Journal of Computers Vol. 30 No. 6, 2019, pp. 100-109
doi:10.3966/199115992019123006008

Research on VCPU Collaborative Scheduling Mechanism Based on Virtual Domain

Xiaodong Liu*, Songyang Li, Miao Wang

School of Computer Science, Henan Institue of Engineering, No. 1, Xianghe Road, Longhu, Zhengzhou, Henan, China
liuxiaodongxht@qq.com

Received 23 January 2018; Revised 25 May 2018; Accepted 4 July 2018

Abstract. Aiming at the problem that the performance and predictability of application are degraded caused by the uncertainty of VCPUs' (Virtual CPU) scheduling order in virtual computing environment, this paper presents a VCPU collaborative scheduling method based on virtual domain (VDS). It combines co-scheduling and virtual domain, which can avoid the uncertainty of VCPUs' scheduling order and improve the performance and predictability of application. The performance of the proposed method is analyzed and validated through various workloads. The results show that the proposed method can effectively improve the performance and predictability of application.

Keywords: priority inversion, virtual domain, virtual machine

1 Introduction

With the advantages of functional isolation, manageability and live migration, virtualization technology has been widely used [1]. The virtualization technology, such as Xen [2], allows multiple virtual machines (VMs) to run simultaneously on the same hardware platform. Like a real physical machine, a VM can run any application, operation system or kernel without modifications. A VM can be configured with different hardware settings, such as the number of VCPUs. A VM with multiple VCPUs, which behave identically, is called a symmetric multiprocessing (SMP) VM [3]. virtualization provides convenience and safety for applications. meanwhile, it can cause problems which do not exist in a non-virtualized environment. Virtualization disrupts the basis of spinlock synchronization in the guest operating system.

In non-virtualized environment, lock-waiter thread never gets scheduled before lock-holder. However, in virtualized environment, scheduler is completely unaware of the association between VCPUs. Scheduler may schedule VCPU executing lock-waiter thread before lock-holder VCPU or it may pre-empt lock-holder VCPU [4]. This is known as lock holder preemption (LHP) problem. It violates the most basic primitive of operating system and causes problems like VCPU stacking due to which VCPUs of the same domain are stacked on the run queues of the same physical processor, which can lead to the performance of applications running on VMs degradation.

To address LHP problem, Usman Rafi et al. propose a co-scheduling method. In this method, either all VCPUs of a VM are scheduled at the same time or none of the VCPU is scheduled. This method removes VCPU stacking problem, but this approach suffers from the serious drawbacks like CPU fragmentation and priority inversion [4]. There are some work uses spinlock detection mechanism [5-6]. Once VCPUs have spinlock, the execution of lock-waiter VCPUs will be terminated or priority scheduling lock-holder VCPUs. However, existing spin lock detection mechanism is not accurate enough [7]. Wang et al. [8] propose a consolidation-aware VCPU scheduling scheme which takes advantages of the low over¬head hardware assisted PLE mechanism to detect lock waiter VCPUs. Based on VCPU over-commitment rate, the CVS scheduling selects appropriate scheduling algorithm. However, existing spin lock detection

* Corresponding Author

mechanism is not accurate enough [9].

This paper presents a VCPU collaborative scheduling method based on virtual domain (VDS). The method combines co-scheduling and virtual domain. A virtual domain is a virtual set of all VCPUs of a physical machine. In each scheduling period, when there is idle CPUs, VCPUs of virtual domain will be selected to schedule. The problem of CPU fragmentation and priority inversion can be avoided. The proposed techniques have been implemented on the Xen VMM. The distinguished features of VCPU collaborative scheduling method compared to previous work are as follows: Firstly, the proposed schedule combines co-scheduling and virtual domain. It can avoid CPU fragmentation and priority inversion. Secondly, our implementation is confined to the VMM layer, without VM dependency. The proposed techniques are evaluated for various workloads. We demonstrate that our scheduling method can improve the performance and predictability of applications.

The remainder of this paper is organized as follows: section 2 presents the background and related work. Section 3 gives theoretical basis. Section 4 describes the VCPU collaborative scheduling method. Section 5 shows the experimental results. Finally, section 6 summarizes our conclusions.

