Research on Real-time Scheduling Algorithm of Partition Operating System in Single Core Environment

Yongchao Tao\textsuperscript{1,\ast}, Qilong Song\textsuperscript{1} and Songhao Piao\textsuperscript{1}

\textsuperscript{1} Shenzhen Academy of Aerospace Technology, Shenzhen, China

*Corresponding author e-mail: taoyongchao@chinasaat.com

Abstract. In an embedded environment, applications have a strong demand for real-time scheduling, but it is difficult to ensure the real-time scheduling performance under a partitioned operating system, mainly because in a multi-operating system environment. It is difficult to determine the real-time schedulability within and between partitions. The research of this paper will focus on the real-time scheduling algorithm of partition operating system with simple task model in single-core environment.

1. Introduction

Parallel operation of multiple systems is one of the characteristics of the partition system. Multiple dedicated partitions "simultaneously" run on the same physical machine, so the partition platform needs to allocate each CPU time slice in this time period to each partition in a fixed time period. The algorithm for allocating time slices is the partition operating system scheduling algorithm.

With the development of partition technology and the strengthening of application requirements, the problem of partition resource scheduling and performance evaluation has attracted widespread attention. In the partition system, the partition manager VMM controls the entire system resources [1]. At present, the most popular partition manager platforms include Virtual PC, KVM, Xen, etc. However, none of these platforms can support real-time systems well in virtual machine scheduling algorithms.

Literature [2] studied the Credit scheduling algorithm of the platform Xen adhering to the principle of fair and efficient sharing of CPU time slices for partition scheduling, and divided different partitions into three priority levels according to the Credit value: under, over, idle, allowing the partition to be in Three states: boost, running, block. The scheduling algorithm can ensure that each partition shares the CPU fairly, but it lacks the guarantee of the real-time performance of the real-time partition.

The author of literature [3] Robert Kaiser divides partitions into non-real-time partitions, time-driven real-time partitions, and event-driven real-time partitions, and discusses the requirements of these three types of partitions on the partition monitoring layer scheduling. It is proposed that a scheduling algorithm should be designed for each type of virtual machine, and then which scheduling algorithm should be selected according to a certain priority strategy to meet the needs of each type of partition.

In order to better support soft real-time applications, the article [4] studied partition scheduling and mainly proposed some target delay indicators to define the performance of soft real-time tasks. In addition, an S algorithm is proposed for SMP to support soft real-time tasks. It mainly reforms the credit algorithm and makes performance comparisons for IP phone applications.
Domestically, the research project "Basic Theories and Methods of Computing System Virtualization" has united various key universities and research institutions in China to carry out in-depth and extensive research at various levels. Among them, literature [5] examines embedded virtual machines from different perspectives.

2. Analysis of real-time scheduling algorithm of partition operating system based on simple task model in single-core environment

A real-time partition is defined as $\text{VM}_k = \langle \Gamma_k, \Theta_k, T_k, \text{intraVM} \rangle$, including 1) The partition management scheduler uses interVM algorithm (SEDF/Credit/BVT...). 2) Periodic resource model, $\Gamma_k$. 3) A task set given by the operating system in a partition, $T_k$. 4) A scheduling algorithm intraVM inside the Guest OS. Among them, $\Pi_k$ represents the period; $\Theta_k$ represents the execution time allocated by inter-VM; $T_k = \{T(p_i, e_i, d_i)\}$ is the task set of the operating system in a partition, contains the task cycle $p_i$, execution time $e_i$, and execution deadline $d_i$.

Then, we define $\text{VM}_k$ is real-time schedulable, if and only if under the scheduling of the algorithm intraVM, all $T_k$ can complete its own execution time $e_i$ before its execution deadline expires $d_i$. In order to verify the schedulability of virtual machines, we can use the definitions of supply-bound function (sbf) and demand-bound function (dbf), and use classic scheduling theory as the basis for analysis.

$sbf_{\tau}(t)$ can be defined as the minimum CPU time required to meet the real-time requirements of virtual clients with the support of high-level scheduling. This article only analyzes the simple periodic task model. $T_k$ inside the client can be boiled down to a task $\tau_k$, which runs under the scheduling of the interVM algorithm.

