

Article

I/O Strength-Aware Credit Scheduler for Virtualized Environments

Jaehak Lee  and Heonchang Yu *

Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea;
smreodmlvl@korea.ac.kr

* Correspondence: yuhc@korea.ac.kr; Tel.: +82-2-3290-2392

Received: 5 November 2020; Accepted: 5 December 2020; Published: 10 December 2020



Abstract: With the evolution of cloud technology, the number of user applications is increasing, and computational workloads are becoming increasingly diverse and unpredictable. However, cloud data centers still exhibit a low I/O performance because of the scheduling policies employed, which are based on the degree of physical CPU (pCPU) occupancy. Notably, existing scheduling policies cannot guarantee good I/O performance because of the uncertainty of the extent of I/O occurrence and the lack of fine-grained workload classification. To overcome these limitations, we propose ISACS, an I/O strength-aware credit scheduler for virtualized environments. Based on the Credit2 scheduler, ISACS provides a fine-grained workload-aware scheduling technique to mitigate I/O performance degradation in virtualized environments. Further, ISACS uses the event channel mechanism in the virtualization architecture to expand the scope of the scheduling information area and measures the I/O strength of each virtual CPU (vCPU) in the run-queue. Then, ISACS allocates two types of virtual credits for all vCPUs in the run-queue to increase I/O performance and concurrently prevent CPU performance degradation. Finally, through I/O load balancing, ISACS prevents I/O-intensive vCPUs from becoming concentrated on specific cores. Our experiments show that compared with existing virtualization environments, ISACS provides a higher I/O performance with a negligible impact on CPU performance.

Keywords: cloud; virtualization; hypervisor; scheduler; I/O performance; workload aware; resource management

1. Introduction

Cloud technology is playing a crucial role in the construction of 21st Century IT infrastructure. It involves building high-performance computing resources into a virtualized environment to provide isolated computing resources in the form of virtual machines (VMs). Managing personal computing resources is a relatively easy task because the user knows the workload type corresponding to applications. However, because users leveraging the cloud server share physical computing resources, the performance of user applications depends on the shared resource management policy of the cloud server and not on the users. More specifically, user application performance depends on the period for which virtual CPUs (vCPUs) in VMs occupy a physical CPU (pCPU) according to the resource management policies of the hypervisor [1–3].

The hypervisor scheduler, which plays a key role in managing the shared resources of a cloud server, schedules the vCPU based on the pCPU run-time of the vCPU, thus allowing multiple user applications to occupy the shared resources fairly. Such hypervisor scheduler policies guarantee relatively good CPU performance for CPU-intensive VMs, but are limited in terms of guaranteeing I/O performance for I/O-intensive VMs [4–9]. One of the main reasons for the poor I/O performance in a virtualized environment is the unpredictable I/O occurrence due to the lack of fine-grained

distinction between the workload types of the vCPU. For example, if I/O occurs on a specific vCPU placed at the tail of the run-queue, I/O latency rises rapidly until all vCPUs that are at the front of the run-queue are scheduled [10–13].

In this paper, we propose an I/O strength-aware credit scheduler (ISACS) to improve the I/O performance in a virtualized environment with a shared resource management policy based on the degree of pCPU occupancy. ISACS operates based on Xen's Credit2 scheduler [14,15] and guarantees pCPU occupancy by providing I/O virtual credits according to the I/O strength for each VM's vCPU in the run-queue. Furthermore, to guarantee CPU performance, it provides CPU virtual credits to CPU-intensive vCPUs. Finally, ISACS performs I/O load balancing to prevent I/O-intensive vCPUs from becoming concentrated on specific run-queues.

This study is organized as follows. In Section 2, to elucidate the architecture of ISACS, we explain Xen's I/O procedure, as well as two credit schedulers. Section 3 describes the motivation for our approach with regard to an existing virtualized environment. In Section 4, we detail the implementation of ISACS to mitigate I/O performance degradation in the existing virtualized environment by expanding the Credit2 scheduler. In Section 5, the performance evaluation of ISACS is presented, and Section 6 discusses related work. Finally, our conclusions are presented in Section 7.

2. Background

Xen has adopted a split-driver model that comprises a driver domain (DOM0) and a user's VM (guest VM). DOM0 has a physical driver that handles the I/O of the guest VMs [9,13,16–18]. The guest VM has a logical driver that sends an I/O request to the backend driver of DOM0 when an I/O appears. This model further simplifies the kernel code of the guest VM and enhances the security of the Xen hypervisor through I/O management based on DOM0. In the Xen hypervisor, two credit schedulers, which are based on the credit mechanism, are employed for fairness. For both credit schedulers, credit means the "time for which a vCPU can occupy a pCPU." A vCPU under the policy of two credit schedulers occupies a pCPU for 30 ms or 10 ms when it is scheduled, and the credit of the vCPU is allocated or deducted according to "weight" which is a scheduling parameter that denotes the time for which the vCPU occupies the pCPU. This section describes the important parts of the Xen architecture to elucidate ISACS.

2.1. Credit1 Scheduler

The Credit1 scheduler schedules vCPUs based on the three priorities of vCPUs. The BOOST priority is the highest priority for guaranteeing I/O performance in the Credit1 scheduler. When the I/O response has arrived from DOM0, a vCPU that is in the sleep state awakens and obtains the BOOST priority. This vCPU preempts a pCPU before vCPUs with the UNDER or OVER state and occupies the pCPU for 10 ms. The UNDER priority is given to vCPUs with more than zero credits, and such vCPUs have a higher scheduling priority than vCPUs with the OVER state. The OVER priority is given to a vCPU with a amount of credit is less than zero; such a vCPU has the lowest scheduling priority. The "weight" is a scheduling parameter that determines the degree of occupancy of the pCPU. The larger the value of the weight, the more credit is allocated to the vCPU. The Credit1 scheduler reallocates credit every 30 ms for each vCPU in the run-queue. The BOOST mechanism improves the I/O performance of the VM; however, the BOOST priority causes the multi-BOOST problem, which arises from I/O performance imbalance due to multiple vCPUs having the same boost priority. Further, a scheduling policy based on round-robin scheduling limits I/O improvement of the Credit1 scheduler [10,16,19,20].

2.2. Credit2 Scheduler

The Credit2 scheduler was adopted as Xen's default scheduler to address the multi-boost problem of the Credit1 scheduler and to improve the I/O performance, fairness, and scalability among VMs. In the Credit2 scheduler, the three priorities (BOOST, UNDER, and OVER) of the Credit1 scheduler

were eliminated. Instead, a vCPU with a large amount of credit is scheduled first. An I/O-intensive vCPU that awakes from the sleep state to handle I/O can immediately preempt the CPU-intensive vCPU in the Credit2 scheduler; Because a vCPU with an I/O-intensive workload will probably have more credits by consuming less credit than that consumed by a vCPU with a CPU-intensive workload due to the frequent sleep state by I/O handling. Therefore, it can preempt the pCPU easily, similar to vCPUs with the BOOST priority in the case of the Credit1 scheduler. Thus, the Credit2 scheduler can enhance the I/O performance of I/O-intensive VMs and can easily overcome the multi-boost problem in the Credit1 scheduler. The “weight” value in the Credit2 scheduler refers to the degree of credit deduction, and the higher the value of the weight, the lower the credit consumption rate when a vCPU occupies the pCPU. Credit reallocation in the Credit2 scheduler occurs for each run-queue, and the condition for credit reallocation is that the amount of credit of the vCPU to be scheduled next should be less than zero. The same amount of credit, which same meaning as default credit, is provided when credit reallocation is performed will be reassigned to all vCPUs inside the run-queue.

2.3. Xen's I/O Procedure

The major mechanisms involved in Xen's I/O procedure are the shared ring buffer and event channel [11,18,21–23]. The shared ring buffer provides a shared memory communication mechanism that is used for communication among VMs and between DOM0 and guest VMs. An event channel exists in each VM, including DOM0, and it performs an asynchronous I/O notification mechanism for the arrival of I/O requests or I/O respond. The overall I/O process of Xen is as follows:

Figure 1 shows the I/O procedure of Xen. In (a), when an I/O occurs on a guest VM, I/O requests, including the request ID and I/O type, are recorded in the shared ring buffer through a logical driver [10,12,24,25]. In (b), the guest VM triggers an I/O request event via a DOM0 event channel. When the corresponding vCPU of DOM0 occupies a pCPU, it checks that there is an I/O request through an I/O request event bit. When a bit is set in the event channel, DOM0 obtains the corresponding information regarding I/O requests from the shared ring buffer. In (c), DOM0 executes an I/O handler that is related to the I/O request [1,4,8,26]. In (d), after the execution of the I/O handler is completed, the result is recorded in the shared ring buffer. Finally, in (e), the backend driver of DOM0 sends an I/O respond event to the guest VM to inform it regarding the execution of its I/O requests. Once the specific vCPU of the VM occupies the pCPU, it checks the event channel of the VM, obtains the result of the I/O request from the shared ring buffer, and performs corresponding I/O handling. We can infer from this I/O processing procedure in the Xen hypervisor that the more I/O-intensive the workloads of the vCPU, the greater the number of I/O events.

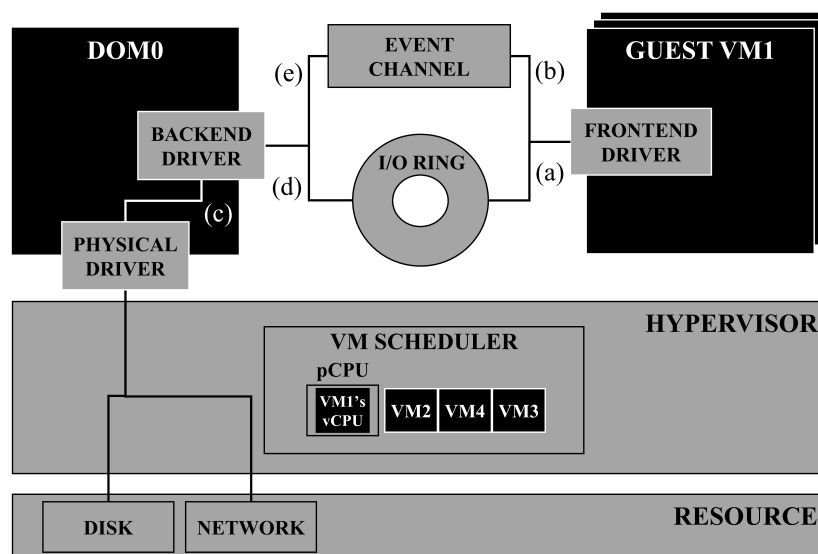


Figure 1. Xen's I/O procedure. DOM, domain; pCPU, physical CPU; vCPU, virtual CPU.

3. Motivation

The existing hypervisor scheduler policy based on the fairness of pCPU occupancy can provide good CPU performance for CPU-intensive vCPUs but poor I/O performance for I/O-intensive VMs. In a virtualized environment, to handle I/O requests or responses to the VM, a vCPU of a VM must occupy a pCPU. However, in the Credit2 scheduler, when I/O occurs for a vCPU that is placed at the tail of the run-queue, I/O latency occurs on that vCPU until all vCPUs that have more credits are scheduled in the Credit2 scheduler; this is necessary to ensure the fairness of pCPU occupancy. In addition, the reason for the poor I/O performance in a virtualized environment is the lack of awareness regarding the workload type of the VM, and more specifically, the lack of awareness regarding the extent of the relative I/O workload that a vCPU handles. In a practical cloud environment, several VMs will run on the host server, which in turn reveals that many vCPUs will share pCPUs with different workloads. The greater the number of vCPUs with different workload types, the more unpredictable is the I/O occurring in a virtualized environment [8,10,11,13]. Numerous researchers have investigated the reason for the poor I/O performance in a virtualized environment.

Some researchers have noted that in a virtualized environment with limited resources, as the number of VMs increases, the number of vCPUs placed in the run-queue also increases, resulting in I/O latency [7,10,20–22,25–27]. This indicates that when an unpredictable I/O occurs on a specific VM, I/O latency rises until all vCPUs placed in front of the VM's vCPU are scheduled. In this case, in a virtualization environment with limited resources, the workload type of the vCPU must be recognized and a scheduling policy that can adapt to the workload type of the vCPU must be devised. Some authors insist that the degradation of I/O performance in a virtualized environment emerges when DOM0, also known as the driver domain, is scheduled with the guest VM on the same core [17,24–26]. More specifically, because DOM0 handles I/O requests from all guest VMs, the vCPUs of DOM0 that handle the VM's I/O requests cannot guarantee pCPU preoccupation by the vCPUs of the guest VM. This situation leads to I/O latency in all I/O-intensive VMs, which results in poor I/O performance in a virtualized environment.

Some researchers noted that assigning the same time slice for all vCPUs without workload type recognition results in poor I/O performance in a virtualized environment [9,11,28–30]. Providing a relatively short time slice to the vCPU of an I/O-intensive VM yields good I/O performance owing to the frequent pCPU occupancy of vCPUs that handle I/O. However, without considering the workload type of the VM, allocating a relatively long time slice to all VMs increases the average run-queue waiting time for all vCPUs in the run-queue, including I/O-intensive vCPUs, resulting in I/O latency for I/O-intensive VMs. Further, concurrently executing a CPU-intensive workload and an I/O-intensive workload in the VM affects I/O performance [18,22,31,32]. For VMs with mixed workloads that perform CPU-intensive and I/O-intensive tasks simultaneously, when the vCPU processes I/O responses from DOM0, it may not complete the processing of tasks for I/O in the current scheduling round. Thus, the time slices or credits of the vCPU, which represent the degree of pCPU occupancy, are consumed to handle CPU-intensive workloads and then become scarce, resulting in a run-queue waiting time for that vCPU until the next scheduling round.

