A Real-Time Scheduling Framework Based on Multi-core Dynamic Partitioning in Virtualized Environment

Song Wu, Like Zhou, Danqing Fu, Hai Jin, Xuanhua Shi

A Real-time Scheduling Framework Based on Multi-core Dynamic Partitioning in Virtualized Environment

Song Wu, Like Zhou, Danqing Fu, Hai Jin, Xuanhua Shi

Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
{wusong, zhoulike, fdq1989, hjin, xhshi}@hust.edu.cn

Abstract. With the prevalence of virtualization and cloud computing, many real-time applications are running in virtualized cloud environments. However, their performance cannot be guaranteed because current hypervisors' CPU schedulers aim to share CPU resources fairly and improve system throughput. They do not consider real-time constraints of these applications, which result in frequent deadline misses. In this paper, we present a real-time scheduling framework in virtualized environment. In the framework, we propose a mechanism called *multi-core dynamic partitioning* to divide *physical CPUs* (PCPUs) into two pools dynamically according to the scheduling parameters of *real-time virtual machines* (RT-VMs). We apply different schedulers to these pools to schedule RT-VMs and non-RT-VMs respectively. Besides, we design a *global earliest deadline first* (*vGEDF*) scheduler to schedule RT-VMs. We implement a prototype in the Xen hypervisor and conduct experiments to verify its effectiveness.

Keywords: Virtualization, Real-time scheduling, Multi-core, Cloud computing.

1 Introduction

Cloud computing is a rapidly emerging paradigm that cloud resources in data centers are leased by users on demand. Cloud data centers, such as Amazon's *Elastic Compute Cloud* (EC2) [1], use virtualization technology to provide such on-demand infrastructure services. In cloud data centers, a *physical machine* (PM) always hosts many *virtual machines* (VMs), and various kinds of applications are running in these VMs. Many of them have real-time constraints, such as streaming server, VoIP server, and real-time stream computing platforms.

Although more and more real-time applications run in virtualized cloud environments, their performance is hardly guaranteed [11][13][18]. The main reason is that virtualization adds an additional layer, called hypervisor such as Xen [8], between *guest operating systems* (guest OSes) and underlying hardware. CPU schedulers in hypervisors are not optimized for real-time applications, such as Xen's default Credit scheduler [9].

Previous studies [13][16][18] present some solutions to support real-time applications in virtualized environments. However, they are not good enough for these applications and cloud environments. RT-Xen [16] does not support VMs with multiple *virtual CPUs* (VCPUs). Schedulability analysis is important in real-time scheduling, but these studies [13][18] do not analyze their schedulability. More importantly, all these solutions favor the RT-VMs running real-time applications, which may affect the performance of non-real-time applications and violate the performance isolation guaranteed by cloud platforms.

Aiming at these problems, this paper presents a real-time scheduling framework based on *multi-core dynamic partitioning*. First, it divides PCPUs into two pools dynamically by taking *non-uniform memory access* (NUMA) architecture into account according to the scheduling parameters of RT-VMs. It allows RT-VMs to run on a pool and non-RT-VMs to run on the other pool, which brings good performance isolation. Second, we design a *global earliest deadline first* (*vGEDF*) scheduler to schedule RT-VMs. Moreover, we implement a working prototype of the **r**eal-t**i**me **s**cheduling fr**a**mework in the Xen hypervisor, named *Risa*, and evaluate its effectiveness through experiments.

In summary, the main contributions of this paper are as follows.

- We present a real-time scheduling framework to support real-time applications in virtualized environment. The framework provides good performance isolation through multi-core dynamic partitioning.
- Considering the domination of multi-core processors in server market, we present the vGEDF scheduler to schedule RT-VMs, which can support real-time applications well and take full advantage of multi-core processors.
- We implement a prototype in the Xen hypervisor, and conduct experiments to verify its effectiveness. The experimental results show that our framework can support real-time applications well, reduce operation expense caused by manual operations in VM management, and improve CPU utilization.

