*Journal of Applied Science and Engineering, Vol. 19, No. 3, pp. 347–356 (2016)* *DOI: 10.6180/jase.2016.19.3.13*

# Parallel Processing-Oriented Hybrid Scheduling of Virtual Machines in Cloud

Haibao Chen, Yuyan Zhao*, Shenghui Zhao and Guilin Chen

*School of Computer and Information Engineering, Chuzhou University, Chuzhou 239000, P.R. China*

## Abstract

In cloud computing environment, parallel applications generally run on symmetric multiprocessing (SMP) virtual machine (VM). Since this type of application requires synchronous operations between processes/threads, all virtual CPUs (vCPUs) of a parallel VM (i.e., the VM running parallel application) should be online simultaneously. At present, relevant studies have been intensively conducted from the perspective of vCPU co-scheduling in virtual machine monitor (VMM). However, the existing co-scheduling methods have the problems of unrestricted preemptions between parallel VMs, which probably results in negative impact on the performance of parallel applications in these VMs.

To address the above problems, in this paper, we first analyze the deficiencies of the existing co-scheduling approaches in virtualized environment. Then we propose an enhanced co-scheduling algorithm to improve the performance of parallel application in SMP VM.

***Key Words*:** Parallel Processing, Virtualized Environment, Co-scheduling, Cloud

## 1. Introduction

Virtualizationcan support efficient use of physical resources. A virtualized system mainly consists of virtual machine (VM) and virtual machine monitor (VMM). Typical VMMs include VMware ESXi, Xen and KVM. A VM, like a real physical machine, can run different types of applications. Generally, system administrators and users can set hardware environments for a VM, such as the number of virtual CPU (vCPU), the size of disk, and the size of memory.

In a virtualized system, the VM with multiple CPUs is called symmetric multiprocessing (SMP) VM. The difference between a SMP VM and a physical one with symmetric multiprocessor lies in that all vCPUs of the former will not be always online (i.e., occupying physical CPUs), and probably be online asynchronously. Generally, when a SMP VM running an application, the VMM will asynchronously schedule all of its vCPUs, on the premise of fair sharing for physical CPUs. This approach has been prevalently applied to VMMs (e.g., Xen) as it can simplify the CPU scheduling in VMMs while maintaining the performance of system. However, when a VM running parallel applications (e.g., multithreaded programs with synchronous operationsor parallel programs), the existing CPU scheduling approach used by VMM may result in some problems which do not exist in non-virtualized environment, thus degrading the performance of parallel applications. For example, a spinlock (used for kernel synchronization) in the non-virtualized environment will be held only for a short time and cannot be preempted. However, the spinlock held by VM can be preempted because of the preemption of vCPU [1], which greatly increases synchronization delay and potentially impedes the progress of other vCPUs waiting to obtain this spinlock. Therefore, the preemption of vCPU will considerably degrade the performance of parallel applications running in SMP VM.

*Corresponding author. E-mail: zhyy@chzu.edu.cn

At present, many studies have been carried out from the perspective of the CPU scheduling in VMM. For example, researchers have presented a hybrid scheduling method suitable for virtualized environment [2], which adopts the co-scheduling strategy for all vCPUs of a parallel VM (i.e., the VM running parallel application) and asynchronous scheduling strategy (e.g., Xen's Credit Scheduling [3,4]) for vCPUs of the non-parallel VM (i.e., the VM running non-parallel applications). However, the co-scheduling strategy will lead to unrestricted mutual preemption between parallel VMs. Thus all vCPUs of the preempted parallel VM cannot be online simultaneously, causing negative impact on parallel applications in these VMs.

In this paper, we first analyze the deficiencies of the existing co-scheduling methods in virtualized environment. Then we propose an enhanced hybrid scheduling method to address the low ratio of all vCPUs online simultaneously in parallel VM. At last, we implement a prototype system based on Xen3.2. Experimental results demonstrate that our method achieves a better performance of parallel applications in SMP VM compared with representative methods such as Xen's Credit scheduling, hybrid scheduling, and balance scheduling.

The contributions of this paper are two-fold:

- We deeply analyze the disadvantages of the existing co-scheduling methods in virtualized environment.
- We present an enhanced hybrid scheduling method, which can be used to optimize the performance of parallel application running in SMP VM.

This paper is organized as follows. Section 2 analyzes the problems of existing approaches. Section 3 introduces an enhanced hybrid scheduling method, which is evaluated in section 4. The related work is discussed in section 5. Section 6 concludes this paper.