2 Background and Related Work

Our work is motivated by previous work on solving spinlock of VCPU. This section presents the background and related work.

2.1 Background

Xen is an open-source VMM that allows multiple operating systems to share the same physical server in a safe and resource managed fashion. Fig. 1 describes the xen architecture. Xen hypervisor provides an abstraction layer between virtual machines and hardware resources. This layer performs functions such as scheduling CPU and allocating memory among virtual machines. There is a privileged domain (Dom0) in the xen VMM which is used to manage guest domains (DomUs). The Dom0 can access to hardware resources directly and DomUs are not allowed to access to hardware resources directly. DomUs can access hardware resources through Dom0.

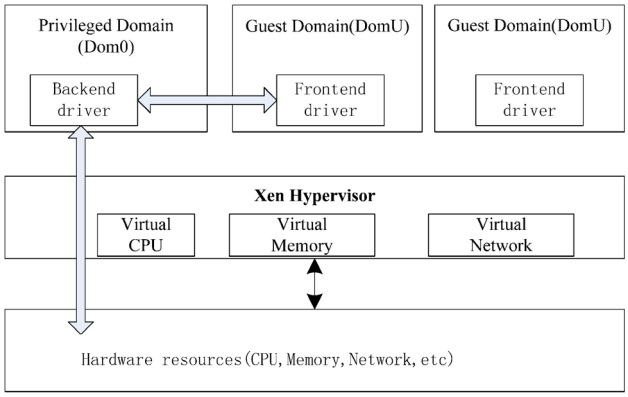

Fig. 1. Xen I/O architecture

In the xen virtualization environment, Guest OS cannot directly schedule physical CPU. Xen establishes the virtual CPU (VCPU) structure and provides one or more VCPUs for every Guest OS. All VCPUs are time-division multiplexing physical CPU. So, xen need reasonable allocation of time slices for VCPUs and schedule them.

Xen provides multiple schedulers, such as borrowed virtual time (BVT)、simple earliest deadline first (SEDF) and credit scheduler. The credit scheduler is xen's default scheduler at present. Its overall objective is to allocate the processor resources fairly. The Guest OS is assigned a weight value when it is created. Each Guest OS is allocated a certain number of credits according to its weight every 30 milliseconds. The credits will be allocated to VCPUs of the Guest OS fairly. As a VCPU runs, it consumes credits. According to the credits of VCPU, a VCPU's priority can be one of the three values: OVER, UNDER and BOOST. If VCPUs are in the OVER state, then they have used up its fair share of

CPU resources. If VCPUs are in the UNDER state, then they have CPU resources that can be consumed. The BOOST state provides a mechanism for domains to achieve low I/O response latency. All VCPUs in BOOST state are placed in front of those in UNDER state in the run queue, while those in OVER state are kept in the tail of the queue. Every physical CPU has a run queue of VCPUs. The queue is sorted by the priority of VCPUs and the head of the queue is always selected to run.

2.2 Related Work

Virtualization disrupts the basis of spinlock synchronization. In virtualization environment, scheduler is completely unaware of the association between VCPUs. Scheduler may schedule VCPU executing lock-waiter thread before lock-holder VCPU or it may pre-empt lock-holder VCPU. This violates the most basic primitive of Operating System and causes problems like VCPU stacking. Co-scheduling [3] is a representative scheme of coordinated scheduling that allows cooperative threads to be synchronously scheduled and de-scheduled. It can remove VCPU stacking problem, but suffer from the serious drawbacks like CPU fragmentation and priority inversion. Usman Rafi et al. [4] propose a dynamic scheduler which VCPUs of a VM are co-scheduled only if concurrency degree of VM is greater than threshold. Hybrid co-scheduling in papers [10-12] are all VCPUs of a VM are scheduled at the same time when they satisfy certain conditions. Kim et al. [5] present a demand-based coordinated scheduling scheme for consolidated SMP VMs that host multithreaded workloads, which diagnose VCPU synchronization through inter processor interrupts (IPIs) between VCPUs. It can effectively reduce synchronization latency and unnecessary CPU consumption. papers in [5-6, 13] are all based on spinlock detection mechanism. However, existing spin lock detection mechanism is not accurate enough. This paper solve the problem of CPU fragmentation and priority inversion. Meanwhile, it need not depend on the spin lock detection mechanism.