The rest of this paper is organized as follows. Section 2 presents the design of the real-time scheduling framework in detail. We explain the experimental environment and show the experimental results in Section 3. Section 4 briefly surveys the related work. Finally, Section 5 concludes this paper.

2 Design of Real-Time Scheduling Framework

In this section, we present the design of our real-time scheduling framework, which is shown in Fig. 1. In the framework, PCPUs are partitioned into two pools (i.e. *rt-pool* and *non-rt-pool*) automatically according to the scheduling parameters of RT-VMs. We apply our vGEDF scheduler to *rt-pool* to schedule RT-VMs and the Credit scheduler to *non-rt-pool* to schedule non-RT-VMs. In the following, we first describe how to partition PCPUs automatically. Then, we propose the design of the vGEDF scheduler.

2.1 Multi-core Dynamic Partitioning Mechanism

In the multi-tenant cloud environment, a PM hosts many VMs that run various kinds of applications from different customers. However, a single CPU sched-

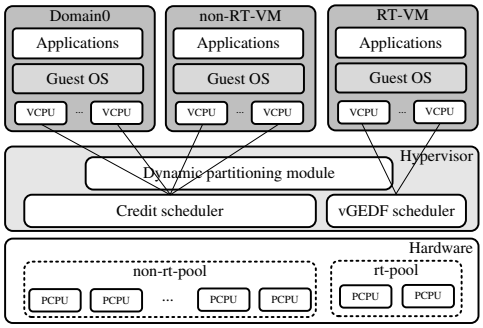


Fig. 1. The real-time scheduling framework.

uler cannot support all the applications well. For example, although the Credit scheduler supports CPU-intensive and memory-intensive applications well, it is not suitable for real-time applications. Accordingly, the schedulers optimized for real-time applications [12][13][16][18] always favor these applications, which may affect the performance of non-real-time applications. Moreover, an important requirement of multi-tenant cloud environment is performance isolation. As a result, it is a challenge to support real-time applications while minimizing the impact on non-real-time applications running on the same PM. In this paper, we present the multi-core dynamic partitioning mechanism to meet this goal.

Currently, although administrators can divide PCPUs into multiple pools and apply different schedulers to these pools manually, this method is not fit for cloud environment. The reasons are as follows. On one hand, administrators need to estimate the requirements of RT-VMs, and statically allocate peak number of PCPUs to a pool, which probably results in resource over-provision and increases operation expense. On the other hand, when the requirements of RT-VMs change, administrators need to manually change the number of PCPUs allocated to the pool. Otherwise, the performance of real-time applications may not be guaranteed any more. Our multi-core dynamic partitioning mechanism addresses these drawbacks well.

If a PM has RT-VMs, the real-time scheduling framework partitions PCPUs into two pools automatically and applies different schedulers to these pools to schedule RT-VMs and non-RT-VMs respectively. So, first of all, we need to determine how many PCPUs should be allocated to *rt-pool*. Then, we allocate corresponding PCPUs to *rt-pool* by taking NUMA architecture into account.

How many PCPUs should be allocated to rt-pool For the convenience of description, we define some variables as follows:

- C_i: the worst-case execution time of the ith task.
- T_i: the inter-arrival period of the ith task (assumed to be equal to the relative deadline).
- N_P: the number of PCPUs should be allocated to *rt-pool*.
- N_{RT}: the number of RT-VMs in a PM.
- NV_i: the number of VCPUs of the ith RT-VM.

- p_i: the *period* parameter of the ith RT-VM, which indicates the relative deadline.
- s_i: the *slice* parameter of the ith RT-VM, which represents the worst-case execution time.

