ON I/O PERFORMANCE FAIRNESS

This experiment confirms whether I-Balancer mitigates the unfair network performance in virtualized systems for network-intensive VMs. In this experiment, a VM server with the same resource configuration as that of the guest VM was set up to act as a server (See Table 3). All vCPUs of the server-VM were pinned to pCPU7. Therefore, all vCPUs of client VMs acting as clients shared pCPU2-6. The reason this experiment is configured differently from that of Section III is that I-Balancer performs traffic control over the outbound-based network I/O to prevent unfair I/O devices occupation. In addition, the target system is not used to measure the fairness of network performance because the network delay resulting from communication with the target system can be eliminated by removing the external network communication; thus, making it possible to more clearly identify changes in the network performance of each client-VM. In addition, the purpose of the I-Balancer is to mitigate the unfair network or disk performance of the virtualized system through in-system I/O traffic control for inter-communication between guest VMs. Thus, measuring internal network communication is sufficient for evaluating the I-Balancer.

The iperf benchmark tool was leveraged to create network workloads in union and mixed experiments with all client-VMs performing TCP/IP-based streaming communication with server-VMs for 120 s using packet sizes of 512 KB. The number of VMs was gradually increased in each round of the experiment. Therefore, in the last step, 12 client VMs were performed, and 48 vCPUs occupied five pCPUs. The experimental results are shown in Figure 10, which is a virtualized system based on the Credit1 scheduler, and in Figure 11, the virtualized system is based on the Credit2 scheduler. In Figures 10 and 11, (a) to (f) show the results of the union experiment, and (g) to (l) show the experimental results obtained in the mixed experiment.

1) I-BALANCER IMPACT ON NETWORK PERFORMANCE FAIRNESS

Figures 10(c), (d), and (e) display the experimental results of the union experiment environment. It can be confirmed that the average network throughput of the I-Balancer is similar to that of the existing system, and there is less variance in the network throughput between Client-VMs. In Table 6, the average network throughput of the I-Balancer is 19.75 GB, and that of the existing system is 19.34 GB. The SD is 0.21 for the I-Balancer and 0.67 for the existing system, showing that Credit1-IB is reduced by approximately 68% compared to Credit1. The MAD value is 0.34 for Credit1-IB, which decreased by approximately 65% for Credit1, which means that the network throughput of Client-VMs in I-Balancer is more centrally distributed compared with the existing system. In Figure 10(e), the average network throughput of Credit1-IB is 17.28 GB and that of the existing system is 17.04 GB; SD and MAD, are 0.12 and 0.17 for Credit1-IB, and 0.41 and 0.6 for Credit1, respectively. The SD and MAD of Credit1-IB, decreased by approximately 70% and 71%, respectively, compared to those of Credit1. In Figures 10(h) to 10(k) and Table 6, which represent the mixed environment, it is observed that I-Balancer has a lower value for SD and MAD than the existing system. The SD and MAD values of Figure 10(j), where eight Client-VMs are running, are 6.34 and 5.43 for Credit1, and 4.51 and 3.52 for Credit1-IB, respectively, displaying a decrease of 40% and 54%, respectively, compared to Credit1.

Figure 11 shows the experimental results of the network throughput for the Credit2 scheduler adopted by the existing system and I-Balancer. As for the overall experimental

TABLE 6. Numerical analysis table based on Figure 10.

		Hybrid Environment					Mixed Environment			
System	**Configuration**	Mean(GB)	SD	MAD	Median(GB)	**Configuration**	Mean(GB)	SD	MAD	Median(GB)
Credit1	(b)	20.27	1.98	1.5	26.4	(h)	19.43	10.92	9.07	22.3
Credit1-IB		20.09	1.54	1.24	25		18.76	7.24	5.59	18.67
Credit1	(c)	19.34	0.67	0.98	19.13	(i)	12.65	7.17	6.21	13.79
Credit1-IB		19.75	0.21	0.34	19.78		10.81	5.34	4.60	8.9
Credit1	(d)	18.58	0.27	0.42	18.5	(j)	9.14	6.34	5.43	6.92
Credit1-IB		18.27	0.13	0.2	18.26		9.83	4.51	3.52	8.87
Credit1	(e)	18.52	0.65	0.42	21.2	(k)	8.81	0.41	0.30	8.87
Credit1-IB		18.74	0.3	0.24	19.48		8.71	0.19	0.15	8.72

TABLE 7. Numerical analysis table based on Firgure 11.

		Hybrid Environment					Mixed Environment			
System	**Configuration**	Mean(GB)	SD	MAD	Median(GB)	**Configuration**	Mean(GB)	SD	MAD	Median(GB)
Credit2	(b)	20.50	0.59	0.48	22.1	(h)	31.09	0.78	0.69	31.13
Credit2-IB		21.29	0.28	0.21	22		31.12	0.51	0.43	31.08
Credit2	(c)	19.28	0.53	0.38	20.72	(i)	18.44	0.49	0.39	18.42
Credit2-IB		19.28	0.29	0.23	19.86		18.37	0.34	0.26	18.34
Credit2	(d)	18.73	0.79	0.62	21.3	(j)	12.37	0.42	0.32	12.38
Credit2-IB		18.82	0.36	0.27	19.6		12.23	0.25	0.16	12.2
Credit2	(e)	18.52	0.65	0.42	21.2	(k)	8.81	0.41	0.30	8.87
Credit2-IB		18.74	0.3	0.24	19.48		8.71	0.19	0.15	8.72

