CIVSched: A Communication-aware Inter-VM Scheduling Technique for Decreased Network Latency between Co-located VMs

Bei Guan, Jingzheng Wu, Yongji Wang, and Samee U. Khan, Senior Member, IEEE

Abstract—Server consolidation in cloud computing environments makes it possible for multiple servers or desktops to run on a single physical server for high resource utilization, low cost, and reduced energy consumption. However, the scheduler in the virtual machine monitor (VMM), such as Xen credit scheduler, is agnostic about the communication behavior between the guest operating systems (OS). The aforementioned behavior leads to increased network communication latency in consolidated environments. In particular, the CPU resources management has a critical impact on the network latency between co-located virtual machines (VMs) when there are CPU- and I/O-intensive workloads running simultaneously. This paper presents the design and implementation of a communication-aware inter-VM scheduling (CIVSched) technique that takes into account the communication behavior between inter-VMs running on the same virtualization platform. The CIVSched technique inspects the network packets transmitted between local co-resident domains to identify the target VM and process that will receive the packets. Thereafter, the target VM and process are preferentially scheduled by the VMM and the guest OS. The cooperation of these two schedulers makes the network packets to be timely received by the target application. Experimental results on the Xen virtualization platform depict that the CIVSched technique can reduce the average response time of network traffic by approximately 19% for the highly consolidated environment, while keeping the inherent fairness of the VMM scheduler.

Index Terms—Communication-aware scheduling, inter-VM, Xen, resource management, vCPU scheduling

1 INTRODUCTION

The cloud is a new paradigm for the provisioning of computing infrastructures. The paradigm migrates the location of the hardware and software resources onto the network to reduce the management costs. Virtualization is one of the primary techniques used within the cloud computing paradigm that is able to abstract, split, assign, and dynamically resize the hardware resources. Virtualization allows multiple operating systems (OSes) and applications to run on independent partitions on a single computer.

Using virtualization, the server consolidation is a solution to efficiently utilize server resources by constraining the total number of servers or server locations that an organization requires. The server consolidation technique orchestrates the service applications deployed on distributed physical servers in different virtual machines (VMs) running on a single virtualization platform [34]. Virtualization can substantially increase the efficient usage of computing resources and reduce the aggregate number of servers; consequently, reducing operational costs and energy consumption. The Quality of Service (QoS) requirements of the applications running in the consolidated VMs have received a considerable attention in the academia and industrial research communities [15], [40]. Fig. 1 presents a typical three-tier Web application that is transferred from a distributed environment to a server consolidation environment. The network communication between different tiers in the Web application is shown by the dashed arrows. When the three-tier Web application is consolidated in a virtualized physical platform, the network communication between various physical servers transfers to that between different co-located VMs (inter-VM communication [2]). In the server consolidation environments, a virtual machine monitor (VMM), also known as a hypervisor, plays an important role to virtualize the physical resource, such as the CPU, memory and I/O resource, and employs a scheduler to manage the physical resource allocation among the upper co-located VMs. However, a semantic gap exists between the VMM and the VMs. The semantic gap is that the VMM is totally agnostic to the behavior of the running tasks in the VMs [14], and the gap will impede the efficient allocation of hardware resources that are required by each of the VMs in the consolidated environments. If the CPU scheduler of the VMM does not take into consideration the communication behavior between different co-located VMs, then the scheduler might make non-optimal scheduling decisions that may lead to increased inter-VM network communication latency [1], [2]. For example, in a TCP/IP connection over a virtual network, a client VM sends a request to a co-located server VM. When the server VM cannot be scheduled timely, the response to the request will not be sent back to the client VM quickly. Such a behavior may introduce additional packet delivery latency.

• Bei Guan, Jingzheng Wu, and Yongji Wang are with the Institute of Software, Chinese Academy of Sciences, Beijing, China. E-mail: {guanbei, jingzheng, yongji}@iis.ac.cn, yongji@bit.edu.cn.
• Samee U. Khan is with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo ND 58108-6050, USA. E-mail: samee.khan@ndsu.edu.
Moreover, when the CPU- and the I/O-intensive workloads in other co-resident VMs also run concurrently, the latency may even further worsen because of the competition for the CPU resources [1], [2].

To further emphasize on the importance of the inter-VM communication and the effect of latency, we provide two illustrative examples. Consider that there are two individuals, playing an online chess game. The game’s client programs run on two co-located desktop VMs, respectively. The VMs must be scheduled by the VMM before the players can independently register moves through their respective clients. If the VMs cannot be scheduled timely, then the clients will not be able to obtain the CPU, leading to latency in registering the moves.

We present a three-tier Web online ticketing system, as a second example, whose Web frontend VM must repeatedly query the co-located database VM to update in real-time the quantity of the remaining tickets. If the database VM is not scheduled when an incoming query event reaches, then the server performance degradation is to be expected. From the above mentioned examples, we can conclude that the network communication between co-located VMs hosting such types of applications is characterized by the light network traffic and is sensitive to high latency.

