Hybrid CPU Management for Adapting to the Diversity of Virtual Machines

Chuliang Weng, *Member*, *IEEE*, Minyi Guo, *Senior Member*, *IEEE*, Yuan Luo, *Member*, *IEEE*, and Minglu Li, *Member*, *IEEE*

Abstract—As an important cornerstone for clouds, virtualization plays a vital role in building this emerging infrastructure. Virtual machines (VMs) with a variety of workloads may run simultaneously on a physical machine in the cloud platform. The scheduling algorithm used in Xen schedules virtual CPUs (VCPUs) of a VM asynchronously and guarantees the proportion of the CPU time allocated to the VM. This proportional sharing (PS) method is beneficial as it simplifies the implementation of CPU scheduling in the virtual machine monitor (VMM), and can deliver near-native performance for some workloads. However, when workloads in VMs are concurrent applications such as multithreaded programs with the synchronization operation, it has been demonstrated that this method in the VMM can reduce the performance, due to the negative impact of virtualization on synchronization. To address this issue, we present a hybrid scheduling framework for CPU management in the VMM to adapt to the diversity of VMs running simultaneously on a physical machine. We implement a hybrid scheduler based on Xen, and experimental results indicate that the hybrid CPU management method is feasible to mitigate the negative influence of virtualization on synchronization, and improve the performance of concurrent applications in the virtualized system, while maintaining the performance of high-throughput applications.

Index Terms—Virtual machine, virtual machine monitor, CPU management, Xen

1 Introduction

With virtualization technology [1], [2], the functionality of multiple standalone computer systems can be aggregated into a single hardware computer, to promote efficient usage of the hardware while decreasing power consumption. As a result, virtualization is an important brick for building the cloud infrastructure such as Amazon's elastic compute cloud (EC2) [3]. Currently, examples of system virtualization include Xen [4], [5], VMware [6], [7], KVM [8], Hyper-V [9], and VirtualBox [10]. Different from the traditional system software stack, a virtual machine monitor (VMM) is sitting between the operating system level and the hardware level in the virtualized system.

It is the VMM that virtualizes physical CPUs (PCPUs) to virtual CPUs (VCPUs), on which guest operating systems run. In a VM system with multicore processors, usually a VM with multiple VCPUs is treated as a virtual symmetric multiprocessing (SMP) system. Different from the non-virtualization scenario, all VCPUs in a virtual SMP system are usually not online all the time, and might not be online at the same time when a certain scheduling strategy is running. This is because the number of VCPUs of all VMs is usually larger than the number of PCPUs, and these VCPUs have to share the limited number of PCPUs in turn. As a result, a fair share of CPU should be guaranteed by the VMM. Proportional share fairness between VMs is usually used in VMMs. For example, Xen uses *weight* and VMware *entitlement*.

Typically, applications in VMs can be classified into concurrent workloads and high-throughput workloads. In a cloud platform, a VM could be created and destroyed on demand, and used for a specific purpose [11]. Therefore, a VM could be assumed to run one kind of workload in the cloud platform. Even if a VM has to run a mix of several computation-intensive applications, as a simple and efficient method, the first-come-first-served strategy could be adopted. Consequently, it is still dedicated to executing tasks of an application in a certain period. When the workload in a VM is a high-throughput application, the VMM schedules VCPUs of the VM asynchronously but only needs to guarantee the fairness of CPU time allocation. This proportional sharing (PS) scheduling method can deliver near-native performance for this kind of workload, and is beneficial to simplify the implementation of the CPU scheduling in the VMM. Therefore, it is widely adopted in the implementation of VMMs such as Xen. However, when workloads in VMs are concurrent applications such as multithreaded programs or parallel programs with synchronization operations, we have demonstrated with theoretical analysis and experiment that this method in the VMM can cause performance deterioration.

To mitigate this kind of performance degradation, coscheduling (alternatively, gang scheduling) can be applied to the CPU scheduling in the VMM, as in VMware [12]. However, we have also demonstrated that coscheduling can degrade the performance of high-throughput workloads in VMs (see Fig. 10a for details), therefore, coscheduling is not suitable for high-throughput workloads.

The motivation of this paper is to not only improve the performance of VMs with concurrent applications, but also

• *The authors are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, China.*
E-mail: {weng-cl, guo-my, luoyuan, li-ml}@cs.sjtu.edu.cn.

Manuscript received 29 Mar. 2011; revised 13 Dec. 2011; accepted 11 Mar. 2012; published online 28 Mar. 2012.
Recommended for acceptance by W. Najjar.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-03-0206.
Digital Object Identifier no. 10.1109/TC.2012.80.

keep the performance of other VMs with high-throughput workloads in the system. In the cloud platform, the workload in a VM is usually known in advance when the VM is created for a specific purpose. Therefore, we propose a hybrid scheduling framework to adapt to the diversity of VMs in the cloud platform. That is, when multiple VMs run simultaneously in a virtualized system, these VMs are classified into two types: concurrent VMs and high-throughput VMs, responding to their workloads' characteristics. Subsequently, the scheduler in the VMM adopts different strategies to schedule VCPUs from the two kinds of VMs, rather than a single strategy for all VMs in the system.

1.1 Contribution

In summary, this paper has made fourfold contributions, as detailed below.

First, we build a model to analyze the impact of the CPU scheduling strategy on the performance of virtualized systems. We find out that, based on the theoretical analysis, the asynchronous VCPU scheduling strategy wastes considerable CPU time when the workload in a VM is a concurrent application. This finding motivates us to present the coproportional share (CPS) scheduling strategy for VMs with concurrent workloads.

Second, we propose the phase-coproportional share (PCPS) scheduling strategy that distributes CPU time among VCPUs based on demand. This is motivated by the observation that the master task in a concurrent application usually consumes more CPU time compared to slave tasks, and traditional equal CPU distribution does not match this fact. The performance of PCPS will be further demonstrated by the theoretical analysis.

Third, we propose a hybrid CPU scheduling framework for the VMM. In a virtualized system, VMs are clustered into two categories based on the characteristics of applications, namely high-throughput VMs and concurrent VMs. Consequently, the hybrid scheduling framework performs a hybrid scheduling operation for the different kinds of VMs in a virtualized system. That is to adopt the PS scheduling strategy for high-throughput VMs and use the CPS or PCPS scheduling strategy for concurrent VMs.

Finally, we have implemented a prototype of the hybrid scheduling framework and conducted a comprehensive experimental study to evaluate its performance. Compared with the default credit scheduler in Xen, the Hybrid Scheduler performs much better. Moreover, compared with the CPS scheduling strategy, the performance of concurrent workloads in a virtualized system has been further promoted by the PCPS scheduling strategy.

1.2 Outline

The remainder of this paper is organized as follows: Section 2 introduces the background, and Section 3 proposes strategies for concurrent workloads. Section 4 presents a hybrid scheduling framework, and Section 5 describes the scheduling algorithm for the VMM. Section 6 discusses experimental results of the implemented Hybrid Scheduler and the default credit scheduler. Section 7 provides a brief overview of related work, and Section 8 concludes the paper.

2 Background

In this section, we first introduce the scheduling issues in the VMM, then give a typical scenario for concurrent workloads running in a virtual SMP system. The scheduling strategies will be proposed and analyzed in the next section.

2.1 Scheduling Issue

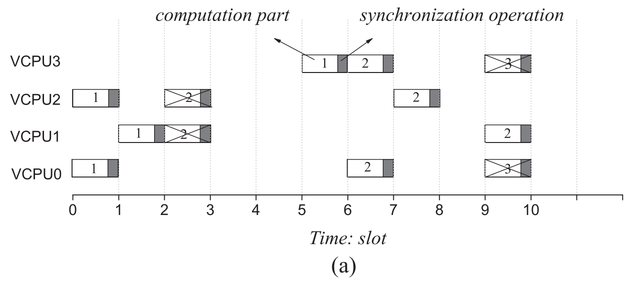
As mentioned above, a VM with multiple VCPUs is treated as a virtual SMP system, where all VCPUs behave identically. There are two issues with the CPU scheduling in the VMM, on which multiple virtual SMP systems run simultaneously. One issue is how to deal with the synchronization issue such as the lock-holder preemption [13] for a virtual SMP system. For example, the VMM may preempt a VCPU with a thread holding a lock, which will result in an extension of the lock holding time. The negative influence of virtualization on synchronization cannot be ignored, otherwise it will bring a performance degradation for the virtualized system. The other issue is how to guarantee the CPU fairness among different VMs in a virtualized system. When multiple VMs share a single physical system, it is not a good practice to allow a VM with a heavy workload to appropriate the CPU resource of other VMs without any limitation. The scheduler in the VMM should keep fairness in resource sharing among VMs. That is, the amount of the CPU time obtained by a VM should be controlled.

To guarantee CPU fairness among different VMs, an effective method is the PS scheduling strategy widely adopted in the implementation of VMMs such as Xen, by which the CPU usage of a VM is in proportion to the *weight* that has been assigned to it. The weight of a VM is an integer parameter, and the *weight proportion* of a VM is the value of its weight divided by the total of all VMs' weights in the system. This determines the proportion of CPU time obtained by the VM compared to the total of CPU time in the virtualized system. Also the CPU time obtained by a VM is distributed equally among its VCPUs in existing methods.