3 Theoretical Basis

This section describes the theoretical basis of system model of our VCPU collaborative scheduling method based on virtual domain.

3.1 VCPU

We suppose there are N VMs running on the same physical machine. VM_i is the $i-th$ (i=1, 2, ... N) VM and N_i is the number of VCPUs of the VM_i. VM_{ij} is the $j-th$ VCPU of the VM_i. $C_1, C_2, ...C_m$ are CPUs of the physical machine (m is the number of CPUs of the physical machine).

Definition 3.1. A VCPU is a three-tuple VCPU=<vid, $vcid$, pri>, where:

- vid is the number of the VM.
- $vcid$ is the number of the VCPU.
- pri is priority of the VCPU.

Definition 3.2. A priority of VCPU can be one of the following:

- *BOOST:* if the VCPU wakes up from a blocked event, its status will be *BOOST*.
- UnBOOST: if the status of a VCPU is schedulable, its status will be UnBOOST.

The priority of VCPUs which status are BOOST are higher than those status are UnBOOST. When a VM is created, VCPUs of the VM are allocated to the CPUs of physical machine (PCPU). VDS allocates PCPU for VCPUs of the VM according to the following definition 3.3.

Definition 3.3. Suppose the $vcid$ of a VCPU is $vcid_i$, cid_j is the number of CPU which is assigned, we have $vcid_i = cid_j$.

For the simplicity of discussion, we introduce a NULL VCPU which is denoted by NULL. At initialization phase of VCPUs, when the number of VCPUs of the VM is less than the number of CPUs, VCPUs which cannot be allocated a CPU will be allocated a NULL VCPU.

3.2 Virtual Domain

A virtual domain is the virtual set of all VCPUs of a physical machine. The VDS scheduling method combines co-scheduling and virtual domain. In each scheduling period, VCPUs of a VM is scheduled. When there exist idle CPUs, VDS will schedule VCPU from virtual domain.

Example 3.1. Suppose VM_1, VM_2 and VM_3 are running on the same physical machine M which has 4 CPUs. The number of VCPU of VM_1, VM_2 and VM_3 are 4, 3 and 2 respectively. Virtual domain (VD) can be expressed as VD ={VC00, VC01, VC02, VC03, VC10, VC11, VC12, VC20, VC21}. Table 1 gives VCPU scheduling results of some scheduling period.

Table 1. VCPU scheduling results of some scheduling period

	C_1	C_2	C_3	C_4
K_0	VC_{00}	VC_{01}	VC_{02}	VC_{03}
K_1	VC_{10}	VC_{11}	VC_{12}	VC_{00}
K_2	VC_{20}	VC_{21}	VC_{01}	VC_{02}

In scheduling period k_0, all VCPUs of VM_1 are scheduled and there is no idle CPU. In scheduling period k_1, VCPUs of VM_2 are scheduled to C_1, C_2, C_3, and C_4 is idle. In order to avoid waste CPU resources, VC_{00} is scheduled to C_4. Similarly, in scheduling period k_3, VC_{01} and VC_{02} are scheduled to idle CPUs. Please here note that VCPUs of virtual domain are actually VCPUs of VMs.

Example 3.2. Suppose VM_1 and VM_2 are running on the same physical machine M which has 4 CPUs. VC_{00}, VC_{01}, VC_{02} and VC_{03} are VCPUs of VM_1. VC_{10}, VC_{11} and VC_{12} are VCPUs of VM_2. The number of VCPUs of VM_2 is same with the number of CPUs of M. There is no idle CPUs when VCPUs of VM_1 are scheduled. When VCPUs of VM_2 are scheduled, idle CPUs can only be filled by VCPUs of VM_1. There is no idle CPUs can be filled by VM_2.

Definition 3.4. *A* VM cannot be filled when there is no idle CPUs. A VM can be filled when there is idle CPUs can be filled.

Now, we give the definition of VD.