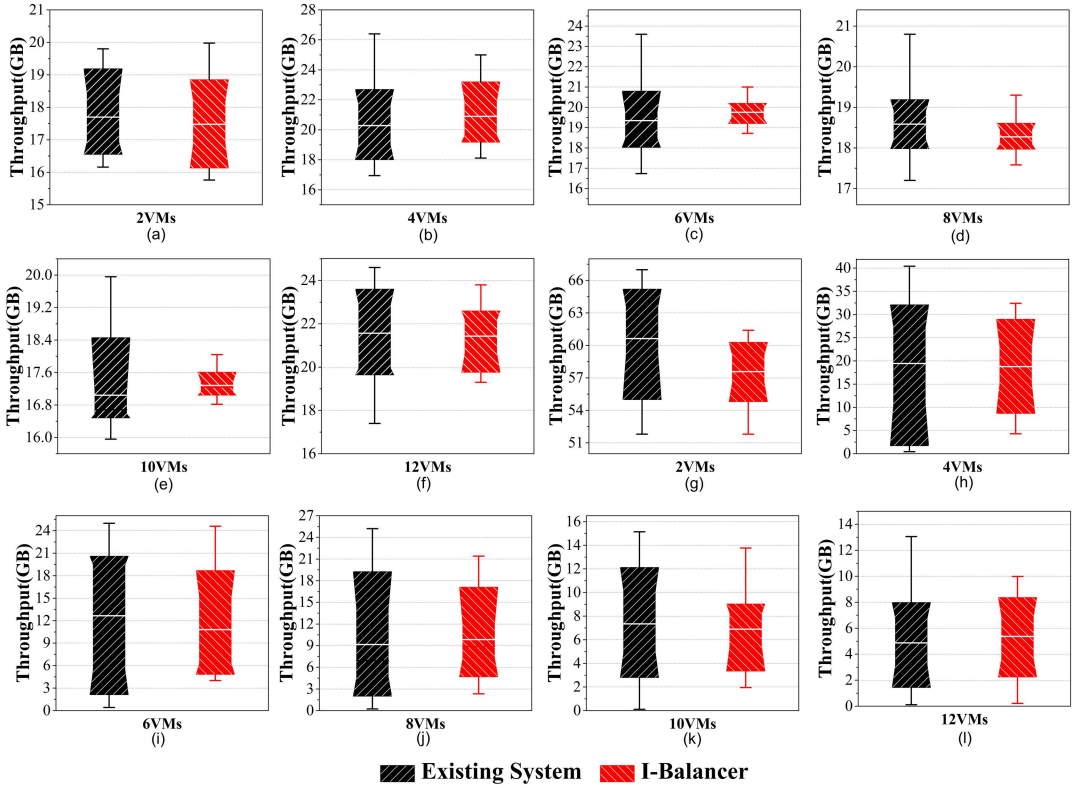

FIGURE 10. Experimental results for network I/O fairness on the virtualized environment with credit1 scheduler, (a) to (f) for union experiment and (g) to (i) for mixed environment.

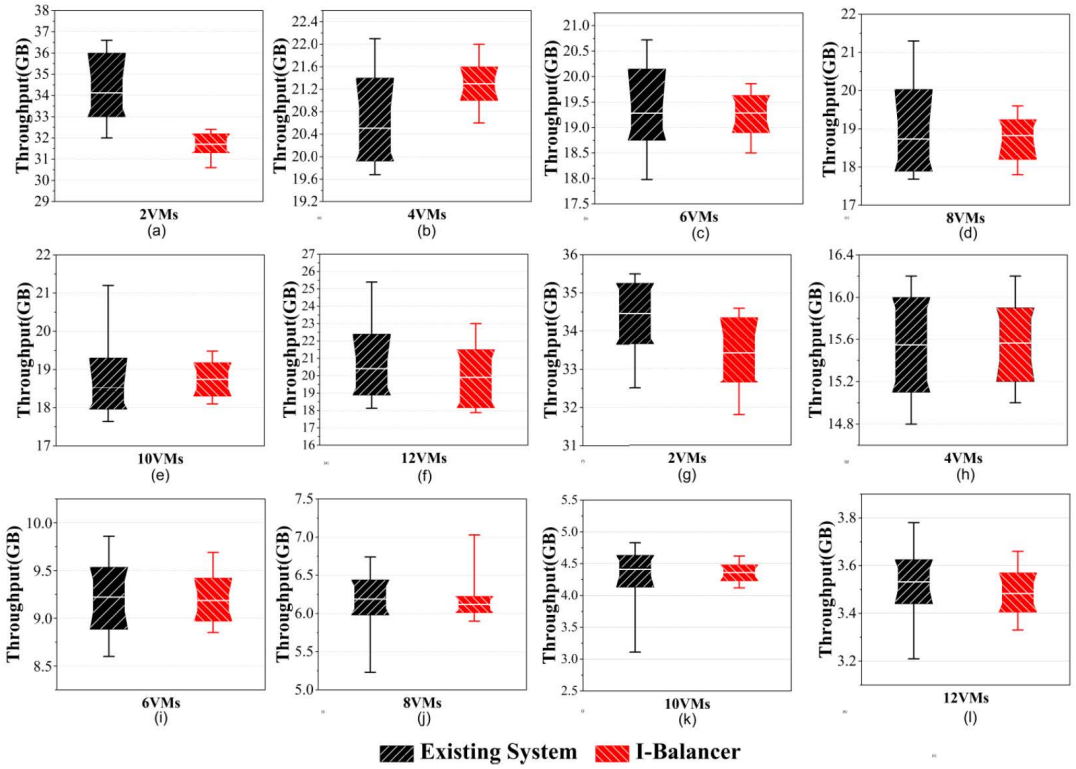

FIGURE 11. Experimental results for network I/O fairness on the virtualized environment with credit2 scheduler, (a) to (f) for union experiment and (g) to (i) for mixed environment.

results, it can be confirmed that the distribution of network throughput in I-Balancer is more concentrated around the average value than that in the existing system. For the union environment, in Figures 11(b), 11(c), 11(d) and Table 7, it can be seen that the average network throughput of the I-Balancer is more centrally concentrated than in the existing system. In Figure 10(d), the I-Balancer SD and MAD are 0.36 and 0.27, which are approximately 53% and 44% lower than those of the existing system, respectively. In the mixed environment of 10(k), although the difference of the results in values is insignificant, I-Balancer has 0.19 for SD and 0.15 for MAD indicating a decrease by 53% and 50%, respectively. From the experimental results of Figure 11, it is confirmed that the network performance of the I-Balancer is fairer among guest VMs on the Credit2 scheduler-based virtualized system.