According to the schedulability test of EDF scheduling [14], a set of real-time tasks is schedulable only if its total utilization does not exceed 100%.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1}$$

In virtualized environments, in order to guarantee the schedulability of RT-VMs in multi-core platforms, the scheduling parameters of RT-VMs and the number of PCPUs must satisfy the following equation:

$$\sum_{i=1}^{N_{RT}} \frac{s_i \times NV_i}{p_i} \leq N_P \tag{2}$$

Hence, derived from (2), N_P can be calculated by (3). Actually, N_P is the minimal number of PCPUs should be allocated to *rt-pool*. It has a strong relationship with the scheduler applied to *rt-pool*. Equation (3) defines how to calculate N_P for the vGEDF scheduler. Our real-time scheduling framework can be easily extended to support other real-time schedulers. The only thing needed to be done is to define how to calculate N_P.

$$N_P = \left\lceil \sum_{i=1}^{N_{RT}} \frac{s_i \times NV_i}{p_i} \right\rceil \tag{3}$$

How to partition PCPUs Cloud is a highly dynamic environment. Various operations are happened in a short time period, such as VM creation, VM destroy, and VM reconfiguration. As a result, N_P is changing as time goes on. Considering such dynamic characteristic, we design a multi-core dynamic partitioning algorithm to support cloud environment. All the operations that may change N_P trigger the algorithm to allocate adequate number of PCPUs to *rt-pool*. The pseudo-code of the algorithm is shown in Algorithm 1.

Algorithm 1: Multi-core Dynamic Partitioning Algorithm

```
1  prev_N_P ← num_pcpus(rt_pool);
2  foreach vm in the list of RT-VMs do
3  |  new_num ← new_num + (vm.nvcpus * vm.slice)/vm.period;
4  end
5  N_P ← ceil(new_num);
6  delete timer;
7  if N_P > prev_N_P then
8  |  partition(non_rt_pool, rt_pool, N_P − prev_N_P);
9  else if N_P < prev_N_P then
10 |  set timer to call partition(rt_pool, non_rt_pool, prev_N_P − N_P);
11 end
```

First, the algorithm reads the scheduling parameters of RT-VMs and calculates N_P (line 2∼5). Then, it compares N_P with the previous one. If N_P is

greater than the previous one, the algorithm allocates more PCPUs to *rt-pool* immediately. On the contrary, if it is less than the previous N_P, the algorithm shrinks *rt-pool*. However, the shrink operation is not executed instantly, because it may cause fluctuation. For example, administrators may destroy a RT-VM belonging to a customer and create the other RT-VM for the other customer immediately. If the algorithm shrinks *rt-pool* immediately, it needs to expand *rt-pool* after the shrink. In order to avoid such fluctuation, we adopt a delayed shrink manner, which uses a timer to delay the shrink operation (line 6∼11).

Nowadays, an increasing number of new multi-core systems use the NUMA architecture. There are multiple memory nodes in modern NUMA systems, and the access latency of local nodes is shorter than that of remote nodes. Aimed at such characteristic, our multi-core dynamic partitioning algorithm takes the NUMA architecture into account when we partition PCPUs. It preferably allocates PCPUs belonging to a NUMA node to *rt-pool* instead of randomly selected PCPUs. The pseudo-code of the algorithm is shown in Algorithm 2, which shrinks *src_pool* and expands *dst_pool*. If *rt-pool* is empty, the algorithm selects the PCPU on which a RT-VM currently running or previously run, and allocates this PCPU to *rt-pool* (line 2∼7). Then, it gets the NUMA topology of the PM and finds the local node associated with *dst_pool* (line 8). Finally, the algorithm preferably allocates PCPUs belonging to this node to *dst_pool*. If all the PCPUs belonging to this node are allocated to *dst_pool*, the algorithm picks PCPUs from other nodes and allocates them to *dst_pool* (line 9∼13).

Algorithm 2: NUMA-aware Partitioning Algorithm

```
1  prev_num ← num_pcpus(rt_pool);
2  if prev_num == 0 and dst_pool == rt_pool then
3  |  select pcpu on which a RT-VM currently running or previously run;
4  |  remove pcpu from src_pool;
5  |  add pcpu to dst_pool;
6  |  pcpu_num ← pcpu_num − 1;
7  end
8  local_node ← the local node associated with dst_pool;
9  while pcpu_num! = 0 do
10 |  remove pcpu from src_pool that belongs to local_node or other nodes if
   |  all PCPUs in local_node are allocated to dst_pool;
11 |  add pcpu to dst_pool;
12 |  pcpu_num ← pcpu_num − 1;
13 end
```

2.2 vGEDF Scheduler

Schedulability analysis is important in real-time scheduling. However, previous solutions [13][18] do not analyze their schedulability. In this paper, we design the vGEDF scheduler based on EDF scheduling algorithm, whose schedulability is analyzed by previous studies [7].