Much of the prior work has been conducted to overcome the semantic gap and decrease the inter-VM communication latency. Govindan et al. proposed a communication-aware VM scheduling mechanism, called XenandCo, in consolidated hosting environments [1]. Their method monitored the I/O ring and preferentially scheduled the VMs that received more data packets or were anticipated to send more data packets. However, XenandCo does not consider the network communication of the latency-sensitive domains with light traffic. Much of the other work uses the shared memory to reduce the data transfer time between the domains [8], [9], [10], [11], [12], [42]. The aforementioned works achieve high performance, but most need to re-implement the program interface of the guest OS for the applications to use the shared memory mechanism. Different from previous work, this paper provides a new solution from the perspective of the CPU resources scheduling. Specifically, this paper proposes to alleviate the inter-VM communication latency under the Xen virtualization platform by using a novel communication-aware scheduler that significantly narrows the semantic gap between the allocation of the CPU resources and the network communication behavior of the VMs. The proposed CIVSched technique, improves the response time of the Xen credit scheduler by taking into account the network communication behavior between inter-VMs. The CIVSched technique is similar to that of Govindan et al.’s solution; however, the CIVSched provides more priority to the network communication between inter-VMs. Firstly, CIVSched inspects the local network packet and extracts the Media Access Control (MAC) address and the network TCP/UDP port number. This information is used to identify the co-located target VM and process that will finally receive the network packet. Secondly, if the packet is sent to a co-resident VM, CIVSched technique informs the VMM scheduler to schedule the VM preferentially. At the same time, CIVSched also notifies the kernel scheduler of the target VM to schedule the target process to receive the network packet. The CIVSched employs a module called AutoCover in Dom0 to automatically discover all of the MAC addresses of the co-resident VMs. Another module PidTrans in the guest OS provides the mapping of a network port number to the listening process’s Process ID (PID). As a comparison, we implemented Govindan et al.’s solution based on our experiment platform. The experiment results show the CIVSched scheduler has a better performance on the network latency. The details will be provided in Section 7. Moreover, live migration of the VM is also supported by the CIVSched technique, which we will detail in Section 6.

We develop a prototype of the proposed CIVSched scheduler in the Xen VMM and explore the performance of the prototype by experimenting with realistic and representative network benchmarks. Both of the simulation and real-world workloads are adopted in the evaluation studies. The evaluations demonstrate the benefits of the CIVSched technique in alleviating the inter-VM latency of highly consolidated environments. For example, the experimental results showed that the CIVSched can effectively reduce the latency by approximately 19%, with little overhead to the fairness of the VMM scheduler and that to the Dom0. Although the focus of this paper is on Xen, the core concept can also be applied to several other virtualization platforms.

**Contribution synopsis.** The development of the model and the solution method for alleviating the inter-VM network latency consists of the following:

(a) We provide the mathematical formulation of the problem of the co-located inter-VMs network latency. The formulation is detailed enough to capture every step that may contribute towards the latency. The formulation also takes into consideration the allocation of the CPU resources between the privileged domains and the unprivileged domains within a consolidated environment. The details on the problem formulation are provided in Section 4.

(b) We propose a hypervisor scheduler algorithm based on Xen, named the CIVSched. The CIVSched is designed to be aware of the behavior of the inter-VMs network communications. The CIVSched narrows the semantic gap between the
VMM scheduler and the tasks running within the VMs. The scheduling technique makes the CPU resources well-allocated among VMs, which we detail in Section 5 and Section 6.

(c) The CIVSched monitors the network packets between the co-located VMs and extracts the useful information to identify the target VM towards which the communication is directed. Thereafter, the CIVSched preferentially schedules the target VM to reduce the waiting latency of the VM in the run queue. We will provide details on the aforementioned contribution in Section 6.1 and Section 6.2.

(d) Much of the research on the VMM scheduling only focuses on how to effectively schedule the VM [1], [14], [17]. Besides scheduling the VM, the CIVSched also takes into consideration the processes running in the VM. Through inspecting the VM’s process queue, the CIVSched judiciously schedules the target process, which can further decrease the network communication latency from the perspective of the process scheduling. The details are presented in Section 6.2.

(e) A prototype scheduler is implemented based on the Xen virtualization platform and the default credit scheduler. Several synthetic and real-world workloads are used to benchmark the performance of the CIVSched, which are detailed in Section 7.

The remainder of this paper is organized as follows. The necessary background information to understand the CIVSched is detailed in Section 2. Section 3 presents the related work. In Section 4, we formulate the scheduling problem, and in Section 5, we provide an overview of the CIVSched architecture. Section 6 provides the design details of all of the key components of the proposed scheduling approach. We also provide a brief introduction to the implementation of the CIVSched in Section 7, which is also used to discuss the performance evaluations of the proposed scheduling technique. Finally, conclusions and future work are provided in Section 8.

2 SERVER CONSOLIDATION AND XEN

2.1 Server Consolidation

Server consolidation is a popular and a highly efficient resource utilization technique in cloud computing. A lot of research has been conducted in the cloud consolidated environment to optimize the resource scheduling algorithm for increasing the utilization of the system resources [32], [33], [29], [30], [41]. Many of the existing cloud infrastructures, such as Amazon EC2 [26], Xen Cloud Platform [27], and Eucalyptus [28], employ the Xen hypervisor as their underlying virtualization component to implement the server consolidation. The Xen hypervisor supports various OSes to run in a virtualized host platform, which makes it easy for the applications to be ported to a virtualized environment. In the server consolidation environment, it is a convention that applications run within different VMs that are co-located in a single server, and share the physical resource, such as CPU cores and I/O bandwidth. Much of the prior work has been studied based on such a convention in the academia and industrial research communities [1], [14], [3], [16], [17], [26], [27]. As mentioned in the introductory section, the server consolidation brings great benefits to the service providers. However, research shows that the performance of the I/O operation, such as the responsiveness, degrades considerably in the consolidated environments [6], [14]. Limited physical resources on a single server makes it necessary to design efficient scheduling algorithms to coordinate between the VMM and VM to select the most appropriate VM and process to use the computing resources during every CPU scheduling time slice. Such a selection process must ensure that high throughput and low latency is maintained.

2.2 Xen I/O Mechanism Overview

The VMM allows multiple OSes to safely share a single physical server. It provides isolation between guest OSes and access controls to hardware devices. As a representative of the open source VMM, Xen [4] is a widely used hypervisor that supports para-virtualization (PV) and hardware-assisted virtualization (HVM) that enables the OSes to run on Xen with or without modifications. The Xen hypervisor adopts a split driver model to manage access to the devices. The front-end driver of the model resides in the unprivileged domain (DomU) and serves as a real device driver. The back-end driver is the counterpart of the front-end driver and is located in the isolated driver domain (IDD) that is Dom0 in Xen. The back-end driver forwards the device requests from the front-end driver to the native device driver, and returns the response to the front-end driver. The IDD can also multiplex and de-multiplex the device requests by using the virtual network bridge (as shown in Fig. 2).