2) IMPACT OF I-BALANCER ON DISK PERFORMANCE FAIRNESS

This experiment measures the degree to which I-Balancer guarantees fairness of disk performance between disk-intensive VMs. The 'dd' tool is leveraged to generate a disk-intensive workload by performing READ/WRITE on /dev/zero, a special file system of Linux, to create a file with a total size of 10 GB. Each credit scheduler divides the union and mixed environments, and the experiment is performed 10 times according to various disk block sizes ranging from 4 KB to 1024 KB. Figure 12 shows the average disk bandwidth for the experimental results. From the experimental results, in Figures 12(a) to 12(d), it is observed that the average bandwidth distribution of the I-Balancer is more centrally concentrated than in the existing system. In particular, SD and MAD in 11(a), are 8.21 and 6.2 for Credit1, whereas 7.32 and 5.58 for Credit2, respectively. In contrast, they are 3.35 and 2.66 for Credit1-IB (reduced by 59% and 57%, respectively, compared to Credit1), and 4.51and 3.51 for Credit2-IB (reduced by 38% and 33%, respectively, compared to Credit2). The SD and MAD in 11(b) for Credit1-IB are 4.79 and 3.7, which are lower by 51% and 61% compared to those of Credit; for Credit2-IB they are 4.34 and 3.5, which are lower by approximately 36% and 32% compared to Credit2, respectively. However, in 11(e) to 11(h), it is observed that the average bandwidth is distributed similarly to that of the existing system. Regarding the block layer, as the disk block size increases, IOPS (I/O per second) decreases; there is a delay in disk I/O delay time, so the ISR processing time increases; and the DISK I/O, requests multiple VMs from DOM0 to accommodate the increase in I/O requests, whereby the DOM0 disk access gets a bottleneck, causing I/O latency and I/O bus congestion [49]–[51], and [52]. As a result, I/O events for disk I/O requests are not that frequent, limiting the I/O fairness guarantee of the I-Balancer as a result of not having the sufficient I/O information required by the I/O traffic control mechanism. However, in 11(a) to 11(d), where the block size is relatively small and I/O events for DISK I/O requests are frequently broadcast, it can be confirmed that I-Balancer provides a fairer disk I/O performance than the existing system.

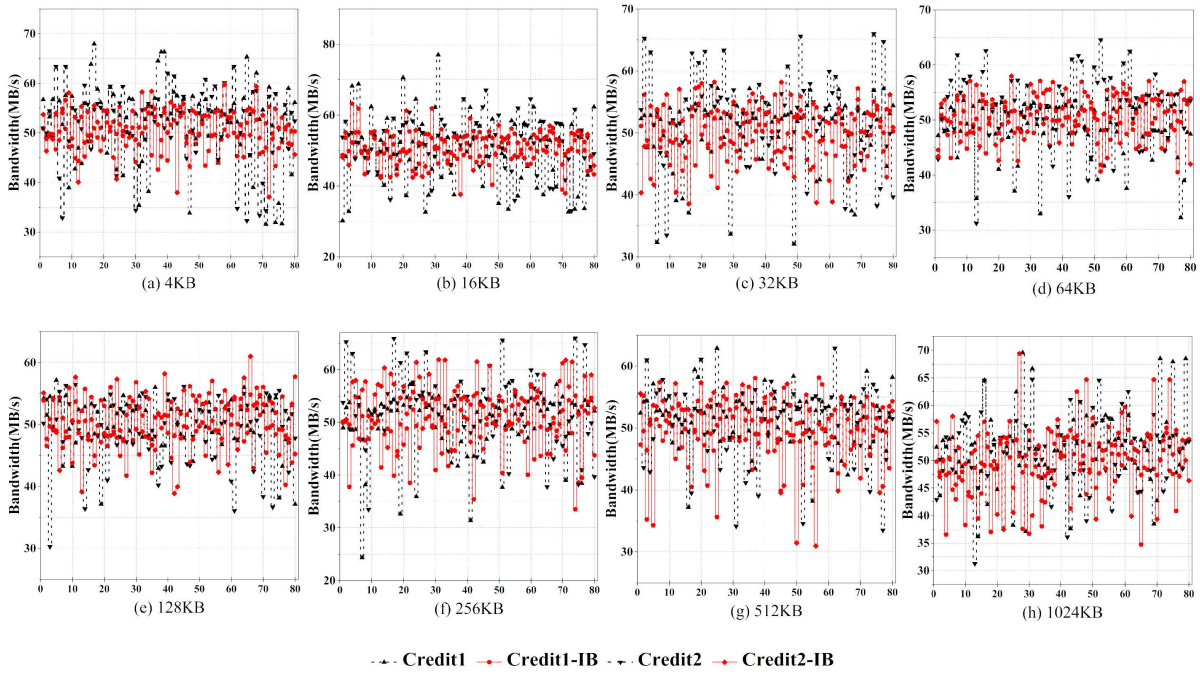

FIGURE 12. Experimental results for disk I/O fairness on the virtualized environment.

3) DISCUSSION FOR THE I-BALANCER's IMPACT ON I/O PERFORMANCE

In the existing virtualized system, in Section A-B of V, it is confirmed through experiments that unfair I/O performance emerges as the density of VMs increases because of the scheduler dependency problem and weak awareness of the user-contention architecture. The existing system focuses only on improving the I/O performance of one VM by increasing the I/O responsiveness degree through the vCPU schedulers and does not consider the fairness degree of I/O performance between VMs. As the density of VMs increases, vCPUs with various workload characteristics as well as the number of vCPUs sharing a pCPU increase, thereby increasing workload unpredictability and resource competition in the virtualized system. Consequently, unfair I/O performance between VMs emerges by adopting only the Linux kernel-based native I/O philosophy for the virtualized system and focusing only on improving I/O performance for VMs and fair pCPU sharing. In the Credit1 scheduler, it was confirmed that unfair I/O performance can occur in the virtualized system because of the multiboost problem caused by I/O-intensive vCPUs. In addition, it was confirmed that the Credit2 scheduler also causes unfair I/O performance because of frequent credit reallocation, as the number of vCPUs sharing the pCPU increases with increasing density of VMs in a mixed environment. In the case of frequent credit reallocation, I/O-intensive vCPUs take the remaining credits making neighboring vCPUs more similar; thus, pCPU preemption cannot always be guaranteed. In addition, the different workload characteristics of the vCPUs placed on each run-queue can result in different pCPU preemption cycles between I/O-intensive vCPUs, resulting in unpredictable I/O performance for virtualized systems.

Through Figures 10, 11, and 12, in a virtualized system with high VM density, I-Balancer can confirm that the network and disk throughput are similar to those of the existing system, and the experimental performance results are generally concentrated around the median. I-Balancer enables the hypervisor to recognize the fine-grained I/O traffic flow of each client-VM by tracking the I/O strength of vCPUs through RTM performing in the scheduler area and ITM performing in the event channel area. Through the I-Balancer's I/O traffic control mechanism, the unfair network performance is mitigated compared to the existing system by controlling the internal network and disk traffic through the I/O traffic control mechanism for I/O requests generated on each client VM. I-Balancer can provide a fairly shared network and disk resources by leveraging the I/O traffic control mechanism for the inter-domain communication of the split-driver architecture in a virtualized system with high VM density.