Nowadays, multi-core processors have dominated server markets. Schedulers must take full advantage of the multi-core processors. The *Simple Earliest Deadline First* (SEDF) scheduler [9] is not suitable for cloud environments because of the lack of load balance among multi-cores. On the contrary, our vGEDF scheduler supports real-time applications in multi-core platform well through global queues. Its architecture is shown in Fig. 2.

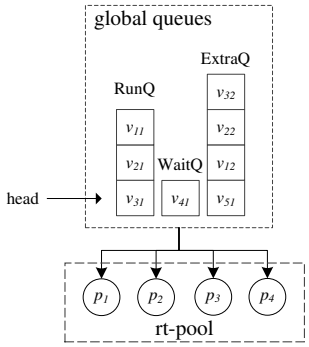


Fig. 2. Architecture of vGEDF.

In the real-time scheduling framework, the vGEDF scheduler is applied to *rt-pool*. All the PCPUs in *rt-pool* share three global queues: runnable queue (*RunQ*), waiting queue (*WaitQ*), and extra queue (*ExtraQ*). Each VM also has following scheduling parameters: p_i, s_i, and x_i. The meaning of p_i and s_i is described in Section 2.1. x_i is a boolean value to indicate whether a VM can get extra CPU time (i.e. work-conserving mode). The scheduler inserts VCPUs into these queues according to their scheduling parameters. If a VCPU has remaining CPU slice in current period, it is inserted into *RunQ*. Otherwise, it is inserted into *WaitQ* or *ExtraQ* according to the value of x_i of the VCPU. The priority of a VCPU is calculated according to their deadlines: the earlier the deadline, the higher the priority. VCPUs in *RunQ* are sorted by their priorities, and the VCPU in the head of *RunQ* has the highest priority. *ExtraQ* is used to support work-conserving mode.

Algorithm 3: vGEDF Scheduling Algorithm

```
1  handle the bookkeeping for current in RunQ or ExtraQ;
2  update_queues(RunQ, WaitQ);
3  snext ← CandidatePick(RunQ);
4  if snext == NULL then
5  |  snext ← CandidatePick(ExtraQ);
6  |  if snext == NULL then
7  |  |  return idle_vcpu[cpu];
8  |  end
9  end
10 ret.task ← snext.vcpu;
11 snext.picked ← 1;
12 return ret.task;
```

The pseudo-code of the vGEDF scheduling is shown in Algorithm 3. The scheduler first conducts bookkeeping for the current running VCPU and updates the parameters of VCPUs in *RunQ* and *WaitQ* (line 1∼2). Then, it picks a VCPU from *RunQ* or *ExtraQ* to run (line 3∼10). For the convenience of bookkeeping and updating queues, the picked VCPU is still in the queues. Therefore, when a VCPU is picked to run, we need to mark it as *picked* (line 11).

Because our vGEDF scheduler picks the VCPU from global queues to run, a VCPU may run on several PCPUs in a short time period, which may increase cache misses. We present some approaches to reduce cache misses. On one hand, the multi-core dynamic partitioning mechanism preferably allocates PCPUs belonging to a NUMA node to *rt-pool* and these PCPUs share the last-level cache. The vGEDF scheduler applied to *rt-pool* will not increase the last level cache misses if all the PCPUs in *rt-pool* belong to a NUMA node. On the other hand, in order to further mitigate the impact of cache misses, we present a cache-aware pick algorithm (shown in Algorithm 4), which takes cache affinity into account, to reduce L1 and L2 cache misses.