We assume that intraVM uses the algo algorithm, then $dbf_{\tau_k}(t)$ should be defined as the minimum CPU time required for all $\text{VM}_k$ run, and $Dbf$ is defined under the constraints of the algorithm algo and the task set $T_k$.

The research in this article will be based on the above-mentioned basic scheduling theory, to theoretically verify the hierarchical scheduling algorithm, and determine the real-time guarantee of the scheduling algorithm through a quantitative method. In terms of experiments, the XEN virtual machine will be used first to study its classic scheduling algorithm, and from the perspective of real-time optimization, the scheduling algorithm will be designed.

3. Research on Scheduling Algorithm with Real-time Preemption in Single-Core Environment

3.1. Introduction to XEN virtual machine scheduling algorithm

Xen's scheduling algorithm is mainly responsible for the allocation of CPU time slices to each guest virtual machine, so that hardware resources are reasonably allocated among each guest virtual machine. At the beginning of the guest virtual machine startup, Xen will configure the number of its CPUs, where the CPUs are called virtual processors, or VCPUs. Xen's virtual scheduling algorithm uses VCPU as the scheduling unit, rather than the guest virtual machine as the granularity for scheduling. The VCPU is dynamically allocated to each real physical CPU for execution, and only VCPUs with the number of physical CPUs can be executed at the same time. The advantage of controlling the granularity of the scheduling algorithm at the VCPU level is that it can increase the processing throughput of the entire virtual system, because each VCPU can migrate between cores according to the load level of the real physical CPU, thereby allowing the utilization of the system's processing capacity to reach maximum. If a guest virtual machine is used as the scheduling unit, when a guest virtual machine with multiple VCPUs is encountered, the same number of physical CPUs are required to complete the simultaneous scheduling of multiple VCPUs, which not only reduces the overall performance, but also is not conducive to system expansion.
3.2. Research on Credit Scheduling Algorithm

Historically, Xen has used Borrowed Virtual Time (BVT) algorithm, Simple Earliest Deadline First (SEDF) algorithm and Credit scheduling algorithm. The BVT algorithm and the SEDF algorithm are gradually being eliminated. Currently, the Credit scheduling algorithm is used by default in Xen. The Credit algorithm is designed and implemented by Emmanuel Ackaouy. It is a non-preemptive scheduling algorithm for proportional sharing. The main advantage of this algorithm is that it can globally manage multiple physical CPUs, thereby allocating CPU time to each virtual CPU fairly and efficiently. It can use SMP to allocate each physical CPU to a virtual CPU to achieve load balancing.

3.2.1. The embodiment of Boost priority. In Xen, a VCPU that is stuck in the Block state due to waiting for an event will call the wake function interface of the currently running scheduling algorithm when the vcpu_wake function is called by Xen. The key parts of the interface code in the Credit scheduling algorithm are as follows:

```c
if ( svc->pri == CSCHED_PRI_TS_UNDER &&
     !(svc->flags & CSCHED_FLAG_VCPU_PARKED) )
{
    svc->pri = CSCHED_PRI_TS_BOOST;
}
/* Put the VCPU on the runq and tickle CPUs */
__runq_insert(cpu, svc);
__runq_tickle(cpu, svc);
```

3.2.2. Credit value calculation. The core of the Credit scheduling algorithm is to perform priority-based queue sorting and adjustment according to the Credit value of each VCPU. Therefore, the update of Credit is necessary. In the algorithm, the function csched_acct completes this function. The function calculates the Credit value of all guest virtual machines every 30ms, and calculates the Credit value of the currently running VCPU every 10ms. The relevant code is as follows:

```c
for_each_online_cpu ( cpu )
  {//Local Timer 10ms
    spc = CSCHED_PCPU(cpu);
    set_timer(&spc->ticker,
              NOW()+MILLISECS(CSCHED_MSECS_PER_TICK));
```
In the code, a 10ms timer is set for each physical CPU to update the Credit value of the running VCPU, and then a global 30ms timer is set for the priority calculation of all guest virtual machines.