C. I/O TRAFFC CONTROL MECHANISM APPLICABILITY

An experiment was conducted to confirm the internal procedure of I-Balancer's I/O traffic control mechanism in detail to determine whether the internal I/O traffic control is well performed. Regarding the configuration of the experimental environment, we define the TAV system, which is an existing system to which only the triggering stage of I-Balancer is applied, and conduct comparative experiments on I-Balancer and TAV-system. In the test scenario, a small portion of the triggering stage of the I-Balancer was modified for analysis.

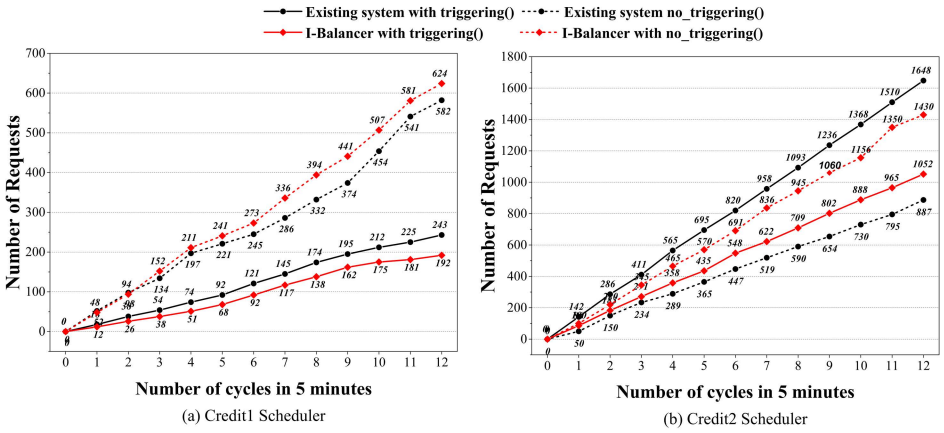

FIGURE 13. Experiment Results for I/O traffic control mechanism applicability.

Based on Equation 2, the modified triggering stage prints '*triggering*()' if high I/O strength deviation is detected for each VM, and '*no_triggering*()' is output if I/O strength deviation is determined to be low. In this case, if '*triggering*()' is printed, the I-Balancer performs the I/O traffic control mechanism, which is a procedure adopted based on Equation 2, and the TAV-System only outputs '*triggering*()' and does not perform the I/O traffic control mechanism. For the TAV-System, only RTM, ITM, and the modified triggering stage among the modules constituting the I-Balancer were adopted, and only the minimum functions necessary for the experiment were applied to generate minimum overhead.

For the TAV-system and I-Balancer, numerous '*triggering*()' outputs indicate that the VM is unfairly sharing network resources, and printing '*no_triggering*()' can be recognized as vice versa. The purpose of this experiment was to determine whether the I-Balancer was beneficial for the virtualized system. Because the comparative experiment is conducted by building a TAV-system in which there is no I/O traffic control mechanism, the configuration of this experimental environment is appropriate. To highlight the scheduling dependency problem and the lack of awareness of the I/O degree, an experiment was also conducted for the mixed environment by generating a network-intensive workload. A total of eight guest VMs of the host system performed TCP/IP-based streaming communication for 1024 KB size packets sent to the target system using 'iperf' for 5 minutes along with 'stress' to handle the CPU-intensive workload.

Figures 13(a) and (b) show the number of '*triggering*()' and '*no_triggering*()' outputs from the TAV system and I-Balancer for Credit1 and Credit2 schedulers. In Figures 13(a) and (b), the x-axis represents the total number of occurrences '*triggering*()' prints in 5 min The y-axis represents the number of triggering module operations accumulated at intervals of 25 s, for a total of 12 times over 5 min. The reason the number of prints in (b) is higher than in (a) is that the default time slice of the Credit1 scheduler is 30 ms and that of Credit2 10 ms, so the triggering module in 11(b) executes frequently. Experimental results show that I-Balancer has a higher number of '*no_triggering*()' than the TAV-system in each scheduler and that the number of '*triggering*()' is less than that of the TAV-system. For the number of '*triggering*()' prints in 11(a) and (b), within 25 seconds (1st monitoring), in (a), the TAV-system prints 18 times and I-Balancer prints 12 times. In the 12th monitoring (within 300 s), the TAV-system prints 243 times and I-Balancer prints 192 times. In 11(b), it is observed that the number of 'triggering()' prints of the I-Balancer is less than that of the TAV system by approximately 15%. As time passes, it is observed that the number of 'triggering()' prints of the TAV system gradually increases compared to I-Balancer. From the number of '*no_triggering*()' prints, it is confirmed that the I-Balancer prints are relatively high compared to the TAV system. In addition, if the I-Balancer determines that the degree of intensity for network I/O in each VM is fair, the I-Balancer does not perform the I/O traffic control mechanism on the inter-communication for all VMs; thus, it is guaranteed that the total network bandwidth is equal to that of the existing system. From the results in Figure 13, it can be concluded that the I/O traffic control mechanism performs well in providing fair network performance between each module and the VMs constituting the I-Balancer.

D. I-BALANCER IMPACT ON CPU PERFORMANCE IN THE VIRTUALIZED ENVIRONMENT

This experiment confirms whether CPU performance interference occurs by ensuring fairness of network performance through I-Balancer's I/O traffic control mechanisms in virtualized systems. In the test scenario, a total of 12 guest VMs were evaluated based on the processing completion time for CPU-intensive workloads. Regarding the workload configuration of guest VMs, the union environment ran one CPU-intensive VM, and the remaining 11 were network-intensive VMs. In a mixed environment, 12 mixed-intensive VMs were run. The vCPUs of all guest VMs were pinned to pCPU2-7;