2.3 Xen Credit Scheduler

The current version of Xen uses the credit scheduler as the default scheduling algorithm to allocate the CPU resources to the VMs [7]. Credit is a proportional fair share CPU scheduler that assigns two parameters to every do-
main, weight and cap. The weight is represented by the amount of credits that are given to a domain in every time slice, while the cap is the maximum amount of the CPU resources that a domain will be able to consume, even if the host system has idle CPU cycles.

There are three priorities for domains in the credit scheduler, UNDER (-1), OVER (2) and BOOST (0). Each physical CPU manages a queue of runnable domains and the queue is sorted by the domains’ priorities. A domain with a positive number of credits has the UNDER priority (indicated by U1 and U2 in Fig. 3). As a domain runs, it consumes credits. When the number of credits reduces to be negative, the domain’s priority changes to OVER (indicated by O1 and O2 in Fig. 3). A new enqueued domain will be arranged after all of the other domains of the equal priority in the queue. To reduce the waiting time of a domain, its priority can be set to BOOST. A domain with BOOST priority will preempt the running domain with the priority of UNDER or OVER via the boosting mechanism. It is noteworthy to mention that only the UNDER domain can be set to the BOOST status. For example, if the U2 domain waiting to run within the vCPU queue in Fig. 3 receives an I/O event, the credit scheduler will set the priority of the U2 domain to BOOST. The U2 domain will be scheduled immediately by preempting the running domain. The credit scheduler can provide the approximate fairness for the CPU resources allocation among the entire guest OSes.

3 RELATED WORK

In this section, we present the related work that is subdivided into three categories: (a) scheduler improvement for the inter-VM communication, (b) network optimization for virtualized environments, and (c) guest OS’s process-aware scheduling within the VMM.

3.1 Scheduler Improvement for Inter-VM Communication

Recently, a lot of research has been conducted to improve the scheduler of the Xen VMM by narrowing the semantic gap between the workload of the guest OS and the VMM scheduler [1], [3], [14], [16], [17]. The most closely related work to the proposed CIVSched is Govindan et al.’s communication-aware VM scheduling mechanism [1], which enables the Xen’s Simple Earliest Deadline First (SEDF) CPU scheduler to be aware of the behavior of the hosted applications. However, two important issues remain unaddressed. First, latency-sensitive domains with light traffic are likely to receive less CPU resources. Consequently, such domains can hardly provide any QoS guarantees on latency for the service applications. Second, when there is much network traffic between the local VMs and the external Internet, the network communication between the inter-VMs may not receive a high throughput and/or a low latency. Kim et al. provided a task-aware VMM scheduling [14], which infers the I/O-intensive tasks of the guest OS based on the inference techniques using the gray-box knowledge and correlates the incoming events with the I/O-intensive tasks. Based on the information acquired by monitoring such events, the task-aware VMM scheduling considers a task that is preemptively scheduled in response to an I/O event and consumes short CPU time as an I/O-intensive task. By taking into account the nature of the I/O workload, Kim et al.’s task-aware VMM scheduling bridges the semantic gap between the CPU resources management and the guest-level tasks. Different from Kim et al.’s work, our proposed method focuses on the network I/O-intensive tasks and takes into account the inter-VM communication behavior in the allocation of the CPU resources.

The guest-aware [3] and preemption-aware [16] schedulers reduce the response time of the interactive applications within consolidated VMs by providing the VMM with the priority of the guest-level tasks and the preemption conditions, respectively. The former is only beneficial for the latency-sensitive applications that have a high priority in the guest OS, and do not always ensure that the inter-VM communication process is preferentially scheduled. The latter focuses on the interactive process and depends on the event channels to be pre-known and registered to the guest OS. Starling [17] is a decentralized affinity-aware migration technique to minimize the communication overhead in virtualization platforms. The starling technique monitors the network affinity between the VMs and dynamically adjusts the VM placement with the information of the job communication patterns, and the heterogeneity and dynamism within the network topology. The technique places two heavily data- or communication-dependent VMs as close to each other as possible, such as the same node. However, it does not solve the inter-VM communication latency.
3.2 Network Virtualization Optimization

The network optimization for a virtualized environment has been studied for many years. Kashif et al. performed detailed research on the models and robustness metrics of the data center network [13]. The Xenoprof [18] analyzed the performance overheads incurred due to the network I/O in a virtualization platform and the results showed significant performance degradation within the Xen environment. The aforementioned approach made the packet transmission path more efficient for para-virtualization I/O and optimized the network performance in the Xen hypervisor [19]. A recent extension of the study reported in [18] resulted in the development of the vSlicer [20], which schedules latency-sensitive VMs more frequently by dividing the time slice into smaller micro time slices. The vSlicer enables more timely processing of I/O events and reduces the network packet round-trip times.

Much of the prior work adopts the shared memory buffer between co-located communicating VMs to bypass the virtualized network interface and the network protocol stack, such as XenSocket [8], XWay [9], Fido [10], XenLoop [11], Socket-Outsourcing [12], and IVC [42]. Such solutions achieve high performance without disturbing the isolation of the VMM. However, significantly different from the previous approaches, the solution provided in this paper improves the network response time among inter-VMs from the perspective of the CPU resources scheduling.

In virtualized data centers, the IDD generally receives more CPU capacity because of the special management role. Based on this fact, the vSnoop [21] and vFlood [22] move some important components of the TCP network communications, such as the congestion control and packet acknowledgement, from the DomU to the IDD. Such a change improves the response time and the network traffic throughput.

3.3 Process Aware Scheduling in VMM

The granularity of the research on the hypervisor scheduler mostly focuses on the virtual machine, such as Govindan et al.’s technique [1]. However, without the knowledge of the processes or tasks within the guest OS, it is difficult to timely schedule an application’s process in a server consolidation environment. Therefore, much of the prior research work reports schedulers that take the process information into consideration.

