Scheduling Model of Virtual Machine Base on Task Type in Multi-core System

Hui-Xing Chen, Kenli Li, and Lin Shi

School of Information Science and Engineering, Hunan University,
Changsha 410082, China
{celine,lkl,shilin}@hnu.edu.cn
http://www.hnu.edu.cn

Abstract. The traditional virtual machine scheduling algorithm does not fully consider the execution efficiency of parallel applications. When multiple virtual machines cooperate to execute the parallel computing tasks, the virtual machine monitor still allocates the physical CPUs by the time-division multiplexing method. That will lead the parallel tasks to be serialized and the efficiency degraded greatly. The modern chip multiprocessors platform involves several available computing cores, to meet the need of the concurrent execution of multiple virtual machines. In this paper, we proposed a dynamic scheduling strategy –CON-Credit scheduler, which helps to speed up the parallel applications in virtual environment with multi-cores or many cores system. The main feature of CON-Credit is to map the virtual CPU to the physical CPU directly, so the virtual machines involves parallel tasks can take fully advantage of the underlying hardware resources. More precisely, the CON-Credit algorithm dynamically allocated processor cores to the virtual domains according to the type of the application. For the parallel applications, CON-Credit chooses to schedule a bulk of physical CPUs at the same time to avoid the extra makespan of discrete dispatch in traditional virtual machine scheduling algorithm. The experimental results show that the CON-Credit algorithm improved the execution efficiency of the parallel application and optimized the overall performance of the virtual machine system.

Keywords: multi-core, scheduler, parallel, VMM, MapReduce.

1 Introduction

System virtualization technology once had a great vogue in the mainframe era. But it fades out as the descent of the computer hardware price and the popular use of the Personal Computer. At the beginning of the 21st century, the prosperity of server consolidation market and the proposition of cloud computing concept offer the opportunity of resuscitation to system virtualization technology. Nowadays the system virtualization technology has become an important infrastructure of cloud computing platforms, and there are more than one hundred affiliated companies and platforms, for instance, KVM, XEN, VMware,

Y. Zhang, K. Li, and Z. Xiao (Eds.): HPC 2012, CCIS 207, pp. 26–39, 2013.

Virtual PC, Hyper-V, and a large number of mature virtual machine platforms have been widely deployed and applied.

Just as the role of process scheduling algorithm in the operating system, the virtual machine scheduling algorithm is a key factor which affecting the overall performance of system virtualization. The virtual machine scheduling is implemented by the VMM(Virtual Machine Manager).At present, the popular VMM scheduling algorithm directly absorbed the essential idea of the operating system scheduler. It cant reflect the characteristics of the virtual machine platform: the virtual machine is a set of hardware and software, and it is far greater in granularity than the traditional process or thread on scheduling. The virtual machine contains an additional scheduling layer, thus forms a double-layer and two-dimensional scheduling architecture. Because the OS of virtual machine may be heterogeneous, uncooperative, and unfriendly to virtualization, the scheduling optimization only in the virtual machine manager layer may be not enough. Therefore, it is important to design a multi-dimension scheduling algorithm for virtualized multi-core processor system according to the very varied types of application. And the most important factor is to design the scheduling algorithm. In the single-core computer system, VMM, Domain0 and Domain U all run in a processor core. But the multi-core platform has the capability to simultaneously run multiple virtual machines.The traditional time-division multiplexing scheduling algorithm should transit to the hybrid schedule mode which combines both the time-division multiplexing and space-division multiplexing.All in all, traditional VMM scheduling algorithms mainly focus on the fairness of processor resources among the virtual machines, but havent considered multipath parallel features of the multi-core processors. Therefore, it leads to the decline in the execution efficiency of the parallel tasks, when request tasks contains some parallel tasks, the default scheduling algorithm in VMM cant fit to the characteristic of the underlying hardware, and fully utilize the hardware.

In this work, we take the virtual machine monitor XEN as an example, design a virtual machine schedule algorithmCON-Credit for the parallel environment. The CON-Credit exploits the virtual machine requests-assign mechanism, and dynamically adjusts the resource allocation pattern; the CPU resources are allocated to the different tasks, according to the current resource status and task characteristics. Without enforce a serious impact on other tasks, it ensures the distribution execution in space and concurrent execution in time of the parallel tasks, and improves the overall performance of the multi-core platform.