## 2. Analysis of Problems

Different VMs in virtualized environment may run different types of workloads (parallel or non-parallel ones). In this paper, the VM running parallel workloads is called parallel VM, and that with non-parallel workloads is called non-parallel VM. Generally, asynchronous scheduling is a suitable strategy for non-parallel VMs and also the default strategy adopted by open-source VMMs, e.g., Xen and KVM [5]. Co-scheduling strategy (i.e., all vCPUs of a VM are simultaneously scheduled to different physical CPUs) fits the parallel VMs better. Hence, when a system contains both parallel and non-parallel VMs, it is a rational choice for VMM to choose scheduling strategy according to the type of VM, which is the common hybrid scheduling approach used by researchers [2,6,7] to optimize the performance of parallel applications in VMs.

With the above methods, all the vCPUs of parallel VM will be co-scheduled to physical CPUs instantly, without consideration about the physical CPUs occupation by vCPUs of other VMs. In this section, a simple scenario of four physical CPUs and five VMs is introduced to describe co-scheduling. The configuration of five VMs (labeled as vm0, vm1, vm2, vm3, vm4, and vm5) and the type of applications running on them are listed in Table 1.

Take $vCPU_{12}$ in Figure 1(a), which is running in CPU2, as an example. When the scheduling period of CPU2 comes, the $vCPU_{12}$ will be de-scheduled from CPU2 to its runqueue by the scheduling program residing in CPU2, and $vCPU_{31}$ at the front of its run queue will be scheduled to occupy CPU2. As parallel applications run on vm3 to which the $vCPU_{31}$ belongs, $vCPU_{30}$ and $vCPU_{31}$ should be scheduled simultaneously according to the existing co-scheduling strategy.

Since $vCPU_{30}$ is in the run queue of CPU1, the scheduling program residing in CPU2 needs to send CPU1 an inter-processor interrupt (IPI) signal to inform that

**Table 1.** The configuration of VMs and the type of applications

| VM | The number of vCPU | Identifier of vCPU | The type of application |
|---|---|---|---|
| vm0 | 1 | $vCPU_{00}$ | No-parallel application |
| vm1 | 3 | $vCPU_{10}$, $vCPU_{11}$, $vCPU_{12}$ | No-parallel application |
| vm2 | 4 | $vCPU_{20}$, $vCPU_{21}$, $vCPU_{22}$, $vCPU_{23}$ | No-parallel application |
| vm3 | 2 | $vCPU_{30}$, $vCPU_{31}$ | Parallel application |
| vm4 | 2 | $vCPU_{40}$, $vCPU_{41}$ | Parallel application |

the scheduling program of CPU1 should compulsorily replace the $vCPU_{22}$ occupying CPU1 with $vCPU_{30}$, as illustrated in Figure 1(b). If the preempted vCPU is running non-parallel applications, the performance of these applications will probably not be affected. However, if the preempted vCPU is running parallel application, the application performance will probably degraded due to the possible lock holder preemption.

Figure 2 gives a simple description of lock holder preemption. In Figure 2(a), when the scheduling period of CPU3 comes, the scheduling program residing in CPU3 will schedule $vCPU_{41}$ at the front of run queue to replace $vCPU_{23}$ which is running in CPU3. Since parallel application runs in vm4 to which $vCPU_{41}$ belongs, the scheduling program of CPU3 will also send an IPI signal to physical CPU2 according to the existing co-scheduling strategy, so as to inform the scheduling program of CPU2 to replace $vCPU_{31}$ occupying CPU2 with $vCPU_{40}$ (see Figure 2(b)). However, in such situation, lock holder preemption may occur after de-scheduling $vCPU_{31}$ from physical CPU2, because what $vCPU_{31}$ runs is parallel application processes/threads. Hence, if there is no constraint on such preemption in co-scheduling, the performance of parallel applications in VM may be affected.

## 3. An Enhanced Hybrid Scheduling Approach

To optimize the performance of parallel applications in a VM, the existing hybrid scheduling approaches can co-schedule all vCPUs of the parallel VM to different physical CPUs, making these vCPUs online at the same time. But in fact, the ratio of these vCPUs online simultaneously with these approaches is still low. One reason is that some vCPUs of a co-scheduled parallel VM relinquish physical CPUs because their CPU share is consumed in current scheduling period. Alternatively, some vCPUs of a co-scheduled parallel VM yield physical CPUs due to the unrestricted preemption by other parallel VMs. This situation usually result in the waste of CPU share, because the rest of vCPUs online in parallel VM cannot realize synchronous operations.

Based on the above analyses, an enhanced hybrid scheduling approach is presented in this section to optimize the performance of parallel applications in SMP VM by elevating the simultaneous online ratio of all vCPUs in SMP VM.