The guest-aware priority-based scheduler [3] collects the information pertaining to the priorities of the processes in each of the VMs and modifies the run-time queue of the VMs to preferentially schedule the domain with the highest effective priority. Because the scheduler only considers the highest priority process in each of the VMs, other processes in the same VM may have to wait for a long time to be scheduled. It is noteworthy to mention that our approach can timely schedule a certain process in the guest OS regardless of the priority of the process. The XenAccess [23] is a library that can be used to transparently gather all of the information pertaining to the processes running within the guest OS. However, the approach adopted by XenAccess cannot affect the process scheduling within the VM. Different from the XenAccess, the Monarch scheduler [24] provides a system-wide process scheduling in the VMM. It can change the process scheduling behavior in all of the domains through the controlled mode. However, the Monarch scheduler breaks the performance isolation of the VMs. The Monarch uses an additional autonomous mode to alleviate the aforementioned issue. However, it is not easy and convenient for the administrators to adjust the ratio of these two modes to make the processes to be scheduled in time. It is also noteworthy to mention that our approach takes into consideration the waiting time of the VMs in the run queue and can effectively guarantee the performance isolation of the VMs.

4 THE SYSTEM MODEL AND PROBLEM DEFINITION

The primary goal of the CIVSched technique is to narrow the semantic gap between the applications running within the VMs and the VMM scheduler. The communication between the VMs can be used to facilitate the scheduling decision of the next executable domain and the corresponding process. In this section, we provide the details on identifying the scheduling delay within the inter-VM communication.

Fig. 4 shows the typical network delay that occurs when a packet is sent from a VM (Dom1) to the process p in another co-located VM (Dom2). The packet delay between the Dom1 and Dom2 can be represented as:

\[ D_p = d1 + d2 + d3 \]  

(1)

The delay \( d1 \) is associated with the scheduling of the IDD. As a crucial domain, the IDD acts as an intermediary between the hardware and the DomUs. The IDD is always assigned more CPU time slices or given exclusive physical CPU core to run [5]. Therefore, the delay \( d1 \) is negligible and can be ignored in (1). However, all of the other unprivileged domains share the same physical CPU core. The VMM scheduler is responsible for picking up a DomU from the run queue and assigning the CPU time slices to the DomU. When the DomU’s time slices expire, the VMM scheduler terminates the running DomU and
picks up another DomU from the run queue. It is highly likely for a DomU to wait in the scheduling queue to gain the CPU time slices. The delay $d_2$ is the waiting time when the Dom2 is scheduled to obtain the CPU resources.

Likewise, all of the processes share the same vCPU inside the DomU. Only when the target process, which is listening on the network TCP/UDP port, is scheduled to run, can the network packet be eventually received by the application. The $d_3$ is associated with this kind of scheduling delay of the target process $p$. There are three scheduling states of the target process $p$.

1. The process $p$ starts to run immediately when the domain is scheduled to run. The delay $d_3$ is nearly equal to 0 and it is ideal for the application to deal with the network packet in time.

2. The process $p$ is scheduled with a delay of time $d_3$, as shown in Fig. 4.

3. The last state is the worst-case scenario. The process $p$ is never scheduled in one of the scheduling time intervals of the domain. That is to say that the delay $d_3$, at times, is longer than the runtime of the VM.

Considering that the delay $d_1$ is very small, a new packet delay can be presented by the following:

$$D_p = d_2 + d_3$$

In (2), the total packet delay between two co-located VMs approximately equals to the sum of $d_2$ and $d_3$. To reduce the latency of receiving the network packet by the target application in the VM, one must keep $d_2$ and $d_3$ as small as possible.

5 Overview of the CIVSched

This section provides an overview of our approach that is developed to reduce the inter-VMs communication latency. Firstly, two design principles that the CIVSched technique must abide by are presented. Thereafter, we provide an overview of the scheduling architecture.

5.1 Design Principles

The design of the CIVSched is based on the following architectural requirements:

(a) **Low latency for the inter-VM.** To reduce the scheduling latency of the target VM and gain short response time between the inter-VMs, the CIVSched must examine all of the network packets transmitted between the co-located VMs, and preferentially schedule the target VMs.

(b) **Low latency for the inner-VM process.** The CIVSched must identify the target process in the target VM for an incoming packet and request the VM’s scheduler to immediately schedule the packet.

These two design principles are developed for the CIVSched to keep the inter-VM latency in the virtualization platform as small as possible. Principle (a) will decrease the delay $d_2$ in (2) while principle (b) decreases the delay $d_3$.

5.2 The CIVSched Scheduling Architecture

Based on the aforementioned desirable requirements, the
6 The Design of the CIVSched

In this section, we will detail the design of the CIVSched. The discussion will encompass all of the key components, such as monitoring of the network data packets, and informing the VMM and DomUs, scheduling the target VM and the target process inside, as well as discovering the co-located VMs and target processes.

6.1 Monitoring of the Network Data Packets

To obtain the characteristics of the network communication between the VMs, the CivMonitor module intercepts all of the outgoing network packets within the IDD (Dom0), which forwards the network packets from all of the VMs to the native network interface card (NIC) driver and then dispatches the reply data packets from the native NIC driver to their target VMs. The CivMonitor module monitors the traffic beneath the network layer. The module only needs to examine the MAC header and the TCP/UDP header of the outgoing data packets. By observing the mapping table, which is a list of entries <DomId, MAC> built by the AutoCover module (detailed in the subsequent section), the CivMonitor module finds out whether the target host of the packet is an inter-VM. If the target VM is co-located, the monitor module will put the DomId of the target VM into a shared ring (Fig. 6) between the Dom0 and the VMM. The VMM checks the ring periodically to read the DomId. In this way, the VMM can be aware of the network communication of the inter-VMs and tune the scheduler to provide better service.