To mitigate the negative impact of virtualization on synchronization, an effective approach is to make the VCPUs of a VM like the CPUs of a physical machine from the viewpoint of the scheduler in the guest operating system. Therefore, when all kinds of applications run on a VM, the VCPUs should all be online at the same time. That is, VCPUs of the VM should be coscheduled. However, coscheduling may also introduce some additional overhead. As a result, intuitively the VMM coschedules VCPUs of a VM when concurrent applications run on the VM; otherwise, the VMM can schedule those VCPUs asynchronously when the workloads are nonconcurrent applications.

2.2 Scenario

In the section, a straightforward scenario is used to emphasize coscheduling for concurrent workloads in VMs. We assume a VM with four VCPUs has a multithreaded program and this VM runs on a physical computer with four PCPUs. In addition, we assume the unit time of the CPU scheduling is one slot. The weight proportion of the VM is $3/10$, that is, the proportion of CPU time consumed by the VM is 30 percent of the total CPU



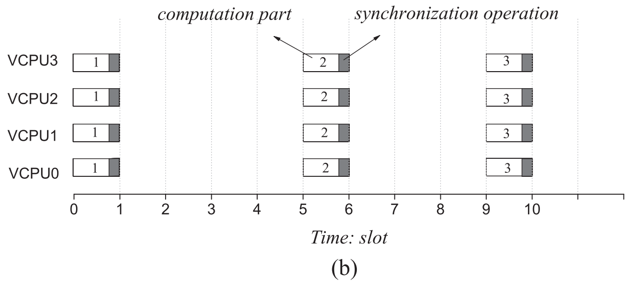


Fig. 1. (a) The scenario of noncoscheduling. (b) The scenario of coscheduling.

time in the virtualized system. There is a synchronization operation between threads at the end of each step, with a length of one slot. This scenario can be extracted from multithreaded applications or parallel applications.

We further assume that there are two kinds of scheduling strategies. One is to asynchronously assign each VCPU to a PCPU to maximize the throughput, while guaranteeing CPU fairness according to the weights. With this noncoscheduling strategy, Fig. 1a depicts a possible scheduling sequence of the above described multithreaded program. It is observed that the multithreaded program only completes two steps in the cycle of 10 slots, with four slots of CPU time wasted due to synchronization.

The alternative is a coscheduling strategy, which synchronously assigns each VCPU to a PCPU to avoid the negative influence of virtualization on synchronization, while guaranteeing CPU fairness according to the weights. Fig. 1b shows a possible scheduling sequence of the multithreaded program generated by this coscheduling strategy, and it is observed that the multithreaded program completes three steps within the cycle of 10 slots.

Intuitively, the coscheduling strategy outperforms the noncoscheduling strategy for concurrent workloads, under the condition that the CPU fairness is guaranteed. This is because the coscheduling strategy considers the correlation of VCPUs, on which a concurrent workload runs, and provides an effective solution to the synchronization problem for concurrent workloads, whereas concurrency is not considered by the noncoscheduling strategy.

3 Scheduling Strategies and Analysis

In this section, we begin by modeling scheduling in the virtualized system, and then propose the CPS scheduling strategy that guarantees the fairness of resource allocation and meanwhile coschedules VCPUs for concurrent workloads. Furthermore, we extend the CPS scheduling strategy to the PCPS scheduling strategy to handle the cases when the master task has a longer phase compared to other tasks in concurrent applications.

3.1 Modeling

Job model. $J = \{J_0, J_1, \ldots, J_{|J|-1}\}$ denotes a concurrent job, which comprises $|J|$ tasks. Each task of the job runs in sequence, and has to synchronize with other tasks in a fixed interval. Task J_i can be decomposed by the synchronization operation into a sequence of $|J_i|$ phases, that is, $\{J_i^0, J_i^1, \ldots, J_i^{|J_i|-1}\}$. Each phase of a task comprises one part of computation and one subsequent part of synchronization. To simplify the presentation, we assume that each phase of a task is equal to one basic time unit. Once one phase of a task begins to run on a PCPU, it will keep running until the completion of the phase.

Scheduling model. In the VM system, the assignment of a concurrent job in a VM includes: the assignment of jobs on VCPUs by the guest operating system, and the assignment of VCPUs on PCPUs by the VMM.

A physical computer includes a group of homogenous PCPUs, denoted by $P = \{P_0, P_1, \ldots, P_{|P|-1}\}$, and the number of PCPUs is $|P|$. VMs running on a physical computer are denoted by $V = \{V_0, V_1, \ldots, V_{|V|-1}\}$, and $|V|$ denotes the number of VMs in the system. The weight proportion of VM V_i is denoted by $\omega(V_i)$, which represents the proportion of CPU time consumed by the VM, with $\sum_i \omega(V_i) = 1$. The set of VCPUs in the ith VM is denoted by $C(V_i) = \{v_{i0}, v_{i1}, \ldots, v_{i(|C(V_i)|-1)}\}$, and $|C(V_i)|$ denotes the number of VCPUs in VM V_i. To avoid the expensive switching cost of VCPUs mapping on PCPUs, the following relation exists, that is, $\forall i, |C(V_i)| \leq |P|$. For the same reason, we assume that $|J| \leq |C(V_i)|$.

The scheduling problem of a concurrent job J in VM V_i is formalized by $\chi = (\tau, \pi)$, where $\tau : J \rightarrow \{0, 1, 2, \ldots, \infty\}$, and $\pi : J \rightarrow P$. τ maps phases of tasks to the set of time slots, and π maps phases of tasks to the set of PCPUs in the system.

As each task of a concurrent job runs in sequence and they synchronize with each other at the end of phases in an interval, we have $\tau(J_k^m) + 1 \leq \tau(J_l^n)$ if $m < n$. The makespan of job J executed on VM V_i by the schedule χ is $makespan_\chi(J) = \max_k\{\tau(J_k^{|J_k|}) + 1\}$.

Objective function. We now formulate the scheduling issue of a concurrent job in a virtualized system as an optimization problem as follows.

Let χ be a schedule of a concurrent job J of VM V_i on a physical machine with P. The objective function is $\min makespan_\chi(J)$, while subject to: 1) the proportion of the CPU time obtained by VM V_i is $\omega(V_i)$; 2) $\tau(J_k^m) + 1 \leq \tau(J_l^n)$ if $m < n$; and 3) $|J| \leq |C(V_i)| \leq |P|$.

Problem reduction. As mentioned above, it could be assumed that a virtual SMP system is dedicated to executing tasks of a concurrent job in a certain period. In this paper, we focus on the CPU scheduling in the VMM, and assume that each task of a concurrent job will be assigned to a fixed VCPU by the guest operating system to reduce the context switching cost. Correspondingly, the job scheduling problem is reduced to the scheduling of VCPUs to PCPUs in the system. Then, in the schedule χ, τ, and π also represent the mapping relation of VCPUs to the set of time slots and the set of PCPUs, respectively.

3.2 CPS Scheduling Strategy

Formally, the time is subdivided into a sequence of fixed-length *slot*s, which are the basic time unit, and slot i

corresponds to the time interval $[i, i+1)$. Within each slot, each PCPU is allocated to a VCPU.

$Received(v_{ij}, t_1, t_2)$ denotes the CPU time obtained by VCPU v_{ij} in the interval $[t_1, t_2)$, and the CPU time obtained by VM V_i is equally distributed among its VCPUs. Then the deviation is defined as follows:

$$Lag(t, v_{ij}) = t \times |P| \times \omega(V_i)/|V_i| - Received(v_{ij}, 0, t). \quad (1)$$

$|Lag(t, v_{ij})|$ is used to evaluate the fairness of the scheduling strategy. For an *ideal* fair scheduling strategy, which can guarantee that the CPU time consumed by a VM is strictly in proportion to its weight, we have $|Lag(t, v_{ij})| \leq 1$, where the time unit is one slot.

CPS scheduling strategy. Besides guaranteeing that the CPU time is allocated to a VM according to its weight proportion, this strategy coassigns the set of VCPUs in a VM to PCPUs, that is, all VCPUs in the VM are coscheduled to PCPUs in the system.

Formula 1. For a concurrent job J in VM V_i, the makespan of job J under the CPS scheduling strategy is:

$$makespan_{cps}(J) = \left\lceil \frac{(\max_k |J_k|) \times |V_i|}{|P| \times \omega(V_i)} \right\rceil$$

Proof. As all VCPUs in VM V_i are coscheduled to PCPUs, the makespan is determined by the time length of the task with the maximal number of phases in job J. The CPU time obtained by a VM is equally distributed among its all VCPUs, then the weight proportion of a VCPU in VM V_i is $\frac{\omega(V_i)}{|V_i|}$. The number of PCPUs is $|P|$, and the execution time of each phase is 1, then the interval of the two successive phases of each task is $\lceil \frac{|V_i|}{|P| \times \omega(V_i)} \rceil$. Then the makespan of $(\max_k |J_k|)$ phases is

$$\left\lceil \frac{(\max_k |J_k|) \times |V_i|}{|P| \times \omega(V_i)} \right\rceil,$$

and then $makespan_{cps}(J) = \lceil \frac{(\max_k |J_k|) \times |V_i|}{|P| \times \omega(V_i)} \rceil$. □

Formula 2. For a concurrent job J in VM V_i, we have

$$\frac{makespan_{ps}(J)}{makespan_{cps}(J)} \approx 2 \times |Lag|$$

when the maximal deviation is $|Lag|$ in the PS scheduling strategy, where $makespan_{ps}(J)$ denotes the supremum of the makespan of J by the PS scheduling strategy.