Definition 3.5. Suppose the number of VCPUs of VM_i is N_i. VCPU set of VM_i can be defined as $VS_i = \{VC_{i0}, VC_{i1}, ...VC_{i(N_i-1)}\}$.

Definition 3.6. A *VD* is a four-tuple *VD*=< s, qb, qv, op >, where:

s is the status of *VD*. *VD* cannot be filled if there is a VM cannot be filled. VD can be filled if all VMs can be filled.

- *qb* is the queue which is consist of VCPUs whose status is *BOOST*.
- *qv* is the queue which is consist of VCPUs whose status is *UnBOOST*.
- *op* is the operation set of *VD*. *VD*::= search| EnQueue| DeQueue| AddAll where search< *vid* , *vcid* > represents that search VCPU whose number is < *vid* , *vcid* > in queue qb or qv. EnQueue (VC_k, Q) represents that put VC_k into *Q*. DeQueue (k, Q) represents that remove k-th elements from *Q*. AddAll (qv) represents that put all VCPUs of the physical machine into *Q*.

In order to let all VCPUs of the physical machine fairly share idle CPUs, VDS put all VCPUs into qv. Once there are idle CPUs, VCPUs of qv will be selected in sequence to fill idle CPUs. When there is no VCPU can be filled, the operation AddAll (qv) will be executed.

4 VCPU Collaborative Scheduling Method Based on Virtual Domain

In the last section, we introduce how to avoid CPU fragmentation through some examples. In this section, we will give how to avoid priority inversion and give the algorithm of our VCPU collaborative scheduling method based on virtual domain (VDS).

The problem of priority inversion is avoided by improving the priority of VCPUs of VD. When a VCPU wakes up from a blocked event, the state of VCPU of the VM is not set to BOOST. The state of VCPU of VD which has the same number is set to BOOST. At the same time, the VCPU will be removed

from the queue qv and put into the queue qb.

Suppose the length of qb is N_b and the number of idle CPU cycles is N_{idle}. VDS scheduling can be divided into three conditions:

(1) If N_b=0, there is no VCPU whose state is BOOST. If there exist idle CPUs, VCPUs of qv will be used to fill the idle CPUs.

(2) If $0 < N_b < N_{idle}$, idle CPU cycles can meet the needs of VPUs whose state is BOOST. Idle CPU cycles are first filled by VCPUs in queue qb. If the number of VCPU is same with the number of VCPUs of VM which is scheduled, the idle CPU cycle will be set to NULL again. If there exist idle CPU cycles When they are filled by qb, idle CPU cycles can be filled by qv.

(3) If $N_b > N_{idle}$, idle CPU cycles cannot meet the needs of VPUs whose state is BOOST. In the next scheduling period, VCPUs will be first selected from the queue qb. If the length of qb more than the number of CPUs which is denoted by m, all VCPUs will be selected from the queue qb in the next scheduling period. If the length of qb less than the number of CPUs, in the next scheduling period, VDS first select all VCPUs of qb and idle CPU cycles will be selected from qv.

The algorithm of our VCPU collaborative scheduling method based on virtual domain is formalized in Algorithm 1.

Algorithm 1. The algorithm of VCPU collaborative scheduling method based on virtual domain

```
1.  IF Nb!=0 and Nb<m THEN
2.  select all VCPUs of one VM
3.    FOR k=0, k<qb.length; k++
4.      VC=DeQueue (k, Q)
5.      IF VC already exists in VCPUs which will be scheduled THEN
6.        VC=NULL
7.      END IF
8.    END FOR
9.  ELSE IF Nb>m THEN
10.   IF qb.length<=m THEN
11.     select all VCPUs of qb
12.  ELSE
13.    select the first m VCPUs into the scheduling queue
14.  END IF
15. END IF
16. IF there is idle CPU THEN
17. select VCPU from qv
18. END IF
```

5 Performance Evaluation

We have implemented the algorithm of our VCPU collaborative scheduling method based on virtual domain in the xen 4.1.2 hypervisor. In this section, we compare the performance of the VCPU collaborative scheduling algorithm based on virtual domain (VDS_xen), Credit scheduling algorithm and Co_scheduling.