Algorithm 4: CandidatePick Algorithm

```
1  ret ← NULL;
2  foreach vcpu in queue do
3      if vcpu.cpu_mask&current_pcpu! = 0 then
4          if vcpu.processor! = current_pcpu&&((vcpu.picked ==
           1&&vcpu! = current)||vcpu is cache hot) then
5              continue;
6          end
7          ret ← vcpu;
8          break;
9      end
10 end
11 return ret;
```

Besides, although the vGEDF scheduler uses global queues to manage VCPUs, the scalability is not a problem for the scheduler. This is because schedulers that use global queues can also scale to a certain number of PCPUs. Moreover, only a part of applications running in cloud environment is real-time applications. As a result, our framework allocates a small amount of PCPUs to *rt-pool* and only these PCPUs share the global queues. Even a PM has many PCPUs and all these PCPUs should be allocated to *rt-pool*, the scalability problem can also be addressed by our framework through partitioning multiple real-time pools.

3 Performance Evaluation

We implement a working prototype of the proposed **r**eal-t**i**me **s**cheduling fr**a**mework in Xen-4.2.1, called *Risa*. In this section, we evaluate the effectiveness of *Risa* through several experiments. We first describe the experimental environment, and then present the experimental results.

3.1 Experimental Environment and Methodology

Our evaluations are conducted on a server which has two quad-core 2.4GHz Intel Xeon CPUs, 24GB memory, 1TB SCSI disk, and 1Gbps Ethernet card. We use Xen-4.2.1 as the hypervisor and CentOS 5.5 distribution with the Linux-2.6.32.40 kernel as the OS. The network I/O of a VM is handled via a software bridge in Domain0. Unless otherwise specified, the configurations of VMs running on the server are as follows: 1VCPU, 1GB memory and 8GB virtual disk. Our experiments are targeted at understanding the effect of each component of *Risa*.

How to evaluate the effect of multi-core dynamic partitioning mechanism As described in Section 2.1, a practical way to support different kinds of applications simultaneously and provide performance isolation in multi-tenant cloud environment is to partition PCPUs into multiple pools and to apply different schedulers to these pools. As a result, we conduct experiments under two multi-core partitioning mechanisms.

One is the multi-core dynamic partitioning mechanism of *Risa*, which can manage *rt-pool* automatically according to the scheduling parameters of RT-VMs. The other is the multi-core static partitioning mechanism. It uses *cpupools*, a new feature of Xen since Xen 4.2, to partition PCPUs into *rt-pool* and *non-rt-pool*, but in a static method. It allocates the peak number of PCPUs to *rt-pool* manually according to the estimation of the requirements of RT-VMs before the creation of them, and deletes *rt-pool* when all the RT-VMs are destroyed.

How to evaluate the effect of the vGEDF scheduler When we evaluate the vGEDF scheduler, we conduct experiments under four strategies to demonstrate the advantages of the vGEDF scheduler. We dedicate four PCPUs to Domain0 to handle communication and interrupts for other VMs, which isolates Domain0 to all other domains. Fourteen VMs (VM1∼VM14) are running on the other PCPUs. VM1 hosts testing real-time applications and the others are interfering VMs which run lookbusy [4]. The details of these strategies are as follows.

baseline is the default configuration in cloud environment that only the Credit scheduler is adopted to schedule VMs.

Risa is our framework. In this strategy, seven VMs (VM1∼VM7) are set as RT-VMs. The scheduling parameters of VM1 are set as (p_i=5ms, s_i=1ms). The others are set as (p_i=10ms, s_i=2ms). Therefore, *Risa* allocates two PCPUs to *rt-pool* according to (3) and applies the vGEDF scheduler to *rt-pool* automatically.

sp+SEDF uses the multi-core static partitioning mechanism to simulate an environment like *Risa*. It partitions PCPUs into two pools and allocates two PCPUs to *rt-pool* manually, and the SEDF scheduler is adopted to schedule RT-VMs in *rt-pool*. Because it does not support load balance among multiple PCPUs, the distribution of these RT-VMs is as follows: a PCPU hosts four RT-VMs (VM1∼VM4) and the other hosts three RT-VMs (VM5∼VM7).

sp+SEDF(overload) is similar with *sp+SEDF*, except that a PCPU is overloaded. Because the SEDF scheduler does not support load balancing among multiple PCPUs, it is possible that a PCPU is overloaded while the other has

slight load. This strategy is used to simulate such situation that a PCPU hosts six RT-VMs (VM1∼VM6) and only one RT-VM (VM7) runs on the other PCPU.