Formula 2 is easily deduced from the above Formula 1 and Formula 6 in *Appendix*, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2012.80. For a practical PS scheduling strategy, the value of $|Lag|$ is usually larger than 1. Especially, in practice it increases as the weight proportion decreases. Therefore, in the worst case, the makespan of a concurrent job J with the PS strategy will be not less than the twice the makespan with the CPS strategy.

3.3 PCPS Scheduling Strategy

For simplicity, the length of phases of each task is assumed to be equal to one slot in Section 3.1. Actually, phases of the master task are usually longer than other tasks in a concurrent application, when the master task has to execute sequential parts as well as parallel parts while other tasks only execute parallel parts. We call the VCPU on which the master task runs the master VCPU, and it is v_{i0} for VM V_i. Intuitively, the master VCPU should obtain more CPU time than other VCPUs, instead of sharing the same amount of CPU time as other VCPUs.

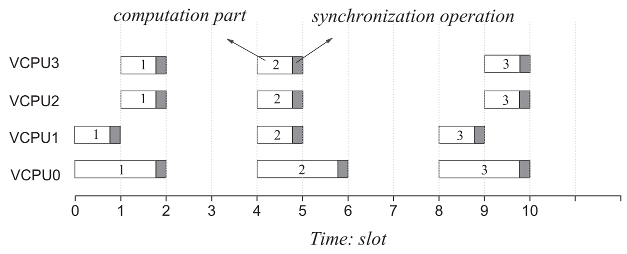


Fig. 2. The scenario of phase-coscheduling.

Let us reconsider the above scenario. We assume phases of the master task are twice as long as phases of other tasks in a concurrent application. Correspondingly, the CPU time obtained by the master VCPU will be twice the CPU time of other VCPUs in this VM, which is illustrated in Fig. 2. Therefore, the concurrent application can still complete three steps within the duration of 10 slots when its weight proportion is 0.375 (15/40). We call this the *PCPS* scheduling strategy. As a comparison, according to the CPS scheduling strategy, the application can only complete up to two steps within the duration of 10 slots.

Formula 3. For a concurrent job J in VM V_i, the master task has the maximum number of phases, and the length of its phases is $1+u$ slots while the length of phases for other tasks is one slot. Here, u is an integer because the unit time is one slot for scheduling. The makespan of job J by the PCPS scheduling strategy is denoted by $makespan_{pcps}(J)$, then,

$$\frac{makespan_{cps}(J)}{makespan_{pcps}(J)} = \frac{|V_i| \times (1+u)}{|V_i| + u}.$$

Proof. In job J, the master task has the maximum of phases. We denote the master task in job J as J_0, then the maximum of phases in job J is $|J_0|$. In the PCPS strategy, it needs $|V_i| + u$ slots for each phase of job J. And the CPU time obtained by VM V_i in each unit time is $|P| \times \omega(V_i)$ slots. Then it can complete $\frac{|P| \times \omega(V_i)}{|V_i| + u}$ phases of job J in each unit time, and then $makespan_{pcps} = \frac{|J_0| \times (|V_i| + u)}{|P| \times \omega(V_i)}$. In the CPS strategy, it needs $|V_i| \times (1+u)$ slots for each phase of job J, then it can complete

$$\frac{|P| \times \omega(V_i)}{|V_i| \times (1+u)}$$

phases of job J in each unit time. Correspondingly,

$$makespan_{cps} = \frac{|J_0| \times |V_i| \times (1+u)}{|P| \times \omega(V_i)}.$$

Then, we have $\frac{makespan_{cps}(J)}{makespan_{pcps}(J)} = \frac{|V_i| \times (1+u)}{|V_i| + u}$. □

In Formula 3, the length of phases of the master task is $1+u$ times the length of phases of other tasks. Actually, the length of phases of the master task can only be either one or two slots, while the length of phases of other tasks is one slot. Let us consider the following situation, for

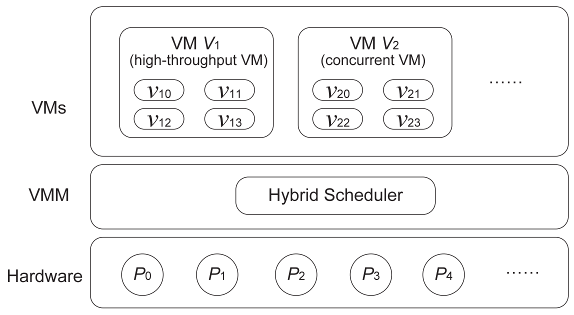

VMs
VM V_1 (high-throughput VM)
v_{10} v_{11} v_{12} v_{13}
VM V_2 (concurrent VM)
v_{20} v_{21} v_{22} v_{23}
……
VMM
Hybrid Scheduler
Hardware
P_0 P_1 P_2 P_3 P_4 ……

Fig. 3. The hybrid scheduling framework.

multithreaded or parallel programs. If the master task has to execute sequential parts as well as parallel parts like other tasks, then the phase of the master task may be two slots when it executes the parallel part and the successive sequential part. Otherwise, the phase of the master task is also one slot. Consequently, we introduce another term u-factor for the master task, which denotes the frequency of the length of its phases being equal to two slots, and $u\text{-factor} \in [0, 1]$. Then, the phase of the master task with two slots occurs in interval $\lceil 1/u\text{-factor} \rceil$. Correspondingly, the master VCPU will be allocated to the two sequential slots in interval $\lceil 1/u\text{-factor} \rceil$. Then, Formula 4 is easily deduced based on Formula 3.

Formula 4. For a concurrent job J in VM V_i, where the master task has the maximum number of phases. The length of phases of the master task is two slots in interval $\lceil 1/u\text{-factor} \rceil$, otherwise the length of phases of the master task is one slot the same as other tasks, then, we have $\frac{makespan_{cps}(J)}{makespan_{pcps}(J)} = \frac{|V_i| \times (1+u\text{-factor})}{|V_i|+u\text{-factor}}$.

4 Scheduling Framework

This section details a hybrid scheduling framework to deal with the CPU scheduling problem when multiple VMs with a variety of workloads coexist in a virtualized system.

4.1 VM Classification

For a high-throughput application, the scheduling goal is to maximize its throughput. Taking a web server application as an example, the performance goal is to maximize the number of access requests that the server can handle, assuming that threads of requests are independent from each other. Obviously, the PS scheduling strategy is suitable for this case, as adopted by the default credit scheduler [14] in Xen. For a concurrent application, the CPS or PCPS scheduling strategy is a potential efficient method as analyzed above.

Correspondingly, from the aspect of CPU scheduling, VMs in a virtualized system can be divided into two kinds, high-throughput VMs and concurrent VMs. When a VM is set as a concurrent VM, VCPUs in the VM will be scheduled by the CPS or PCPS strategy. Otherwise, the VM is a high-throughput VM, and its VCPUs are scheduled by the PS strategy.

4.2 Scheduling Framework

Given that fact that both high-throughput applications and concurrent applications could run on VMs simultaneously in a physical machine in the cloud platform, we present a hybrid scheduling framework, illustrated as Fig. 3. We have the high-throughput VM to run high-throughput applications and the concurrent VM to run concurrent applications. By default, a VM is set as a high-throughput VM. However, it can be set as a concurrent VM when the majority of its workload are concurrent applications, to reduce the cost of synchronization. Consequently, a *user interface* should be provided to the system administrator, so that the type of a VM can be set by the system administrator.

As shown in Fig. 3, it is the *Hybrid Scheduler* that implements these functions according to the scheduling algorithm (see Section 5 for details). For example, VCPUs in VM V_1 are mapped into PCPUs asynchronously as it is a high-throughput VM. On the other hand, VCPUs in VM V_2 are mapped into PCPUs synchronously because it is set as a concurrent VM. Moreover, during the lifetime of a VM, its type can be changed by the system administrator, according to the characteristics of its workload. That means, a VM can be changed from the high-throughput type to the concurrent type and vice versa, based on the types of applications it has.

Besides the number of VCPUs, the capacity of memory, and the location of the file system, the weight and the type should be set when a new VM is created. If it is a concurrent VM, its u-factor should also be set. The CPS strategy will be adopted for the VM when u-factor equals 0, otherwise the PCPS strategy will be adopted. All above information is included into the configuration file of the VM.

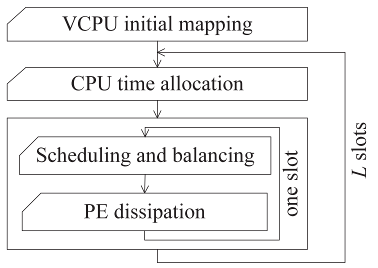


Fig. 4. The skeleton of the scheduling algorithm.

5 Scheduling Algorithm

This section presents the CPU scheduling algorithm adopted by the hybrid scheduling framework. We begin with the skeleton of the scheduling algorithm, followed by its details.

For VM V_i, $VT(V_i)$ denotes its type, with $VT(V_i) = HIT$ indicating a high-throughput VM, and $VT(V_i) = CON$ indicating a concurrent VM. We also define $VCT(v_{ij}) = HIT$ if V_i is a high-throughput VM, and $VCT(v_{ij}) = CON$ if V_i is a concurrent VM, where v_{ij} is a VCPU in VM V_i.

5.1 Algorithm Skeleton

The CPU scheduling algorithm includes four components, namely, *VCPU initial mapping*, *CPU time allocation*, *Scheduling and balancing*, and *PE dissipation*. Its skeleton is shown in Fig. 4, and these four components will be described in detail in the following sections.