5.1 Experiments Setup

Our system is installed on the physical machine equipped with two Inter(R) Xeon(R) 4-core CPU running at 2.40GHz, 32G of RAM. The operation system is running the 64-bit version of Ubuntu 12.04. The system is configured with multiple VMs, and each VM is configured with 2G memory. The operation system of VMs is also running the 64-bit version of Ubuntu 12.04. The application of VMs is matrix multiplication C=AB which is implemented by BSPCloud [14].

5.2 Performance Indices

In order to improve the accuracy of experiments results, each experiment will be performed several times. Let N denotes the number of experiments (N is set to 50 in this paper). Test values are denoted by X_1,

X_2,...X_N. The results are measured by two performance indices. One is the mean value of test values which can be expressed by the following equation.

$$E(x) = \frac{X_1 + X_2 + ... + X_N}{N}. \quad \textbf{(1)}$$

The other is the standard deviation of test values which can be expressed by the following equation.

$$\sigma(x) = \sqrt{E(x - E(x))^2}. \quad \textbf{(2)}$$

Standard deviation can reflect the discrete degree of test values. The less the value, the better the stability of applications.

5.3 Experiment I

In this set of experiments, we evaluate the effect of VDS scheduling on the performance of application. We create 4 VMs on the physical machine which is introduced in the above subsection 5.1. The number of VCPUs is 4, 6, 6 and 6 respectively. Matrix multiplication C=AB which is implemented by BSPCloud run on the VM which has 4 VCPUs. The size of A and B are 2000×2000, 3000×3000 and 4000×4000 respectively. Six threads are worked on the rest 3 VMs, and each thread run a compute-intensive application. For simplicity, the application of each thread run a multiplication of variable a and b in a loop function block.

Fig. 2 shows the average completion time of application under different data size. Compared with xen credit scheduling, Co_scheduling and VDS scheduling can both improve the performance of application. The average completion time of application of Co_scheduling declines 17.3%, 17.41% and 17.38% under data size 2000×2000, 3000×3000 and 4000×4000. The average completion time of application of VDS declines 12.69%, 12.78% and 12.72% under data size 2000×2000, 3000×3000 and 4000×4000. The results show that both VDS scheduling and Co_scheduling can improve the performance of application. We can also see that the performance of application under VDS scheduling is better than the application under Co_scheduling. This is because that VDS can avoid CPU fragment.

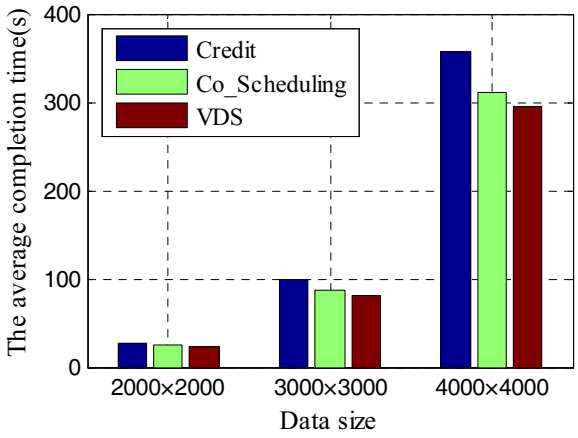

Fig. 2. The average completion time of application under different data size

Fig. 3 shows standard deviation of application under credit scheduling, Co_scheduling and VDS scheduling. The standard deviation under credit scheduling is relatively large because of the uncertainty of VCPUs' scheduling order of VCPUs. Compared with xen credit scheduling, Co_scheduling and VDS scheduling can both decline the standard deviation of application. The standard deviation of Co_scheduling declines 84.49%, 74.84% and 74.89% under data size 2000×2000, 3000×3000 and 4000×4000. The standard deviation of VDS declines 90.28%, 86.91% and 90.05% under data size 2000×2000, 3000×3000 and 4000×4000.