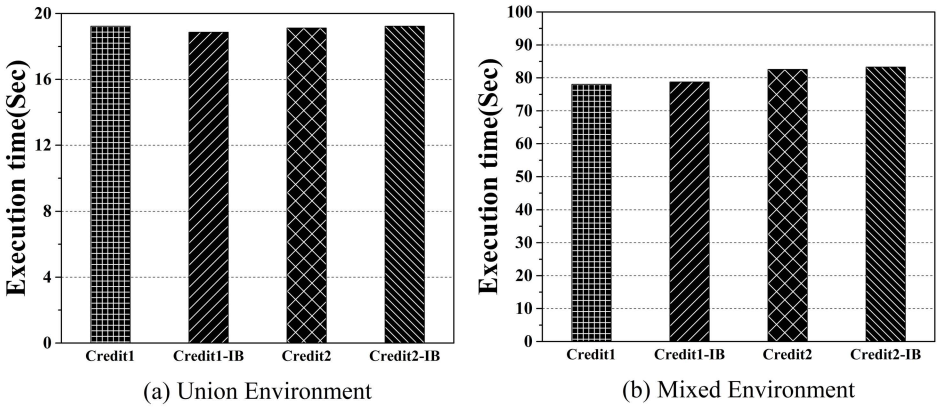

FIGURE 14. Experimental results on the effect on CPU performance.

therefore, a total of 48 vCPUs shared six pCPUs; 'iperf' was used to set the packet size to 512 KB and TCP/IP streaming was performed for communication with the target system. CPU-intensive workloads were generated by leveraging 'sysbench' and four threads were used to find the prime numbers between 2 and 10,000,000. Figures 14(a) and 14(b) show the average job completion time for 10 experiments on a virtualized system to which the Credit1 and Credit2 schedulers were applied. The average job completion time in 13(a) was 18.87 s for Credit1, 18.96 s for Credit1-IB, 19.11 s for Credit2, and 19.23 s for Credit2-IB. The average job completion time in 14(b), was 78.01 s for Credit1, 78.75 s for Credit1-IB, 82.57 s for Credit2, and 83.25 s for Credit2-IB. From the experimental results, it can be confirmed that the average job completion time for the I-Balancer and the existing system are not significantly different.

In the experiment, the reason that I-Balancer does not significantly degrade the CPU performance of the VM is that the actual network I/O control is performed in the event channel area independent of the scheduling area; therefore, I-Balancer does not affect the scheduling policy of the vCPU scheduler. In addition, I-Balancer applies an I/O traffic control mechanism only to network-intensive vCPUs with fine-grained I/O strength based on the traffic control table obtained from ITM and RTM; thus, CPU performance of neighbor-vCPUs or itself does not dramatically degrade. This means that I-Balancer does not degrade CPU performance in a space separated from the scheduling area and performs I/O traffic control effectively by providing the hypervisor with high awareness of the network I/O strength of the vCPUs.

E. IMPACT OF I-BALANCER ON DISK PERFORMANCE IN THE VIRTUALIZED ENVIRONMENT

In this experiment, to determine the degree of performance interference between network and disk-intensive VMs, the virtualized system was divided into a union environment and a mixed environment by applying the I-Balancer. A total of 10 VMs were run to configure a high VM density virtualized system such that in the union environment, if one VM was disk-intensive, the other nine were network-intensive or when three VMs were disk-intensive, seven VMs were network-intensive. All vCPUs of the VMs were pinned to pCPU2-7 such that 40 vCPUs shared six pCPUs, and all vCPUs of DOM0 were pinned to pCPU0-1. In addition, disk bandwidth was selected as an evaluation metric of performance interference. In the union environment, the disk bandwidth was checked for one to three disk-intensive VMs. In the mixed environment, all VMs simultaneously handled network- and disk I/O-intensive workloads. To generate network-intensive workloads, the 'iperf' tool was used to perform 512 KB TCP/IP streaming to the target system. To create disk-intensive workloads, the 'dd' tool was used to operate read/write to /dev/zero with a block size of 1024 bytes.

In this experiment, both the network and disk generated I/O events on the event channel; however, because the I/O handling procedure for each I/O device is different, the pattern of I/O events exposed on the event channel is also different. Because the network I/O performs streaming-based data processing, it has relatively low latency for NIC access through memory page flipping. The NIC that receives the packet from the Net-Backend driver of DOM0 performs short-term I/O processing by simply forwarding the packet to the exterior, which makes ISR completion time very short. Therefore, I/O events of the network are frequently exposed in the event channel. However, disk I/O involves long-term I/O handling procedures that include disk seek times for fixed-size blocks, data communication latency, relatively deep I/O stacks, and long I/O paths. The block-based data I/O of the disk device is generated in units of I/O event sets in the event channel. The occurrence of such a set unit spreads out the event frequency between sets of I/O events and creates an intermittent I/O event pattern in the event channel. In this case, I-Balancer's I/O traffic control mechanism naturally focuses on network I/O traffic control rather than disk I/O traffic control because I/O events of naturally network-intensive VMs are frequently broadcast on event channels.

Figure 15 shows the average DISK bandwidth for ten rounds of the experiment. In Figure 15(a), when one disk I/O

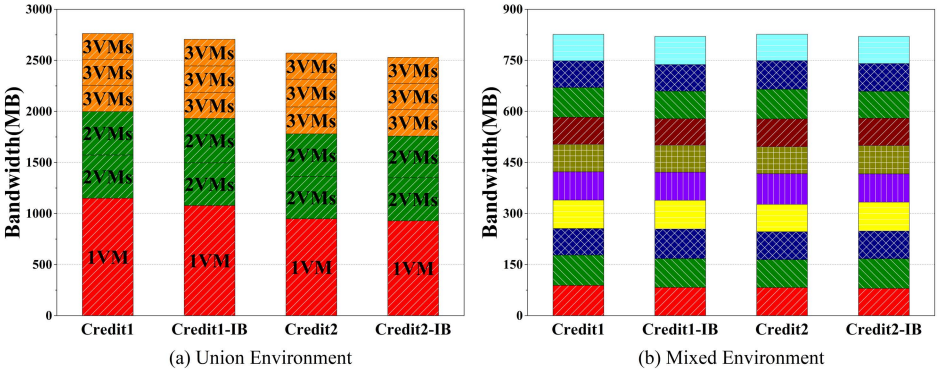

FIGURE 15. Experimental results on the effect on DISK performance.