When a VM is created, its VCPUs will be inserted into the run queues of PCPUs in the system, and it is the component of *VCPU initial mapping* that is responsible for executing the initial mapping of VCPUs. For a concurrent VM, its VCPUs have to be inserted into the run queues of different PCPUs, because its VCPUs will be coassigned to these PCPUs. It is the component of *CPU time allocation* that allocates the PCPU time to all VCPUs in an interval, according to the weight proportion that is assigned to the VMs. The concept of potential energy (PE) is adopted for CPU time allocation, borrowed from mechanics. The PE value of a VCPU represents how much CPU time it can consume. The specific scheduling operation is performed in the component of *Scheduling and balancing*. The coscheduling operation for VCPUs in a concurrent VM is implemented in this component, and the scheduling decision is based on the PE value of VCPUs. Once a VCPU is assigned to a PCPU, it will run for one slot. After a VCPU has run for one slot, its PE is decreased. This work is performed in the component of *PE dissipation*.

5.2 VCPU Initial Mapping

When a VM is created, each VCPU of the VM will be inserted into the run queue of a PCPU. All available PCPUs for VCPU v_{ij} make up a set of PCPUs, denoted by $AP(v_{ij})$, and we have $AP(v_{ij}) \subseteq P$. The VCPU initial mapping is shown in Algorithm 1. For VCPU v_{ij} from a concurrent VM V_i, its $AP(v_{ij})$ should not include any other VCPU from V_i, because VCPUs of a concurrent VM have to be inserted into the run queues of different PCPUs.

Algorithm 1. VCPU initial mapping

```
1: if $VCT(v_{ij}) = CON$ then
2:     Determine $AP(v_{ij})$, so that there does not exist any
       VCPU $v_{ik}$ ($k = 0, \dots, |C(V_i)| - 1$, and $k \neq j$) in the
       run queue of any PCPU $P_l \in AP(v_{ij})$;
3: else if $VCT(v_{ij}) = HIT$ then
4:     $AP(v_{ij}) \leftarrow P$;
5: end if
6: PCPU $P_{l^0}$ with the minimal workload in $AP(v_{ij})$ is
   chosen;
7: VCPU $v_{ij}$ is inserted into the run queue of PCPU $P_{l^0}$;
```

5.3 CPU Time Allocation

The CPU time of the system is allocated to VMs according to the weight proportion assigned to them. PE is adopted for CPU time allocation, and the PE value of each VCPU is reset according to the weight proportion in a certain interval.

The PE value of VCPU v_{ij} is $PE(v_{ij})$, and the dissipation of PE per slot is denoted as PE_{unit}. The time length of the allocation interval is denoted as L slots, and the time of allocating CPU time is denoted as a *allocation event*. Then at each allocation event, the procedure of allocating CPU time is shown in Algorithm 2, where VCPU v_{i0} is the master VCPU in VM V_i when it is a concurrent VM, and $uf(v_{i0})$ denotes u-factor of the master VCPU in VM V_i.

Algorithm 2. CPU time allocation

```
1: The interval of allocation events is $L$ slots;
2: The total PE of the system is $PE_{total} \leftarrow |P| \times PE_{unit} \times L$;
3: for each VM $V_i$ do
4:     The PE increment $PE_{inc} \leftarrow PE_{total} \times \omega(V_i)$;
5:     if $V_i$ is a concurrent VM then
6:         $PE(v_{i0}) \leftarrow PE(v_{i0}) + PE_{inc} \times \frac{1+uf(v_{i0})}{|C(V_i)|+uf(v_{i0})}$;
7:         for each VCPU $v_{ij} (j > 0)$ in VM $V_i$ do
8:             $PE(v_{ij}) \leftarrow PE(v_{ij}) + \frac{PE_{inc}}{|C(V_i)|+uf(v_{i0})}$;
9:         end for
10:    else
11:        for each VCPU $v_{ij}$ in VM $V_i$ do
12:            $PE(v_{ij}) \leftarrow PE(v_{ij}) + \frac{PE_{inc}}{|C(V_i)|}$;
13:        end for
14:    end if
15: end for
```

At the allocation events, each VM gets its PE increment in proportion to its weight. To a concurrent VM, the master VCPU v_{i0} may obtain more CPU time depending on the value of $uf(v_{i0})$, the remaining CPU time is distributed equally among the other VCPUs in this VM. To a high-throughput VM, the PE increment is just distributed equally among its all VCPUs. After the PE value of VCPUs is updated, VCPUs in the run queue of a PCPU will be sorted in decreasing order of their PE values.

5.4 PE Dissipation

We call the VCPU currently running on PCPU P_i $PV(P_i)$, and it is the head element in the run queue of P_i, and the run queue of P_i is denoted by $runq(P_i)$. At the end of each slot, the PE value of the VCPU running on a corresponding PCPU is decreased by PE_{unit}. The processing procedure is shown in Algorithm 3, where the boot strap processor in the system is abbreviated as BSP. Besides, a allocation event may be invoked.

Algorithm 3. PE dissipation

```
1: for each PCPU $P_k$ do
2:     $PE(PV(P_k)) \leftarrow PE(PV(P_k)) - PE_{unit}$;
3:     if $P_k$ is BSP, and a allocation event achieves then
4:         Perform CPU time allocation (Algorithm 2);
5:     else
6:         Sort the run queue of VCPUs in the decreasing
           order of PE;
7:     end if
8: end for
```

5.5 Scheduling and Balancing

The specific scheduling operation is performed in this component. At the beginning of each slot, a VCPU should be selected for each PCPU. The scheduling and balancing algorithm is described in Algorithm 4. If the PE value of the VCPU at the head of the run queue of a PCPU is more than zero, a VCPU will be selected from its run queue. If the selected VCPU is a master VCPU in a concurrent VM, other VCPUs in the VM will be coscheduled to the corresponding PCPUs. This VCPU may be scheduled not only at this slot, but also at the next slot according to its u-factor. Otherwise, a VCPU from a high-throughput VM will be selected and be scheduled to this PCPU.

Algorithm 4. Scheduling and balancing

```
1: for each PCPU $P_k$ do
2:     if $PE(PV(P_k)) > 0$ then
3:         if $VCT(PV(P_k)) = CON$ and $PV(P_k)$ is a master
           VCPU then
4:             Lookup all other VCPUs that belong to the
               same VM as $PV(P_k)$, and then their PCPUs;
```

```
5:          PCPU $P_k$ sends Inter-Processor Interrupts
            (IPIs) to these PCPUs, and all VCPUs in this
            VM are scheduled;
6:          $PV(P_k)$ is scheduled to PCPU $P_k$ for this slot,
            and will also be scheduled to PCPU $P_k$ for the
            next slot in the interval $\lceil \frac{1}{uf(PV(P_k))} \rceil$;
7:        else if traverse its run queue until it finds the first
          $v_{ij}$, where $VCT(v_{ij}) = HIT$ then
8:          VCPU $v_{ij}$ is scheduled to PCPU $P_k$;
9:        end if
10:     else
11:       Lookup $v_{i^*j^*}$ at the head of the run queue of
          $P_{k'}(k' \neq k)$, where $PE(v_{i^*j^*}) = \max_{k'} PE(PV(P_{k'}))$,
          and $runq(P_k) \cap C(V_{i^*}) = \phi$ if $VT(V_{i^*}) = CON$;
12:       Migrate $v_{i^*j^*}$ to $runq(P_k)$ and schedule it to PCPU
          $P_k$;
13:     end if
14: end for
```

If the PE value of the VCPU at the head of the run queue of a PCPU is not more than zero, VCPUs in its run queue will not be scheduled in this slot, because its PE has been used up for this period. A VCPU in the run queue of the other PCPU may be migrated to this PCPU by the balancing mechanism. Meanwhile, it should avoid having two VCPUs from a concurrent VM stay in the run queue of the same PCPU, incurred by the VCPU migration. If no VCPU can be chosen for a PCPU in Algorithm 4, the idle VCPU from the idle VM will be scheduled to the specific PCPU.

This algorithm implements the coscheduling of VCPUs from a concurrent VM and the asynchronous scheduling of VCPUs from a high-throughput VM, while keeping load balancing between multiple PCPUs.

6 Performance Evaluation

To evaluate the performance of hybrid scheduling, we perform a series of experiments with benchmarks. First, we test the performance of a single VM running with workloads, and analyze the performance of schedulers in the VMM while varying the weight of the VM. Then, we test the performance of multiple VMs running simultaneously, and analyze the benefit and influence of hybrid scheduling on the performance of the whole system and individual applications.

6.1 Experimental Methodology

We have implemented a working prototype of a Hybrid Scheduler in a VMM. It is based on open-source Xen [5], and the guest operating system is Linux. We choose Linux and Xen because of their broad acceptance and the availability of their open-source code. Specifically, the virtualized system ran Xen 3.3.0, and all VMs ran the Fedora Core 6 Linux distribution with the Linux 2.6.18 kernel. The user interface of the Hybrid Scheduler is implemented in the C and Phyton languages. Specifically, we added a new option "`xm sched-pe`" into the Xen management tool (i.e., `xm`). Through this user interface, the VM's type, weight, and u-factor can be set directly by the system administrator.