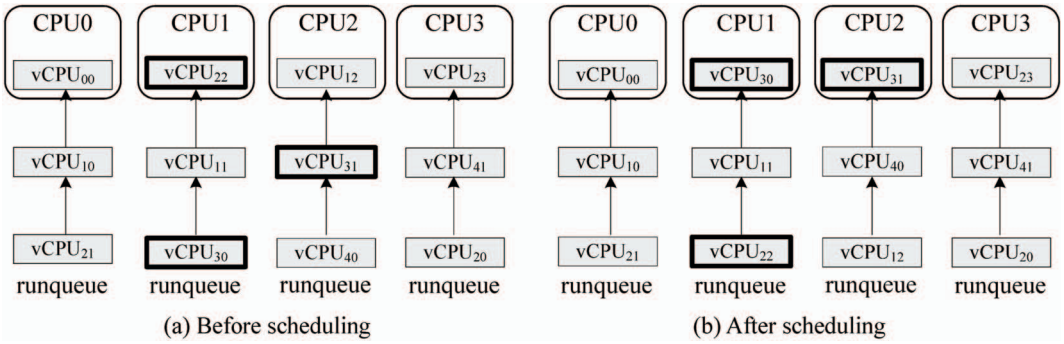


**Figure 1.** The scheduling process of the existing co-scheduling methods.

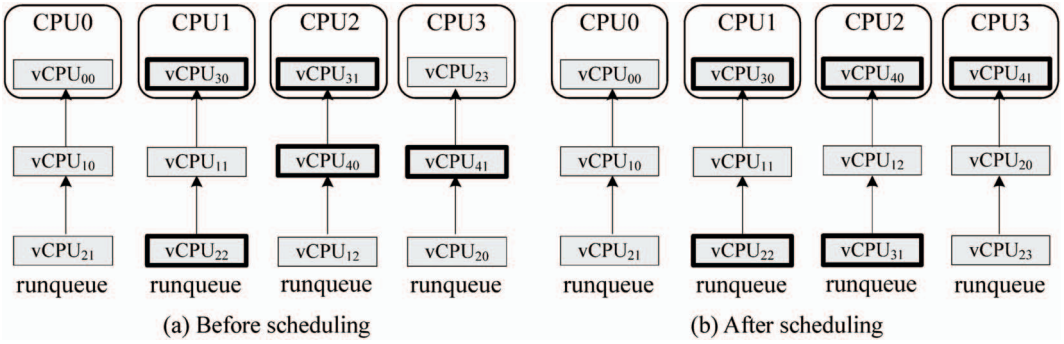


**Figure 2.** The preemption problem of co-scheduling parallel VM.

### 3.1 Overview of the Approach

The applications running in VMs include parallel and non-parallel applications. As for the non-parallel applications, the objective of scheduling is the maximization of throughput. Obviously, when non-parallel application is running in a VM, it is suitable to schedule the vCPUs of VM with asynchronous scheduling (AS) strategy based on proportional share, e.g., Xen's Credit Scheduler. As for the parallel applications, co-scheduling strategy is more effective, as indicated by the above analyses. Thus we propose an enhanced hybrid scheduling (EHS) approach.

As seen in Figure 3, the scheduler of the VMM contains two scheduling strategies: (1) the enhanced co-scheduling strategy is adopted in the case of parallel VM; (2) the asynchronous scheduling strategy (based on proportional share) is used in the case of non-parallel VM.

### 3.2 Common Issues

The primary issue to be solved in this approach is the fairness of system. Besides, load balancing and initial mapping should be taken into account in method design to optimize the performance of virtualized system.

**System fairness:** In a virtualized system, especially the cloud computing system based on virtualization, how to fairly allocate physical CPU resources is a crucial issue. To ensure the system fairness, this paper employs a method that can achieve the proportional share of CPU resources, which has been widely applied to open-source VMMs like Xen and KVM. For each VM in a virtualized system, their CPU share will be proportional to their weights. And the CPU share will be updated regularly. The total CPU share of the virtualized system in each interval of allocation is represented by $TotalShare_{system}$. It can be calculated by equation (1), where |PCPU| denotes the number of physical CPUs in the system and $Share^{CPU}_{interval}$ denotes the allocable share of each physical CPU in each interval of allocation. The share obtained by $vm_i$ in each interval of allocation is represented by $Share^{vm_i}_{interval}$ and can be calculated by equation (2). In each allocation, the CPU share obtained by $vm_i$ will be evenly allocated to all of its vCPUs. More precisely, the CPU share is equally distributed to all of its runnable vCPUs. Then when the current CPU share allocation is over, the actual vCPU share of any vCPU is the sum of the remaining (unconsumed) share in last interval of allocation and the share just obtained in this interval.

**Load balancing:** Load balancing among physical CPUs in the virtualized system is one task of VMMs. Load pulling is a feasible method to achieve load balancing. Specifically, when the run queue of a physical CPU is empty (no runnable vCPUs), this method enables the scheduler of VMM to pull one load (i.e., vCPU) from the run queue of other physical CPUs, thereby avoiding the physical CPU idling.