intensive VM is running, the disk bandwidth is 1150.7 MB/s for Credit1 and 1112.8 MB/s for Credit-IB. In Figure 15(a), when one disk I/O intensive VM is running, the disk bandwidth is 1150.7 MB/s for Credit1, 1112.8 MB/s for Credit-IN, 948.9 MB/s for Credit2, and 928.9 MB/s for Credit2-IB. Overall, the I-Balancer has low disk bandwidth due to performance interference. This is because there is no competition for disk device resources, and I-Balancer's I/O traffic control for the network causes overhead in the event channel. However, when two disk-intensive VMs are running, the average bandwidth is approximately 423 MB/s for Credit1, 426 MB/s for Credit1-IB, 415 MB/s for Credit2, and 414 MB/s for Credit2-IB. In addition, it can be confirmed that the average disk bandwidths of Credit1-IB and Credit2-IB are similar to those of the existing system, including the case of the three VMs running. In Figure 15(b), the average disk bandwidths of Credit1 and Credit1-IB are 82.6 MB/s and 82 MB/s, respectively, and Credit2 and Credit2-IB have average disk bandwidths of 82.6 MB/s and 82.7 MB/s, respectively. Regarding the overall experimental results, the I/O traffic control mechanism of I-Balancer is judged to have negligible interference between the network and disk I/O, and low I/O overhead in the virtualized system.

VI. RELATED WORK

Various studies have been conducted to alleviate unfair I/O performance in virtualized systems. Asyabi [5] proposed the vCPU scheduler ppXen, which can differentiate user service complexity through processor time (PT) and I/O quality (IOQ) for I/O- and CPU-intensive workloads to provide fair I/O performance. The ppXen derives complementary vCPUs based on the demand side of PT and IOQ and organizes a set of vCPUs so that they can be scheduled to share the same pCPU as much as possible. This allows vCPUs with a similar workload tendency to share the same pCPUs, thereby minimizing performance interference and ensuring fair I/O resource access.

Jang [[9]] presented a vCPU scheduler that applies a loan and redeem mechanism for credits based on the Credit1 scheduler so that BOOST priority is set fairly for each vCPU. By controlling the occurrence of BOOST priority, this loan and redeem mechanism provides a similar degree of I/O responsiveness to I/O-intensive vCPUs to ensure fair I/O performance in the virtualized system. I-Balancer only modifies Xen's abstract scheduling interface, which allows various vCPU schedulers to be applied to virtualized systems, to flexibly combine vCPU schedulers. In addition, the modified vCPU schedulers provide inbound I/O fairness considering I/O responsiveness, and when combined with the I-Balancer, provide outbound I/O fairness, so that a more delicate I/O fairness can be provided to the virtualized system.

Software defined network (SDN)-based external network control techniques [53]–[56] have also been studied extensively. These techniques aim to improve the network performance of VMs that are unfair as a result of traditional static network bandwidth allocation which controls the data center network from the exterior of the host system. The overall method of the external network control technique is to dynamically allocate the optimal network bandwidth by configuring SDN controller policies that are adaptive to user traffic to meet the QoE demand of user services. However, it cannot provide a solution for the network I/O unfairness of the virtualized system, and there is a limit to the I/O control of the disk area. If this SDN-based external network control technique is combined with I-Balancer, it can provide very high I/O fairness among user services. If external network I/O fairness and internal network I/O fairness for virtualized systems are satisfied, it is expected that a more elastic QoE can be derived for the user service.

Research is also being actively conducted on disk and network bandwidth allocation technology-based software units to alleviate the unfair I/O performance of existing virtualized systems. Li [57] proposed V-Scheduler, a disk I/O scheduler that allocates a fair disk bandwidth between the VMs. This technology analyzes the disk I/O request information of the backend driver based on the Shadow-Proc mechanism by recognizing the I/O context of the user service in the VM. Based on the derived disk I/O information, the existing disk I/O scheduler, which considers only the number of disk I/O

requests, is extended to present a disk I/O scheduling technique that also considers the data size of the disk I/O requests. Tan [58] proposed VMCD, a virtual multichannel-based disk I/O scheduler. The VMCD isolates disk I/O streams by allocating virtual channels to the I/O device for all guest VMs in DOM0. Then, the CFQ scheduler is advanced by applying a lottery algorithm and credit-based bandwidth adjustment technique to provide fair disk I/O performance.

In a similar technique, Songe [59] proposed I-Share, which assigns a weight to each VM and allocates a proportional time slice based on the weight. Then, I-Share dynamically adjusts the time slice according to the disk status for each VM to fairly allocate bandwidth. The dynamic network bandwidth allocation technique [60], [61] provides fair network performance through the allocation of the virtual interface function (VIF), which is a network virtualization unit of the virtualized system. These technologies configure network performance metrics for network-intensive VMs and monitor the related real-time network I/O information. The network performance parameters defined by the user and actual network performance metrics are analyzed to differentiate the required network bandwidth of the VM. It then provides fair network performance by allocating VIF credits (a credit mechanism for network bandwidth adjustment, not a vCPU scheduler credit policy) to each network-intensive VM, based on the differentiated required network bandwidth. Most of these technologies operate on the I/O driver network or disk driver of DOM0's Linux kernel layer, but I-Balancer operates on the virtualization layer, which is higher than the kernel layer, and is more portable.

VII. I-BALANCER's LIMITATION AND FUTURE WORK

Figure 15 shows I-balancer performing biased I/O traffic control for specific workload types that preferentially handle network I/O requests rather than disk I/O; and in a high VM density virtualized system when the I-Balancer handles the massive set of disk I/O, it may be seen that the efficiency of controlling disk I/O traffic decreases as the block size increases(As Figure 12, when the block size is more than 128KB). Also, I-Balancer provides static I/O fairness to all VMs without considering the I/O resource information required by each user, so it is not adaptive to each user's SLAs related to I/O performance.

Our future work to solve these drawbacks of I-Balancer is as follows. The first is expanding the I/O performance metrics collection area of I-Balancer. I/O metrics obtained from Grant tables, Xenbus, or header dumping of memory I/O, etc., without being limited to event channels and scheduling areas, can provide I-Balancer with a more accurate and clear I/O traffic control mechanism. In addition, we plan to study the I/O request event type expansion and classification (network I/O request event or disk I/O request event) technique in the event channel. This can ensure more granular asynchronous event notification mechanisms when I/O handling by building network- and disk-specific isolated I/O event communication paths. Finally, we will develop a dynamic I/O traffic control mechanism that takes into account SLAs related to per-user I/O performance. This mechanism is expected to provide differentiated I/O performance fairness for each VM based on the required I/O resources by allowing I-Balancer to recognize and distinguish various user-specific SLA levels.