Besides notifying the VMM scheduler of the domain ID, the CivMonitor extracts the destination port number from the TCP/UDP header within the network packet and saves the port number to the XenStore, which is a file system-like database, shared between domains. The Dom0 and Xen hypervisor can read and write all other domains’ entries in the XenStore, while the DomU just has the permission to write and read its own part within the XenStore. The CivMonitor creates a new entry named “destport” under the target VM’s XenStore hierarchy (presented as /local/domain/<DomId>/destport). The DomU will obtain the port number by reading the entry.

6.2 The CIVSched

The network latency of the inter-VM communication arises from: (a) the long wait time of a packet for the target VM and (b) the wait time for scheduling the target process. Intuitively, if the scheduler makes the target VM and process to be scheduled preferentially, then the delay can get alleviated. The CIVSched comprises of two parts the: (a) CivScheduler within the VMM and (b) PidScheduler within the guest OS. The CivScheduler chooses the proper domain when an inter-VM network packet arrives at the back-end driver, while the PidScheduler makes the target process to run in a timely manner to deal with the packet, when the packet arrives at the front-end driver within the domain.

Algorithm 1 presents the core algorithm of the CivScheduler. When the VMM obtains the target VM ID, which is involved in the current communication activity, from the shared ring, the CivScheduler will check the run queue. Once the target VM is found in the queue, it will be moved to the head to preempt the processor while the running domain is enqueued as the next VM to be scheduled. To maintain the fairness, the CivScheduler reorders the run queue but the credit values of the VMs do not change. However, because of the aforementioned procedure, domains may wait for a considerably long time in the run queue. For example, the domain at the head of the queue will not be scheduled when the preemption occurs very frequently. To circumvent this issue, the CivScheduler checks the domain at the head of the queue before the preemption. If the wait time of the domain is larger than a certain threshold, such as 60ms, the target VM must not preempt the CPU. The above mentioned procedure also contributes to the fairness of the CIVSched scheduler.

It seems that what the CivScheduler does is similar to the boosting mechanism within the Xen credit scheduler. We may gather this notion due to the fact that the CivScheduler focuses on the network I/O within the network transactions, and makes the target VMs preferentially scheduled. However, the boosting mechanism responds to all of the I/O events and it rarely aids the light network traffic communication between inter-VMs. We have illustrated the above mentioned phenomenon in the evaluation section 7.2.

It is also noteworthy to mention the issues arising from the underlying system with multiple domains. When there are multiple domains as the targets of the network traffic, the CivScheduler obtains them from the shared ring and schedules them on a first-come-first-served basis.

The PidScheduler algorithm within the guest OS is similar to the CivScheduler in the VMM that we described above. The module PidTrans running in the kernel of the guest OS monitors in real-time its own XenStore entry “destport” mentioned in the previous subsection. When the VMM saves the target port number into the XenStore, the PidTrans translates it into a process PID using the Port

<table>
<thead>
<tr>
<th>Algorithm 1 CivScheduler in VMM (Pseudo code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If not_empty(run_queue) and not_empty(shared_ring) then</td>
</tr>
<tr>
<td>/<em>get destination DomId from the shared ring</em>/</td>
</tr>
<tr>
<td>dest_domid = get_from_ring();</td>
</tr>
<tr>
<td>If running_domid != dest_domid then</td>
</tr>
<tr>
<td>if dest_domid in run_queue then</td>
</tr>
<tr>
<td>/<em>if other domain waits too long, don’t preempt</em>/</td>
</tr>
<tr>
<td>if run_queue_head_dom_wait_time &lt; threshold then</td>
</tr>
<tr>
<td>/<em>preempt the running domain</em>/</td>
</tr>
<tr>
<td>move dest_domid to the head of the run_queue;</td>
</tr>
<tr>
<td>insert running_domid behind dest_domid;</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end if</td>
</tr>
</tbody>
</table>

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
to PID translation mechanism (detailed later). Thereafter, the PidTrans notifies the VM’s process scheduler of the target process through a sharing data ring. Once the process scheduler finds the target process PID within the shared ring, it will search the run queue, and then immediately schedule the target process. The PidScheduler also checks the process at the head of the run queue to ensure that the process will not wait too long. (The usual delay is kept under 60ms.) With the cooperation of the schedulers within the VMM and VM, the CIVSched can effectively reduce the delay time of a packet to be dealt with by the application within the VM.

6.3 Discovering Co-resident VMs/Target Processes

The CIVSched employs a domain automatic discovery mechanism that discovers the co-located VMs and maintains a mapping table of the domain IDs and the corresponding MAC addresses. In the Xen hypervisor, whenever a new VM is created, its configuration information, such as domain ID and MAC address, is registered within the XenStore. Likewise, when a VM stops running or is destroyed, the corresponding information in the XenStore is also destroyed. The AutoCover module within the CIVSched is responsible for periodically checking the XenStore. It collects all of the domain IDs and the MAC addresses, and then combines them as the identity pairs <DomId, MAC> to compose the mapping table.

We design a translation mechanism to translate the socket port number extracted from the network packet to the PID of the process, which is listening on the port, awaiting packets. The procs (proc file system) is a special file system within UNIX-like OS that provides the kernel application program interface (API) to gather information about the processes and other system information. All of the information gathered is presented as a file-like structure. We analyze the readily available information that is provided by the procs file system. In the Linux OS, such information is located in the directory /proc. Our PidTrans module, firstly, traverses the two files /proc/net/tcp and /proc/net/udp to find out all of the current network ports that are in use. Secondly, the module traverses through every file within the directory /proc/<pid>/fd to obtain information pertaining to the sockets, namely the interfaces associated with the port numbers that are used by the processes to connect to the network. Lastly, we can obtain the mapping table between the port number and the PID. The PidTrans can easily translate a port number to the listening process’s PID with the mapping table. Meanwhile, the mapping table is updated in a certain time interval with the purpose of gaining the latest statuses of the platform.