3.2 Effect of Multi-core Dynamic Partitioning Mechanism

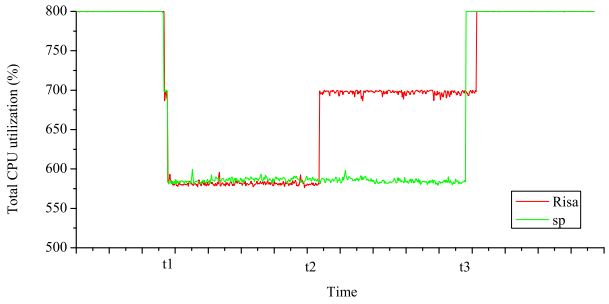


Fig. 3. Total CPU utilization of non-RT-VMs under different partitioning strategies. *Risa* uses dynamic partitioning, and *sp* means static partitioning.

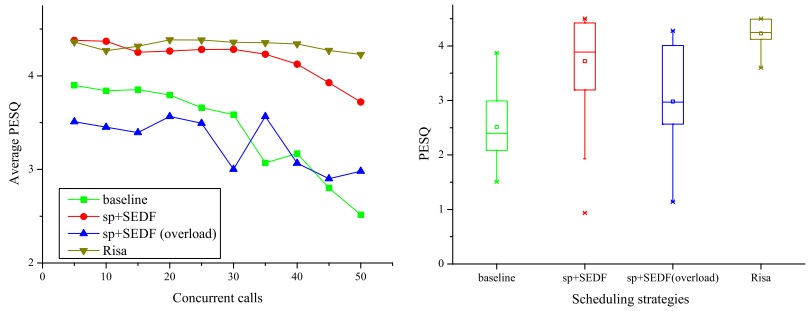
In this test, we evaluate the effect of the multi-core dynamic partitioning mechanism. We launch two non-RT-VMs with 8 VCPUs on the server and each runs eight hungry loop applications as non-real-time applications, which can exhaust the available CPU resources. We monitor the total CPU utilization of these VMs, which is the performance metric in this test. A shell script is running to create and destroy RT-VMs as time goes on, and the tasks of this script are as follows. 1) at time *t1*, it creates two RT-VMs and sets their scheduling parameters as (p_i=10ms, s_i=6ms); 2) at time *t2*, it changes s_i of a RT-VM to 2ms; 3) at time *t3*, it destroys these RT-VMs. In the multi-core static partitioning mechanism, the first thing needs to be done is to estimate the number of PCPUs which should be allocated to *rt-pool*. Then, two PCPUs are allocated to *rt-pool* before the creation of RT-VMs according to (3). Finally, *rt-pool* is destroyed at *t3* and the number of PCPUs of *non-rt-pool* is increased (it cannot be increased automatically when *rt-pool* is destroyed). Moreover, RT-VMs need to be assigned to *rt-pool* by administrators explicitly. In the multi-core dynamic partitioning, the only thing needs to be done is to run the shell script. The test results are shown in Fig. 3.

From the test results, we can observe that *Risa* automatically reduces the number of PCPUs of *rt-pool* at time *t2*. This is because the needed number of PCPUs of *rt-pool* turns to 1 according to (3) when the script adjusts the scheduling parameter of the RT-VM. Besides, because *Risa* adopts a delayed shrink manner, the increase of CPU utilization at *t3* under *Risa* is 15 seconds (implementation defined) later than *sp*. As a result, compared to the multi-core static partitioning mechanism, the multi-core dynamic partitioning mechanism of *Risa* can reduce operation expense and improve CPU utilization.