As mentioned above, numerous factors lead to poor I/O performance in a virtualization environment. The following subsections describe certain representative causes for poor I/O performance in existing virtualized environments and present our contributions to mitigate the poor I/O performance.

3.1. Lack of Awareness of I/O Load

Suppose that a specific vCPU simultaneously executes a mixed workload comprising both I/O and CPU workloads. When I/O handling occurs on that vCPU by receiving I/O respond event that indicates completion of the I/O request, it is likely that the vCPU will have a poor amount of credit because of the continuous handling of CPU-intensive workloads. Therefore, the I/O respond event cannot be immediately processed because the amount of credit is similar to that of the neighboring

vCPUs inside the run-queue. This situation leads to I/O latency in the run-queue until the amount of credit of the that vCPU is higher than that of the other vCPUs.

3.2. No Trade-Offs for Resource Sharing by Workload Type

Scheduling policies based on the degree of pCPU occupancy for fair resource usage can cause I/O performance degradation in I/O-intensive vCPUs with unpredictable I/O occurrences. To prevent such I/O performance deterioration, a method that guarantees pCPU occupancy for I/O-intensive vCPUs must be adopted. However, in a virtualization environment with limited resources, guaranteeing pCPU occupancy for some I/O-intensive vCPUs might decrease the performance of CPU-intensive vCPUs. Assume that the I/O-intensive vCPU1 and CPU-intensive vCPU2 have a total pCPU occupancy rate of 100%, corresponding to an individual pCPU occupancy rate of 50% each. If we change the pCPU occupancy rate of vCPU1 to 70% to enhance its I/O performance, then vCPU2 will always have a 30% pCPU occupancy rate per scheduling round, and the CPU performance will always be lower than it was prior to. Therefore, the trade-off relationship in real time for each of the vCPUs must be considered when determining which vCPU must be assigned a higher pCPU occupancy rate.

3.3. Meaningless Priority

I/O-intensive vCPUs consume less credit than CPU-intensive ones because of the frequent sleep states required to wait for the I/O response. If time has elapsed since vCPU scheduling began after credit reallocation, an I/O-intensive vCPU is more likely to occupy the pCPU than a CPU-intensive vCPU; this is because the I/O-intensive vCPU has a higher credit amount than the CPU-intensive vCPU. However, if the CPU-intensive vCPU frequently leads to credit reallocation, which reallocates credits to all vCPUs in the run-queue, the I/O-intensive vCPU is likely to have the same amount of credits as a CPU-intensive vCPU. Therefore, I/O-intensive vCPUs are more likely to be scheduled with the same priority as CPU-intensive vCPUs, resulting in I/O latency increases.

In virtualized environments, these causes of I/O degradation decrease the quality of service (e.g., response time, throughput, etc.) for network-intensive services (e.g., streaming-based audio/video services, web-based services, IoT services, etc.) and disk-intensive services (e.g., big data services, image processing services, etc.) [15,18,20,22,33]. To overcome these limitations of the existing virtualized environment, we introduce ISACS to address the aforementioned limitations of existing virtualized environments. ISACS assigns I/O virtual credits (IVCredits) to I/O-intensive vCPUs by measuring the I/O strength based on a per-vCPU unit to ensure the pCPU occupancy of I/O-intensive vCPUs. In addition, to achieve a trade-off between I/O performance and CPU performance, ISACS sets up a CPU-BOOST STATE to allocate CPU virtual credits (CVCredits) to prevent the deterioration in CPU performance. This guarantees the maximum CPU performance of CPU-intensive vCPUs while ensuring the pCPU occupancy of I/O-intensive vCPUs that are assigned IVCredit. Lastly, ISACS performs I/O load balancing to fairly balance the I/O load among cores. This study makes the following contributions to improving I/O performance in a virtualized environment:

- Providing fine-grained I/O strength awareness with respect to vCPUs;
- Ensuring good I/O performance on mixed workloads;
- Guaranteeing good CPU performance of CPU-intensive vCPUs; and
- I/O load balancing for run-queues.

4. Design and Implementation

4.1. Idea and Overview

The awareness of workloads per vCPU rather than per VM can provide a diverse scheduling environment that provides a chance to improve the I/O performance of VMs. ISACS recognizes the degree of I/O workloads for each VM's vCPU using the split-driver architecture in Xen without

leveraging the exterior monitoring module. Figure 2 shows the architecture of ISACS. In ISACS, EVENT CHECKER, shown in Figure 2, gathers the I/O information regarding the vCPUs that handle I/O-intensive workloads for all guest VMs. Then, using the information gathered by EVENT CHECKER, ISACS provides an I/O virtual credit (IVCredit) that is mapped to the degree of I/O workloads of I/O-intensive vCPUs to guarantee pCPU occupancy. To avoid the degradation of CPU performance in VMs, ISACS gives a CPU virtual credit (CVCredit) to CPU-intensive vCPUs to ensure CPU performance. In this section, we explain ISACS's design and structure with regard to guaranteeing the I/O performance of I/O-intensive vCPUs for various workloads in a virtualized environment.

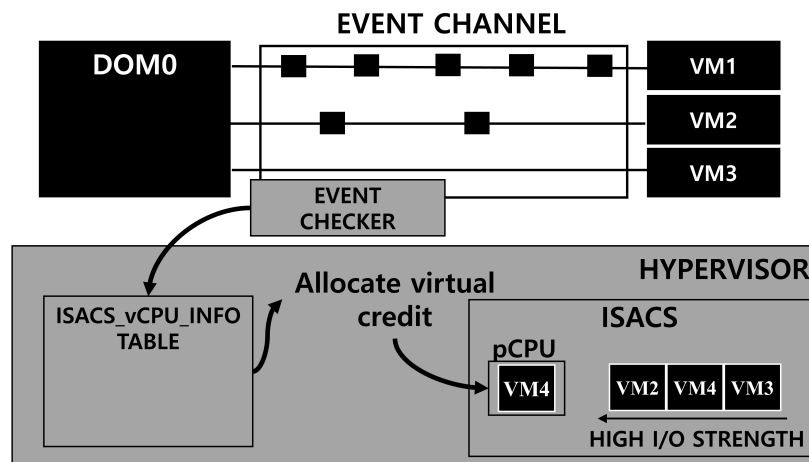


Figure 2. I/O strength-aware credit scheduler (ISACS) architecture.

4.1.1. Mapped vCPU Information of Three Areas

ISACS uses extended scheduling parameters that are not dependent on the exterior monitoring module that measures the resource usage of each VM. As the exterior monitoring module acts independently, it has to be synchronized with the scheduling parameter of the hypervisor scheduler. Moreover, DOM0 adds additional overload because of unnecessary information investigation and continuous load tracking of all VMs. Therefore, leveraging the exterior monitoring module creates an unnecessary overhead for resource management in a virtualized [9,26,34–37].

In the Credit1 scheduler, the degree of I/O load on the vCPU can be known easily through only the BOOST mechanism in the scheduling area. However, because there is no BOOST mechanism in the Credit2 scheduler, it is difficult to know the degree of I/O occurrence for the vCPU based on only the scheduler. To avoid the overhead arising from leveraging the exterior monitoring module and to determine I/O occurrence based on the Credit2 scheduler, ISACS leverages the event channel mechanism in Xen.

In the split-driver architecture of Xen, the asynchronous I/O notification mechanism based on the event channel and vCPU scheduling are independently executed over different areas; thus, the event channel cannot know the scheduling information for each vCPU, and the scheduler cannot know the number of I/O response events occurring for each vCPU [3,21,23,38]. Furthermore, credit reallocation for each run-queue in the Credit2 scheduler is independently performed. This is because it depends on the number of vCPUs waiting for scheduling and because the degree of credit consumption of each vCPU placed in the run-queue depends on the workload distribution trend of each vCPU. To integrate the up-to-date status information of vCPUs that can be obtained from these three independent areas and expand the range of scheduling parameters that can be derived, ISACS manages the extended vCPU scheduling structure ISACS_vCPU_INFO, which can allow bidirectional communication through shared memory communication. Figure 3 shows the structure of the scheduling parameter of ISACS. ISACS_vCPU_INFO in Figure 3 is configured based on the extended scheduling information of the vCPU. ISACS_vCPU_INFO consolidates and maintains the up-to-date information of the

vCPU regarding I/O response events, which can be obtained from the event channel through the shared memory mechanism and the run-queue information, which is that the vCPU is assigned. Thus, in ISACS, the event channel and scheduler can recognize each other's existence and keep up-to-date status of three areas including run-queue information.

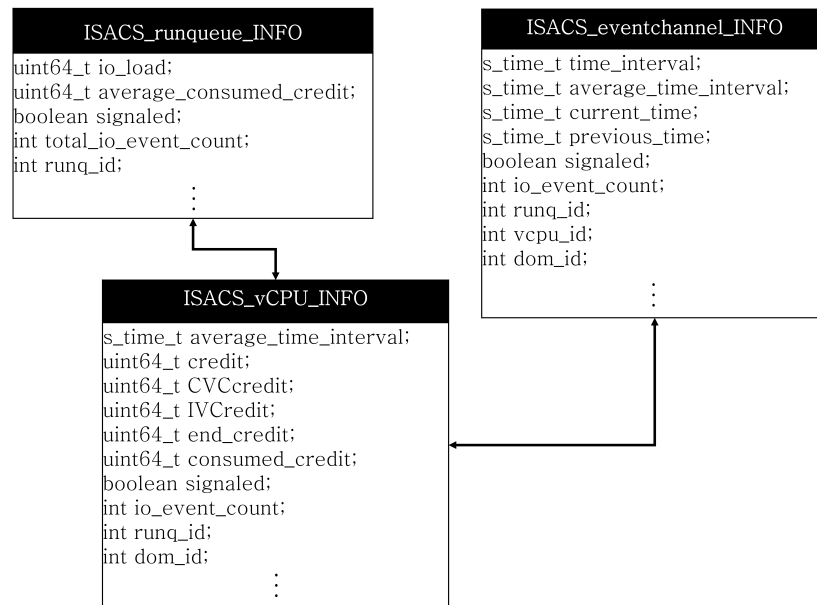


Figure 3. ISACS scheduling structure.

4.1.2. Finding a Good Degree of pCPU Occupancy for vCPU with Different Workloads

In the Credit2 scheduler, the amount of credits denotes the run-time of the vCPU on a pCPU and the priority of the vCPU for pCPU occupancy. Because the virtualized server has limited resources, if a specific vCPU preempts more pCPUs, it means that the vCPU seizes more resources than the other vCPUs. Therefore, we leverage and carefully select the additional scheduling parameters mentioned above to derive the virtual credits to be assigned to vCPUs. ISACS gives IVCredits to I/O-intensive vCPUs, giving them the opportunity to occupy pCPUs more frequently than CPU-intensive vCPUs. The key aspect of the ISACS model is to derive the degree of guaranteed pCPU occupancy according to the I/O strength of each vCPU in the run-queue. In addition, the pCPU occupancy of I/O-intensive vCPUs should not interfere with that of CPU-intensive vCPUs; this is to guarantee good CPU performance. Therefore, ISACS allocates CVCredits based on the amount of the IVCredit for all I/O-intensive vCPUs in the run-queue to prevent CPU performance degradation because of guaranteed pCPU occupancy of the I/O-intensive vCPUs. Thus, it can ensure the maximum performance of CPU-intensive vCPUs. Finally, ISACS calculates the I/O load for each run-queue and performs vCPU migration to prevent a biased I/O-intensive vCPU placement among the run-queues.

4.2. Recognizing I/O Strength For vCPUs

In a virtualized environment, a hypervisor cannot be aware of the workload type of guest VMs [25,30]. Therefore, ISACS obtains information from the event channel to know the degree of I/O strength for a specific vCPU. ISACS places EVENT CHECKER on the event channel to obtain information regarding I/O events, which is transferred from DOM0 to a specific vCPU of the guest VM. EVENT CHECKER tracks the particular event channel of a guest VM bound to DOM0's event channel and measures the number of I/O response events that DOM0 transfers to guest VMs and the average interval of the I/O response events to which a simple exponential moving average(EMA) is applied. When credit reallocation is invoked on one of the run-queues, ISACS sends a signal to the EVENT CHECKER along with the id of that run-queue to obtain I/O information regarding

the I/O response events and the average interval of the I/O response events for all vCPUs in that run-queue. When EVENT CHECKER receives this signal, it sends the I/O response event information of each vCPU in the run-queue through shared memory communication and reinitializes the I/O response event information of all vCPUs in that run-queue. EVENT CHECKER does not manage the vCPU status based on VM units; instead, it manages it based on individual vCPU units arranged in the run-queue; thus, it can provide more fine-grained I/O information regarding the degree of I/O occurrence for vCPUs.

4.3. IVCredit Derivation

An IVCredit is a virtual credit provided by ISACS to guarantee the pCPU occupation of an I/O-intensive vCPU. Each vCPU's I/O resource occupancy is determined based on the IVCredit allocated to it by considering I/O strength. Tracking the detailed I/O operations of each VM to determine its workload type requires modification of the real I/O driver in DOM0, making the hypervisor scheduler less portable. Analyzing the I/O request types of all VMs increases the memory overhead. In ISACS, to avoid the complicated process of recognizing the completion of I/O by modifying the physical driver of DOM0 that performs the actual I/O operation in a virtualized environment, ISACS obtains the I/O request processing time of the specific vCPU of the guest VM based on the interval of the I/O response event transmitted from DOM0. As explained earlier, DOM0 is a driver domain that can handle all I/O requests from all guest VMs. Therefore, the interval between I/O response events of vCPUs of the guest VM can be referred to as the processing time of DOM0 for I/O requests. To measure the I/O strength of a vCPU, ISACS obtains the number of I/O response events and the average interval between I/O response events for I/O-intensive vCPUs of the VM from the EVENT CHECKER.