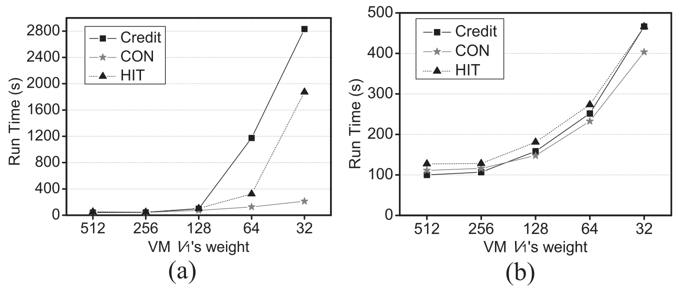


Fig. 5. (a) LU workload. (b) SP workload.

A virtual SMP system could be used to run a high-throughput workload with multiple tasks running independently, or a concurrent workload with frequent synchronization operations between multiple tasks. Correspondingly, the SPEC rate metrics of SPEC CPU2000 [15] is adopted; it measures the throughput of a machine by running multiple copies of the benchmark simultaneously. The NAS parallel benchmarks [16], [17] are adopted as the concurrent workload with multiple threads; they are multithreaded programs with synchronization operations.

All experiments were executed on a Dell Precision T5400 workstation, with dual quad-core Xeon X5410 CPUs and 8GB of RAM. The number of PCPUs is therefore eight in all experiments.

6.2 Testing a Single VM

In this testing scenario, a VM V_1 is configured with four VCPUs and 1,024 MB memory, with its weight decreasing from 512 to 32. Besides, VM V_0 is the manager VM (i.e., Domain-0 in Xen), and is configured with eight VCPUs and 1,024 MB memory, and its weight is fixed as 256 without workload on it. In the default Xen with the credit scheduler and the modified Xen with the Hybrid Scheduler, VM V_1 is configured to use the *nonwork-conserving* mode [18]. That is, the CPU time obtained by the VM is strictly in proportion to its weight, and it cannot receive any extra CPU time. For example, the percentage of time of a VCPU in VM V_1 being mapped to a PCPU is 66.7 percent ($= \frac{8 \times \frac{128}{128+256}}{4}$), if its weight is 128 and the CPU time obtained by the VM is equally distributed among its all VCPUs.

For each situation, the performance of VM V_1 in the default Xen is denoted by `Credit`. In the modified Xen with the Hybrid Scheduler, the CPS scheduling strategy is adopted for the concurrent VM. The performance of VM V_1 is denoted by `CON` when it is set as a concurrent VM, while the performance of VM V_1 is denoted by `HIT` when it is set as a high-throughput VM.

First, the scalar pentadiagonal (SP) benchmark and the lower upper (LU) symmetric Gauss-Seidel benchmark are chosen from the NAS parallel benchmarks [16], [17], and the problem size is configured as *Class A* with the default parameters. The number of threads of the two parallel benchmarks is set as 4, and the runtime of each benchmark is shown in Figs. 5a and 5b.

According to Figs. 5a and 5b, a concurrent workload obtains similar performance in `Credit` and `CON`, when its weights are 512, 256, and 128. A VCPU cannot run on two PCPUs at the same time, as a result, when its weight is 512 or 256, the CPU time actually obtained by VM V_1 is the

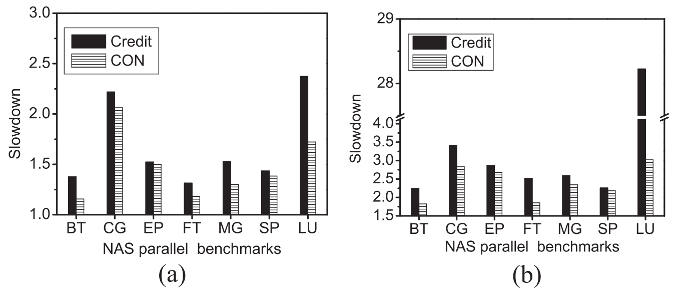
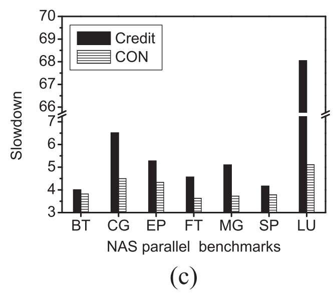



Fig. 6. Slowdowns of NAS parallel benchmarks in VM V_1, where its weight is (a) 128, (b) 64, and (c) 32.

same, and its four VCPUs are online all the time. When its weight reduced to 128, the percentage of time of each VCPU being mapped to a PCPU becomes 66.7 percent, and there is an expected increase in the runtime for concurrent workloads. It seems that virtualization has little impact on synchronization in the guest operating system. However, when the weight is reduced to 32, it is observed that the runtime has a larger increase in `Credit` than in `CON`, especially when the workload is LU, as there are many synchronous communication operations in LU, which leads to performance degradation.

Next, we run other benchmarks in NAS parallel benchmarks on VM V_1, and test their runtimes. When its weight is 512 or 256, its four VCPUs are online all the time, and the impact of virtualization on synchronization is relatively small. Therefore, we test V_1 with its weight decreasing from 128 to 32. To provide an intuitive comparison, we define the slowdown of a benchmark running on a VM with a weight (<256) as the ratio of its runtime to the runtime of the same benchmark running on the same VM scheduled by the default Xen with the weight equaling 256. According to the result shown in Fig. 6, the CPS strategy (shown in `CON`) could improve the performance of all parallel benchmarks including LU and SP. For simplicity, in the subsequent experiments, we select LU and SP from NAS parallel benchmarks. The former has the maximal performance degradation while the later has normal performance degradation, with the default Xen.

Now, we study the performance of the high-throughput workload on the virtualized system. The workload in VM V_1 is SPEC CPU2000, and four copies of the benchmarks run on it simultaneously. We measure the throughput of VM V_1 by the rate metrics in the SPEC CPU2000. The results are shown in Figs. 7a and 7b, and indicate that the performance of the Hybrid Scheduler is close to the performance of the credit scheduler when workloads are high-throughput applications. This is because there is no synchronization between individual tasks in a high-throughput application, and only one VM with a workload runs in the system. It is

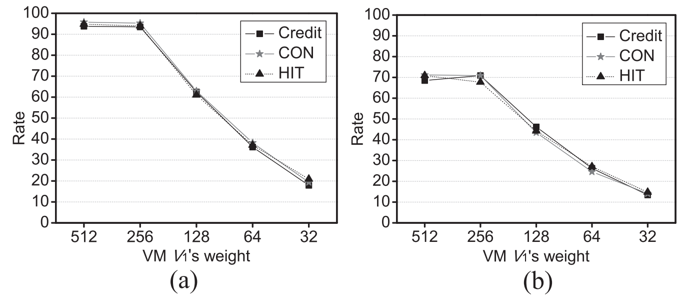


Fig. 7. (a) 176.gcc workload. (b) 256.bzip2 workload.

simply that VCPUs in the VM are mapped to PCPUs in a certain percentage of time in response to the weight.

Besides the performance issue, another significant issue is CPU fairness in the VM system. The concept of weight is adopted to solve this issue, and the CPU usage of a VM is in proportion to the weight that the VM has been assigned. We use XenMon [19] to gather the CPU usage information of VMs for analyzing fairness. We monitor the CPU usage percentage of VM V_1 over 60 seconds while the LU benchmark is testing, and the results for the three cases are shown in Fig. 8. During the monitoring procedure, the weight of VM V_1 is 128 while the weight of VM V_0 is 256, therefore, the theoretical CPU utilization percent of VM V_1 is 33.3 percent. The experimental result shows that the average CPU utilization of VM V_1 is approximately equal to the theoretical value in the three cases. Therefore, both the credit scheduler and the Hybrid Scheduler can guarantee the CPU fairness among VMs in the virtualized system, and the CPU time obtained by a VM can be controlled according to its weight. As VM V_1 is dedicated to running benchmarks, it is guaranteed that performance difference is due to the difference in scheduling strategies, rather than the system noise [20].

6.3 Extending CPS to PCPS

In the above section, for comparing hybrid scheduling with the default scheduling in Xen, the CPS strategy was adopted for a concurrent VM. Now, we adopt the PCPS strategy for a concurrent VM, and then the performance of

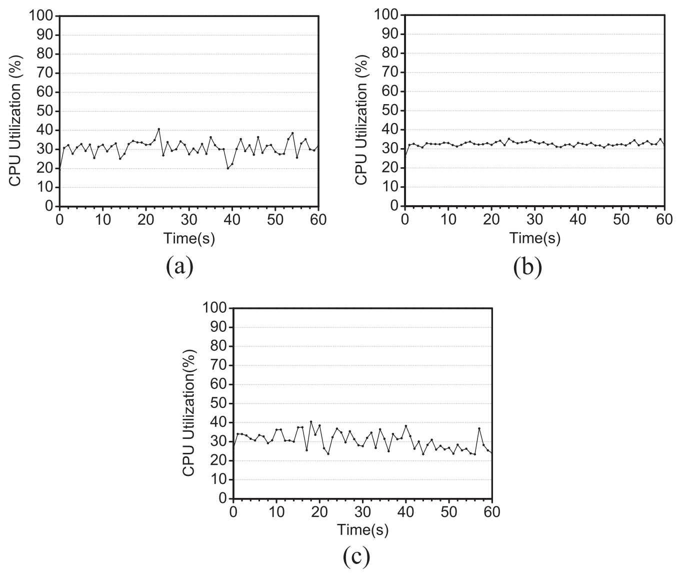


Fig. 8. (a) VM V_1 (`Credit`). (b) VM V_1 (`CON`). (c) VM V_1 (`HIT`).