3.2.3. Dynamic load balancing. In the Credit scheduling algorithm, when a physical CPU is idle or the VCPU to be scheduled on another physical CPU has a higher priority than all the VCPUs in the current scheduling queue, the physical CPU will "steal" the high priority VCPU Come and execute. Although the Credit scheduling algorithm has obvious advantages, including the ability to globally manage multiple physical CPUs. In this way, the CPU time is fairly and efficiently allocated to each guest virtual machine, and each physical CPU can be allocated to each VCPU by SMP to achieve load balancing.

3.3. Optimized algorithm design capable of real-time preemption
The real-time preemptible scheduling algorithm is based on the Credit scheduling algorithm to asymmetrically increase the support for real-time client virtual machines, and improve the real-time performance of real-time client virtual machines.

3.3.1. Real-time guest virtual machine friendliness. Since the Credit scheduling algorithm is a non-preemptive scheduling algorithm that shares the CPU fairly and proportionally, at the same time, regardless of whether the guest virtual machine running on Xen has real-time requirements, it will be treated as a type of guest virtual machine for scheduling. In this way, all guest virtual machines including real-time guest virtual machines are scheduled according to the default scheduling method, and the position of the real-time guest virtual machine in the scheduling queue is also determined by its priority. When a real-time guest virtual machine needs to process events from the hardware, it needs to wait for all the guest virtual machines in front of it to consume its allocated time slice or fall into the Block state due to the event waiting before it has the opportunity to use processor resources. And this situation will inevitably bring additional response delays to real-time guest virtual machines.

In the implementation process, avoid modifying the support part of the Credit scheduling algorithm for non-real-time guest virtual machines, including the calculation of the Credit value and the adjustment of the queue, thereby reducing the impact on the fairness of resource usage among non-real-time guest virtual machines. At the same time, in the algorithm implementation, the support for real-time guest virtual machines is increased asymmetrically. First, the real-time identification flag Isrealtime is added to the structure rtfs_dom and the scheduling unit rtfs_vcpu initialized by the algorithm for each guest virtual machine.

```c
struct rtfs_dom {
    struct list_head active_vcpu;
    struct list_head active_sdom_elem;
    struct domain *dom;
    uint16_t active_vcpu_count;
    uint16_t weight;
    uint16_t cap;
    uint16_t isrealtime;
};
struct rtfs_vcpu {
    struct list_head runq_elem;
    struct list_head active_vcpu_elem;
    struct rtfs_dom *sdom;
}...
```
struct vcpu *vcpu;
    atomic_t credit;
    uint16_t flags;
    int16_t pri;
    uint16_t rtvcpu;
}

When a new guest virtual machine is started, the algorithm can judge whether it is a non-real-time guest virtual machine according to the ID of the guest virtual machine, and initialize the corresponding structure, and set the corresponding flag. During the execution of the algorithm, these flag bits can be used to provide real-time preemptible scheduling services for real-time client virtual machines. At the same time, in the scheduling queue sorting every 30ms, the real-time client virtual machines are arranged at the head of the queue.

3.3.2. Boost priority preemption mechanism. In the Credit scheduling algorithm, the execution of entering the Boost state and grabbing the current processor resources is handled by the Wake function. Therefore, if it is necessary to give the real-time guest virtual machine the ability to seize the current processor resources, it needs to be implemented in this function. First, through the real-time identification flag of the scheduling unit structure, the real-time virtual machine is given Boost state priority.

```c
//if the vcpu is a rt one, boost it anyway
if (svc->rtvcpu == RTFS_VCPU_IS_RT)
{
    svc->pri = RTFS_PRI_TS_BOOST;
}
```

Then, in the adjustment of the scheduling queue, the real-time guest virtual machine is taken as the next scheduling object.

```c
//add the rtvcpu to the runq's as the next task
if(svc->rtvcpu ==RTFS_VCPU_IS_RT)
{
    list_add_tail(&svc->runq_elem,runq->next);
    //printk("RT vcpu inserted\n");
}
```

Finally, in the function that notifies the current physical CPU for scheduling, the most critical asymmetric comparison is added; because in the Credit scheduling algorithm, only the guest virtual machine that enters the Boost state has a higher priority than the currently running guest virtual machine. The preemption of processor resources is allowed. Therefore, the judgment for real-time guest virtual machines is added asymmetrically in this part. If the real-time guest virtual machine is the new priority of the Boost state, it is allowed to preempt the current processor resources, and a scheduling soft interrupt occurs to realize a real-time preemption mechanism.