Figure 4 shows the procedure of measuring the I/O strength of each vCPU of the VM in ISACS. The first dotted box at the top of Figure 4 depicts the perspective of only DOM0 processing the I/O requests of each VM. The boxes in the run-queue indicate the amount of time for which DOM0's vCPU has processed I/O requests received from a specific VM. The second dotted box at the top in Figure 4 shows an amount of accumulated time interval in which DOM0 generates an I/O response event for each VM's I/O request based on a split-driver mechanism. Here, the EVENT CHECKER tracks the vCPUs receiving the I/O response event and derive the average interval of the I/O response events with EMA applied. The third dotted box at the top indicates the process of allocating IVCredits according to the I/O strength by leveraging the ISACS_vCPU_INFO structure, which is synchronized with the EVENT CHECKER through shared memory communication. At this time, the VM box indicates the I/O-intensive vCPU of a specific VM which receives and handles the I/O response event from DOM0. As shown in Figure 4, the I/O strength of the vCPU is determined by the processing time of DOM0 for I/O requests of VMs, which has the same meaning for the interval between I/O response events from DOM0 to VM in ISACS. In Figure 4a,b represent the procedure corresponding to Equations (1) and (2) respectively.

$$\bar{I}_e = 0.9 \times I_e + (1 - 0.9) \times \bar{I}_{e-1}, \quad \text{where } t_e - t_{e-1} \text{ is } I_e \quad (1)$$

$$IVCredit = E_n \times \bar{I}_e \times \frac{\text{consumed credit}}{\frac{\sum_{i=1}^n cc_i}{n}} \quad (2)$$

Equations (1) and (2) represent the procedure of deriving IVCredit for each vCPU in the run-queue when credit reallocation is performed. For Equation (1), which entails the same procedure as that shown in Figure 4a, t_e denotes the arrival time of the e th I/O response event that vCPU received from DOM0; thus, $t_e - t_{e-1}$ denotes the time interval between the I/O response events and is expressed as I_e . Here, in order to avoid allocating IVCredits to the vCPUs with infrequent I/O, ISACS will not reflect I/O information for that vCPU when the value of $t_e - t_{e-1}$ exceeds 10 ms. Just ISACS initializes t_e to the current time for that vCPU. Therefore, vCPUs with infrequent I/O are allocated a relatively small

amount of IVCredit. For the average interval between I/O response events until credit reallocation is performed, the EMA with a smoothing constant of 0.9 is applied. By applying EMA to Equation (1), ISACS can apply the latest trend for the change in the interval between I/O response events to IVCredit based on the degree of the current I/O workload of I/O-intensive vCPUs.

In Equation (2), which has the same procedure as that shown in Figure 4b, E_n denotes the number of I/O response events that vCPU receives from DOM0 until credit reallocation is performed. Based on $E_n \times \bar{I}_e$, which multiplies the number of I/O events received from DOM0 for a vCPU and the average interval of I/O events after applying the EMA, we can derive the differentiated I/O strength of a specific vCPU among the vCPUs inside the run-queue. The value of $E_n \times \bar{I}_e$ for a CPU-intensive vCPU will be zero or a value smaller than that for an I/O-intensive vCPU because CPU-intensive vCPU has a low probability of receiving an I/O response event from DOM0 or generating an I/O request event to DOM0. Therefore, ISACS recognizes vCPUs with an IVCredit that is greater than zero as I/O-intensive vCPUs considering the value of $E_n \times \bar{I}_e$ in Equation (2). Then, to reflect the credit consumption for the actual workload of I/O-intensive vCPUs in the run-queue where vCPU is placed, $\frac{\text{consumed credit}}{\sum_i^n cc_i}$ is multiplied by the result of $E_n \times \bar{I}_e$. For $\frac{\text{consumed credit}}{\sum_i^n cc_i}$, *consumed credit* represents the amount of credit consumed before credit reallocation is performed for the vCPU to which the IVCredit will be allocated. n denotes the total number of vCPUs in the run-queue and cc_i means the *consumed credit* of each vCPU in the run-queue. Using Equations (1) and (2), IVCredit is provided to the I/O-intensive vCPUs inside each run-queue when credit reallocation is completed. After credit reallocation, the I/O-intensive vCPUs will have more credits than the CPU-intensive vCPUs. Thus, the I/O-intensive vCPUs have a higher probability of pCPU occupancy than the CPU-intensive vCPUs. However, the CPU performance may be degraded on CPU-intensive vCPUs because IVCredit is assigned to the I/O-intensive vCPUs. To avoid CPU performance degradation on the CPU-intensive vCPUs, ISACS provides CVCredits to vCPUs considering the degree of CPU-intensiveness. In the Xen architecture, CPU-intensive vCPUs have a low degree of I/O request events and I/O response events occurring owing to the I/O processes of VMs. Therefore, because the amount of IVCredit represents the I/O-intensiveness of vCPUs, the vCPUs with a relatively low I/O workload can be categorized based on their IVCredit amount. Equation (3) refers to the condition that defines the vCPU to which CVCredit should be allocated. In Equation (3), $IVCredit_i$ is the amount of IVCredit allocated to $vCPU_i$, which is placed in the run-queue. n denotes the total number of vCPUs in the run-queue and k represents the number of vCPUs that the amount of IVCredit greater than zero in the run-queue. $IVCredit_c$ represents the amount of IVCredits allocated through Equation (2) for the vCPU to which Equation (3) applies.

$$\frac{\sum_{i=1}^n IVCredit_i}{k} \times \frac{1}{IVCredit_c + 1} > 1 \quad (3)$$

$$CVCredit = \frac{\sum_{i=1}^n IVCredit_i}{j} \times \frac{\text{consumed credit}}{\text{default credit}} \quad (4)$$

Regarding Equation (3), a vCPU with a relatively high amount of IVCredit in the run-queue will have a value less than 1; thus, CVCredit will not be assigned to such a vCPU by ISACS. After ISACS judges all vCPUs in the run-queue and decides whether to assign CVCredit or not, it places the vCPUs satisfying Equation (3) in the CPU-BOOST GROUP of specific run-queue. The CPU-BOOST GROUP is the pool of CPU-intensive vCPUs that satisfy Equation (3) and is managed by a run-queue unit. After ISACS configures the CPU-BOOST GROUP, it calculates the amount of CVCredit to be given to all vCPUs that belong to the CPU-BOOST GROUP using Equation (4). In Equation (4), j denotes the number of vCPUs that belong to the CPU-BOOST GROUP, and n is the total number of vCPUs in the run-queue. $\frac{\sum_i^n IVCredit_i}{j}$ represents amount of CVCredits be allocated to the CPU-intensive vCPU inside the run-queue based on total amounts of IVCredits for all I/O-intensive vCPUs. Here, ISACS can guarantee pCPU occupancy for CPU-intensive vCPUs based on the amount of actual pCPU run-time for vCPU of DOM0 to handle the I/O request from I/O-intensive VMs. Then, $\frac{\text{consumed credit}}{\text{default credit}}$ is

multiplied to provide a differentiated CVCredit according to the actual degree of CPU workload for the target vCPU. CVCredit is assigned to all vCPUs belonging to the CPU-BOOST GROUP after IVCredit has been assigned to the vCPUs. When the next vCPU to be scheduled satisfies the credit reallocation condition of ISACS, it enters the CPU-BOOST STATE, where only the vCPUs in the CPU-BOOST GROUP are scheduled. While CVCredit is consumed in the CPU-BOOST STATE, if the CVCredit of the next scheduled vCPU becomes less than zero, credit reallocation occurs for all vCPUs in the run-queue.

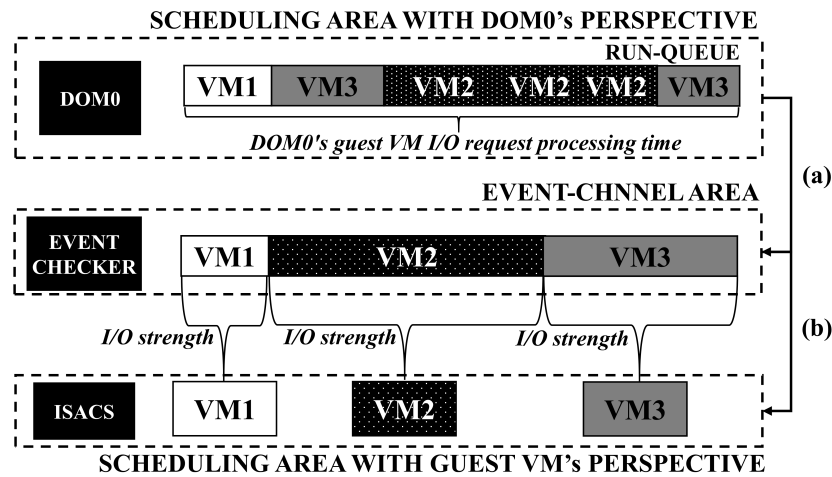


Figure 4. ISACS vCPU I/O strength measurement procedure.

4.4. Credit Concealing Method

In the previous section, we described equations to derive the amount of virtual credits, such as IVCredit and CVCredit, according to the degree of workload and workload type of the vCPU unit. However, a vCPU that is assigned an IVCredit can have more credits than a neighbor vCPU. Therefore, a fair pCPU run-time for each vCPU is not guaranteed, and this can lead to biases in I/O and CPU performance. For example, when allocating IVCredits to vCPUs, I/O-intensive vCPUs obtain more credits than the CPU-intensive ones, resulting in unfair pCPU run-time. Therefore, ISACS should ensure that all vCPUs in the run-queue have the same amount of credit as much as possible. To avoid unfair pCPU run-time over all vCPUs in the run-queue, ISACS adopts a credit concealing method for all vCPUs to allocate a fair amount of credit for all vCPUs in the run-queue. In this section, we explain the credit concealing method that mitigates the unfairness arising from the amount of credit allocated to each vCPU placed in the same run-queue. This method provides a fair pCPU run-time for each vCPU by defining the limit of the amount of credit that can be consumed by considering the type of virtual credit.

The Credit2 scheduler first schedules the vCPU that has a high amount of credits. ISACS leverages this scheduling policy to increase the probability of preempting the pCPU for I/O-intensive vCPUs using a credit concealing method. Figure 5 shows the result of the credit concealing method after credit reallocation. The concealing area refers to the virtual credit provided by ISACS. For concealing area in the Figure 5, vCPU1 and vCPU2 are I/O-intensive vCPUs and have an IVCredit granted by ISACS. vCPU4 and vCPU5 are CPU-intensive vCPUs and are allocated a CVCredit by ISACS to guarantee the CPU performance. vCPU3, unlike vCPU1 and vCPU2, meets Equation (3) for the average IVCredit in the run-queue. Therefore, IVCredit and CVCredit together represent the assigned state. In the Credit2 scheduler, all vCPUs inside the run-queue are assigned a default credit when the credit amount for the next scheduled vCPU is less than zero. Therefore, for the credit reallocation condition in the Credit2 scheduler, we can state that $default\ credit - consumed\ credit < 0$. When IVCredit is assigned to the I/O-intensive vCPU, we get $(default\ credit + IVCredit) - consumed\ credit < 0$. However, if the credit reallocation condition is less than IVCredit and not zero, then $(default\ credit + IVCredit) - consumed\ credit < IVCredit$. IVCredit is erased from both sides of the credit reallocation condition;

thus, $default\ credit - consumed\ credit < 0$. Therefore, in ISACS, all vCPUs in the run-queue takes the same amount of credits whenever credit reallocation is performed, regardless of the type of workload that each vCPU handles when ISACS is not in CPU-BOOST STATE. ISACS guarantees that the same amount of credit can be assigned to all vCPUs in the run-queue by ensuring that credit reallocation is performed when the amount of credit is less than the value of $vCPU.endcredit$ in Figure 5, not when the amount of credit is less than zero.

Figure 6 shows the order of vCPU placement in the run-queue when credit reallocation is completed and when the CPU-BOOST STATE occurs based on Figure 5. As shown in Figure 5, ISACS schedules a vCPU with a high IVCredit first after credits are reallocated in a specific run-queue. If the next vCPU to be scheduled has less credit than $vCPU.endcredit$, ISACS enters a CPU-BOOST STATE and schedules the vCPU with the highest CVCredit first. Finally, in the CPU-BOOST STATE, when the CVCredit amount of the next vCPU to be scheduled is less than zero, credit reallocation is performed for all vCPUs within the run-queue. Algorithm 1 shows the entire scheduling procedure of ISACS.

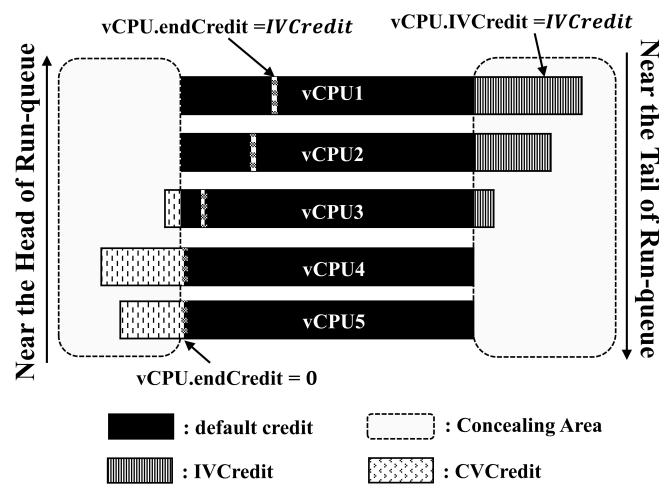


Figure 5. ISACS credit concealing method. IVCredit, I/O virtual credit; CVCredit, CPU virtual credit.