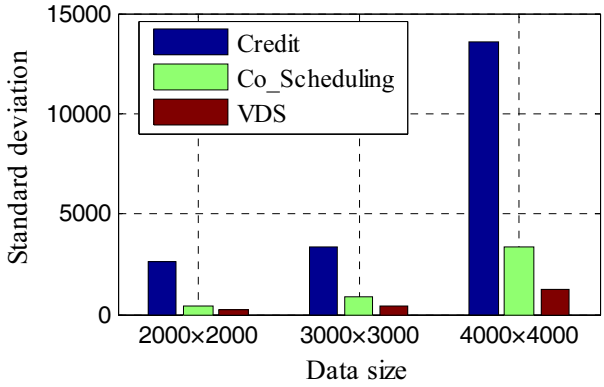

Fig. 3. The standard deviation of application under different data size

Because Co_scheduling can only guarantee either all VCPUs of the VM are scheduled at the same time or none of the VCPU is scheduled. The allocation of VCPUS on the physical CPU is not considered. The migration of VCPUs may influence the stability of the application. VDS scheduling establishes the stable mapping relationship between VCPUs and physical CPUs. The influence of VCPUs migration is eliminated. The standard deviation of application is further reduced.

5.4 Experiment II

In this set of experiments, we consider the effect of VCPUs number on the performance of application. We create 4 VMs on the physical machine which is introduced in the subsection 5.1. The number of VCPUs of three VMs is 4, and each VM run a 4 threads application. Each thread run a compute-intensive application. For simplicity, the application of each thread run a multiplication of variable a and b in a loop function block. The number of VCPUs of one VM changes from 3 to 8. Matrix multiplication C=AB which is implemented by BSPCloud run on the VM. The size of A and B are both 3000×3000.

Fig. 4 shows the average completion time of application under different number of VCPUs. Compared with xen credit scheduling, Co_scheduling and VDS scheduling can both improve the performance of application when the number of VCPUs changes from 3 to 8. The average completion time of application of Co_scheduling declines 13.43%, 15.55%, 14.76%, 16.24%, 16.35% and 16.14% respectively. The average completion time of application of VDS declines 11.72%, 14.84%, 10.73%, 12.89%, 13.06%, and 13.11% respectively. The performance of application under VDS scheduling is better than the application of application under Co_scheduling. This is because that VDS can avoid CPU fragment. We can see that the performance of application is related to the number of VCPUs. For example, when the total number of VCPUs is 16 which is twice as many as the total number of CPUs. The CPU fragment is not generated. Thus, the performance improves better than others. We can also see that the performance of application is not always increase with the increase of the number of VCPUs. The performance of application begins to declines when the total number of VCPUs increases to some extent. This is because the overhead of scheduling is increases and the allocated CPU resources of each VCPU is decreases.

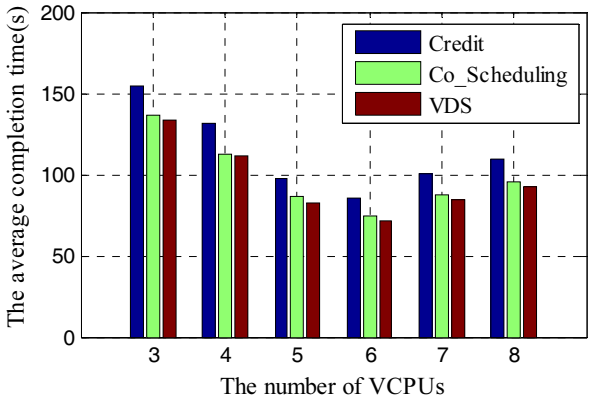

Fig. 4. The average completion of application under different number of VCPUs

Fig. 3 shows standard deviation of application under different number of VCPUs. Compared with xen credit scheduling, Co_scheduling and VDS scheduling can both reduce the standard deviation of application when the number of VCPUs changes from 3 to 8. The standard deviation of application of Co_scheduling declines 91.92%, 90.19%, 90%, 90.36%, 91.9% and 92.53% respectively. The standard deviation of application of VDS scheduling declines 80.75%, 76.28%, 81.99%, 74.53%, 78.62% and 80.88% respectively. The reason is same as which has been explained in the subsection 5.3.