**Initial mapping from vCPU to physical CPU:** After creating $vm_i$, the VMM will insert all vCPUs of $vm_i$, i.e., VCPU ($vm_i$), into the run queue of physical CPUs, respectively. In order to reduce the potential cost for vCPU migration during the co-scheduling of VM, we introduce two principles in the initial mapping of VM inspired by balance scheduling [7].

- In the case of parallel VM, all vCPUs are separately put into the run queues of physical CPUs and any two vCPUs of VM should not coexist in the run queue of the same physical CPU. Besides, load balancing must alsobe taken into consideration.
- In the case of non-parallel VM, vCPUs mapping is performed based on the principle of load balancing among physical CPUs, not constrained by principle (1).

### 3.3 Enhanced Hybrid Scheduling Algorithm

As described in section 3.2, the enhanced hybrid scheduling approach will perform co-scheduling for parallel VMs and asynchronous scheduling for non-parallel VMs in the system. It optimizes the co-scheduling algorithm in the existing hybrid scheduling approaches, i.e.,

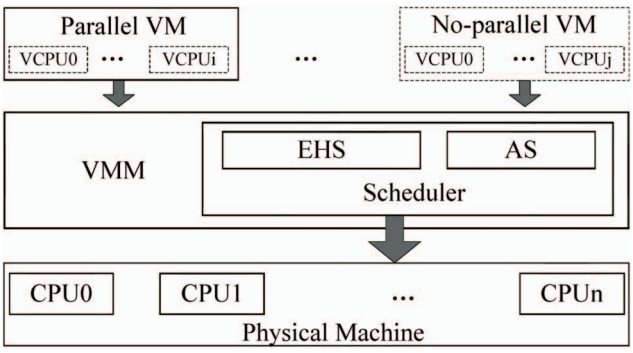


**Figure 3.** The framework of enhanced hybrid scheduling approach.

presenting an enhanced co-scheduling algorithm.

The basic idea of enhanced co-scheduling algorithm proposed in this paper is as follows: When one vCPU of parallel VM is de-scheduled from physical CPU, the scheduler will initiatively de-schedule the rest vCPUs of that parallel VM to avoid the potential waste of physical CPU share due to the failure of synchronization operations of vCPUs. The pseudo-codes are shown below.

**Algorithm 1.** Enhanced co-scheduling algorithm
Input: *parallel_vcpu_id*/*the ID of vCPU that issues co-scheduling request*/
*curr_vcpu_id*/* the ID of running vCPU before co-scheduling request is issued */
*co_scheduling_doms*/* the list of parallel VMs that are co-scheduled and running */
Output: void
Steps:
1. *dom* = get_dom_by_vcpu_id (*vcpu-> id*)
2. *deschedule_dom* = get_dom_by_vcpu_id (*curr_vcpu_id*)
3. *need_preempt_cpu_ids* = {}
4. *need_deschedule_cpu_ids* = {}
5. **for** each vcpu of dom
6. **do**
7. **if** the ID of vCPU is not equal to *parallel_vcpu_id* **then**
8. *need_preempt_cpu_ids* += get_cpu_id_by_vcpu_id (*vcpu-> id*)
9. **end if**
10. **end for**
11. **for** each vcpu of *deschedule_dom*
12. **do**
13. **if** the ID of vCPU is not equal to *curr_vcpu_id* **then**
14. *need_deschedule_cpu_ids* += get_cpu_id_by_vcpu_id (*vcpu->id*)
15. **end if**
16. **end if**
17. sending IPI to physical CPU in *need_preempt_cpu_ids*
18. sending IPI to physical CPU in *need_deschedule_cpu_ids*

The input information of the algorithm includes the ID of the vCPU requesting co-scheduling (*parallel_vcpu_id*), the ID of the running vCPU before the request of co-scheduling is issued (*curr_vcpu_id*), and the list of parallel VMs that are co-scheduled and running in the system (*co_scheduling_doms*). In the above pseudo-codes, Lines 5–10 are used to acquire the IDs of all physical CPUs needing co-scheduling, which will be stored in the set *need_preempt_cpu_ids*. Lines 11–16 are used to acquire the IDs of all physical CPUs needing co-de-scheduling, which will be stored in the set *need_deschedule_cpu_ids*. Lines 17 and 18 are used to initiate IPI for the physical CPUs in these two sets, so as to achieve co-scheduling and co-de-scheduling, respectively.

Based on Algorithm 1, the pseudo-codes of the enhanced hybrid scheduling approach are developed, as shown in Algorithm 2.