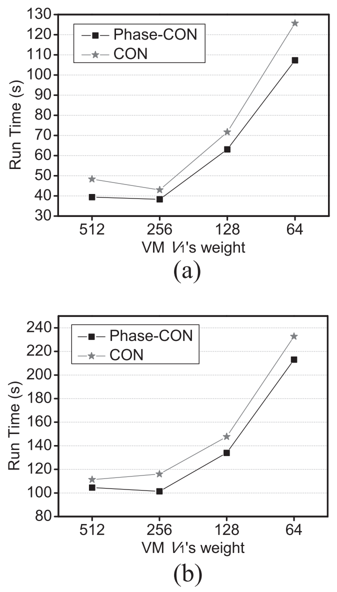


Fig. 9. (a) LU workload. (b) SP workload.

VM V_1 is retested with the PCPS strategy when benchmarks are LU and SP, which is denoted by `Phase-CON`. Parallel benchmarks including LU and SP consist of sequential parts and parallel parts, and the sequential parts are executed in the master task. Therefore, u-factor is set as 0.25 in the PCPS strategy according to the proportion of the sequential part in parallel benchmarks. The result is shown in Fig. 9.

According to Fig. 9, it is observed that the PCPS strategy (`Phase-CON`) achieves better performance than the CPS strategy (`CON`) for a concurrent VM, when the master task has to execute the sequential parts in a parallel program. Therefore, the runtime of this kind of parallel program can be further reduced when more CPU time is allocated to the master VCPU instead of the CPU time being equally distributed among all VCPUs.

6.4 Testing Multiple VMs

In this scenario, there are six VMs (VM V_1, VM V_2, ..., VM V_6), and we test various combinations of these VMs, in which multiple VMs run on the system simultaneously. VM V_0 is also configured as in Section 6.2, without workload on it. The configuration of VMs is shown as Table 1. In experiments, VMs are configured to use the *work-conserving* mode [18]. That is, the shares are merely guarantees, and a VM is eligible to receive extra CPU time if other VMs are blocked or idle. Workloads and performance metrics are the same as in Section 6.2, except that the metrics for SPEC CPU2000 is the runtime of four copies of benchmarks.

The runtimes of workloads may differ from each other, as a result, multiple workloads will not finish at the same time when they simultaneously start to run on corresponding VMs. Therefore, we run each workload repeatedly with a batch program. The number of repetitions of each workload is set to be large enough that all other workloads are still running when each workload finishes its first 10 rounds. The result is the average value of the runtimes of the first 10 rounds for each workload. Besides, the confidence interval of the runtimes is also given, in which the confidence level is 0.95.

TABLE 1
VMs' Configuration

VM	the number of VCPUs	memory (MB)	weight
V_0	8	1024	256
$V_{1\sim6}$	4	1024	256

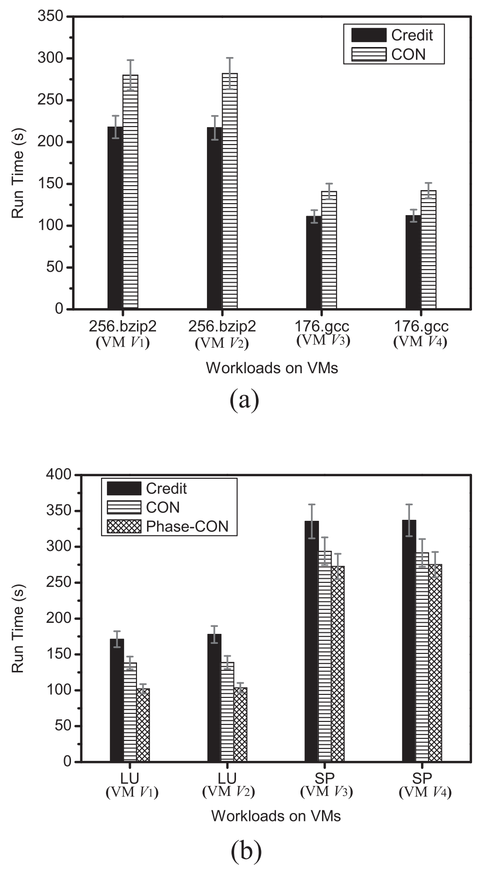


Fig. 10. (a) Four high-throughput workloads run simultaneously on four VMs. (b) Four concurrent workloads run simultaneously on four VMs.

First, we test the combination of four VMs running simultaneously. As depicted in Fig. 10a, we run four high-throughput workloads on these VMs. The VMs are set as concurrent VMs in the modified Xen with the Hybrid Scheduler, and the CPS strategy is adopted (shown in `CON`), to verify whether coscheduling has negative influence on the high-throughput workload. Although it has little negative impact when only one VM with this workload runs in the virtualized system, illustrated as Figs. 7a and 7b. According to Fig. 10a, coscheduling leads to a runtime increase of about 25 percent in the scenario of multiple VMs running simultaneously. It is because coscheduling also introduces additional cost to scheduling in the VMM, which may incur performance loss when multiple VMs run simultaneously.

Next, we run four concurrent workloads on these VMs, illustrated as Fig. 10b. The four VMs are also set as concurrent VMs with the Hybrid Scheduler, and the CPS strategy is adopted (shown in `CON`). It is observed that concurrent workloads obtain better performance in `CON` than in `Credit`, and the runtime is reduced by about 22 percent for LU and about 15 percent for SP. Moreover, the PCPS strategy is also used to coschedule VCPUs from concurrent VMs (shown in `Phase-CON`). It can obtain better

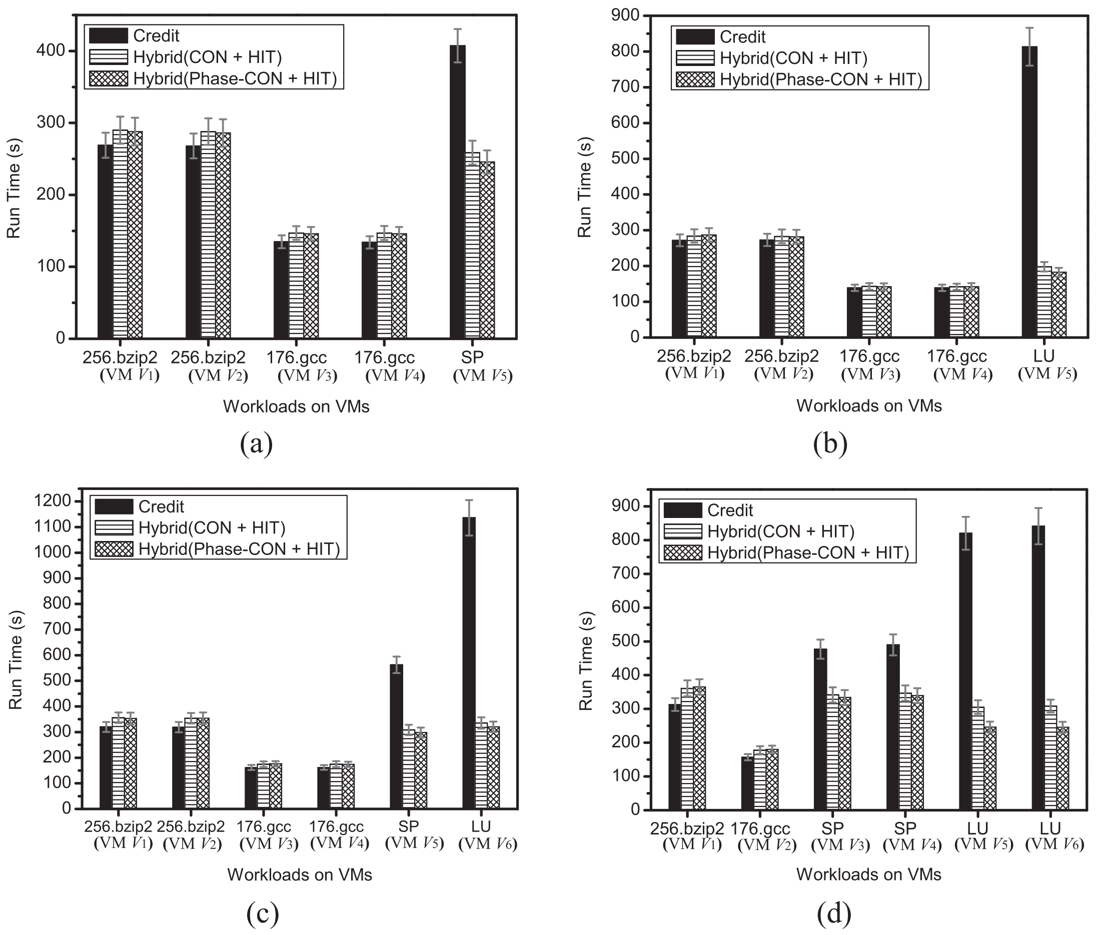


Fig. 11. (a) and (b) Four high-throughput workloads and one concurrent workload run simultaneously on five VMs, where VM V_5 is a concurrent VM, and other VMs are high-throughput VMs. (c) Four high-throughput workloads and two concurrent workloads run simultaneously on six VMs, where VM $V_1,\ldots,$ VM V_4 are high-throughput VMs, and VM V_5 and VM V_6 are concurrent VMs. (d) Two high-throughput workloads and four concurrent workloads run simultaneously on six VMs, where VM V_1 and VM V_2 are high-throughput VMs, and VM $V_3,\ldots,$ VM V_6 are concurrent VMs.

performance as it allocates more CPU time to the master task in concurrent workloads, and the runtime is reduced by about 40 percent for LU and about 19 percent for SP, compared with `Credit`.

