Future Generation Computer Systems 56 (2016) 229–237

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient consolidation-aware VCPU scheduling on multicore virtualization platform

Bei Wang [a], Yuxia Cheng [a], Wenzhi Chen [a], Qinming He [a], Yang Xiang [a,b,*], Mohammad Mehedi Hassan [c], Abdulhameed Alelaiwi [c]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[b] School of Information Technology, Deakin University, Melbourne, Australia
[c] College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

HIGHLIGHTS

- We propose an efficient consolidation-aware vCPU (CVS) scheduling scheme.
- The CVS scheduling scheme adaptively selects three vCPU scheduling algorithms.
- The CVS scheme can effectively improve virtual machine performance.
- The CVS scheme works in different consolidation scenarios.

ARTICLE INFO

Article history:
Received 20 January 2015
Received in revised form
28 July 2015
Accepted 16 August 2015
Available online 28 August 2015

Keywords:
Multicore
Virtualization
Lock holder preemption
vCPU scheduling
Consolidation

ABSTRACT

Multicore processors are widely used in today's computer systems. Multicore virtualization technology provides an elastic solution to more efficiently utilize the multicore system. However, the Lock Holder Preemption (LHP) problem in the virtualized multicore systems causes significant CPU cycles wastes, which hurt virtual machine (VM) performance and reduces response latency. The system consolidates more VMs, the LHP problem becomes worse. In this paper, we propose an efficient consolidation-aware vCPU (CVS) scheduling scheme on multicore virtualization platform. Based on vCPU over-commitment rate, the CVS scheduling scheme adaptively selects one algorithm among three vCPU scheduling algorithms: co-scheduling, yield-to-head, and yield-to-tail based on the vCPU over-commitment rate because the actions of vCPU scheduling are split into many single steps such as scheduling vCPUs simultaneously or inserting one vCPU into the run-queue from the head or tail. The CVS scheme can effectively improve VM performance in the low, middle, and high VM consolidation scenarios. Using real-life parallel benchmarks, our experimental results show that the proposed CVS scheme improves the overall system performance while the optimization overhead remains low.

1. Introduction

Multicore processors are commonly deployed in computer systems from high-end servers to power efficient embedded devices. With the increasing number of processing cores integrated into the system, how to efficiently utilize the multicore processing power becomes a big challenge. To more flexibly utilize physical resources, virtualization technology is widely used in today's cloud computing environment. Virtualization technology enables multiple virtual machines (VMs) concurrently run in one physical machine, which provides an elastic resource provisioning method.

However, multiple VMs consolidated into the same physical machine will contend for shared CPU resources [1]. In a typical VM consolidation scenario, one physical core usually has multiple virtual CPUs (vCPUs) running on it (named vCPU overcommitment) [2]. The virtual machine monitor (VMM), which provides a virtual abstraction of machine hardware for each guest operating system (OS), schedules the vCPUs based on their time slice and priority [3]. The VMM has little awareness of the code being executed inside each vCPU [4]. Therefore, the vCPUs that are holding

* Corresponding author at: School of Information Technology, Deakin University, Melbourne, Australia.
E-mail addresses: wangbei@zju.edu.cn (B. Wang), rainytech@zju.edu.cn (Y. Cheng), chenwz@zju.edu.cn (W. Chen), hqm@zju.edu.cn (Q. He), yang.xiang@deakin.edu.au (Y. Xiang), mmhassan@ksu.edu.sa (M.M. Hassan), aalelaiwi@ksu.edu.sa (A. Alelaiwi).

http://dx.doi.org/10.1016/j.future.2015.08.007
0167-739X/

spinlocks may be preempted [5] by other vCPUs due to the VMM's vCPU scheduling. The vCPUs that are waiting for the spinlocks have to spin for a much longer time when the vCPUs holding the locks are preempted. This is known as lock holder preemption (LHP) problem [6–9], which decreases VM performance and reduces system scalability.

In the non-virtualized environment, the LHP problem can be avoided [10,11] by preventing the lock holder from being preempted until the lock holder releases the lock. The spinlock is designed to wait for a very short time and is used in the situations where context switches to yield CPU time slices are deemed as more costly than spinning. The lock holder can be easily detected by the native OS kernel. But when the OS is running inside a VM, the VMM cannot easily figure out whether the vCPU is holding the spinlock.

To address the LHP problem, researchers have proposed multiple solutions. The VM co-scheduling technique [5,12] was proposed to simultaneously co-schedule all or part of the vCPUs in one VM onto physical cores to avoid the LHP problem. However, co-scheduling has some weaknesses [13,24], such as CPU utility fragmentation [14,26] and increased system latency. Para-virtualization technique provides another solution to address the LHP problem by modifying the spinlock primitives in the guest OS to cooperate with the underlying VMM [4,15,16]. In this way, the VMM can detect the lock holder and avoid preempting the vCPU that are holding the spinlock. But modifying the primitive spinlock cannot be easily applied in those proprietary OSes such as Windows and Mac OS X [17].