**Algorithm 2.** Enhanced hybrid scheduling
Input: the run queue information of physical CPUs
Output: void
Steps:
1. **if** there is no runnable vCPU in runqueue **then**
2. migrate a suitable vCPU from neighbor physical CPU with load balancing /*load balancing among physical CPUs*/
3. schedule this vCPU to physical CPU
4. **else**
5. **if** the first vCPU in runqueue belongs to the VM running non-parallel application **then**
6. schedule this vCPU to physical CPU /*asynchronous scheduling */
7. **else**
8. call Algorithm 1
9. **end if**
10. **end if**

When there is no runnable vCPU in the run queue of physical CPU, this algorithm will perform load balancing, i.e., migrating a suitable vCPU from the neighbor physical CPU to this physical CPU (Lines 1–3). When there are runnable vCPUs in the run queue, two scenarios exist: (1) If non-parallel applications are running on the VM to which the vCPU at the head of the run queue belongs, this algorithm will perform asynchronous scheduling, i.e., directly scheduling this vCPU to the current

physical CPU (Lines 4–6); (2) otherwise, algorithm 2 will be called to perform the enhanced co-scheduling (Lines 7–10).

### 3.4 Implementation of the Enhanced Hybrid Scheduling Method

Xen is an open-source virtualization software extensively adopted in academic and business communities. In this paper, we develop a prototype system of enhanced hybrid scheduling method based on the Credit scheduler of Xen3.2. Xen provides schedulers with an abstract interface defined by structure in C programming language. To be specific, the structure contains pointers pointing to functions implemented in scheduler. When adding a new scheduler to Xen3.2, a structure needs to be created to point to the implementation of new scheduling method. After that, the scheduler will be added to a static array storing all available schedulers. VMM (i.e., Xen3.2 in this paper) can select a scheduler according to the parameters specified by system administrator.

In order to implement the prototype of our method, we add a new scheduler (i.e., sched_enhanced_def) to Xen3.2, which is defined as follows.

```
struct scheduler sched_enhanced_def ={
.name=”Enhanced-hybrid-scheduling Scheduler”;
….
.do_schedule=enhybridsched_schedule;
…..
};
```

As the system fairness implemented by Xen's Credit scheduler is well recognized, we employ fairness module of Credit scheduler (i.e., credit allocation and consumption for proportional share of physical CPUs) to develop our prototype system. Meanwhile, we also extend Credit scheduler's mapping module from vCPUs to physical CPUs by introducing constraint, i.e., any two vCPUs of a VM cannot coexist in the run queue of the same physical CPU. Besides, the original load balancing module in Credit scheduler is also extended so that the above-mentioned constraint can also be satisfied during load balancing (i.e., the scheduler pulls vCPUs to idle physical CPUs from their neighbor physical CPUs).

The role of do_schedule defined in structure is to select a vCPU for physical CPU from its run queue, where the algorithms 1 and 2 are implemented.

The information about the type of VM is required in implementing do_schedule. The type of VM can be obtained through many ways, for example, manual setting used in [2], and fuzzy inference in [8]. In this paper, we manually set the type of VM according to the work in [2].

## 4. Performance Evaluation

In this section, our enhanced hybrid scheduling method and current typical scheduling methods are evaluated with a representative test program. Experimental setting and results are shown in 4.1 and 4.2, respectively.

### 4.1 Experimental Setting

**Experimental environment:** The physical machine is equipped with two 8-core Intel Xeon E5620 CPUs@ 2.40 GHz/24 GB memory/1 TB disk), CentOS5.5 (Linux kernel 2.6.18), and Xen3.2. There are six VM (vm1–vm6) launched in this physical machine. Each VM has four vCPUs/1 GB memory/8 GB disk, and its operating system is also CentOS5.5 (Linux kernel 2.6.18). The privileged VM (i.e., Dom 0 in Xen) is interconnected with other guest VMs through bridge network interface.

**Benchmark:** The NAS Parallel Benchmarks (NPB) [9] are a small set of programs designed to help evaluate the performance of parallel supercomputers. We choose three typical benchmarks from NPB, i.e., *is*, *lu* and *ep* (*is* > *lu* > *ep* in terms of communication intensity), and one benchmark measuring the throughput of web server. Specifically, the web server is Apache http server. The performance of web server is measured with Httperf [10], which provides flexible tools to generate various HTTP loads for the evaluation of web server.

**Scheduling methods:** The compared methods in this section include the Credit scheduling (CREDIT) method adopted in Xen, the hybrid scheduling (HS) method proposed in [2], the balance scheduling (BS) method presented in [7], and our enhanced hybrid scheduling (EHS).

**Experiment classification:** Two experiments are carried out for evaluating the performance of different scheduling methods. In the first experiment, we test the performance of scheduling methods when all VMs run the same test program, with the metric of average time spent in running test program once. In the second experi-

ment, we test the performance of scheduling methods when all VMs run different test programs, with the metrics of average time spent in running test program once and the throughput of apache server.