VIII. CONCLUSION

In this study, I-Balancer was proposed to mitigate the unfair I/O performance caused by the scheduling dependency problem and the low perception of the I/O procedure of the virtualized system with high VM density. The key aspects of I-Balancer are to increase the awareness degree of the hypervisor's user-contention architecture in virtualized systems and provide fair access to the I/O devices of the guest VM based on the degree of I/O occupancy of the vCPU. To this end, the proposed I-Balancer controls the internal I/O traffic for each VM through an I/O traffic control mechanism for inter-domain communication in a split-driver architecture. Through ITM, the I/O degree is derived in detail in units of vCPUs, not VMs, using an asynchronous notification mechanism in the event channel for network or disk I/O requests generated by each guest VM. At this time, the independent I/O strength of only vCPU is derived by masking the scheduling dependency problem and control parameters are derived through runtime tracking (virtual and pure runtime based on each vCPU scheduler's default time slice) to vCPU using the abstract scheduling interface of RTM. Subsequently, I-Balancer converts the I/O degree of vCPU into the I/O strength of vCPU based on the control parameters and optimal delay time derived from empirical experiments. Finally, an I/O traffic control mechanism provides fair access to I/O devices according to the I/O strength of each vCPU. A prototype of the I-Balancer was built based on the Xen 4.12 hypervisor, and various experiments were conducted. The experimental results show that I-Balancer provides fair I/O performance than existing virtualized systems with negligible overhead.

REFERENCES

[1] M. Al-Ruithe, E. Benkhelifa, and K. Hameed, "Key issues for embracing the cloud computing to adopt a digital transformation: A study of Saudi public sector," *Proc. Comput. Sci.*, vol. 130, pp. 1037–1043, Jan. 2018.

[2] A. A. Alli and M. M. Alam, "The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications," *Internet Things*, vol. 9, Mar. 2020, Art. no. 100177.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[4] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[5] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, Jun. 2015, pp. 37–42.

[6] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Singapore: Springer, 2018, pp. 103–130.

[7] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *Very Large Data Base (VLDB) Endowment*, vol. 11, no. 12, pp. 2046–2049, 2018.

[8] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1–24, Jun. 2020.

[9] J. Jang, J. Jung, and J. Hong, "An efficient virtual CPU scheduling in cloud computing," *Soft Comput.*, vol. 24, no. 8, pp. 5987–5997, Apr. 2020.

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

[11] E. Asyabi, S. SanaeeKohroudi, M. Sharifi, and A. Bestavros, "TerrierTail: Mitigating tail latency of cloud virtual machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2346–2359, Oct. 2018.

[12] S. Kim, D. Kang, and J. Choi, "Fine-grained I/O fairness analysis in virtualized environments," in *Proc. ACM Res. Appl. Comput. Symp. (RACS)*, 2012, pp. 403–408.

[13] D. Kim, H. Kim, M. Jeon, E. Seo, and J. Lee, "Guest-aware priority-based virtual machine scheduling for highly consolidated server," in *Proc. Eur. Conf. Parallel Process.* Berlin, Germany: Springer, Aug. 2008, pp. 285–294.

[14] W. Jia, C. Wang, X. Chen, J. Shan, X. Shang, H. Cui, and Y. Wang, "Effectively mitigating I/O inactivity in vCPU scheduling," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 267–280.

[15] C. Li, S. Xi, C. Lu, R. Guérin, and C. D. Gill, "Virtualization-aware traffic control for soft real-time network traffic on Xen," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 257–270, Feb. 2022.

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Dec. 2003.

[17] A. K. Qumranet, Y. K. Qumranet, D. L. Qumranet, U. L. Qumranet, and A. Liguori, "KVM: The Linux virtual machine monitor," in *Proc. Linux Symp.*, vol. 1, 2007, pp. 225–230.

[18] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 181–194, Dec. 2002.

[19] Microsoft. *Hyper-V*. Accessed: Apr. 2022. [Online]. Available: http://en.wikipedia.org/wiki/Hyper-V

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

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

[22] Y. Gao, Y. Zhang, and Y. Zhou, "Building a virtual machine-based network storage system for transparent computing," in *Proc. Int. Conf. Comput. Sci. Service Syst.*, Aug. 2012, pp. 2341–2344.

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

[24] H. R. Mohebbi, O. Kashefi, and M. Sharifi, "ZIVM: A zero-copy inter-VM communication mechanism for cloud computing," *Comput. Inf. Sci.*, vol. 4, no. 6, pp. 1–10, Oct. 2011.

[25] Q. Shen, M. Wan, Z. Zhang, Z. Zhang, S. Qing, and Z. Wu, "A covert channel using event channel state on Xen hypervisor," in *Proc. Int. Conf. Inf. Commun. Secur.* Cham, Switzerland: Springer, Dec. 2013, pp. 125–134.

[26] L. Cheng and F. C. M. Lau, "Offloading interrupt load balancing from SMP virtual machines to the hypervisor," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3298–3310, Nov. 2016.

[27] *Common/Schedule.c*. Accessed: Apr. 2022. [Online]. Available: https://github.com/mirage/xen/blob/master/xen/common/schedule.c

[28] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in *Proc. 9th ACM Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2011, pp. 39–48.

[29] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing compositional scheduling through virtualization," in *Proc. IEEE 18th Real Time Embedded Technol. Appl. Symp.*, Apr. 2012, pp. 13–22.

[30] *Credit Scheduler*. Accessed: Apr. 2022. [Online]. Available: https://wiki.xenproject.org/wiki/Credit_Scheduler

[31] L. Zeng, Y. Wang, D. Feng, and K. B. Kent, "XCollOpts: A novel improvement of network virtualizations in Xen for I/O-latency sensitive applications on multicores," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 2, pp. 163–175, Jun. 2015.

[32] S. Xi, C. Li, C. Lu, and C. Gill, "Prioritizing local inter-domain communication in Xen," in *Proc. IEEE/ACM 21st Int. Symp. Quality Service (IWQoS)*, Jun. 2013, pp. 1–10.

[33] *Credit2 Scheduler*. Accessed: Apr. 2022. [Online]. Available: https://wiki.xenproject.org/wiki/Credit2_Scheduler

[34] G. Lettieri, V. Maffione, and L. Rizzo, "A study of I/O performance of virtual machines," *Comput. J.*, vol. 61, no. 6, pp. 808–831, Jun. 2018.