3.3 Effect of vGEDF Scheduler

In this test, we perform two experiments to evaluate the effectiveness of the vGEDF scheduler of *Risa*. They are conducted under different guest OSes. One

is *general purpose operating system* (GPOS). The other is *real-time operating system* (RTOS), which is designed to serve real-time application requests.



(a) Average PESQ of concurrent calls (b) Statistics of 50 concurrent calls

Fig. 4. Call quality under different strategies.

Experiments with VoIP server running in GPOS *Voice over Internet Protocol* (VoIP) server is a typical soft real-time application. Asterisk [2] is a famous and open source telephone private branch exchange. In this test, we use Asterisk to conduct experiments to evaluate the vGEDF scheduler of *Risa*.

We use VM1 to host Asterisk, and run SIPp [6] on a machine in the same LAN as a VoIP client. We start up several concurrent calls that range from 5 to 50 to simulate the real world environment, and measure call quality with the ITU-T PESQ (*Perceptual Evaluation of Speech Quality*) metric [15], which ranges from 0 to 4.5. Typically, if the value is greater than 4, it means that the VoIP service has good quality. The test results are shown in Fig. 4.

Seen from Fig. 4(a), *Risa* is the best among these scheduling strategies, and the call quality is guaranteed under *Risa*. This is because *Risa* is designed for real-time applications and takes full advantage of underlying multi-core processors. The Credit scheduler is a proportional fair share scheduler and does not consider real-time constraints. Thus, it even cannot guarantee the call quality with small concurrent calls. With the increase of concurrent calls, the SEDF scheduler cannot support the VoIP server any more. This is because the SEDF scheduler cannot make full use of multi-core processors. Besides, the call quality under the strategy of *sp+SEDF(overload)* is very low, which also shows the importance of load balancing among multiple PCPUs. Compared with the Credit scheduler, *Risa* achieves 68.1% improvement in call quality according to the average PESQ when we start up 50 concurrent calls. Accordingly, compared with the SEDF scheduler, *Risa* enhances the call quality by 13.7%.

Moreover, Fig. 4(b) shows the statistics of the call qualities of 50 concurrent calls under different strategies. We find that call quality is very steady under *Risa*, which is crucial for the VoIP server to provide stable services.

Experiments with Cyclictest running in RTOS Cyclictest [3] is a widely used real-time testing tool, which can evaluate kernel latencies of real-time Linux kernel. In this test, we use cyclictest to conduct experiments under a RTOS to demonstrate whether *Risa* supports the RTOS and hardware-assisted VMs (HVMs).

Table 1. Cyclictest test results under different strategies

Strategy	Min Latencies (us)	Avg Latencies (us)	Max Latencies (us)
Credit	5	5862	181559
sp+SEDF	0	3224	58634
Risa	0	2342	55700

The guest RTOS is CentOS 5.5 with Linux-2.6.32.40 kernel plus PREEMPT-RT patch [5], which is installed in a HVM. We replace VM1 in the four strategies with the HVM, and use cyclictest to evaluate the kernel latency of the RTOS by collecting data for 500,000 times. However, we observe that the RTOS is not responded under the strategy of *sp+SEDF(overload)* because of the features of the HVM. As a result, the experimental results only include three strategies, which are shown in Table 1. From the test results, we can find that the kernel latency is the smallest under *Risa*. Compared with the Credit scheduler and the SEDF scheduler, the kernel latency is reduced by 60% and 27.4% according to the average latencies, respectively. However, the reduction on maximum latencies is small compared to the SEDF scheduler. This is because both SEDF and vGEDF are based on the EDF scheduling algorithm.

4 Related Work

Hu *et al.* [10] present an I/O scheduling model of VM based on multi-core dynamic partitioning. They divide PCPUs into three subsets, and apply an identical scheduler with different strategies to these subsets. However, real-time scheduling is much more complex than I/O scheduling. Designing different schedulers for various subsets is more suitable for supporting real-time applications.