### 4.2 Experimental Results and Analyses

**Experiment 1**

Six VMs (vm1–vm6) run the same program of NPB simultaneously in this experiment. Each VM runs a script to control the repeated running of test program. The time spent in the first 10 runs of test program on each VM is recorded. Then the average time of all six VMs spent in running a NPB program once is used to compare the performance of different scheduling methods. The experimental data are normalized with that obtained under CREDIT method, as shown in Figure 4.

Figure 4 indicates that compared with the CREDIT method, the performance of the other three scheduling methods is better. Specifically, the EHS method exhibits the best performance, followed by HS and BS method successively. The reasons are as follows:

(1) HS method is able to co-schedule the vCPUs of parallel VMs, so as to alleviate the synchronization delay of parallel applications running in VMs. Thus it obviously outperforms the CREDIT method.

(2) When adopting the BS method, all vCPUs of parallel VM will not be co-scheduled explicitly. Instead, these vCPUs will be put into the run queues of different physical CPUs while guaranteeing that any two vCPUs of a parallel VM will not coexist in the run queue of the same physical CPU. That is, BS method is a probabilistic co-scheduling method. Hence, it is inferior to HS method but overmatches CREDIT method.

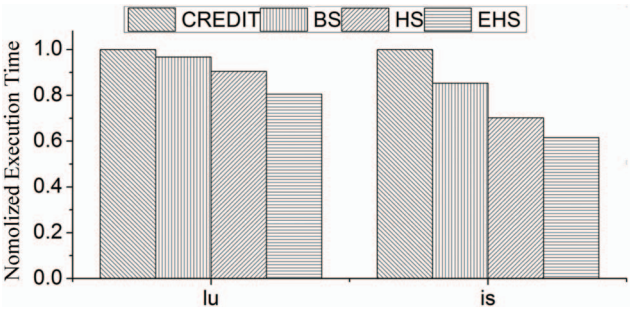


**Figure 4.** Normalized execution time.

(3) As for the EHS method proposed in this paper, it introduces a co-descheduling mechanism based on the existing co-scheduling method. When a vCPU in parallel VM needs to yield physical CPU, the EHS method will de-schedule the rest of vCPUs inthat VM from physical CPUs. Thus the CPU share allocated to the parallel VM will not be wasted, which leads to a better performance of EHS method compared with HS method.

**Experiment 2**

In this experiment, VMs run different NPB programs and web server at the same time. For example, vm1 and vm2 run *lu* of NPB, vm3 and vm4 run *is* of NPB, vm5 runs *ep* of NPB, and vm6 runs web server (i.e., Apache). Generally, it spends different time to run different NPB programs once. Therefore, a script is used to run on each VM to control the running times of its NPB program, in order to guarantee that all VMs will keep running until the time of the first 10 runs of NPB program on each VM is recorded. Meanwhile, Httperf runs on another physical machine (with the same configuration as the physical machines in experiment) in the same LAN to test the throughput of Apache server in vm6. The experimental results of NPB programs are normalized with that under CREDIT method, as seen in Figure 5(a). In addition, the throughput data of Apache server under different methods are also normalized with that under CREDIT method, as shown in Figure 5(b).

It can be seen in Figure 5(a) that different scheduling methods show the similar performance trend as that in Experiment 1, when running parallel applications *lu* and *is*. That is, the EHS method proposed in this paper has the best performance, followed by HS and BS method successively. It can be attributed to the same reasons as shown in Experiment 1. Meantime, it can be observed from Figure 5(a) that CREDIT and BS methods have the same performance and overmatch HS and EHS methods, when running the program *ep*. It is because both HS and EHS method adopt co-scheduling with preemption, which makes the VM running compute-intensive applications perform frequent context switches. Accordingly, the performance of compute-intensive applications is affected.