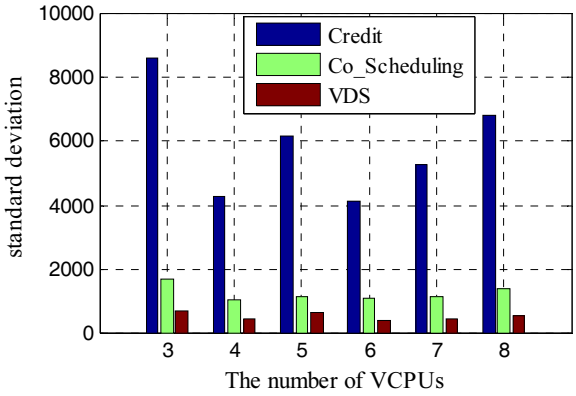

Fig. 5. The standard deviation of application under different number of VCPUs

5.5 Experiment III

In this set of experiments, the number of VCPUs of the VM which run BSPCloud application is fixed. We evaluate the performance of application when the number of VCPUs of the physical machine is changes. We create 3 VMs on the physical machine which is introduced in the subsection 5.1. The number of VCPUs of one VM is 4, and it run the application of matrix multiplication C=AB which is implemented by BSPCloud. The size of A and B are both 3000×3000. The number of VCPUs of the other VMs (auxiliary VCPUs) changes from 3 to 8, and the number of threads of the VM is also corresponding changes from 3 to 8. Each thread run a multiplication of variable a and b in a loop function block.

As we can see from Fig. 6, compared with xen credit scheduling, Co_scheduling and VDS scheduling can both improve the performance of application when the number of auxiliary VCPUs changes from 3 to 8. The average completion time of application of Co_scheduling declines 16.17%, 15.77%, 16.06%, 14.83%, 14.16% and 13.59%. The average completion time of application of VDS scheduling declines 12.56%, 12.49%, 13%, 12.75%, 12.78% and 12.8%. The results show that both VDS scheduling and Co_scheduling can improve the performance of application. Because the VDS scheduling can effectively avoid CPU fragment, the performance of application under VDS scheduling is better than the application under Co_scheduling. Similarly as the situation in subsection 5.3. The performance of application is also related to the number of VCPUs of the physical machine. The reason is same as which has been explained in the subsection 5.3.

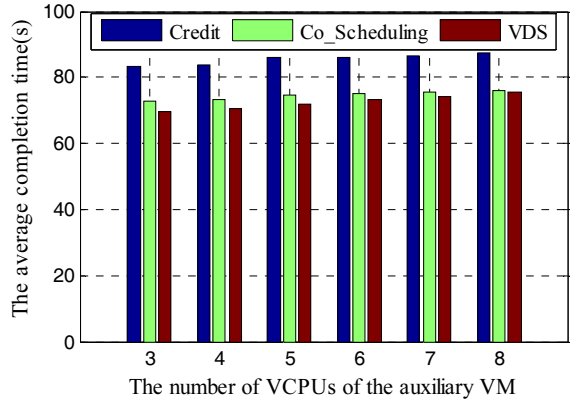

Fig. 6. The average completion of application under different number of auxiliary VCPUs

Fig. 7 shows standard deviation of application under different number of auxiliary VCPUs. Compared with xen credit scheduling, Co_scheduling and VDS scheduling can both reeuce the standard deviation of application when the number of auxiliary VCPUs changes from 3 to 8. The standard deviation of application of Co_scheduling declines 86.92%, 88.61%, 89.07%, 91.34%, 90.19% and 86.13% respectively. The standard deviation of application of VDS scheduling declines 78.04%, 77.68%, 75.74%, 78.76%, 78.85% and 76.32% respectively. The reason is same as which has been explained in the subsection 5.3.

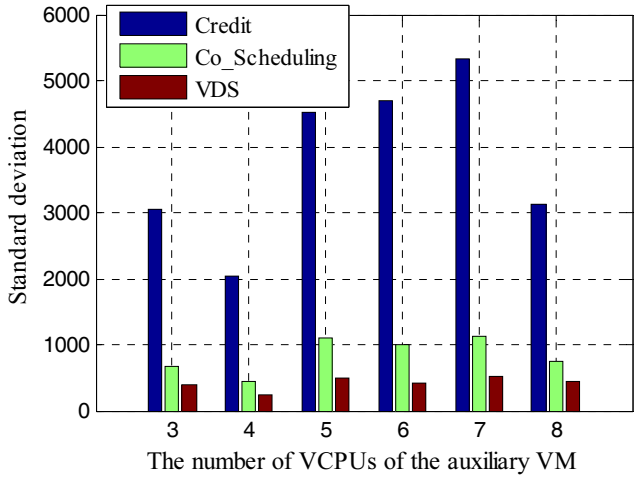

Fig. 7. The standard deviation of application under different number of auxiliary VCPUs