Lee *et al.* [13] introduce a concept named *laxity* to denote the scheduling latency that a VM desires. The VCPU of a VM running soft real-time applications is inserted into the middle of run queue according to its *laxity* so that it can be scheduled within its desired deadline. Kim *et al.* [12] present an approach to reallocate credits for the VMs running client-side multimedia applications adaptively according to their qualities. Our previous work [17][18] proposes a parallel soft real-time scheduling algorithm, which addresses real-time constraints and synchronization problems simultaneously, to support parallel soft real-time applications in virtualized environment. Hwang *et al.* [11] design a soft real-time scheduling to support virtual desktop infrastructures. However, all these studies lack the schedulability analysis, which is important for real-time scheduling. RT-Xen [16] presents a hierarchical real-time scheduling framework for Xen, but it only supports single core VMs.

5 Conclusion

In this paper, we present a real-time scheduling framework based on multi-core dynamic partitioning in virtualized environment. If the system has RT-VMs, PCPUs are partitioned into two pools (*rt-pool* and *non-rt-pool*) automatically according to the scheduling parameters of RT-VMs. *rt-pool* uses the vGEDF

scheduler, which takes full advantage of multi-core processors, to schedule RT-VMs. Non-RT-VMs are scheduled by the Credit scheduler in *non-rt-pool*. We implement a prototype in the Xen hypervisor and evaluate its effectiveness. The experiments results show that *Risa* supports real-time applications well, reduces operation expense, and improves CPU utilization.

Acknowledgments. The research is supported by National Science Foundation of China under grant No.61232008, National 863 Hi-Tech Research and Development Program under grant No.2013AA01A208, Doctoral Program of MOE under grant 20110142130005, EU FP7 MONICA Project under grant No.295222, and Chinese Universities Scientific Fund under grant No. 2013TS094.

References

1. Amazon's Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2/`
2. Asterisk. `http://www.asterisk.org/`
3. Cyclictest. `https://rt.wiki.kernel.org/index.php/Cyclictest`
4. Lookbusy - a synthetic load generator. `http://www.devin.com/lookbusy/`
5. Real-Time Linux Wiki. `https://rt.wiki.kernel.org`
6. SIPp. `http://sipp.sourceforge.net/`
7. Baker, T.P.: An analysis of edf schedulability on a multiprocessor. IEEE Trans. Parallel Distrib. Syst. **16**(8), 760–768 (2005)
8. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proc. SOSP'03, pp. 164–177 (2003)
9. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three cpu schedulers in Xen. SIGMETRICS Perform. Eval. Rev. **35**(2), 42 (2007)
10. Hu, Y., Long, X., Zhang, J., He, J., Xia, L.: I/O scheduling model of virtual machine based on multi-core dynamic partitioning. In: Proc. HPDC'10, pp. 142–154 (2010)
11. Hwang, J., Wood, T.: Adaptive dynamic priority scheduling for virtual desktop infrastructures. In: Proc. IWQoS'12 (2012)
12. Kim, H., Jeong, J., Hwang, J., Lee, J., Maeng, S.: Scheduler support for video-oriented multimedia on client-side virtualization. In: Proc. MMsys'12, pp. 65–76 (2012)
13. Lee, M., Krishnakumar, A.S., Krishnan, P., Singh, N., Yajnik, S.: Supporting soft real-time tasks in the Xen hypervisor. In: Proc. VEE'10, pp. 97–108 (2010)
14. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) **20**(1), 46–61 (1973)
15. Rix, A.W., Beerends, J.G., Hollier, M.P., Hekstra, A.P.: Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In: Proc. ICASSP'01, vol. 2, pp. 749–752 (2001)
16. Xi, S., Wilson, J., Lu, C., Gill, C.: RT-Xen: Towards real-time hypervisor scheduling in Xen. In: Proc. EMSOFT'11, pp. 39–48 (2011)
17. Zhou, L., Wu, S., Sun, H., Jin, H., Shi, X.: Supporting parallel soft real-time applications in virtualized environment. In: Proc. HPDC'13, pp. 117–118 (2013)
18. Zhou, L., Wu, S., Sun, H., Jin, H., Shi, X.: Virtual machine scheduling for parallel soft real-time applications. In: Proc. MASCOTS'13, pp. 525–534 (2013)