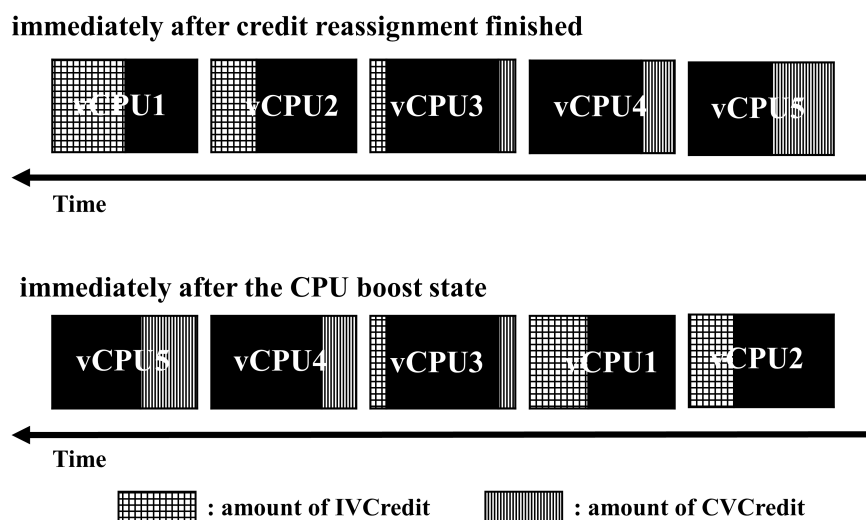


Figure 6. ISACS run-queue ordering.

Algorithm 1 ISACS: scheduling algorithm.

$vCPU_c$: Current vCPU on the pCPU
 $vCPU_n$: Next vCPU to Run

```

1: if CPU – BOOST STATE == FALSE then
2:   Burn the  $vCPU_c.credit$ 
3:   Set vCPU with highest credits for all vCPUs in the RunQ as  $vCPU_n$ 
4:   if  $vCPU_n.credit \leq vCPU_n.endcredit$  then
5:     CPU – BOOST STATE  $\leftarrow$  TRUE
6:     Set vCPU with highest CVCredits over all vCPUs in CPU – BOOST GROUP as  $vCPU_c$ 
7:   end if
8: end if
9: if CPU – BOOST STATE == TRUE then
10:  Burn the  $vCPU_c.CVCredit$ 
11:  Set vCPU with highest CVCredits over all vCPUs in CPU – BOOST GROUP as  $vCPU_n$ 
12:  if  $vCPU_n.credit \leq vCPU_i.credit$  then
13:    RunQ.IOload  $\leftarrow$  0
14:    Init the member of CPU – BOOST GROUP
15:    CPU – BOOST STATE  $\leftarrow$  FALSE
16:    for  $vCPU_i \in$  RunQ do
17:      Derive IVCredit for  $vCPU_i$  using Equation (1) and Equation (2)
18:       $vCPU_i.credit \leftarrow$  default credit + IVCredit
19:       $vCPU_i.endcredit \leftarrow$  IVCredit
20:      RunQ.IOload+ = IVCredit
21:    end for
22:    for  $vCPU_i \in$  RunQ do
23:      if Equation (3) is satisfied then
24:        Join  $vCPU_i$  to the CPU – BOOST GROUP
25:         $vCPU_i.CVCredit \leftarrow$  CVCredit which is calculated by using Equation (4)
26:      end if
27:    end for
28:  end if
29: end if
30: return  $vCPU_n$ 

```

4.5. I/O Load Balancing

The Credit2 scheduler has a limitation of performing load balancing according to the degree of I/O load on each run-queue. This is because, in the Credit2 scheduler, vCPUs are migrated considering the run-queue load, which considers only the number of vCPUs and the execution time of each vCPU in the run-queue. In ISACS, we add a very simple load-balancing mechanism with considering I/O load of each run-queue after credit reallocation for all vCPUs in the run-queue is finished. When migrating vCPUs to prevent I/O workload concentration to a specific core, the degree of I/O load of the run-queue to which a vCPU is migrated should be carefully considered.

ISACS employs a migration condition based on the total number of I/O events received from all vCPUs in the run-queue and the degree of I/O load of the run-queue. To explain the migration conditions of ISACS, we define several terms. $RunQ_s.d.IOevent$ is the total number of I/O events received by all vCPUs in the run-queue, and $vCPU_m$ denotes the vCPU to be migrated. $RunQ_d$ denotes the run-queue to which $vCPU_m$ will be migrated, and $RunQ_s$ denotes the run-queue that performs credit reallocation. The condition for selecting $RunQ_d$ is $RunQ_i.IOevent < RunQ_s.IOevent$ where $RunQ_i$ is the candidate of the $RunQ_d$ excluding $RunQ_s$. Once $RunQ_d$ is selected, to prevent it from having an increased I/O load among run-queues, ISACS finds the vCPU which meets the following conditions: $RunQ_d.IOload + vCPU.IVCredit < RunQ_s.IOload - vCPU.IVCredit$. When ISACS finds a vCPU that meets the above condition, it sets the vCPU as $vCPU_m$ and migrates it to $RunQ_d$. Algorithm 2 shows the I/O load balancing performed by ISACS.

Algorithm 2 ISACS: I/O load-balancing.

```

1: for  $RunQ_i \in RunQ$  do
2:   if  $RunQ_i.id == RunQ_s.id$  then
3:     continue
4:   end if
5:   if  $RunQ_i.IOevent < RunQ_s.IOevent$  then
6:      $RunQ_d \leftarrow RunQ_i$ 
7:   end if
8: end for
9: for  $vCPU_j \in RunQ_s$  do
10:  if  $RunQ_d.IOload + vCPU_j.IVCredit < RunQ_s.IOload - vCPU_j.IVCredit$  then
11:     $vCPU_m \leftarrow vCPU_j$ 
12:  end if
13: end for
14: if  $vCPU_m \neq \text{NULL}$  then
15:   Migrate  $vCPU_m$  to  $RunQ_d$ 
16:   Return
17: end if
18: Return

```

5. ISACS: Performance Evaluation

In this section, we evaluate the I/O performance of ISACS by comparing it with the existing virtualized environment in which the Credit2 scheduler is adopted. The reason we compared ISACS only with the Credit2 scheduler is because ISACS aims to increase the I/O performance of the credit-based general-purpose hypervisor scheduler and is designed based on the Credit2 scheduler. Therefore, for the experiments in Section 5, we do not compare ISACS with the Credit1 scheduler because Credit1 scheduler has a relatively low I/O performance compared to the Credit2 scheduler [8,14,33,39,40].

In the experiments, we highlight the following: (1) the effects of ISACS on network performance; (2) the effects of ISACS on disk performance; (3) the effects of ISACS on the degree of CPU performance interference over neighboring VMs. Based on the VM workload type, we designate VMs with a CPU-intensive workload as CPU-intensive VMs, those with an I/O-intensive workload as network- or disk-intensive VMs, and those with I/O- and CPU-intensive workloads as mixed-intensive VMs. We define the execution environment for VMs as a hybrid environment which comprises I/O-intensive VMs and CPU-intensive VMs and a mixed environment that only consists of mixed-intensive VMs. Lastly, we define the simple expressions about VM configuration in the experiments. ‘c’ represents a CPU-intensive VM, ‘i’ represents an I/O-intensive VM. For example, “i2c2” means that two CPU-intensive VMs, and two I/O-intensive VMs perform simultaneously in the hybrid environment. ‘m’ means a mixed-intensive VM and “m2” means the two mixed-intensive VMs perform simultaneously in the mixed environment. The experimental environment is shown in Table 1. For resource configuration of VMs, all vCPUs of DOM0 are pinned to pCPU0 and pCPU1 for isolating the performance of DOM0 with respect to the guest VMs [2,9,41].

Table 1. Experiment environment.

CPU	Intel core i9-9900 CPU@3.6Ghz, 8 cores without hyper-threading technique
RAM	DDR4 32GB
OS	Ubuntu 18.04 for all VMs, including DOM0
Hypervisor	Xen 4.12
Guest VM	4 vCPUs with 2048MB of RAM
DOM0	6 vCPUs, 8096MB of RAM, all vCPUs pinned to pCPU0 and pCPU1

For the network-related experiment, we measured the internal network performance of the virtualized environment. The reason for not using external physical machines to measure network performance is that inter-domain communication does not require external networks, and changes in the I/O performance in the virtualized environment can be clearly identified by eliminating external network latency arising from communication with external physical machines. In addition, the purpose of this study is to improve the overall I/O performance in a virtualized environment by increasing the I/O performance of inter-domain communication; thus, simply measuring the internal network performance of the physical machine is sufficient to evaluate our approach. We use a *CLIENT – VM* that acts as a client, and all vCPUs of the *CLIENT – VM* are pinned to pCPU7 to prevent performance interference from guest VMs. All vCPUs of the guest VMs that act as a server are pinned from pCPU2 to pCPU6. Thus, for the network performance experiment, five cores are allocated for all guest VMs, two cores are allocated to DOM0, and one core is allocated to the *CLIENT – VM*. For the disk-related experiment, all vCPUs of the guest VM are pinned from pCPU2 to pCPU7. Thus, six cores are allocated for all guest VMs, and two cores are allocated for DOM0. Further, DOM0 has six vCPUs and guest VMs, including *CLIENT – VM*, have four vCPUs.

Table 2 presents the benchmark tool used to evaluate the performance of ISACS. To evaluate the network performance of ISACS, we leverage Iperf and Ping, and to evaluate the disk performance of ISACS, we leverage DD. Stress is used to generate a CPU-intensive workload on a VM in the background process. Finally, we employ Sysbench to evaluate the CPU performance of the VM based on the execution time of the Sysbench command.

Table 2. Benchmark tool.

Iperf [42]	Network performance test for network I/O throughput measurement over TCP/IP communication from CLIENT-VM to Guest VM
Ping [43]	Network performance test for network latency measurement over TCP/IP communication from CLIENT-VM to Guest VM
DD [44]	Disk performance test for disk I/O bandwidth measurement
Stress [45]	Generating CPU workload on VM and performed in the background
Sysbench [46]	CPU performance test for process execution time

5.1. ISACS Impacts on Network Latency

In this experiment, we measure the extent to which the network latency is decreased by ISACS compared to the existing environment. For the VM configuration, for a total of 13 VMs, twelve guest VMs running as a server are pinned to pCPU2–pCPU6 and the *CLIENT – VM* is pinned to pCPU7. To measure the network latency, we leverage Ping, which is a computer network management software utility provided by Linux. For CPU-intensive workloads, we leverage Stress and configure Stress using four threads so that all vCPUs of CPU-intensive VMs or mixed-intensive VMs handle CPU-intensive workloads.

5.1.1. Effect of ISACS on Hybrid Environment

In this experiment, we confirm the extent to which ISACS reduces the network latency in a hybrid environment by ensuring pCPU occupancy according to the I/O strength of the vCPU. The experimental environment consisted of 12 guest VMs. In addition, to measure the actual network I/O latency while handling the network-intensive workload of the guest VMs, we perform 1000 Ping tests while the *CLIENT – VM* executes Iperf, which generates 1024 KB of streaming data to network-intensive VM. The rest of the VMs, except the network-intensive VMs, handle a CPU-intensive workload, based on the Stress command line utility, using four threads. For all experiments, we conducted a total of 1000 Ping tests; however, we only show the results for 500 Ping

tests, excluding 25% of the Ping test results obtained immediately after the start of the Ping test and 25% obtained before the end of the Ping test.

Figure 7 shows the distribution of network latency for the Ping test when one network-intensive VM to three network-intensive VMs are executed. The experimental results show that ISACS generally has a lower network latency than the existing environment. In Figure 7a, the tail latency emerges in the existing environment including Figure 7c. However, in ISACS, a small tail latency exists on n Figure 7c and the maximum tail latency at VM3 is 1.68 ms; in contrast, in the existing environment, the maximum tail latency is 5.87 ms on VM3. Table 3 shows the average network latency according to the VM configuration in the hybrid environment based on Figure 7. The network latency in ISACS is approximately 43% lower than that in the existing environment when running 1 VM, and it is 55% lower when 3 VMs are employed. However, when running 2 VMs, a small difference exists in network latency between ISACS and the existing environment.

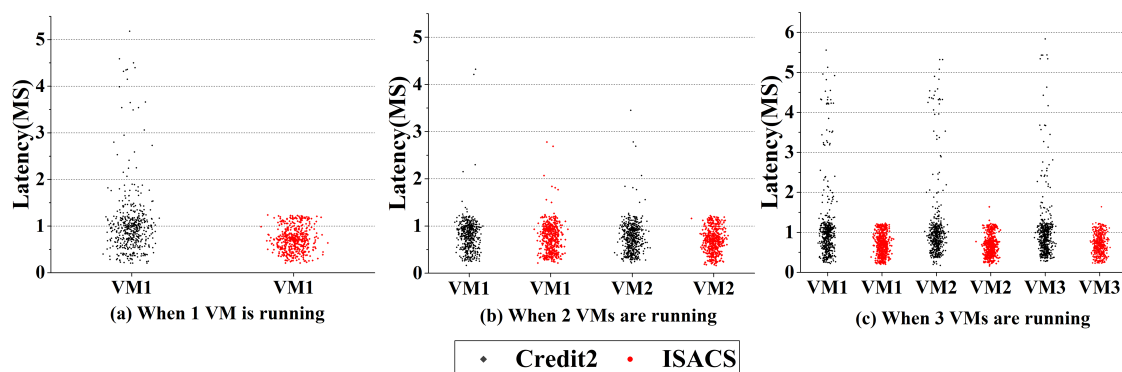


Figure 7. Network latency distribution between 25% and 75% for the result of 1000 experiments per experiment rounds according to the VM configuration in a hybrid environment.