According to Figs. 10a and 10b, coscheduling improves the performance of concurrent workloads, while degrading the performance of high-throughput workloads in the virtualized system. Therefore, hybrid scheduling is an ideal solution for running various VMs simultaneously in the system.

Finally, in the modified Xen with the Hybrid Scheduler, we set the VM's type corresponding to the workload running on it. For a concurrent VM, we test its performance with the CPS strategy and the PCPS strategy. When the performance of VMs is tested and the CPS strategy is adopted for concurrent VMs, it is denoted by `Hybrid(CON+HIT)`. `Hybrid(Phase-CON+HIT)` denotes that the PCPS strategy is adopted for concurrent VMs, and u-factor is 0.25 as Section 6.3. For each combination of multiple VMs, we also test the performance of VMs with the default credit scheduler in Xen, denoted by `Credit`. The experimental result is shown in Fig. 11. It is observed that the runtime of LU with the credit scheduler is about three to four times the runtime with the Hybrid Scheduler, that is, in the best case the Hybrid Scheduler can save up to about 75 percent of the runtime for LU. Although the Hybrid Scheduler cannot reduce the runtime for SP as well as LU, it can still save up to about 30 percent of the runtime for SP in the worst case, compared with the credit scheduler.

6.5 Discussion

The credit scheduler in Xen can achieve good performance when there are only high-throughput workloads in VMs. However, the characteristics of concurrent applications should be considered in the virtualized system, otherwise the virtual SMP system could introduce significant overhead and consequent performance degradation. This result has been analyzed theoretically in Section 3, and is validated again by the above experiments.

As a VM with a concurrent workload is set as a concurrent VM, and then its VCPUs are coscheduled to PCPUs. This is helpful to mitigate the negative influence of virtualization on synchronization. Therefore, a concurrent workload can obtain better performance with the Hybrid Scheduler. Meanwhile, a high-throughput workload with the Hybrid Scheduler steals less CPU time through the load balancing mechanism, therefore its runtime will increase a little compared to the credit scheduler. It is a tradeoff between improved performance and additional cost. However, experiments demonstrate that the performance profit of concurrent workloads is far greater than the performance loss of high-throughput workloads. It is also demonstrated that the PCPS strategy could further improve the performance of concurrent workloads in the virtualized system.

It is also noted that an appropriate type should be chosen for a VM according to the characteristics of its workloads, otherwise, the Hybrid Scheduler may reduce the performance. In the cloud platform, a VM is usually created for a specific purpose, and then the workload characteristics could be known in advance. As a result, the system administrator could determine the type of VMs in the system and set the value of u-factor. Moreover, the system administrator could also change the type and u-factor of a VM via the user interface "xm sched-pe," responding to the change of its workloads.

7 Related Work

The scheduling issue in the VMM had a close relationship with scheduling of operating systems. A simple notion of priority for process scheduling is usually used in conventional operating systems. A task with a higher priority is scheduled prior to a task with a lower priority, and priorities may be static or be dynamically recalculated. There are many sophisticated priority schemas such as decay usage scheduling [21], which is a priority and usage-based mechanism for CPU scheduling employed by BSD Unix [22]. For ensuring that a particular application receives a certain percentage of the CPU usage, the fair-share scheduler is introduced [23], [24], and it can provide PS among processes and users in a way compatible with a UNIX-style time sharing framework. However, experiments indicate that the fair-share scheduler provides reasonable proportional fairness over relatively large time intervals [25]. Lottery scheduling provides a more disciplined PS approach than fair-share schedulers [26].

Applications can be typically divided into two types: computation-intensive applications and I/O-intensive applications. Correspondingly, current research efforts are focused on the scheduling issue in the VMM with computation-intensive workloads and I/O-intensive workloads.

For improving the performance of computation-intensive workloads in the VM system, especially for a VM with multiple VCPUs, VMware, and Xen provide their schedulers in the VMM. Coscheduling is adopted in VMware's series products [12], which is an add-on software module. VMkernel always coschedules VCPUs of a multi-VCPU VM, although it adopts a relaxed coscheduling to allow VCPUs to be scheduled on a slightly skewed basis. We have demonstrated that coscheduling could degrade the performance of high-throughput workloads in VMs, therefore, coscheduling is not suitable for high-throughput workloads. As an open-source product, the current default scheduler in Xen [4] is the credit scheduler [14], and this scheduler does not attempt to coschedule VCPUs. It adopts a proportional share scheduling strategy, and tries to maximize the throughput of the system while guaranteeing fairness. However, it is also demonstrated that synchronization in concurrent workloads incurs significant overhead in VMs when these workloads run with the default scheduler in Xen. Therefore, the credit scheduler is not suitable for concurrent workloads in the virtualized system. Besides, there are some research efforts [27], [28] based on the previous conference version of our paper, which emphasize the importance of coscheduling for the virtualized system as well as improve it.

Differing from above schedulers in VMMs, in this paper, we propose a hybrid scheduling framework, in which VCPUs from a concurrent VM are coscheduled while VCPUs from a high-throughput VM are scheduled asynchronously, to improve the performance of concurrent workloads, while keeping the performance of high-throughput workloads. As the separation of mechanism and policy is an important principle in computer science [29], we argue that the hybrid scheduling framework is the fundamental approach for adapting to the diversity of VMs in the cloud platform (i.e., mechanism), and when and how to coschedule VCPUs on PCPUs will depend on the specific policy, which is implemented via the user interface in the hybrid scheduling framework. Moreover, the hybrid scheduling framework supports distributing CPU time among VCPUs based on demand, as well as distributing equally.

In the aspect of I/O-intensive workloads, a communication-aware CPU scheduler for Xen is proposed [30], [31]. The SEDF scheduler is modified so that it counts the number of packets sent or received by each VM and preferentially schedules I/O-intensive VMs. To improve I/O responsiveness while keeping CPU fairness, a partial boosting mechanism is adopted in [32]. If a VCPU has at least one inferred I/O-bound task and an event is pending for the VCPU, the VMM initiates partial boosting for the VCPU regardless of its priority. The SEDF and credit schedulers are evaluated with different configurations in [33]. Furthermore, they make some extensions such as fixing event channel notification and ordering the run queue within the CPU scheduler of the VMM to improve I/O performance. VMM-bypass I/O [34] is also a method to improve I/O performance, which allows guest domains to carry out I/O operations without the involvement of the VMM and the privileged domain. Self-virtualized devices [35], [36] also share the same idea. Another relevant work is concerned with monitoring the performance [19], [37], [38], [39] (especially, the overhead for I/O processing), and the scheduler is configured with the gained insight from those information to achieve a high performance.

Our work is also related to scheduling on multiprocessor or distributed systems. Coscheduling (or gang scheduling) and backfilling [40] are two major strategies for scheduling parallel jobs. Coscheduling [41], [42], [43], [44] tries to schedule related threads or processes to run simultaneously on different processors. When scheduling any of processes in a related group, all of these processes are scheduled for execution so that they can communicate efficiently. Otherwise, one process may wait to send or receive a message to another. Backfilling scheduling such as [45], [46], on the other hand, attempts to schedule jobs that are behind in the priority queue of waiting jobs to unutilized nodes, rather than keep them idle. To prevent starvation of larger jobs, it requires that the execution of a job selected out of order will not delay the start of jobs that are ahead of it in the priority queue. This method is based on the estimation of job execution time. Some papers [47], [48] propose combining these two strategies for better performance. Besides, a group of physical machines could also be hooked together by a network to be used as a single VM for executing parallel programs, and there are a lot of research efforts such as [49], [50], [51].

Coscheduling is a straightforward concept for scheduling multiple computing entities (VCPUs, processes, or threads); it is expected to reduce the execution time of concurrent applications, meanwhile, it may incur additional overhead. The challengeable issue is how to implement coscheduling of computing entities effectively on a specific type of computing systems. Those research efforts for coscheduling on multiprocessor or distributed systems could not be applied directly to the VMM scheduling. The design of schedulers in the VMM should consider the VMM-level scheduling issues such as proportional fairness, load balancing, and minimizing additional overhead as well as coscheduling. Our proposed hybrid scheduling framework with corresponding strategies could meet the requirements with a better performance.

8 Conclusion

In this paper, we present a hybrid scheduling framework for CPU management in the VMM. We also propose CPS and PCPS strategies for concurrent workloads. For adapting to the diversity of VMs in the cloud platform, a VM can be set as the concurrent type when the majority of its workloads are concurrent applications, and then the CPS strategy or the PCPS strategy is adopted to mitigate the negative influence of virtualization on synchronization. Otherwise, it is set as the high-throughput type as the default, and the PS strategy is adopted. Experiments indicate that the hybrid CPU management is feasible to improve the performance of concurrent applications while maintaining the performance of high-throughput applications, which run simultaneously on VMs in the cloud platform.

Acknowledgments

This work was supported partly by National Natural Science Foundation of China (Nos. 60970008, 90715030, and 60503043), and National Key Basic Research and Development Plan (973 Plan) of China (No. 2007CB310900), and Research Fund for the Doctoral Program of Higher Education of China (No. 20100073110016).