[35] C. Li, S. Xi, C. Lu, C. D. Gill, and R. Guerin, "Prioritizing soft real-time network traffic in virtualized hosts based on Xen," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 145–156.

[36] Y. Liang and H. Dai, "Application virtualization: An agent encapsulation of software in virtual machines to archive the execution performance in hosts," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, Sep. 2021, pp. 618–625.

[37] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, L. Iannone, and J. Roberts, "Comparing the performance of state-of-the-art software switches for NFV," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2019, pp. 68–81.

[38] X. Ling, H. Jin, S. Ibrahim, W. Cao, and S. Wu, "Efficient disk I/O scheduling with QoS guarantee for Xen-based hosting platforms," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2012, pp. 81–89.

[39] X. Wang, X. Xie, H. Jin, X. Shi, W. Cao, and X. Ke, "A disk bandwidth allocation mechanism with priority," *J. Supercomput.*, vol. 66, no. 2, pp. 686–699, Nov. 2013.

[40] J. Lee and H. Yu, "I/O strength-aware credit scheduler for virtualized environments," *Electronics*, vol. 9, no. 12, p. 2107, Dec. 2020.

[41] V. Venkatesh and A. Nayak, "Optimizing I/O intensive domain handling in Xen hypervisor for consolidated server environments," in *Green, Pervasive, and Cloud Computing*. Cham, Switzerland: Springer, 2016, pp. 180–195.

[42] *Ubuntu Manuals 'Stress.1'*. Accessed: Apr. 2022. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/stress.1.html

[43] *Ubuntu Manuals 'iPerf.1'*. Accessed: Apr. 2022. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/iperf.1.html

[44] *Ubuntu Manuals 'dd.1posix'*. Accessed: Apr. 2022. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/dd.1posix.html

[45] *Ubuntu Manuals 'SysBench.1'*. Accessed: Apr. 2022. [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/en/man1/sysbench.1.html

[46] *Kali 'hping3'*. Accessed: Apr. 2022. [Online]. Available: https://www.kali.org/tools/hping3/

[47] R. Chi, Z. Qian, and S. Lu, "Be a good neighbour: Characterizing performance interference of virtual machines under Xen virtualization environments," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2014, pp. 257–264.

[48] L. Liu, H. Wang, A. Wang, M. Xiao, Y. Cheng, and S. Chen, "VCPU as a container: Towards accurate CPU allocation for VMs," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments (VEE)*, 2019, pp. 193–206.

[49] G. Joshi, S. T. Shingade, and M. R. Shirole, "Empirical study of virtual disks performance with KVM on DAS," in *Proc. Int. Conf. Adv. Eng. Technol. Res. (ICAETR)*, Aug. 2014, pp. 1–8.

[50] B. Mao, S. Wu, and L. Duan, "Improving the SSD performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.

[51] W. Cheng, C. Li, L. Zeng, Y. Qian, X. Li, and A. Brinkmann, "NVMM-oriented hierarchical persistent client caching for Lustre," *ACM Trans. Storage*, vol. 17, no. 1, pp. 1–22, Feb. 2021.

[52] T. I. Papon and M. Athanassoulis, "A parametric I/O model for modern storage devices," in *Proc. 17th Int. Workshop Data Manage. New Hardw. (DaMoN)*, Jun. 2021, pp. 1–11.

[53] F. Xu, W. Ye, Y. Liu, and W. Zhang, "UFalloc: Towards utility max-min fairness of bandwidth allocation for applications in datacenter networks," *Mobile Netw. Appl.*, vol. 22, no. 2, pp. 161–173, Apr. 2017.

[54] J. Tian, Z. Qian, M. Dong, and S. Lu, "FairShare: Dynamic max-min fairness bandwidth allocation in datacenters," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 1463–1470.

[55] Y. Yoo, G. Yang, M. Kang, and C. Yoo, "Adaptive control channel traffic shaping for virtualized SDN in clouds," in *Proc. IEEE 13th Int. Conf. Cloud Comput. (CLOUD)*, Oct. 2020, pp. 22–24.

[56] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Bandwidth isolation guarantee for SDN virtual networks," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[57] D. Li, M. Dong, Y. Tang, and K. Ota, "A novel disk I/O scheduling framework of virtualized storage system," *Cluster Comput.*, vol. 22, no. S1, pp. 2395–2405, Jan. 2019.
[58] H. Tan, C. Li, Z. He, K. Li, and K. Hwang, "VMCD: A virtual multi-channel disk I/O scheduling method for virtual machines," *IEEE Trans. Services Comput.*, vol. 9, no. 6, pp. 982–995, Nov. 2016.
[59] S. Wu, S. Tao, X. Ling, H. Fan, H. Jin, and S. Ibrahim, "IShare: Balancing I/O performance isolation and disk I/O efficiency in virtualized environments," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 2, pp. 386–399, Feb. 2016.
[60] Z. Shao, K. Zhang, and H. Jin, "Improving fairness of network bandwidth allocation for virtual machines in cloud environment," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, Jun. 2016, pp. 1–5.
[61] L. Mei and X. Lv, "Optimization of network bandwidth allocation in Xen," in *Proc. 17th Int. Conf. High Perform. Comput. Commun., IEEE 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 1558–1566.

JAEHAK LEE (Member, IEEE) received the B.S. degree in computer science and engineering from the Tech University of Korea, Gyeonggi-do, South Korea, in 2017. He is currently pursuing the Ph.D. degree with Korea University, Seoul. His current research interests include virtualization, distributed systems, and resource management.

HWAMIN LEE received the B.S., M.S., and Ph.D. degrees in computer science education from Korea University, Seoul, South Korea, in 2000, 2002, and 2006, respectively. She is currently a Professor with the Department of Medical Informatics, College of Medicine, Korea University. Her research interests include cloud computing, time series data prediction, medical data analysis, deep learning, machine learning, and SW convergence.

HEONCHANG YU (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Korea University, Seoul, South Korea, in 1989, 1991, and 1994, respectively. Since 1998, he has been a Professor of computer science and engineering with Korea University. From January 2015 to December 2020, he was the Vice President of the Korea Information Processing Society, South Korea. From September 2019 to August 2020, he was a Visiting Professor of electrical and computer engineering with the UTSA. His research interests include cloud computing, virtualization, distributed computing, and fault-tolerant systems. He was awarded the Okawa Foundation Research Grant of Japan in 2008.