4. Experimental data and summary

Hardware environment: Intel I7-5600U Processor, 4GB memory, 1TB hard drive.

Software environment: The version of the Xen virtual machine platform is 3.4.1, install with source code. The operating system of Domain0 is Cent OS 5.5; a real-time guest virtual machine and two processor-intensive guest virtual machines are run on the Xen platform, and the response delay of the real-time guest virtual machine to events is tested; the configuration of each virtual machine is passed The respective Config files are set to 1 VCPU and 256M memory.

In the test, the experiment separately tested the event response delay performance of the real-time guest virtual machine in the case of the Credit scheduling algorithm and the real-time preemptive scheduling algorithm. Cyclitest was used to perform a 100,000 kernel delay test for the data delay test. The real-time virtual machine and Two CPU-intensive non-real-time virtual machines are running on one physical CPU.
Test results: Under the Credit scheduling algorithm, the kernel latency of the real-time virtual machine below 20 microseconds accounted for 68.00% of the total number of tests, indicating that the real-time performance of the real-time guest virtual machine under the Credit scheduling algorithm is unstable. The same test is performed in an environment where the scheduling algorithm can be preempted in real time. The kernel latency of less than 20 microseconds accounts for 91.59% of the total number of tests, and it can be seen that the range of kernel latency of real-time guest virtual machines is effectively reduced. At the same time, the kernel latency of 20 microseconds has been greatly increased, which means more stable real-time performance.

In order to further analyze the performance of real-time guest virtual machines, the experiment uses Xen Monitor of the Xen virtual machine platform to further observe the waiting status of real-time guest virtual machines in the queue under different scheduling algorithms. The data results are shown in Figure 2.

![Figure 2. Comparative analysis](image)

In the figure, the vertical axis represents the percentage of real-time guest virtual machines in a waiting state per second, and the horizontal axis represents the test time in seconds. From the above figure, it can be found that under the Credit scheduling algorithm, the waiting time percentage of real-time guest virtual machines fluctuates greatly, the highest can reach 9.17%, while the lowest is only 0.15%, and the average waiting time percentage is 2.43%. Under the real-time preemptible scheduling algorithm, the waiting time percentage of real-time guest virtual machines is very stable and maintained at a low level, and the average waiting time percentage is only 0.13%.

5. Acknowledgments
This work was financially supported by Shenzhen Special Fund for Strategic Emerging Industry Development JCYJ20170816151055158 and Key Basic Research Projects of the Military Science and Technology Commission Basic Strengthening Plan 2019-JCJQ-ZD-097-00.

References
[1] Willmann P, Shafer J, Carr D, et al. Concurrent direct network access for virtual machine monitors. High Performance Computer Architecture[C]. 13th International Symposium on IEEE, 2007:306-317
[2] Yu Peijie. Research on asymmetric real-time scheduling algorithm based on Xen. Shanghai: Shanghai Jiao Tong University Software Engineering Master Degree Thesis, 2011
[3] Robert Kaiser. Alternatives for Scheduling Virtual Machines in Real-Time Embedded Systems[C]. 1st Workshop on Isolation and Integration in Embedded Systems 2008
[4] Min Lee, A. S. Krishnakumar, P. Krishnan. supporting soft real-time task in XEN hypervisor[C]. VEE’10, March 17–19, 2010
[5] Wang Z, Weng C, Wang Y, et al. A resource management mechanism and its implementation for virtual machines[C]. Systems and Virtualization Management. Standards and New Technologies. Springer Berlin Heidelberg, 2008:113-118