2 Background

2.1 Xen's Credit Scheduler

Xen[1] is a virtual machine monitor based on the open source Linux kernel, it comprises two parts: the hypervisor which located in the highest privilege level management monitor and the frontend/backend driver in the virtual domain. On recent versions of Xen, the Credit scheduler[2] is used by default. For this

scheduler, each domain has two properties associated with it, a weight and a cap. The weight determines the share of the PCPU time that the domain gets, whereas the cap represents the maximum that the domain can get. In contrast, the cap is an absolute value, representing a proportion of the total CPU that can be used. The credit scheduler manages multiple PCPUs and distributes CPUs time to each VCPU equally, which in order to realize the load balance. However, it focus mainly on the fairness, ignores the specific characteristics of different tasks, which leads to the inefficiency for the collaborative workload in the multi-core system.

2.2 MapReduce

MapReduce is a distributed computing model by Google. It is simply represented in two functions: Map and Reduce. Map function was written by the user, processes a key-value pair to generate a set of intermediate key/value pairs. Reduce function was written by the user too, which merges all intermediate values associated with the same intermediate key. The Map usually contains independent operation adapt to the characteristics of the large-scale parallel and distributed computation. In MapReduce clusters consist of a large number of nodes for redundancy and fault tolerance, each node in the cluster is assigned a certain number of data chunks, which are in turn duplicated on several nodes based on a distributed file system[3][4]. In the Map and Reduce operations, the Map is a highly parallel, and Reduce is dependent on the result of Map operations. The parallelism and balance of the Map operation determine the overall efficiency of the MapReduce program.The MapReduce isalso widely used in the system-level virtual machine platform, such as CLOUDLET [5] and Cloud BLAST [6] and other projects. They combined the flexibility of the virtual machine and the distributed parallelism of MapReduce framework, so the MapReduce can easily be deploied in the cloud platform. It is important for cloud platform to support this kind of parallel computing.

3 Scheduling Strategies and Analysis

3.1 Motivations and Modeling

In this paper, we take the *MapReduce* structure model for an example, which is shown in the figure 1, where nodes $V = V_{11}, V_{12}, \cdots, V_{21}, V_{22}, \cdots, V_{i1}, \cdots, V_{ij}$ denote the set of different tasks. Assume that the computation time of all the nodes at the same level is the same. The processing time of the $i - thlevel$ task is $T(V_i), T(V_i) = T(V_{i1}) = T(V_{i2}) = \cdots = T(V_{ij}), i = 1, 2, 3, \cdots, I$.Assume there are 4 cores available to execute this task. If a node V_{ij} has already started in the core, we define the node can occupy the processor core resources until its processing completed. The target of scheduling is to find out a dispatch scheme to guarantee the minimum of total execution time.

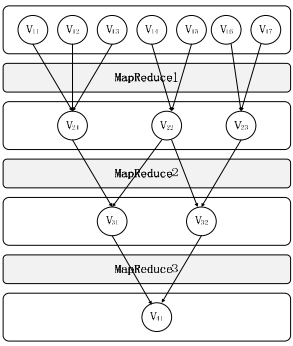

Fig. 1. The Structure model of *MapReduce*

The main drawbacks of the existing default credit scheduler when applied to *MapReduce* workloads are as follows. Figure 2(a) shows the scheduling time of the nodes on the 4 cores physical machine under *MapReduce* workloads. The total execute time is $T = 4 \cdot T(V_1) + 2 \cdot T(V_2) + T(V_3) + T(V_4)$.

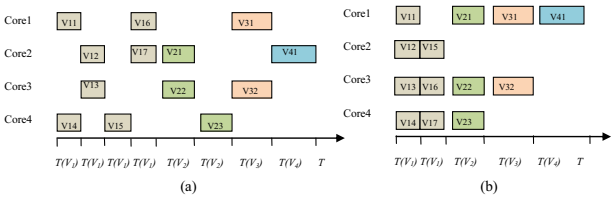

Fig. 2. (a)A possible order distribution of the *MapReduce* workloads with Credit algorithm.(b)A possible order distribution of the *MapReduce* workloads with concurrent scheduling algorithm.

The concurrent scheduling algorithm uses a novel mechanism to optimize the Credit algorithm. For a node, the tasks of next level must wait for all the tasks of the previous level were disposed, before it is implemented. To make full use of processor resources, a core can be adopted to maximize the parallel process the various tasks of the same level.The total time$T = 2 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$.A order distribution of the *MapReduce* workloads with concurrent scheduling algorithm base on this idea is as shown in the figure 2(b).