References

[1] P.H. Gum, "System/370 Extended Architecture: Facilities for Virtual Machines," *IBM J. Research and Development,* vol. 27, no. 6, pp. 530-544, 1983.

[2] J.E. Smith and R. Nair, *Virtual Machines: Versatile platforms for systems and processes.* Elsevier, 2005.

[3] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/, 2012.

[4] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," *Proc. ACM Symp. Operating Systems Principles (SOSP),* pp. 164-177, 2003.

[5] Xen Hypervisor, http://www.xen.org/, 2012.

[6] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI),* pp. 181-194, 2002.

[7] VMware Virtualization Software, http://www.vmware.com/, 2012.

[8] Kernel Based Virtual Machine, http://www.linux-kvm.org, 2012.

[9] Hyper-V, http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx, 2012.

[10] VirtualBox, http://www.virtualbox.org/, 2012.

[11] Amazon Case Studies, http://aws.amazon.com/solutions/case-studies, 2012.

[12] S. Lowe, *Mastering VMware vSphere 4.* Wiley Publishing, Inc., 2009.

[13] V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski, "Towards Scalable Multiprocessor Virtual Machines," *Proc. Third Virtual Machine Research and Technology Symp.,* 2004.

[14] Credit Scheduler, http://wiki.xensource.com/xenwiki/credit scheduler, 2012.

[15] Standard Performance Evaluation Corporation, http://www.spec.org/, 2012.

[16] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga, "The NAS Parallel Benchmarks—Summary and Preliminary Results," *Proc. ACM/IEEE Conf. Supercomputing,* pp. 158-165, 1991.

[17] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance," Technical Report NAS-99-011, NASA Ames Research Center, 1999.

[18] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 35, no. 2, pp. 42-51, 2007.

[19] D. Gupta, R. Gardner, and L. Cherkasovah, "XenMon: QoS Monitoring and Performance Profiling Tool," Technical Report HPL-2005-187, HP Labs, 2005.

[20] D. Tsafrir, Y. Etsion, D.G. Feitelson, and S. Kirkpatrick, "System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications," *Proc. 19th Ann. Int'l Conf. Supercomputing (ICS),* pp. 303-312, 2005.

[21] J.L. Hellerstein, "Achieving Service Rate Objectives with Decay Usage Scheduling," *IEEE Trans. Software Eng.,* vol. 19, no. 8, pp. 813-825, Aug. 1993.

[22] M.K. McKusick, K. Bostic, M.J. Karels, and J.S. Quarterman, *The Design and Implementation of the 4.4 BSD Operating System.* Addison-Wesley, 1996.

[23] J.J. Kay and P. Lauder, "A Fair Share Scheduler," *Comm. ACM,* vol. 31, no. 1, pp. 44-55, 1988.

[24] G.J. Henry, "The Fair Share Scheduler," *AT&T Bell Labs Technical J.,* vol. 63, no. 8, pp. 1945-1957, 1984.

[25] R.B. Essick, "An Event Based Fair Share Scheduler," *Proc. Winter USENIX Conf.,* pp. 147-161, 1990.

[26] C.A. Waldspurger and W.E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proc. First Symp. Operating Systems Design and Implementation (OSDI),* pp. 1-11, 1994.

[27] O. Sukwong and H.S. Kim, "Is Co-Scheduling Too Expensive for SMP VMs?" *Proc. European Conf. Computer Systems (EuroSys),* 2011.

[28] Y. Bai, C. Xu, and Z. Li, "Task-Aware Based Co-Scheduling for Virtual Machine System," *Proc. ACM Symp. Applied Computing,* pp. 181-188, 2010.

[29] B.W. Lampson and H.E. Sturgis, "Reflections on an Operating System Design," *Comm. ACM,* vol. 19, no. 5, pp. 251-265, 1976.

[30] S. Govindan, A.R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Scheduling for Consolidated Xen-Based Hosting Platforms," *Proc. ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE),* pp. 126-136, 2007.

[31] S. Govindan, J. Choi, A.R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Management in Consolidated Xen-based Hosting Platforms," *IEEE Trans. Computers,* vol. 58, no. 8, pp. 1111-1125, Aug. 2009.

[32] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-Aware Virtual Machine Scheduling for I/O Performance," *Proc. ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE),* pp. 101-110, 2009.

[33] D. Ongaro, A.L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," *Proc. ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE),* pp. 1-10, 2008.

[34] J. Liu, W. Huang, B. Abali, and D.K. Panda, "High Performance VMM-Bypass I/O in Virtual Machines," *Proc. USENIX Ann. Technical Conf.,* 2006.

[35] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A.L. Cox, and W. Zwaenepoel, "Concurrent Direct Network Access for Virtual Machine Monitors," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* pp. 306-317, 2007.

[36] H. Raj and K. Schwan, "High Performance and Scalable I/O Virtualization via Self-Virtualized Devices," *Proc. 16th Int'l Symp. High Performance Distributed Computing (HPDC),* pp. 179-188, 2007.
[37] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," *Proc. USENIX Ann. Technical Conf.,* 2005.
[38] A. Menon, J.R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing Performance: Overheads in the Xen Virtual Machine Environment," *Proc. ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE),* pp. 13-23, 2005.
[39] S.T. Jones, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Antfarm: Tracking Processes in a Virtual Machine Environment," *Proc. USENIX Ann. Technical Conf.,* 2006.
[40] D.G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Scheduling - A Status Report," *Proc. 10th Int'l Conf. Job Scheduling Strategies for Parallel Processing (JSSPP),* pp. 1-16, 2004.
[41] J.K. Ousterhout, "Scheduling Techniques for Concurrent Systems," *Proc. Third Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 22-30, 1982.
[42] D.G. Feitelson and L. Rudolph, "Gang Scheduling Performance Benefits for Fine-Grain Synchronization," *J. Parallel and Distributed Computing,* vol. 16, no. 4, pp. 306-318, 1992.
[43] A. Batat and D.G. Feitelson, "Gang Scheduling with Memory Considerations," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 109-114, 2000.
[44] D.G. Feitelson and M.A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," *Proc. Job Scheduling Strategies for Parallel Processing Conf. (JSSPP),* pp. 238-261, 1997.
[45] B. Lawson, E. Smirni, and D. Puiu, "Self-Adapting Backfilling Scheduling for Parallel Systems," *Proc. Int'l Conf. Parallel Processing (ICPP),* pp. 583-592, 2002.
[46] E. Shmueli and D.G. Feitelson, "Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling," *Proc. Job Scheduling Strategies for Parallel Processing Conf. (JSSPP),* pp. 228-251, 2003.
[47] Y. Wiseman and D.G. Feitelson, "Paired Gang Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 6, pp. 581-592, June 2003.
[48] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 133-142, 2000.
[49] Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/, 2012.
[50] R.B. Yehezkael, Y. Wiseman, H.G. Mendelbaum, and I.L. Gordin, "Experiments in Separating Computational Algorithm from Program Distribution and Communication," *Proc. Fifth Int'l Workshop Applied Parallel Computing,* pp. 268-278, 2001.
[51] M. Geva and Y. Wiseman, "Distributed Shared Memory Integration," *Proc. IEEE Int'l Conf. Information Reuse and Integration,* pp. 146-151, 2007.

Chuliang Weng received the PhD degree in computer software and theory from Shanghai Jiao Tong University (SJTU), China, in 2004. He is currently an associate professor of computer science with the Department of Computer Science and Engineering at SJTU, and a visiting associate research scientist with the Department of Computer Science at Columbia University, New York. His research interests include parallel and distributed system, system virtualization/cloud, operating system, and system security. He is a member of the IEEE, ACM, and CCF.

Minyi Guo received the PhD degree in computer science from the University of Tsukuba, Japan. He is currently a chair professor and a head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He received the national science fund for distinguished young scholars from NSFC in 2007. His research interests include parallel and distributed computing, compiler optimizations, embedded systems, pervasive computing, and bioinformatics. He has more than 300 publications in major journals and international conferences in these areas. He is on the editorial board of the journals *IEEE Transactions on Parallel and Distributed Systems* and *IEEE Transactions on Computers*. He is a senior member of the IEEE, a member of ACM, IEICE IPSJ, and CCF.

Yuan Luo received the PhD degree in probability and mathematical statistics from Nankai University, China, in 1999. From July 1999 to April 2001, he held a postdoctoral position in the Institute of Systems Science, Chinese Academy of Sciences, China. From May 2001 to April 2003, he had a postdoctoral position in the Institute for Experimental Mathematics, University of Duisburg-Essen, Germany. He is currently a full professor of computer science with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His research interests include information theory, coding theory, and computer security. He is a member of the IEEE.

Minglu Li received the PhD degree in computer software from Shanghai Jiao Tong University (SJTU), China, in 1996. Now, he is a full professor and the vice dean of the School of Electronic Information and Electrical Engineering, and the director of Grid Computing Center of SJTU. Currently, his research interests include grid and cloud computing, services computing, wireless sensor networks, and Internet of things. He has presided more than 20 projects supported by the National Natural Science Foundation, 863 Program, 973 Program, and Science and Technology Commission of Shanghai Municipality. He has published more than 200 papers in academic journals and international conferences. He is on the editorial board of Elsevier's *journal Computer Communications*, and the Executive Committee of the Technical Committee on Services Computing, and the Technical Committee on Parallel Processing of the IEEE Computer Society. He is a member of the IEEE.