6 Conclusion

In this paper, we have proposed a VCPU collaborative scheduling method based on virtual domain (VDS). In this method, either all VCPUs of the VM are scheduled at the same time or none of the VCPU is scheduled. The mapping relationship between VCPUs and physical CPUs is also established. The CPU fragmentation is filled by VCPUs of virtual domain. The problem of CPU fragmentation and priority inversion can be avoided. The proposed method is implemented on the VMM. We evaluate the effect of VDS scheduling on the performance of multi-thread application. Compared with the existing scheduling algorithm, the performance has increased by 5%. Meanwhile, stability has also increased by 5%. In the future, we will further optimize our performance model.

Acknowledgements

This work is supported by scientific and technological research project of Henan provincial science and technology department (182102310919), Foundation of He'nan Educational Committee (16A520041).

References

[1] S. Jin, J. Ahn, S. Cha, J. Huh, Architectural support for secure virtualization under a vulnerable hypervisor, in: Proc. 2011 International Symposium on Microarchitecture, 2011.

[2] B.D.P. Barham, S.H.K. Fraser, A.H.T. Harris, R. Neugebauer, I. Pratt, Xen and the art of virtualization, ACM SIGOPS Operating Systems Review 37(8)(2003) 164-177.

[3] C. Ahn, C.A.C. Guzman, B. Egger, POSTER: NUMA-aware power management for chip multiprocessors, in: Proc. 2017 International Conference on Parallel Architectures and Compilation Techniques, 2017.

[4] U. Rafi, M.A. Zia, A. Razzaq, S. Ali, M.A. Saleem, Multi-queue priority based algorithm for CPU process scheduling, in: Proc. 2017 International Conference on Management Science and Engineering Management, 2017.

[5] H. Kim, S. Kim, J. Jeong, J. Lee, S. Maeng, Demand-based coordinated scheduling for SMP VMs, in: Proc. 2013 International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.

[6] Y. Ye, R. West, J. Zhang, Z. Cheng, MARACAS: a real-time multicore VCPU scheduling framework, in: Proc. 2017 IEEE Real-Time Systems Symposium, 2017.

[7] L. Zhang, Y. Chen, Y. Dong, C. Liu, Lock-visor: an efficient transitory co-scheduling for MP guest, in: Proc. 2012 International Conference on Parallel Processing, 2012.

[8] B. Wang, Y. Cheng, W. Chen, Q. He, Y. Xiang, Efficient consolidation-aware VCPU scheduling on multicore virtualization platform, Future Generation Computer Systems 56(3)(2016) 229-237.

[9] S. Kashyap, C. Min, T. Kim, Opportunistic spinlocks: achieving virtual machine scalability in the clouds, Acm Sigops Operating Systems Review 50(1)(2016) 9-16.

[10] Y. Yua, Y. Wang, H. Guo, X. He, Optimisation schemes to improve hybrid co-scheduling for concurrent virtual machines, International Journal of Parallel, Emergent and Distributed Systems 28(1)(2013) 46-66.

[11] C. Weng, Z. Wang, M. Li, X. Lu, The hybrid scheduling framework for virtual machine systems, in: Proc. 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2009.

[12] Y. Yu, Y. Wang, H. Guo, X. He, Hybrid co-scheduling optimizations for concurrent applications in virtualized environments, in: Proc. 2011 International Conference on Networking, Architecture and Storage, 2011.

[13] H. Kim, S. Kim, J. Jeong, J. Lee, S. Maeng, Demand-based coordinated scheduling for SMP VMs, in: Proc. 2013 International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.

[14] X. Liu, W. Tong, Z. Fu, W. Liao, BSPCloud: a hybrid distributed-memory and shared-memory programming model, International Journal of Grid and Distributed Computing 6(1)(2013) 87-97.