The precise lock holder detection in the full virtualization environment is currently not available [1,8]. An alternative hardware assisted technology was introduced to approximately detect the lock waiter, such as Intel PAUSE Loop Exiting (PLE) [18] and AMD PAUSE Filter (PF) [19]. The PLE/PF mechanism is designed to detect the guest OS execution of the PAUSE instruction that is used in the spin loop code. By detecting the potential lock waiter, the VMM can then make the lock waiter vCPU yield its CPU cycles and donate them for other vCPUs [17]. Therefore, the overall system throughput can be improved. However, the vCPU yield policy can lead to unfairness for the donating vCPUs and impact their response latency [7,8].

In this paper, we propose an efficient consolidation-aware vCPU scheduling (CVS) scheme that considers different VM consolidation scenarios. The CVS scheme takes advantages of the low overhead hardware assisted PLE mechanism to detect lock waiter vCPUs. Based on vCPU over-commitment rate, the CVS scheduling selects one algorithm among three vCPU scheduling algorithms: co-scheduling, yield-to-head, and yield-to-tail. The main contributions are described as follows:

(1) We evaluate three vCPU scheduling algorithms in the multicore virtualized system under different VM consolidation scenarios. We observe that different scheduling algorithms have performance advantages and disadvantages under different vCPU over-commitment rate settings. We analyze the behavior of three vCPU scheduling algorithms and find the most suitable algorithm in a certain VM consolidation scenario.
(2) After analyzing the characteristics of three vCPU scheduling algorithms, we propose an efficient consolidation-aware vCPU scheduling (CVS) scheme that adaptively selects the most suitable vCPU scheduling algorithm online based on the vCPU over-commitment rate. The CVS scheme dynamically changes vCPUs scheduling strategies according to CPU run queue status and the positions of the lock waiter vCPU and the lock holder vCPU. The CVS can more effectively address the LHP problem and improve VM performance than the single policy vCPU scheduling algorithms.

Table 1
The properties of three algorithms.

Algorithm	YTT	YTH	Co-sched
Overhead	Low	Low	High
Performance	Low	Mid	High
Degradation	Low	Mid	High

(3) We extensively evaluate the proposed CVS scheme in the multicore virtualized platform. We run the real-life parallel benchmarks in the multicore VMs. The evaluation results demonstrate that our proposed CVS scheme can effectively improve the overall VM performance, and the optimization overhead remains low.

The rest of this paper is organized as follows. Section 2 describes the motivation of this paper and shows the experimental observations. Section 3 presents our proposed consolidation-aware vCPU scheduling scheme in detail and describes its implementation in the Xen virtualized platform. Section 4 shows the evaluation results. Finally, we conclude this paper in Section 5.

2. Motivation

The lock holder preemption (LHP) problem in the virtualized environment can be mitigated via three different vCPU scheduling algorithms: co-scheduling, yield-to-head (YTH), and yield-to-tail (YTT), because the actions of vCPU scheduling are split into many single steps such as scheduling vCPUs simultaneously or inserting one vCPU into the run-queue from the head or tail.

The co-scheduling algorithm, shown in Fig. 1(a), schedules all vCPUs of a VM synchronously on physical cores. The YTH algorithm detects the lock waiter vCPU and inserts it to the head of the corresponding CPU run queue. The YTH algorithm can be illustrated with Fig. 1(b). The lock waiter vCPU yields its CPU time slice to other vCPUs and waits in the head of CPU run queue for the next round. The YTT algorithm is the default scheduling methods for Xen Hypervisor. As shown in Fig. 1(c), YTT detects the lock waiter vCPU and inserts it to the tail of the corresponding CPU run queue. In the YTT algorithm, the lock waiter vCPU also yields its CPU time slice to other vCPUs but waits in the tail of CPU run queue for the next round.

From the analysis above, we fill Table 1 with the properties of three algorithms. Table 1 shows that three algorithms have their own advantages and disadvantages. To study the performance characteristics of these algorithms, we implemented three algorithms in the Xen hypervisor and evaluated their performance using parallel benchmarks. We run 4 VM guests with 3 GB simultaneously and concurrently using the Kernbench [20] as the workload. Each VM guest has the same number of vCPUs. We gradually increase the number of vCPUs in each guest to increase the probability of the LHP problem. Since only the number of CPUs or VMs could not reflect the contention for CPU resources between VMs, we consider the vCPU over-commitment rate (VOR) to represent the current system's VM configurations. VOR is calculated as the following equation:

$$\mathrm{VOR} = \frac{\sum_{i=1}^{n} VN(vm_i)}{Total\ Number\ of\ pCPUs}, \tag{1}$$

where $VN(vm_i)$ represents the number of vCPUs in the ith VM. For example, if the system has 8 physical CPUs (pCPUs) and 4 VMs with each configured with 2 vCPUs run on the system, then the VOR value equals to $(2 + 2 + 2 + 2)/8$. The higher the value of VOR, the higher probability the LHP problem occurs.

Fig. 2 shows the total execution time of the Kernbench running inside 4 VMs under three vCPU scheduling algorithms: YTT, YTH,