Further we will discuss the situation that hybrid types of tasks run simultaneously. Assuming that the *MapReduce* model above and the other five common tasks send request at the same time, there are six physical processor cores available and fix 3 processor cores to each type of task before execute. The total time$T = 3 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$, a possible schedule order distribution of the hybrid tasks as shown in the figure 3(a).

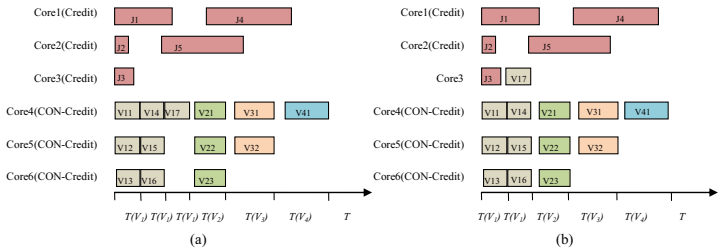

Fig. 3. (a)A possible order distribution of the *MapReduce* workloads and common tasks with Credit schedule.(b)A possible order distribution of the *MapReduce* workloads and common tasks with concurrent scheduling algorithm.

If we design a mechanism to adjust the allocation of resources dynamically according to the number of tasks for the different types of tasks, this allows us to make full use of the resources, improve the overall real-time and throughput, and reduce the total execution time. At this point, due to the *core*3 has been idle after performing task J_3, if its resources assigned to V_{17} dynamically, the total time$T = 2 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$. Now, a schedule order distribution of the hybrid tasks as shown in the figure 3(b).

From the above scheduling situation, we can see that the default credit algorithm for parallel tasks has obvious defects, but the concurrent scheduling algorithm in multi-core system show better performance. In addition, the analysis of the hybrid tasks shows that dynamic scheduling give more conducive to improve the resource utilization and application performance than static scheduling, thereby enhancing the throughput of the entire system, reducing the minimum total execution time the event($MAKESPAN$).

3.2 Scheduling Framework

Based on the above analysis, this paper proposes CON-Credit scheduling model(concurrent credit scheduler) that according to the types of the tasks to divide multi-core processor core dynamically. The task status monitoring and scheduling decision-making module are added in this model on the basis of the original VMM scheduler structure. The former collect time and task status information in the process of device access by reading the data in the event channel, while the latter process the information that have collected in real-time, and dispatch cores resources to each VCPU dynamically to deal with the tasks of corresponding types, make a scheduling decision when some conditions are satisfied.

The four main parts to realize this scheduling model are as follows:1) Cores are divided into the COM-Core and CON-Core dynamically, COM-Core corresponding to ordinary task and CON-Core corresponding parallel tasks; 2) The VMM scheduler dispatch tasks according to the application type in the different sets; 3) Task status monitoring and real-time decision-making. 4) Scheduling virtual machine in parallel and collaboratively. The separation of CON-core and

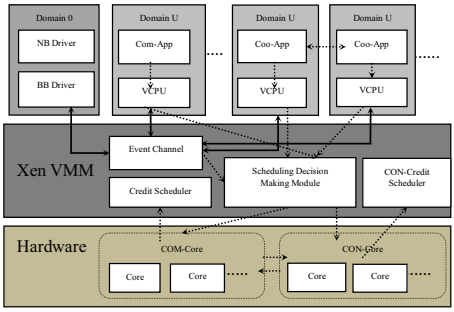

Fig. 4. The overall structure of the scheduling strategy model in VMM

COM-Core is used to reduce the time of the physical CPU resource allocation and to respond events request quickly. The task of condition monitoring and dynamic division of decision-making are used to reduce the performance penalty of context switching of the different task types. Parallel and collaboration of the virtual machine are conducive to the separation between the parallel tasks, and the implementation of the synchronous collaboration to achieve the task parallel processing. The overall structure of the scheduling model was shown as figure 4.