Table 3. Average of network latency according to the VM configuration based on Figure 7.

Hybrid Environment	Average Network Latency (MS)		
	1 VM	2 VMs	3 VMs
Credit2	1.02	0.88	1.07
ISACS	0.71	0.72	0.69

5.1.2. Effect of ISACS on Mixed Environment

In this experiment, we measured the network latency of the existing environment and ISACS when six VMs were run in a mixed environment. Thus, twenty-four vCPUs are allocated to five pCPUs, and all VMs in this experiment undergo 2000 Ping tests while performing CPU-intensive tasks. Figure 8 shows the network latency distribution for 1000 Ping tests, which are range from 25% to 75% for a total of 2000 Ping tests for six VMs. The results of the experiment show that the network latency of ISACS is more densely distributed, in the range of 0 ms to 6 ms, than that of the existing environment. In contrast, the network latency in the existing environment, more than 6 ms, is higher than that of ISACS. For all VMs, the longest tail latency is 19.87 ms in the existing environment and 11.27 ms in ISACS, which is shorter than that in the existing environment.

Table 4 shows the average network latency of each VM for the Ping test results. The highest average of the network latency in an existing environment is 6.12 ms for VM5, and the lowest is 3.58 ms for VM1. ISACS, in contrast, shows the lowest average network latency of 3.19 ms for VM1, and VM3 has the highest average network latency of 3.49 ms. The average network latency of all VMs is 4.51 ms in the existing environment, and in ISACS, its value is 3.36 ms, which indicates that the average network latency is approximately 34% lower than that of the existing environment. The experimental results confirm that ISACS ensures immediate pCPU occupancy of vCPUs by recognizing the I/O strength. The existing environment has no awareness of the I/O-intensiveness of vCPUs; therefore,

when credit reallocation occurs, they are scheduled with the same priority as that of the vCPUs performing other workload types, resulting in tail latency. However, ISASC recognizes the I/O strength of vCPUs and provides a clear priority through IVCredit allocation, ensuring the greater pCPU occupancy of vCPUs that are more I/O-intensive, allowing them to have a shorter tail latency. The results show that ISACS provides lower network latencies to latency-sensitive VMs compared to the existing environment.

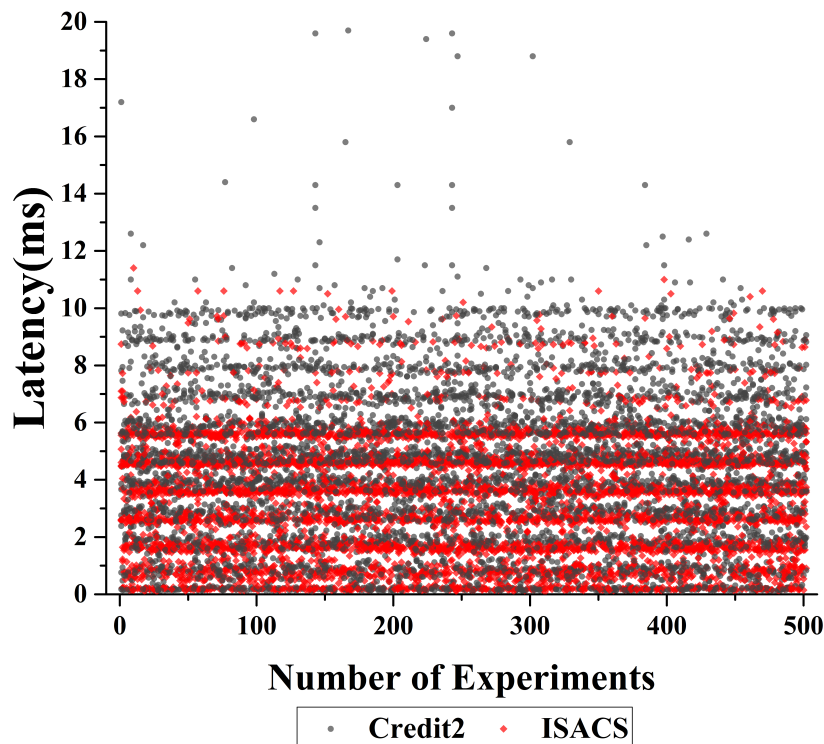


Figure 8. Network latency distribution between 25% and 75% for the results of 2000 experiments with six VMs in a mixed environment.

Table 4. Average network latency according to the VM configuration based on Figure 8.

Mixed Environment	Average Network Latency (MS)						Average
	VM1	VM2	VM3	VM4	VM5	VM6	
Credit2	3.58	4.54	5.08	3.95	6.12	3.83	4.51
ISACS	3.19	3.31	3.49	3.27	3.48	3.41	3.36

5.2. ISACS's Impacts on Network Throughput

In this study, we identify the degree of improvement in network performance provided by ISACS compared with that provided by the existing environment. Regarding the VM configurations, for a total of 13 VMs, twelve VMs are run as a server and are pinned from pCPU2 to pCPU6, and *CLIENT – VM* is pinned to pCPU7. To measure the network throughput of ISACS, we leverage Iperf and *CLIENT – VM* to perform TCP/IP communication with all guest VMs for 60 seconds using 1024 KB streaming packets. We measured the total throughput for all network-intensive VMs and mixed-intensive VMs. For the CPU-intensive workload, we leveraged the Stress with the four threads configuration. All experiments were conducted 15 times according to the VM configurations.

5.2.1. Effect of ISACS in the Hybrid Environment

In this experiment, we confirmed the effects of ISACS on network throughput in a hybrid environment. At the beginning of the experiments, 11 network-intensive VMs and a CPU-intensive

VM (i11c1) were executed. We gradually decreased the number to two network-intensive VMs and increased the number to two CPU-intensive VMs after 15 rounds of measuring the network throughput on each scheduler. At the final stage of the experiments, a network-intensive VM and 11 CPU-intensive VMs (i1c11) were executed. Figure 9 and Table 5 represent the distribution of the network throughput and average network throughput for 15 rounds of each experiment with different VM configurations. For “i11c1”, the averages of the throughput of the existing environment and of ISACS for the entire round are 153.1 GB and 154.1 GB, respectively. There is no difference between the two schedulers at “i11c1”. This is because the number of CPU-intensive vCPUs is too small for such vCPUs to effectively compete with the I/O-intensive vCPUs for the pCPUs. However, for “i9c3”, the average throughput of ISACS is 175.5 GB, which is 22% higher than that of the existing environment, which is 135.5 GB. For “i7c5” and “i5c7”, ISACS has approximately 10.6% and 10.62% higher average network throughputs than the existing environment, respectively. Finally, for “i1c11”, the existing environment has 19.3 GB of throughput and ISACS has 23.7 GB of throughput, which is 22% higher.

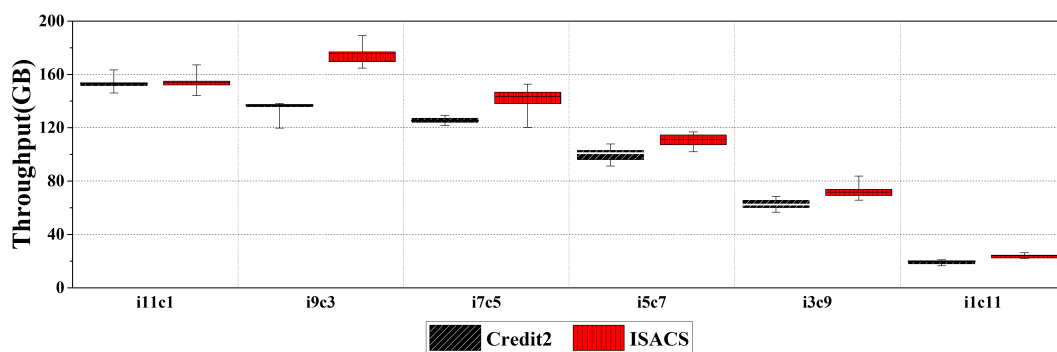


Figure 9. Distribution of the network throughput of each scheduler for 15 iterations of each experiment with a different VM configuration in the hybrid environment.

Table 5. Average network latency according to the VM configuration based on Figure 9.

Hybrid Environment	Average Network Throughput (GB)					
	i11c1	i3c9	i5c7	i7c5	i9c3	i11c1
Credit2	19.3	62.7	99.8	125.6	135.5	153.1
ISACS	23.7	72.1	110.4	140.5	175.5	154.1

In the results for ISACS, we can observe that the network throughput significantly decreases from “i5c7”. This is because credit reallocation occurs relatively more frequently as the number of CPU-intensive vCPUs for the entire run-queue increases compared to when the I/O-intensive vCPUs increase in number. Therefore, owing to the frequent credit reallocation, ISACS has an insufficient period for measuring the I/O strength to define the I/O strength of the vCPUs using the EVENT CHECKER. However, from “i9c3” to “i1c11”, we confirm that ISACS has a higher network throughput than the existing environment. Thus, we can state that ISACS has better network performance in a hybrid environment than that in the existing environment.

5.2.2. Effect of ISACS in the Mixed Environment

Experiments were conducted to illustrate the network performance of ISACS in a mixed environment compared with that in the existing environment. All VMs perform mixed workloads with Iperf and Stress running simultaneously. The first round of the experiment starts with one VM (m1), and the number of VMs is gradually increased by two when two VMs (m2) are running until 12 VMs (m12) are running. Figure 10 and Table 6 represent the distribution of the network throughput and average of the network throughput of each scheduler for 15 iterations of each experiment with different VM configurations.

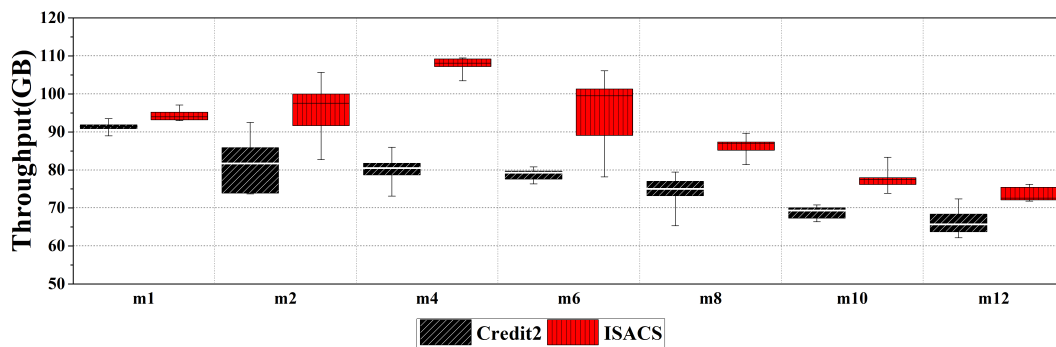


Figure 10. Distribution of the network throughput of each scheduler for 15 iterations of each experiment according to the VM configurations in a mixed environment.

Table 6. Average network throughput according to the VM configuration based on Figure 10.

Mixed Environment	Average Network Throughput (GB)						
	m1	m2	m4	m6	m8	m10	m12
Credit2	85.3	81.4	80.3	78.2	74.7	68.5	66.2
ISACS	94.2	95.5	107.4	96.6	86.4	77.3	73.4

The results show that ISACS has a more fine-grained I/O strength and provides a higher network throughput than the existing environment. In “m4” of Figure 10, ISACS has a network throughput of 107.4 GB, which is 22.79% higher than that of the existing environment with a network throughput of 80.3 GB. Furthermore, in “m12” with the most CPU-intensive vCPUs assigned, the existing environment and ISACS have an average network throughput of 73.4 GB and 66.2 GB, respectively, indicating that ISACS has approximately a 9.8% higher throughput than the existing environment. ISACS’s network throughput in this mixed environment is higher than that of the existing environment because vCPUs that handle mixed workloads have higher credits with the same amount of credits than other vCPUs that handle CPU-intensive workloads through ISACS’s credit concealment method. By doing so, ISACS can better detect the I/O strength of vCPUs processing I/O and can ensure their pCPU occupancy. Thus, ISACS provides higher network performance than existing environments in a mixed environment.

5.3. ISACS Impacts on Disk Bandwidth

To evaluate the disk performance of ISACS, we leverage DD command which is a command tool that can copy and convert files in a block unit. We configure read/write DD with a block size of 1024 bytes for /dev/zero and /dev/urandom, which are special files provided by the Linux system, resulting in a file size of 5 GB. Note that /dev/zero always returns bytes containing zero (‘\0’ characters) in a streaming form. Furthermore, /dev/urandom returns a non-blocking pseudo-random number from the entropy pool configured by the system, and /dev/urandom typically has a lower disk I/O bandwidth than /dev/zero because of the process in which the entropy pool is formed and the system calculation time for generating pseudo-random numbers. For the CPU-intensive workload, we leverage Stress configured with four threads. In this experiment, vCPUs of guest VMs are pinned from pCPU2 to pCPU7 except for the vCPUs of DOM0, and a maximum number of 48 vCPUs can be allocated to five pCPUs. Similar to the above experiments, all experiments were performed 15 times, and the disk performance of ISACS in the hybrid environment and in the mixed environment was compared with that in the existing environment.