Live migration is one of the most attractive features provided by the VM technologies. It is not difficult to support live migration within the CIVSched. When a VM migrates to another CIVSched supported physical platform, the corresponding <DomId, MAC> entry is removed from the local mapping table. Under the domain automatic discovering mechanism, a new <DomId, MAC> entry associated with the VM will be added into the new physical platform’s mapping table. The network communication between the migrating VM and the VMs on the original physical platform is beyond this paper’s inter-VM problem.

7 IMPLEMENTATION AND EVALUATIONS

7.1 Implementation

A large category of resource scheduler work in VMM initially focuses on the server that has a single physical CPU core to study their proposed approach, such as Govindan et al.’s XenandCo scheduler [1] and Cong et al.’s vSlicer [20]. In such a single CPU core environment, it is feasible to observe the performance optimization without considering the complexity raised by the multi-processor system. To study the proposed scheduler could effectively reduce the inter-VM communication latency in a consolidation environment, we make an assumption that the server has only one physical CPU core. That is to say that all of the co-located VMs must share the CPU core. Extending our CIVSched scheduler to multi-processor is part of the future work.

We have implemented the CIVSched based on the Xen 4.1.2 hypervisor and its default credit scheduler. The portion within the guest OS is developed based on the Fedora platform with Linux kernel 2.6.18.8. The Xen hypervisor provides a data structure shared_info, which is mapped into a shared memory page between each domain and the VMM at runtime. We placed our implemented shared ring within the above mentioned shared page so that the VMM scheduler can easily fetch the scheduling information. As mentioned in the Xen I/O mechanism section, every front-end driver has an associated back-end driver. When multiple domains are running simultaneously, there are more than one back-end drivers accessing the shared ring (Fig. 6). Therefore, an atomic lock for writing into the shared ring is adopted within the CIVSched. Considering the efficiency of looking up the mapping table, we keep the table into a memory page instead of a file in the disk. To discover all of the running co-located domains in time without introducing much performance degradation, the frequency for the AutoCover module to check the XenStore is set to once every 1s.

For the portion that takes place within the guest OS, the XenStore entry /local/domain/<DomId>/destport will be set to 0, whenever the PidTrans module finishes reading the port number from the entry. Consequently, the CIVSched will not read the old port value more than once. The CivMonitor and the PidTrans modules write into the same XenStore entry using the Xenbus interfaces,
which can ensure the atomic write operation on the Xen-Store entry. The mapping table of the port number and the process PID is also kept within the guest OS’s memory and updated every 1s.

In the later subsections, we will present several experiment evaluations of the CIVSched. We adopted a small scale cluster with five nodes as our experiment environment. Each node employs the Dell Optiplex 960 with four Intel Core 2 Duo 2660 MHz processors, 8GB main memory, and Intel integrated graphics card. All of the nodes are interconnected through 10 Gigabit Ethernet network. We chose the Xen VMM 4.1.2 as the virtualization platform and the Linux kernel 2.6.18.8 as the Linux Dom0. Each platform can run up to seven DomUs and all of the DomUs are equipped with the Linux kernel 2.6.18.8. We installed the VMM, the Linux Dom0, and our CIVSched scheduler on all of the nodes in the cluster. We made one of the nodes as our primary virtualization platform to run the tests. On the primary test platform, Dom1 and Dom2 are used to measure the communication latency. Besides the Dom1 and Dom2, there are five other domains with synthetic CPU- and network I/O- intensive workloads. We adopted a Pi computing program as a CPU-intensive workload, while using the netperf [35] tool and the ping command to access the external Internet websites as a network I/O-intensive workload. Each DomU is configured with one vCPU and 1GB memory while Dom0 is configured with four vCPUs and 1GB memory. Because we assumed that all VMs shared a single CPU core, we pinned all of the DomUs’ vCPUs and one of the Dom0’s vCPUs to a single physical CPU core using the application interfaces provided by the Xen hypervisor in all the evaluations. Meanwhile, the other three vCPUs of Dom0 were pinned to the other three physical CPU cores. The aforementioned configuration ensures that the Dom0 gets enough of the CPU resources.

As a comparison, we implemented Govindan et al.’s solution (XenandCo scheduler) based on our Xen hypervisor version and the Xen credit scheduler. To achieve the best performance in the credit scheduler, we let the XenandCo scheduler operate under the same previously defined strategy, which was to preferentially schedule the VM with the most network packets. Moreover, for fair comparisons, we performed the same network latency experiments on the XenandCo scheduler.

7.2 Network Latency

Three types of experiments were performed to study the network latency that is decreased by the CIVSched. These experiments are composed of a ping-pong test, a simulation test using the netperf [35] tool, and a real-world Web application scenario using an Apache Web Server [37] but with synthetic workloads. The real-world Web application scenario runs a Web service and the Apache Bench [36] is used to demonstrate the performance of the HTTP connectivity.

We performed the ping-pong test in a client-server environment [3] to observe the inter-VM network communication latency in a highly consolidated environment. In the ping-pong experiment, we mandated that Dom1 and Dom2 communicate with each other and recorded the response time with various consolidated scenarios. We ran the domains Dom3 through Dom7 one by one to simulate various consolidated scenarios. The Dom3-Dom7, concurrently ran the CPU and the network I/O workloads. The ping-pong test was achieved by running the ping command in the Dom1 (client) to continuously send small requests to the destination host, Dom2 (server) and receive the response messages. The CIVSched can capture the network packet and extract the destination MAC address. Subsequently, the target VM is identified and preferentially scheduled. The frequency of the packets sent from Dom1 (client) to Dom2 (server) was approximately one hundred packets per second and the size of each packet was set to 64 bytes. In each of the consolidated scenarios, we ran the ping-pong test five times, and one minute at a time. The results of the average I/O response time are reported in the Fig. 7. The boosting mechanism is enabled in the Xen’s default credit scheduler, but with the increase in the number of consolidated domains, the network latency between Dom1 and Dom2 increases under the credit scheduler. Compared to the Xen’s default credit scheduler, the CIVSched reduces the delay of communication by 14.0%-17.7% even in highly consolidated environment, such as when there are simultaneously seven VMs running. Fig. 7 also depicts the CIVSched exhibits lower network latency compared to the XenandCo scheduler. However, we also found that the network latency still increased with the increase in the number of the consolidated VMs under the CIVSched scheduling. It is due to the fact that the CIVSched only schedules one target VM at a time. When more target VMs are in the run queue, the latency is bound to increase for the reason that the target VMs wait to be scheduled one by one. We believe that the above mentioned issue will be further alleviated in the multi-core platform where more than one target VMs would be scheduled to different physical cores.