As shown in Figure 5(b), in term of the web server

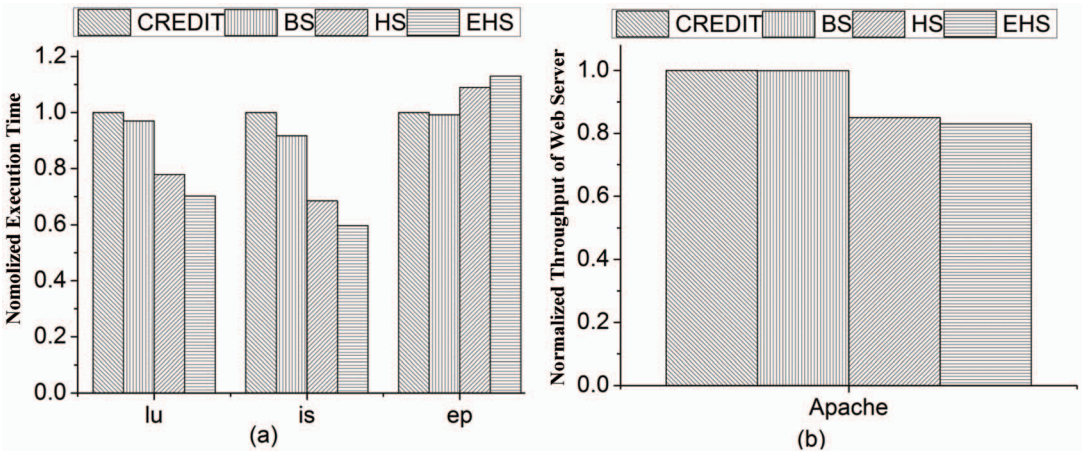


**Figure 5.** Performance of scheduling methods with mixed NPB programs and web server.

throughput, CREDIT and BS methods perform similarly and are superior to EHS and HS methods. The reasons are summarized as follows:

(1) CREDIT exhibits the best performance because it is a throughput-oriented scheduling method.

(2) Although BS method is a probabilistic co-scheduling method, it will not change the scheduling priority of vCPUs. So its throughput of web server is equivalent to that of CREDIT method.

(3) As employing the preemptive co-scheduling strategy for VMs, both EHS and HS have a certain negative impact on the throughput of web server in a mixed environment with parallel applications and web application.

## 5. Related Work

In this section, we summary the related work.

A hybrid scheduling framework that needs a system administrator to set the type of application in a VMis proposed in [2] to optimize the performance of parallel applications running in the single SMP VM. This framework can perform co-scheduling for the SMP VM running multithreaded parallel applications, i.e., scheduling all vCPUs of that VM simultaneously. Meanwhile, it performs asynchronous scheduling for SMP VM running compute-intensive applications. A dynamic adaptive VM scheduling method is presented in [6] to improve the performance of multithreaded parallel applications in SMP VM. By monitoring the spinlock latency in guest operating system, this method can judge the type of applications running inSMP VM, thus providing a basis for subsequent work of scheduling. It avoids the manual setting of application type in the former method, but needs to perform intrusive detection in the guest operating system. So it does not fit such commercial operating systems as Windows. A co-scheduling method based on task-aware is introduced in [11]. It judges the type of applications by monitoring the scheduling state of vCPUs and the state of threads in vCPUs, so as to provide a basis for the virtualization system to make decision on co-scheduling.

VMware, the leading provider of virtualization solutions, has proposed a relaxed co-scheduling method [12]. It tries to maintain synchronous progress of vCPU siblings by deferring the advanced vCPUs until the slower ones catch up. However, this scheme is too rigorous for the SMP machine with less demand on synchronization.

Different from the above-mentioned co-scheduling and relaxed co-scheduling methods, the authors of [7] present balance scheduling method. It aims at allocating the vCPUs of the same SMP VM evenly into the run queues of different physical CPUS (i.e., any two vCPUs of a SMP VM will not exist simultaneously in the run queue of the same physical CPU), rather than co-scheduling the vCPUs of SMP VM accurately at the same time.

Virtualization may cause many issues including lock-holder preemption, vCPU stacking, CPU fragmentation, and priority inversion. In [13], the authors intro-

duce FlexCore, a scheduling scheme using vCPU ballooning, which dynamically adjusts the number of vCPUs of a VM at runtime. The authors of [14] propose vINT (scheduling status based virtual INterrupt remapping adapTer), a scheme that leverages hardware-assisted interrupt mapping, so as to alleviate the problem of unpredictable I/O responsiveness of SMP VM. In [15], the authors propose a self-boosted co-scheduling (SBCO) algorithm to reduce synchronization latency among consolidated VMs, which reorders all these sibling vCPUs threads coarsely at the same level in their respective run queue, then schedules them at the same time window, and maintains global fairness between consolidated VMs. The authors of [16] propose a consolidation-aware vCPU (CVS) scheduling scheme on multicore virtualization platform. Based on vCPU over-commitment rate, the 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.

Different from our work, although these existing approaches improve the performance of parallel application running SMP VM, they have done nothing about the low ratio of all vCPUs online simultaneously in parallel VM after co-scheduling.

## 6. Conclusions

Virtualization technology may result in some problems not existing in non-virtualized environment, such as lock holder preemption. To co-schedule the vCPUs of VMs in VMM is an effective path to alleviate the performance degradation of parallel applications in virtualized environment, and many approaches have been developed. However, the existing approaches pay little attention to the unrestricted preemption between parallel VMs when performing co-scheduling. Moreover, no one focuses on the low ratio of all vCPUs online simultaneously in parallel VM after co-scheduling. Actually, these two issues may also bring negative impact on the performance of parallel applications in SMP VM. That is why we develop the enhanced hybrid scheduling method. Experimental results show that our proposed method outperforms the existing Credit scheduling, hybrid scheduling, and balance scheduling methods in optimizing the performance of parallel applications in SMP VM.