5.3.1. Effect of ISACS in the Hybrid Environment

Figure 11 shows the experimental results for the disk bandwidth for each experimental round according to the VM configuration for ISACS and the existing environment. Table 7 presents the average bandwidth for 15 rounds of experiments according to the VM configurations for ISACS and the existing environment. The results of the experiment confirmed that ISACS has a high disk bandwidth in a hybrid environment than in the existing environment. The largest difference in bandwidth between ISACS and the existing environment is “i9c3”, as shown in Figure 11b. The average bandwidth for (b) in Figure 11 in the existing environment is 687 MB and 548.4 MB, respectively, and its values in ISACS are 768.4 MB and 602.3 MB, respectively, which increases the disk bandwidth for /dev/zero and /dev/urandom by 10.2% and 8.9%, respectively, compared with that in the existing environment. The least disk bandwidth improvement of ISACS is for “i1c11” in Figure 11f. For /dev/zero and /dev/urandom, these values in the existing environment are 285.4 MB and 28.2 MB, and in ISACS, the values are 298.4 and 29.3 MB, which denote an improvement of 4.3% and 3.7% respectively.

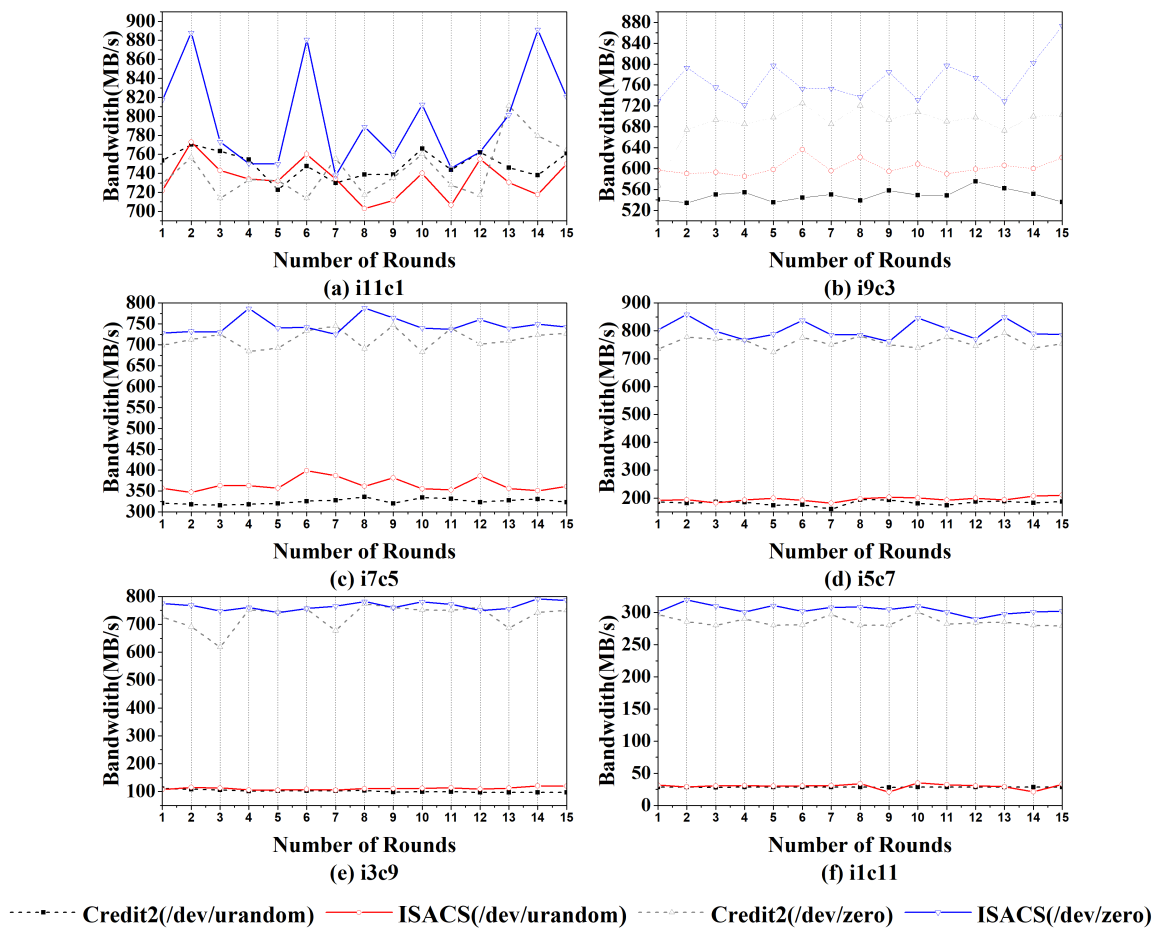


Figure 11. Disk bandwidth in 15 experiments with a different VM configuration in the hybrid environment.

In the experimental results, /dev/urandom has a lower degree of disk I/O bandwidth improvement than /dev/zero over the entire experiment. The reason is that /dev/zero has a relatively short disk I/O processing time; therefore, DOM0's I/O response event period can occur immediately. However, in the case of /dev/urandom, the entropy of the actual system is collected, and then, a random number is generated based on that entropy information. Therefore, the disk I/O processing time is considerably longer than that for /dev/zero. Therefore, the frequency of occurrence of I/O response events on DOM0 is less than that of /dev/zero. This means that ISACS does not collect enough I/O information for all vCPUs until credit reallocation occurs on a specific run-queue, so the

recognition of vCPU I/O strength becomes weaker. However, according to the overall experimental results, the disk bandwidth of ISACS was higher than that of the existing environment. It can be said that ISACS can improve disk I/O performance in a hybrid environment.

Table 7. Average disk bandwidth according to the VM configuration based on Figure 11.

Hybrid Environment	Average Disk Bandwidth (MB)					
	ic11	i3c9	i5c7	i7c5	i9c3	i11c1
Credit2 (/dev/urandom)	28.2	101.1	182.3	325.3	548.4	715.3
ISACS (/dev/urandom)	29.3	109	195.2	365.9	602.3	749.2
Credit2 (/dev/zero)	285.4	729.5	758.5	714.5	687	734.4
ISACS (/dev/zero)	298.4	765.8	802.1	747.5	768.4	798

5.3.2. Effect of ISACS in the Mixed Environment

Figure 12 shows the disk I/O bandwidth results for each round of the experiment, and Table 8 presents the average disk I/O bandwidth for ISACS and the existing environment, which is also shown in Figure 12. The results of the experiment show that the overall degree of the disk I/O bandwidth of ISACS is higher than that of the existing environment. However, the degree of the disk I/O bandwidth for /dev/urandom was not significantly high. The reason for the low degree of disk I/O bandwidth improvement in /dev/urandom is that the processing cost for random number generation in /dev/urandom, as discussed in Section 5.3.1, is high, resulting in a lack of I/O response event information, which weakens the I/O intensity awareness of ISACS. Therefore, we will not discuss the results of the DD experiment for /dev/urandom herein because the increase in disk bandwidth is too low.

For the results of the DD test on /dev/zero, the VM configuration with the highest disk bandwidth increase rate is shown as “m4” in Figure 12c; the average bandwidth of the existing environment is approximately 738.2 MB, and the average bandwidth of ISACS is approximately 812.4 MB, which is 9.1% higher than that of the existing environment. In contrast, the VM configuration with the lowest average disk bandwidth increase rate for ISACS is “m2” in Figure 12b, and the corresponding value is 873.7MB, which is just 1.3% higher than compared with that of the existing environment. The reason for the low disk bandwidth increase rate is that the performance interference between vCPUs does not occur because the number of pCPUs is larger than that of vCPUs. From “m12” in Figure 12g, we can observe that the disk bandwidth improvement rate of ISACS is low. This is because as the number of VMs increases, the number of vCPUs also increases, resulting in a lack of pCPU resources. However, when performing the DD test for /dev/zero in Figure 12, the overall disk bandwidth is increased according to each VM configuration. Therefore, ISACS can improve disk I/O performance in a mixed environment.

Table 8. Average disk bandwidth according to the VM configuration based on Figure 12.

Mixed Environment	Average Disk Bandwidth (MB)						
	m1	m2	m4	m6	m8	m10	m12
Credit2 (/dev/urandom)	219.2	291.5	319	323.9	326.5	320.3	330.4
ISACS (/dev/urandom)	219.1	291.5	326.4	332.6	332.1	336.7	336.2
Credit2 (/dev/zero)	1035.3	861.4	738.2	752.7	716.5	743.1	746.2
ISACS (/dev/zero)	1068.7	873	812.4	777.8	752.9	797.4	773.9

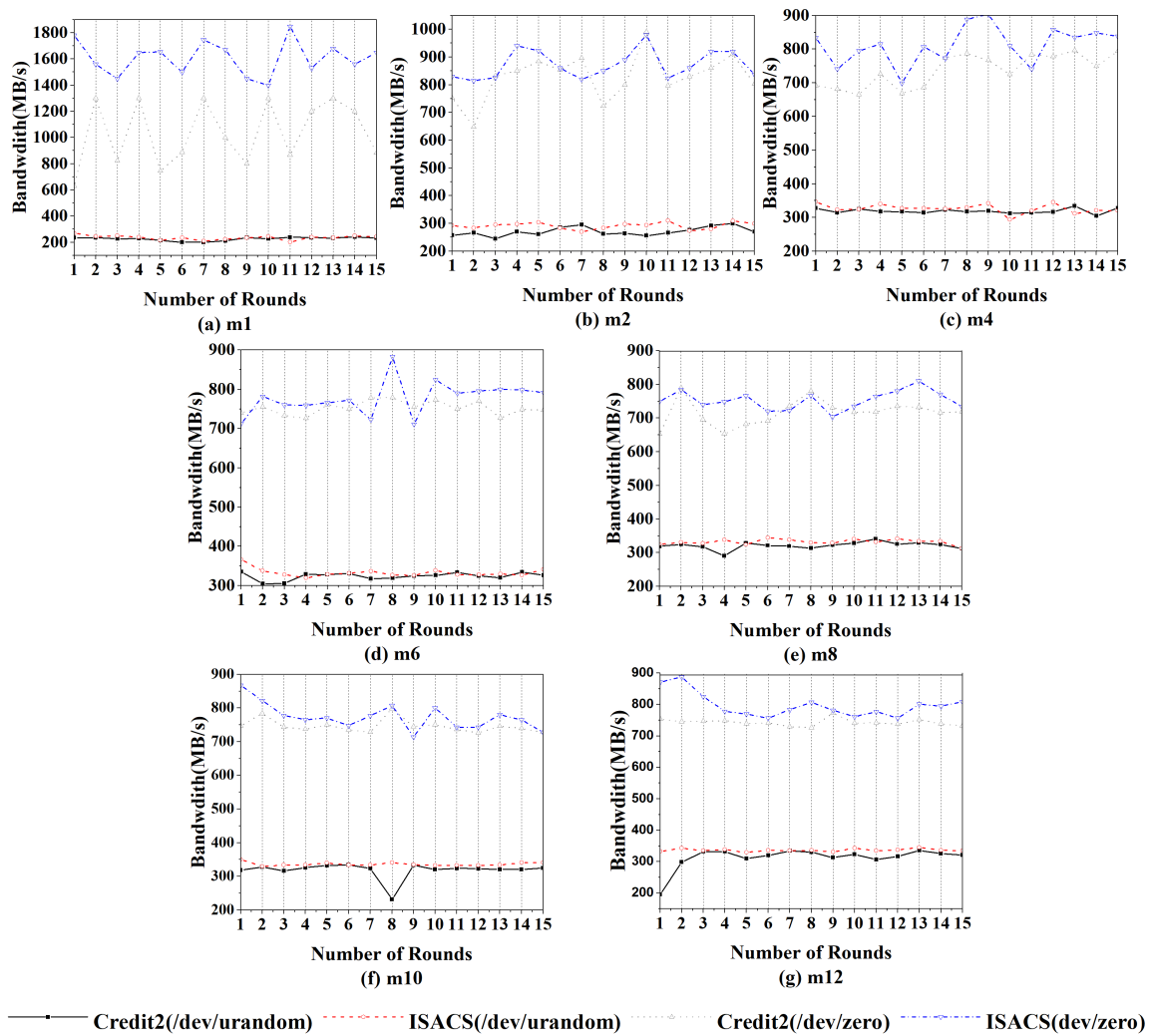


Figure 12. Disk bandwidth in 15 experiments according to the VM configuration in the mixed environment.

5.4. Degree of CPU Performance Impact Owing To ISACS

In this experiment, the degree of impact on the CPU performance of the VMs caused by the increase in I/O performance owing to ISACS is compared with that for the existing environment. We measure the degree of impact on the internal CPU performance of a VM with mixed workloads and the external CPU performance for a neighbor VM. To measure CPU performance based on execution time, we leverage Sysbench, a benchmark tool that finds prime numbers from one to a predefined number. We set up a Sysbench to find prime numbers from one to 1,000,000 for all experiments in this section. Further, Iperf is used to generate I/O-intensive workloads. All vCPUs of *CLIENT – VM* were fixed to pCPU7. All vCPUs of the guest VMs were fixed from pCPU2 to pCPU6, and in this experiment, twenty-four vCPUs corresponding to six guest VMs were assigned to five cores.

5.4.1. ISACS Impacts on Internal CPU Performance of VM

The purpose of this experiment is to measure the degree of CPU performance degradation inside a VM when ISACS guarantees pCPU occupancy of the I/O-intensive vCPUs via the credit concealing method; this performance degradation is compared with that in the existing environment based on the Sysbench execution time. The experiment was conducted in a mixed environment to generate the CPU and I/O workloads to be handled simultaneously on the VM. For experiments, the number of VMs in the VM configuration was changed to six VMs by increasing the VMs one by one. Figure 13 shows the average execution time for 15 rounds of experiments according to the VM configuration.