As mentioned above, the ping-pong test was performed by using the Linux ping command. The ping command sends and receives the Internet Control Message Protocol (ICMP) network data packet, which does not contain the source and destination port numbers. Without the destination port numbers, the modules PidTrans and PidScheduler running in the DomU do not take effect. Only the CioScheduler running in the VMM can
preferentially schedule the target VM in the ping-pong test. In that case, the ping-pong test only depicts the performance improved at the VM level. The performance does not include the part of preferentially scheduling the target process in the guest OS, which is performed by the modules PidTrans and PidScheduler. To determine the holistic performance improved by our CIVSched technique, we need other experiments to transfer TCP/UDP packets between Dom1 and Dom2.

We use netperf version 2.6.0 as our primary benchmark to measure the performance of the CIVSched in various types of network environments, such as the TCP and UDP. The netperf tool provides tests for both unidirectional throughput and end-to-end latency [35]. There are two different tests in netperf for simulating the: (a) database transactions (TCP_RR test) and (b) HTTP connection service (TCP_CRR test). The TCP_RR test in netperf assesses multiple request and response transactions in one TCP connection. It can be used to simulate the database transaction application. Another test TCP_CRR provided by the netperf is used to test the establishment of TCP connections. The TCP_CRR is similar to the HTTP connection establishment. We ran the server-side of the netperf in Dom2 while the client-side in Dom1. Thereafter, the client sent the request data packets to the server and received the response data packets from the server. To make the network traffic in the simulation more realistic, the size of the request and the response data between the client and server was set to 128 bytes in the TCP_RR test while the size was set to 32 bytes in the TCP_CRR test. Each of the tests lasted for 120 seconds. The average round-trip latency can be obtained by inverting the transaction rate. We examined the network latency in the TCP_RR test and the TCP_CRR test with various number of consolidated domains, which were running with external network I/O workloads and CPU workloads.

Fig. 8 reports the results that show that the CIVSched decreases the latency by up to 16%, when compared to the Xen credit scheduler. We also compared the CIVSched with Govindan et al.’s XenandCo scheduler. The results clearly depict that the CIVSched has a better performance on the network latency than the XenandCo scheduler. The rate of the performance growth in Fig. 8 also shows that the network latency of the inter-VM communication is not well guaranteed in Govindan et al.’s scheduler. The aforementioned can be attributed to the fact that the network latency in the XenandCo scheduler at times is higher than the credit scheduler, which is represented by a negative number as the rate of the performance growth, in Fig. 8. From Fig. 8(a) and Fig. 8(b), we can also observe that the rate of the performance growth increases in the CIVSched scheduler with the increase in the number of

Fig. 9. HTTP Throughput and Request Numbers

Fig. 8. The network latency in netperf benchmark.

Fig. 9. Average Latency of HTTP Request

HTTP request measurement with Apache Bench.
simultaneously running domains. In our experimental evaluations, when the number of co-located VMs was seven, we obtained the biggest performance increase. The decrease in the network latency was 15.87% (Fig. 8(a)) and 16.23% (Fig. 8(b)) in TCP_RR and TCP_CRR test cases, respectively. The two numbers are much higher than the amounts of decrease (no more than 5%) when there were only two or three consolidated VMs. The aforementioned can be attributed to the fact that there were more opportunities to identify the target VM in the run queue and move it to the head of queue, when the system was heavily consolidated.

To record the performance of the proposed CIVSched technique in a real-world Web application scenario that has heavy synthetic network traffic, an Apache Web Server of version 2.2.6 was configured with the Apache Bench (AB) 2.0.40. The AB was used to simulate multiple concurrent browser clients accessing the application, and then measure the HTTP throughput and request latency. The Apache Web Server was running in the Dom1 while the other DomUs were allowed to access the Web Server. We ran the Apache Bench within the Dom2. The rest of the DomUs, namely Dom3-Dom7, were booted one by one and ran the network I/O and the CPU workloads. To be as close as possible to real-world scenarios, we ran the Apache Bench within the Dom2 to concurrently send a large number of HTTP requests to the Apache Web Server. The number of the concurrent requests was from 1,000 to 5,000 every time, and lasted for 50 times. The network traffic in the Dom3-Dom7 was configured to send and receive hundreds of packets one second. Fig. 9(a) reports that the CIVSched obtained a slightly improved throughput of approximately 14%. Meanwhile, the number of requests per second was larger than that reported by the credit scheduler. The requests increased by approximately 20% in one second. Most importantly, Fig. 9(b) shows the average amount of time it took for a single request to be processed by the Apache Web Server. For every HTTP request, the average time (latency) decreased by approximately 19% within the CIVSched, when the environment is heavily consolidated. When compared to the XenandCo scheduler, the CIVSched also reported a better performance on the HTTP throughput, the HTTP requests per second, and the average network latency for the HTTP connection application that is sensitive to high latency.

7.3 Fairness Guarantees

The fairness of the VMM scheduler directly affects the fairness of the CPU resources allocation among all of the running domains. We evaluated our proposed CIVSched technique to ensure that it does not incur significant overhead to the CPU fairness. In this experiment, we kept all of the seven domains (Dom1-Dom7) running with the same CPU- and network I/O- intensive workloads. We mandated that Dom1 continuously sends small requests to Dom2 with a frequency of approximately one hundred packets per second and a packet size of 64 bytes using the ping command. We obtained the CPU usage of each VM using the xentop [38] tool. The xentop tool provides detailed information about the CPU utilization of every guest OS. Fig. 10 depicts the CPU usage of each of the VMs every second of time, up to about 200s, as well as the average CPU utilization of each of the VMs in the same time. The result demonstrates that our proposed technique is able to guarantee the CPU fairness of the Xen hypervisor scheduler.