## Acknowledgements

This work is supported by the Anhui Natural Science Foundation of China under Grant (No. 1608085QF147 and No. 1408085MF126), and Key Project of Support Program for Excellent Youth Scholars in Colleges and Universities of Anhui Province (No. gxyqZD2016332). It is also supported by Scientific Research Staring Foundation of Chuzhou Univeristy (No. 2014qd016).

## References

[1] Uhlig, V., LeVasseur, J., Skoglund, E., et al., "Towards Scalable Multiprocessor Virtual Machines," Proc. of 3rd Virtual Machine Research & Technology Symposium (VM'04), Berkeley, CA, USA: USENIX Association, pp. 43–56 (2004)

[2] Weng, C., Wang, Z., Li, M., et al., "The Hybrid Scheduling Framework for Virtual Machine Systems," Proc. of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, New York, NY, USA: ACM Press, pp. 111–120 (2009). doi: 10.1145/1508293.1508309

[3] Chen, H., Wu, S., Di, S., et al., "Communication-driven Scheduling for Virtual Clusters in Cloud," Proc. of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, New York, NY, USA: ACM Press, pp. 125–128 (2014). doi: 10.1145/2600212.2600714

[4] Wu, S., Chen, H., Di, S., et al., "Synchronization- aware Scheduling for Virtual Clusters in Cloud," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 10, pp. 2890–2912 (2015). doi: 10.1109/TPDS.2014.2359017

[5] Kivity, A., Kamay, Y., Laor, D., et al., "kvm: the Linux Virtual Machine Monitor," Proc. of the Linux Symposium, pp. 225–230 (2007).

[6] Weng, C., Liu, Q., Yu, L., et al., "Dynamic Adaptive Scheduling for Virtual Machines," Proc. of the 20th International Symposiumon High Performance Distributed Computing, New York, NY, USA: ACM Press, pp. 239–250 (2011). doi: 10.1145/1996130.1996163

[7] Sukwong, O. and Kim, H. S., "Is Co-scheduling Too

Expensive for SMPVMs?" Proc. of the 6th European Conference on Computer Systems, New York, NY, USA: ACM Press, pp. 257–272 (2011).

[8] Kim, H., Lim, H., Jeong, J., et al., "Task-aware Virtual Machine Scheduling for I/OPerformance," Proc. of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, New York, NY, USA: ACM Press, pp. 101–110 (2009). doi: 10.1145/1508293.1508308

[9] Ramachandran, A., Vienne, J., Van Der Wijngaart, R., et al., "Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi," Proc. of 42nd International Conference on Parallel Processing (ICPP), IEEE, pp. 736–743 (2013). doi: 10.1109/ICPP.2013.87

[10] Mukherjee, J., Wang, M. and Krishnamurthy, D., "Performance Testing Web Applications on the Cloud," Proc. of IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, pp. 363–369 (2014). doi: 10.1109/ICSTW.2014.57

[11] Bai, Y., Xu, C. and Li, Z., "Task-Aware Based Coscheduling for Virtual Machine System," Proc. of the 2010 ACM Symposium on Applied Computing, New York, NY, USA: ACM Press, pp. 181–188 (2010). doi: 10.1145/1774088.1774126

[12] McDougall, R. and Anderson, J., "Virtualization Performance: Perspectives and Challenges Ahead," *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 4, pp. 40–56 (2010). doi: 10.1145/1899928.1899933

[13] Miao, T. and Chen, H., "FlexCore: Dynamic Virtual Machine Scheduling Using VCPU Ballooning," *Tsinghua Science and Technology*, Vol. 20, No. 1, pp. 7–16 (2015). doi:10.1109/TST.2015.7040515

[14] Li, J., Ma, R., Guan, H. B., et al., "vINT: Hardware-Assisted Virtual Interrupt Remapping for SMP VM with Scheduling Awareness," Proc. of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada: IEEE, pp. 234–241 (2015). doi: 10.1109/CloudCom.2015.18

[15] Wang, K., Wei, Y., Xu, C. Z., et al., "Self-boosted Coscheduling for SMP Virtual Machines," Proc. of the 2015 IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Atlanta, GA, USA: IEEE, pp. 154–163 (2015).

[16] Wang, B., Cheng, Y., Chen, W., et al., "Efficient Consolidation-aware VCPU Scheduling on Multicore Virtualization Platform," *Future Generation Computer Systems*, Vol. 56, pp. 229–237 (2016). doi: 10.1016/j.future.2015.08.007

***Manuscript Received: Feb. 22, 2016***
***Accepted: Apr. 21, 2016***