In the experiment, the largest difference in execution time is when six VMs are running, and the existing environment takes 56.11 s and ISACS takes 56.93 s, which ISACS shows the execution time with $0.98\times$ in speed-up. However, the overall results of the experiment for the existing environment and ISACS are similar. The reason for the similar execution time emerged is that ISACS provides an appropriate degree of pCPU occupancy for I/O-intensive vCPU using Equations (1) and (2) and guarantees the consumption of CVCredit for CPU-intensive vCPUs through the CPU-BOOST status. Therefore, CPU-intensive vCPU can be guaranteed pCPU occupancy from I/O-intensive vCPUs that IVCredit is allocated. Through this experiment, we confirm that ISACS prevents CPU performance degradation arising from ensuring the pCPU occupancy of I/O-intensive vCPUs inside the VM.

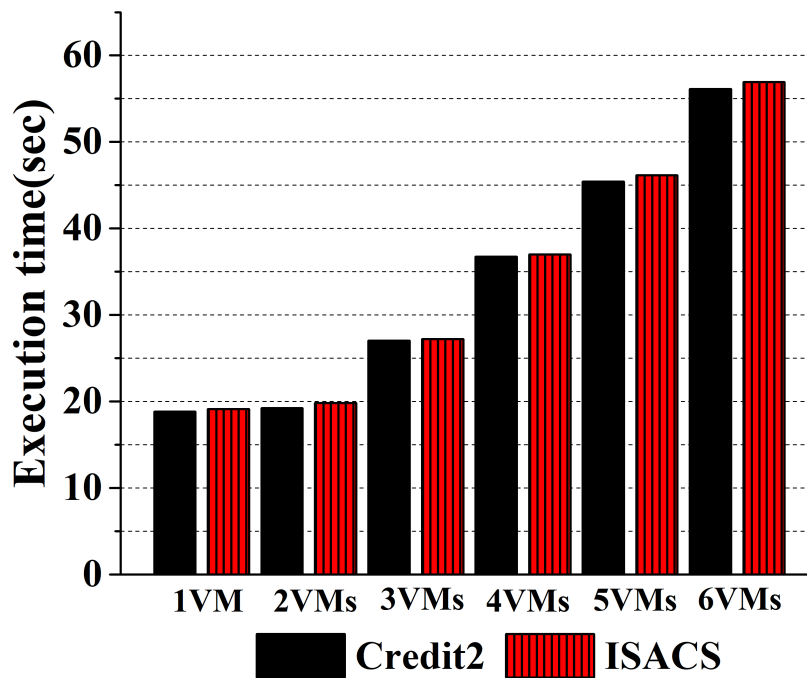


Figure 13. Average execution times for 15 experiments according to the VM configurations in the mixed environment.

5.4.2. Impacts of ISACS on CPU Performance of the Neighboring VM

The purpose of this experiment is to measure the degree of neighbor CPU performance interference arising from ISACS in hybrid and mixed environments. In the experiment, we set up the target-VM that handles only a CPU-intensive workload and determine whether ISACS affects the target-VM's CPU performance more than the existing environment. To increase the degree of CPU performance interference for the target-VM, we gradually increase the number of VMs in the hybrid and mixed environments to determine the changes in the bench execution time of the target-VM according to the number of VMs. Figures 14 and 15 present the results of the execution time for target-VM according to the execution environment.

As shown in Figure 14, in the hybrid environment, the CPU task processing time of the target-VM is similar in the existing environment and in ISACS. However, as shown in Figure 15, in a mixed environment, ISACS entailed a slight delay compared to the existing environment. In the experiments, the largest differences in execution time for the existing environment and ISACS, in a hybrid environment, are 10.14 s and 10.65 s respectively. For the mixed environment, these are 53.12 s and 55.94 s respectively when four VMs are running. For the above two experimental results, ISACS appears the execution time with $0.95\times$ approximately in speed-up than that of existing environment. The reason for the lower CPU performance of the target-VM in a mixed environment than that in the existing environment is that the vCPUs of the VMs other than those of the target-VM are assigned IVCredits for each scheduling round due to I/O handling. Because I/O-intensive vCPUs are assigned IVCredits

immediately after the credit reallocation, which leads to a higher amount of credits on average for vCPUs other than those of the target-VM, the scheduling priority of the vCPUs of the target-VM in the mixed environment is lower in the experiment. In the experiments, the However, the difference in the execution time of Sysbench is very small and thus is negligible.

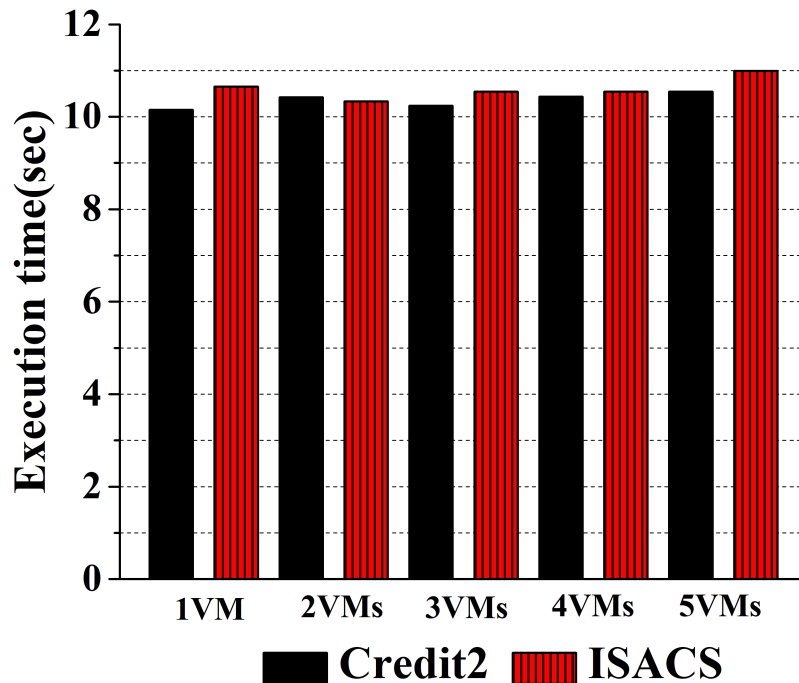


Figure 14. Average execution time of target-VM for 15 experiments according to the VM configuration in the hybrid environment.

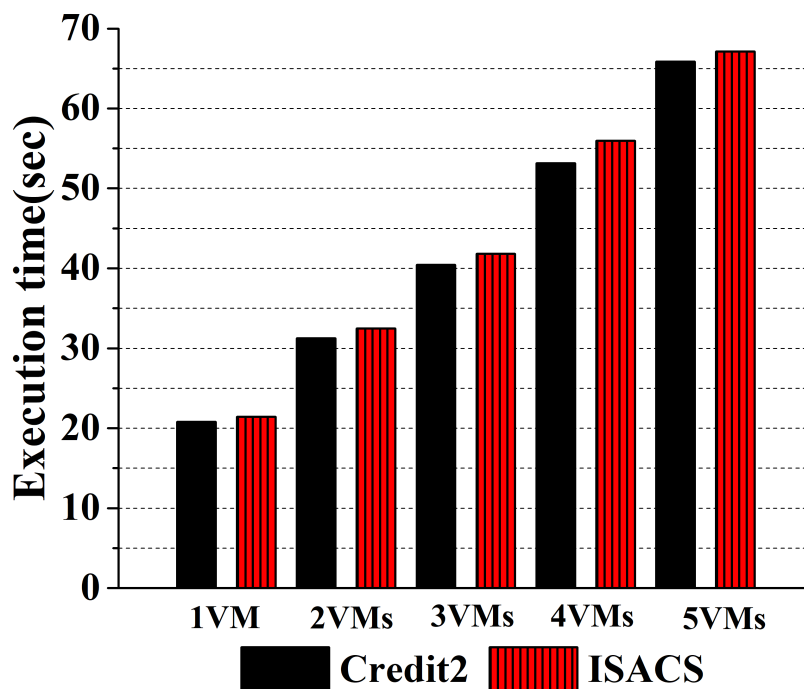


Figure 15. Average execution time of target-VM for 15 experiments according to the VM configuration in the mixed environment.

Note that in this experiment, Equations (3) and (4) prohibit the allocation of CVCredits for I/O-intensive vCPUs with relatively large amounts of IVCredits and prevent CPU performance

degradation as much as possible. In addition, the CPU-BOOST STATE, which is a practical mechanism for ensuring CPU performance, guarantees CVCredit consumption of CPU-intensive vCPU from I/O-intensive vCPUs in the run-queue. Through the above mechanism, ISACS performs a trade-off between I/O-intensive vCPUs and CPU-intensive vCPUs in a virtualization environment with limited resources to achieve fairness for maximum resource occupancy.

6. Related Work

Many researchers are attempting to solve the aforementioned problems of virtualized environments. Techniques have been proposed to improve the I/O performance of VMs by modifying and adding scheduling parameters of the scheduler to predict the workload type of VMs for existing virtualized environments [8,47–49].

Caglar et al. [21] used machine learning algorithms to derive optimized scheduling parameters for the workload type of each VM to enhance I/O performance of VMs. First, this scheme analyzes logs of each VM through modified Xenmon monitoring tools [34] and clusters them according to the characteristics of each VM. Second, it derives optimized scheduling parameters through machine learning for clustered groups. Finally, using the optimized scheduling parameters, the proposed technique could minimize VM latency in the scheduler. However, additional tasks such as monitoring and load analysis on DOM0 that handle I/O requests on all VMs may result in an I/O overhead in all VMs. In addition, they do not consider the latency and overhead that occurs until parameters optimized by machine learning are applied to the scheduler. However, ISACS can instantly schedule vCPUs according to the vCPU's I/O workload by deriving the I/O strength of each vCPU in the run-queue before credit reallocation occurs and applying it directly to the hypervisor scheduler. ISACS does not need the external monitoring modules, so it can prevent the overhead caused by synchronizing parameters with the hypervisor scheduler.

Jang et al. [22] presented a loan and redeem mechanism in the Credit1 scheduler through the BOOST controlling method. They allocated BOOST credits derived from the previous scheduling round to all vCPUs in the current round; this process considers the BOOST frequency to control the I/O resource sharing of I/O-intensive VMs. However, allocating BOOST credits can limit continuous I/O resource occupancy in situations in which there are sufficient I/O resources and a small number of I/O-intensive VMs and can lead to performance interference of VMs with different workloads. ISACS does not limit the I/O performance of vCPUs because it guarantees pCPU occupancy only based on the independent I/O strength of each vCPU, regardless of the I/O degree of the neighboring vCPUs.

Another strategy for increasing I/O performance in a virtualized environment is resource management technology that takes into account the time slice of vCPU according to the workload type of the VM [28,29,47]. In another previous study [11], a technique was proposed to improve I/O performance using the CPU-pool mechanism. They used monitoring tools to determine the workload type of a VM. Then, through the CPU-pool mechanism, they pinned the VM to a pCPU that fits the workload type of the VM and assigned a dynamic time slice optimized for the characteristics of VMs pinned to the pCPU. Thus, for each pCPU, time slices optimized for VMs improve the I/O performance for I/O-intensive VMs. However, the allocation of time slices by the VM unit is a coarse-grained characterization of workloads, and unoptimized time slices can be derived for vCPUs that simultaneously perform I/O-intensive and CPU-intensive workloads. pCPU pinning creates a trade-off relationship between load balancing for each pCPU and the accuracy of the time slice. ISACS provides fine-grained workload type awareness by measuring I/O strength in units of vCPUs and not in units of VMs. In addition, ISACS can ensure a fair I/O load for all cores by performing I/O load balancing; thus, ISACS does not have problems that may arise from pCPU pinning for vCPUs with various time slice and vCPUs of all VMs can be allocated to all cores more flexibly.

Jia W et al. [50] proposed the vMigrator that increases the I/O performance of I/O-intensive VMs through I/O-intensive task migration techniques to prevent I/O latency caused by frequent I/O activity of vCPU. vMigrator tracks the time slice consumption pattern of all vCPUs and migrates

I/O-intensive tasks to vCPUs with the longest remaining time slices among vCPUs with the active state of the VM so that I/O operations can be processed immediately. However, if the vCPU workload changes rapidly, the patterned time slice length which is derived from the previous scheduling round will not be hit, resulting in lower overall performance, and there is a limitation not considering network I/O intensive vCPU. ISACS is adaptive for workload changes on vCPU by measuring the I/O strength of all vCPUs in the run-queue and scheduling the vCPUs based on the I/O strength of vCPU before the scheduling round begins. It also provides pCPU occupancy of the I/O-intensive vCPU by allocating IVCredit based on the I/O strength of the vCPU obtained by tracking the disk I/O information as well as network for all vCPUs in the event channel and proven by experiments, which has been proven through the experiment.

Xu et al. [27] proposed a dynamic time slice technique to improve I/O performance in a virtualized environment. In this study, a scheme allocates a short time slice to I/O-intensive vCPUs by considering the default time slice in the Credit1 scheduler; further, it shares time slices between the vCPUs of each I/O-intensive VM. Thus, the I/O latency is reduced in a virtualized environment as the average number of pCPU occupancies of an I/O-intensive vCPU increases in each scheduling round.

Another method to improve the I/O performance involves modifying the hypervisor structure in an existing virtualization environment. Zeng et al. [3] proposed an SR-IOV allocation technique by leveraging the ACPI hotplug to improve I/O performance. After differentiating the I/O occurrence degree of each VM through the modified Xenmon, they allocated first VF(virtual function) and SR-IOV, which enables direct communication between VM and paravirtual network interface cards, to I/O-intensive VMs. As a result, the I/O performance in the overall virtualized environment was improved by devising a scheduling technique that dynamically allocates the VFs and SR-IOV to I/O-intensive VMs.