7.4 Performance Overhead

The UnixBench suite 4.1.0 was adopted to evaluate the host’s performance overhead affected by our proposed CIVSched technique. The purpose of UnixBench is to provide a basic indicator of the performance of a Unix-based system [31]. The tests in the UnixBench suite focus on the various aspects of the OS functionality, such as process spawning, inter process communication, file system throughput, floating point arithmetic, and system calls.

We measured the performance overhead at two different levels of the consolidated environment. The first level is the light consolidation and there are only two simultaneously running VMs (Dom1 and Dom2). The second level is the heavy consolidation and there are seven simultaneously running VMs (Dom1-Dom7). In both of the environments, we mandated that Dom1 sends network packets to Dom2 using the ping command (at a frequency of approximately one hundred packets per second). We also mandated all of the domains keep running the CPU- and network I/O- intensive workloads. We ran the UnixBench to obtain the index values of the different aspects of the OS functionality under the Xen’s default credit scheduler and our proposed CIVSched scheduler, respectively. Fig. 11 shows the relative performance as a ratio of the CIVSched to the credit scheduler. Each index value is the average of five trials. Most of the data points demonstrate that the overhead of our prototype is small except for the file copy test, which exhibits at the most 18% overhead (see Fig. 11(a)). The file copy test measures the rate at which data can be transferred from one file to another [31]. The file read, write, and copy tests require high disk throughput. The module AutoCover within the CIVSched, running in Dom0 also requires disk throughput to frequently read the XenStore that is a file system-like a database. Consequently, the file copy operation will not get enough I/O bandwidth to gain a high throughput. At last, the performance of the file copy operation is affected by the CIVSched. However, when compared to the
default credit scheduler, each of the index values of other tests are no more than 7% (Fig. 11(a)). The number means the overhead is very small. The holistic performance of a system is measured by a final score in UnixBench [31]. A higher final score means the performance of the system is better. We also provided the final scores that were got under the CIVSched and credit schedulers, in Fig. 11(a) and Fig. 11(b). Comparing the two final scores, we can see that the overhead of CIVSched is smaller in the heavily consolidated environment (99%) than that in the light consolidated environment (95%). It is because that the overhead introduced by the simultaneously running VMs is much more than that introduced by the CIVSched technique.

7.5 Further Discussion

Our research was focused on improving the response time of the inter-VM network communication in a consolidated virtualized environment from the perspective of the CPU resources scheduling and allocation. The design principle behind the CIVSched was to timely schedule the target VM and process of the network traffic to reduce the wait time of the network packets. The CIVSched was implemented based on the Xen virtualization platform and the Xen’s default credit scheduler. However, the central idea described in this research, on how to impose an upper bound to virtual network, is also highly suitable for many other virtualization platforms, such as Kernel-based Virtual Machine (KVM) [25] and VMware ESX Server [39].

Though the proposed CIVSched technique mainly focused on the communication latency between the colocated inter-VMs, it can also be used to decrease the network communication latency when both communicating VMs are located on different virtualized host. For convenience, we call the latter situation as the external network communication. In this situation, wherever the source packet comes from, VMM can timely schedule the target VM and the target process to reduce the amount of time in dealing with the network packets. As a result, the network communication latency is mitigated. The performance of the external network communication under the CIVSched technique is beyond this article’s scope.

The current implementation of the CIVSched scheduler needs a little modification to the kernel of the guest OS. It will be removed when the transparent monitoring and controlling mechanism [23], [24] is adopted in the VMM scheduler. An important issue pertaining to the CIVSched is that the research was performed based on a single CPU as all of the DomUs shared one CPU core. The multi-core environment was not involved in our current research. However, extending the current work to the multi-core domain will not be infeasible and we will pursue it as the subsequent work.

8 CONCLUSIONS AND FUTURE WORK

This paper proposed a novel communication-aware scheduler based on the Xen virtualization platform, called the CIVSched. The CIVSched makes the scheduling decision by taking into account the co-located target VMs and processes of the network traffic for the purpose of providing better QoS for the latency sensitive application running in the virtualization platform. The back-end driver within the Xen PV mechanism monitors the network packets and passes the communication pattern of the inter-VMs to the VMM scheduler and the guest OS’s process scheduler, respectively. The two schedulers cooperatively work to reduce the wait time of network packets to be processed by the application. Consequently, the CIVSched technique decreases the response time of the network communication between two inter-VMs. The CIVSched technique adopts an automatic discovery mechanism to identify the inter-communication packets and applies a Port to PID mapping mechanism to translate the port number extracted from a TCP/UDP network packet to a PID of the corresponding target process. The comparative evaluations depicts that the CIVSched can reduce the response time of the inter-VM communication while maintaining the fairness of the VMM scheduler.

The current CIVSched technique only studies the situation when all of the DomUs are pinned to one physical CPU core. However, multi-core architecture is being adopted very rapidly, nowadays. We will build on our current work to develop an implementation of the CIVSched, which can adapt to the multi-core environment. Besides, we will provision for the CIVSched to
support several kinds of VMs, such as the PV guest OSes and the HVM guest OSes, within the VMM. In the meanwhile, further improvements will be investigated to reduce the complexity of the DomU, such as making the CIVSched technique transparent to the guest.

**ACKNOWLEDGMENTS**

This work is jointly supported by the National Science and Technology Major Project under Grant No.2012ZX01039-004, the National Natural Science Foundation of China under Grant No.61170072, No.61303057, the Knowledge Innovation Key Directional Program of Chinese Academy of Sciences under Grant No.KGCX2-YW-174, No.KGCX2-YW-125, and the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No.XDA06010600.

**REFERENCES**