Kim et al. [38] proposed a guest-aware scheduling method to provide an awareness of the I/O degree of VMs to a hypervisor by modifying the kernel of the guest VMs. They inspected the information of the blocked tasks that await an I/O response to the operating system of the guest, and then, shared the task information in the blocked state of the specific VM with the hypervisor. Then, a high scheduling priority was given to the specific VM with high blocking levels to ensure pCPU preoccupation and increase I/O performance. However, modifying the VM's I/O device driver takes a lot of code modification at a high cost and is less portable. ISACS tracks I/O information from all I/O device drivers wherein the event channel, which is the above layer of all I/O device drivers, making it more portable and less costly to modify.

7. Conclusions

The main reason for the degradation in I/O performance in a virtualized environment is the scheduling policy, which is based on pCPU occupancy and the lack of workload-type awareness. To increase the I/O performance of I/O-intensive VMs, pCPU occupancy should be guaranteed according to the degree of I/O occurrence for each vCPU. In addition, the performance of CPU-intensive vCPUs should be simultaneously ensured. Thus, it is important to consider the trade-off between handling I/O and CPU workloads. In this paper, we propose ISACS to mitigate I/O performance degradation in virtualized environments. Notably, we modified the event channel and the Credit2 scheduler to determine the I/O strength corresponding to each vCPU such that ISACS could perceive the degree of I/O occurrence. Further, ISACS reflects the I/O strength based on virtual credits (IVCredit, CVCredit) provided to vCPUs to optimize the I/O performance and CPU performance of VMs. Finally, ISACS performs I/O load balancing in each run-queue considering the I/O strength of each vCPU. Our experiments showed that ISACS provides higher network and disk performances than those of the existing virtualized environment by optimizing the I/O performance.

Author Contributions: Conceptualization, J.L.; Data curation, J.L.; Formal analysis, J.L.; Funding acquisition, H.Y.; Investigation, J.L.; Methodology, J.L.; Project administration, H.Y.; Resources, J.L. and H.Y.; Software, J.L.; Supervision, H.Y.; Validation, J.L. and H.Y.; Visualization, J.L. and H.Y.; Writing—original draft, J.L. and H.Y.;

Writing—review & editing, J.L. and H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-0-01405) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation). This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00480, Developing the edge cloud platform for the real-time services based on the mobility of connected cars).

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

References

- Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. Xen and the art of virtualization. *ACM SIGOPS Oper. Syst. Rev.* **2003**, *37*, 164–177. [\[CrossRef\]](#)
- Asyabi, E.; Sharifi, M.; Bestavros, A. ppXen: A hypervisor CPU scheduler for mitigating performance variability in virtualized clouds. *Future Gener. Comput. Syst.* **2018**, *83*, 75–84. [\[CrossRef\]](#)
- Zeng, L.; Wang, Y.; Fan, X.; Xu, C. Raccoon: A novel network i/o allocation framework for workload-aware vm scheduling in virtual environments. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 2651–2662. [\[CrossRef\]](#)
- Pu, X.; Liu, L.; Mei, Y.; Sivathanu, S.; Koh, Y.; Pu, C.; Cao, Y. Who is your neighbor: Net I/O performance interference in virtualized clouds. *IEEE Trans. Serv. Comput.* **2012**, *6*, 314–329. [\[CrossRef\]](#)
- Ram, K.K.; Santos, J.R.; Turner, Y. Redesigning Xen’s memory sharing mechanism for safe and efficient I/O virtualization. In Proceedings of the 2nd Workshop on I/O Virtualization, Pittsburgh, PA, USA, 17–19 March 2010.
- Rodrigues, H.; Santos, J.R.; Turner, Y.; Soares, P.; Guedes, D.O. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. *WIOV* **2011**, *1.3*, 784–789.
- Ongaro, D.; Cox, A.L.; Rixner, S. Scheduling I/O in virtual machine monitors. In Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Seattle, WA, USA, 3–5 March 2008; pp. 1–10.
- Vanga, M.; Gujarati, A.; Brandenburg, B.B. Tableau: A high-throughput and predictable Hypervisor scheduler for highdensity workloads. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–16.
- Li, J.; Ma, R.; Guan, H.; Wei, D.S. Accurate CPU proportional share and predictable I/O responsiveness for virtual machine monitor: A case study in Xen. *IEEE Trans. Cloud Comput.* **2015**, *5*, 604–616. [\[CrossRef\]](#)
- Hu, Y.; Long, X.; Zhang, J.; He, J.; Xia, L. I/O scheduling model of virtual machine based on multi-core dynamic partitioning. In Proceedings of the the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, IL, USA, 21–25 June 2010; pp. 142–154.
- Lin, L.; Liu, X.; Ma, R.; Li, J.; Wang, D.; Guan, H. vSimilar: A high-adaptive VM scheduler based on the CPU pool mechanism. *J. Syst. Archit.* **2019**, *98*, 361–373. [\[CrossRef\]](#)
- Asyabi, E.; SanaeeKohroudi, S.; Sharifi, M.; Bestavros, A. TerrierTail: Mitigating Tail Latency of Cloud Virtual Machines. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 2346–2359. [\[CrossRef\]](#)
- Cheng, L.; Wang, C.L. vBalance: Using interrupt load balance to improve I/O performance for SMP virtual machines. In Proceedings of the Third ACM Symposium on Cloud Computing, San Jose, CA, USA, 29–31 October 2012; pp. 75–84.
- Credit2 Scheduler. Available online: <https://wiki.xenproject.org/wiki/Credit2Scheduler> (accessed on 9 December 2020).
- Kang, M.; Lee, S. Mcredit2: Enhanced High-Performance Xen Scheduler via Dynamic Weight Allocation. *Hindawi J. Sens.* **2018**, 1–10. [\[CrossRef\]](#)
- Zeng, L.; Wang, Y.; Shi, W.; Feng, D. An improved xen credit scheduler for i/o latency-sensitive applications on multicores. In Proceedings of the IEEE 2013 International Conference on Cloud Computing and Big Data, Washington, DC, USA, 16–19 December 2013; pp. 267–274.
- Liu, L.; Wang, H.; Wang, A.; Xiao, M.; Cheng, Y.; Chen, S. vCPU as a container: Towards accurate CPU allocation for VMs. In Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Providence, RI, USA, 14 April 2019; pp. 193–206.

18. Jiang, C.; Fan, T.; Qiu, Y.; Wu, H.; Zhang, J.; Xiong, N.N.; Wan, J. Interdomain I/O optimization in virtualized sensor networks. *J. Sens.* **2018**, *18*, 4359. [[CrossRef](#)] [[PubMed](#)]
19. Mei, L.; Lv, X. Optimization of network bandwidth allocation in Xen. In Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, Los Alamitos, CA, USA, 24–26 August 2015; pp. 1558–1566.
20. Xu, Y.; Musgrave, Z.; Noble, B.; Bailey, M. Bobtail: Avoiding long tails in the cloud. In Proceedings of the Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), Berkeley, CA, USA, 28–30 April 2013; pp. 329–341.
21. Caglar, F.; Shekhar, S.; Gokhale, A.S. iTune: Engineering the performance of Xen hypervisor via autonomous and dynamic scheduler reconfiguration. *IEEE Trans. Serv. Comput.* **2016**, *11*, 103–116. [[CrossRef](#)]
22. Jang, J.; Jung, J.; Hong, J. An efficient virtual CPU scheduling in cloud computing. *J. Soft Comput.* **2019**, *24*, 1–11. [[CrossRef](#)]
23. Gamage, S.; Xu, C.; Kompella, R. R.; Xu, D. vPipe: Piped I/O offloading for efficient data movement in virtualized clouds. In Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 3–4 November 2014; pp. 1–13.
24. Santos, J.R.; Turner, Y.; Janakiraman, G.J.; Pratt, I. Bridging the Gap between Software and Hardware Techniques for I/O Virtualization. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, 22–27 June 2008; pp. 29–42.
25. Cheng, L.; Lau, F.C.M. Offloading interrupt load balancing from smp virtual machines to the hypervisor. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 3298–3310. [[CrossRef](#)]
26. Kangarlou, A.; Gamage, S.; Kompella, R. R.; Xu, D. vSnoop: Improving TCP throughput in virtualized environments via acknowledgement offload. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 13–19 November 2010; pp. 1–11.
27. Ding, X.; Ma, Z.; Da, X. Dynamic time slice of credit scheduler. In Proceedings of the 2014 IEEE International Conference on Information and Automation (ICIA), Yunnan, China, 28–30 July 2014; pp. 654–659.
28. Xu, C.; Gamage, S.; Rao, P. N.; Kangarlou, A.; Kompella, R. R.; Xu, D. vSlicer: Latency-aware virtual machine scheduling via differentiated-frequency CPU slicing. In Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, Delft, The Netherlands, 12–15 June 2012; pp. 3–14.
29. Ma, R.; Li, J.; Lin, L.; Guan, H. DaSS: Dynamic Time Slice Scheduler for Virtual Machine Monitor. *Int. Conf. Algorithms Archit. Parallel Process.* **2015**, *10*, 77–90.
30. Kang, D.J.; Kim, C.Y.; Kim, K.H.; Jung, S.I. Proportional disk I/O bandwidth management for server virtualization environment. In Proceedings of the 2008 International Conference on Computer Science and Information Technology, Washington, DC, USA, 7–10 December 2008; pp. 647–653.
31. Guan, B.; Wu, J.; Wang, Y.; Khan, S. CIVSched: A communication-aware inter-VM scheduling technique for decreased network latency between co-located VMs. *IEEE Trans. Cloud Comput.* **2014**, *2*, 320–332. [[CrossRef](#)]
32. Xu, C.; Ma, X.; Shea, R.; Wang, H.; Liu, J. MemNet: Enhancing Throughput and Energy Efficiency for Hybrid Workloads via Para-virtualized Memory Sharing. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing, San Francisco, CA, USA, 27 June–2 July 2016; pp. 980–983.
33. Venkatesh, V.; Nayak, A. Optimizing I/O Intensive Domain Handling in Xen Hypervisor for Consolidated Server Environments. In Proceedings of the 11th International Conference on Green, Pervasive and Cloud Computing, Xi'an, China, 6–8 May 2016; pp. 180–195.
34. Gupta, D.; Gardner, R.; Cherkasova, L. *Xenmon: Qos Monitoring and Performance Profiling Tool*; Hewlett-Packard Labs Technical Reports HPL-2005-187; HP Development Company: Spring, TX, USA, 2005; Volume 16, pp. 1–13.
35. Zhan, D.; Li, H.; Ye, L.; Zhang, H.; Fang, B.; Du, X. A Low-overhead Kernel Object Monitoring Approach for Virtual Machine Introspection. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications, Shanghai, China, 20–24 May 2019; pp. 1–6.
36. Sato, M.; Yamauchi, T. Secure log transfer by replacing a library in a virtual machine. In Proceedings of the 8th International Workshop on Security, Okinawa, Japan, 18–20 November 2013; Volume 10, pp. 1–18.
37. Xiang, G.; Jin, H.; Zou, D.; Zhang, X.; Wen, S.; Zhao, F. VMDriver: A driver-based monitoring mechanism for virtualization. In Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems, Delhi, India, 31 October–3 November 2010; Volume 10, pp. 72–81.

38. Kim, D.-S.; Kim, H.; Jeon, M.; Seo, E.; Lee, J. Guest-Aware Priority-Based Virtual Machine Scheduling for Highly Consolidated Server. In Proceedings of the European Conference on Parallel Processing, Las Palmas de Gran Canaria, Spain, 26–29 August 2008; pp. 285–294.
39. Hnarakis, R. In Perfect Xen, “A Performance Study of the Emerging Xen Scheduler”. Ph.D. Thesis, the Faculty of California Polytechnic State University San Luis Obispo, San Luis Obispo, CA, USA, 2013.
40. Hughes, A.; Amro, A. Quantifying Performance Determinism in Virtualized Mixed-Criticality Systems. In Proceedings of the 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing, Valencia, Spain, 7–9 May 2019; pp. 181–184.
41. Guo, D.; Liao, G.; Bhuyan, L.N. Performance characterization and cache-aware core scheduling in a virtualized multicore server under 10GbE. In Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, 4–6 October 2009; pp. 168–177.
42. IPERF, Networking with Iperf. Available online: <https://sourceforge.net/projects/iperf/> (accessed on 9 December 2020).
43. Ping. Available online: <http://manpages.ubuntu.com/manpages/bionic/en/man1/ping.1.html> (accessed on 9 December 2020).
44. DD. Available online: <http://man7.org/linux/man--pages/man1/dd.1p.html> (accessed on 9 December 2020).
45. Stress. Available online: <https://linux.die.net/man/1/stress> (accessed on 9 December 2020).
46. Kopytov, A. SysBench: A Scriptable Database and System Performance Benchmark. Available online: <https://github.com/akopytov/sysbench> (accessed on 9 December 2020).
47. Fang, W.; Lu, Z.; Wu, J.; Cao, Z. Rpps: A novel resource prediction and provisioning scheme in cloud data center. In Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 609–616.
48. Raghunath, B.R.; Annappa, B. Dynamic resource allocation using fuzzy prediction system. In Proceedings of the 2018 3rd International Conference for Convergence in Technology, Pune, India, 6–8 April 2018; pp. 1–6.
49. Kousiouris, G.; Cucinotta, T.; Varvarigou, T. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *J. Syst. Softw.* **2011**, *84*, 1270–1291. [[CrossRef](#)]
50. Jia, W.; Wang, C.; Chen, X.; Shan, J.; Shang, X.; Cui, H.; Ding, X.; Cheng, L.; Francis, C.M.; Wang, Y. Effectively mitigating I/O inactivity in vCPU scheduling. In Proceedings of the 2018 USENIX Annual Technical Conference, Boston, MA, USA, 11–13 July 2019; pp. 267–280.

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).