3.3 CON-Credit Strategy Analysis

Taking into account $|P|$ processor cores system, use$P = P_1, P_2, \cdots, P_{|P|}$ to represent all the multi-core processors. Let$J = J_1, J_2, \cdots, J_n$ means be the set of the common tasks which have n tasks. For each task$J_i \in J, T(J_i)$means the processing time of task i. Besides, with $V = V_{11}, V_{12}, \cdots, V_{21}, V_{22}, \cdots, V_{i1}, \cdots, V_{ij}$ denotes the set of the nodes of parallel tasks, and m nodes. $T(V_{ij})$ denotes the process time of node i, and let $W = J \cup V_t$. Assume the set of the $|D|$ VCPUs is $D = D_1, D_2, \cdots, D_{|D|}$. For a running time, there is $|W_R| \leq |D_R| \leq |P_R|$, and$|W_R|, |D_R|, |P_R|$ denotes the number of tasks are running at the moment, VCPUs and cores respectively. In the multi-core virtual machine monitor system, the task scheduling includes allocation of the tasks in VCPUs to the *domainU*, denoted by $D_k = \lambda(W_i)$, and the VMM allocate the cores to the each *domainU*, denoted by $P_j = \mu(D_k)$. So $P_j = \mu\left(\lambda(W_i)\right)$ denotes that allocate the processor *core* j to the *node* i. (((((())))))We make the weight of PCPU process the common types of task is$\varpi(P_J)$, and the parallel types of task is$\varpi(P_V)$ respectively, then$\varpi(P_J) + \varpi(P_V) = 1$. Of course, the cores that perform two types of tasks can interconversion according to the value of$\varpi(P_J), \varpi(P_V)$.

When the event request, allocate processor core resources to the various VCPUs dynamically depending on the value of$\varpi(P_J), \varpi(P_V)$ to. Among the$|P|$ processor cores system, we assume that each half of the processor cores are assigned to the two types of tasks. In operation, record the number of two types of tasks and the processing time required to be processed within a time unit in real-time by state monitoring mechanism. We assume the number of common tasks is

C_1, then the total processing time can express as$\sum_1^{C_1} T(J_{K_1}), k = 1, 2, 3, \cdots, C_1$. At the same time, we assume the numbers of parallel tasks type are C_2, then the total processing time can expressed as $\sum_1^{C_2} T\left(V_{K_2}\right), k = 1, 2, 3, \cdots, C_2$. So, the ratio of the task duration is denoted by $R = \frac{\sum_1^{C_1} T\left(J_{K_1}\right)}{\sum_1^{C_2} T\left(V_{K_2}\right)}$, and let $\frac{\varpi(P_J)}{\varpi(P_V)} = R = \frac{\sum_1^{C_1} T\left(J_{K_1}\right)}{\sum_1^{C_2} T\left(V_{K_2}\right)}$, $\varpi(P_J) + \varpi(P_V) = 1$. As a result, the number of processor cores are assigned to the type of the common tasks can be defined as:$R_J = \lfloor \varpi(P_J) \cdot |P| \rfloor = \left\lfloor \frac{\sum_1^{C_1} T\left(J_{K_1}\right)}{\sum_1^{C_1} T\left(J_{K_1}\right) + \sum_1^{C_2} T\left(V_{K_2}\right)} \cdot |P| \right\rfloor$ .The number of processor cores are assigned to the type of the parallel tasks can be defined as: $R_V = \lceil \varpi(P_V) \cdot |P| \rceil = \left\lceil \frac{\sum_1^{C_2} T\left(V_{K_2}\right)}{\sum_1^{C_1} T\left(J_{K_1}\right) + \sum_1^{C_2} T\left(V_{K_2}\right)} \cdot |P| \right\rceil$ .Here $|P| = R_J + R_V$.

In the CON-Credit algorithm, when a task node W_i has begun executed in the processor core, it will occupy the core until it finished. The question is how to find out a task scheduling strategy that has minimum total execution time for the request event so as to improve the real-time performance and throughput. When an event sends a request, each task nodes are assigned to each processor core according to $\mu\left(\lambda(W_i)\right)$, $T\left(\mu_{W_i}^{P_j}\right)$ denote the process time of the task W_i that assigned to the processor core P_j. If the total number of tasks that has already completed is K within the time T, then the throughput is:$\varphi = \frac{K}{T}$, and:

$$T = \max\left\{\sum_J T(J_i), \sum_V T(V_i)\right\} + \sum_{i=1}^{n+m} T\left(\mu_{W_i}^{P_j}\right) . \tag{1}$$

When the parallel task comes, each node is scheduled according to their level or their sequential dependencies. The tasks at the same level are allocated to each virtual machine in accordance with the$\lambda(V_i)$, and corresponds to the processor core for concurrent processing in accordance with the$\mu(\lambda(V_i))$. Then allocate the appropriate number of processors to process the task nodes at each level according to the value of$\varpi(P_J), \varpi(P_V)$, and disposed of these tasks in a period of time as much as possible to maximize concurrent processing of parallel tasks. When$R_V < |V_i|$, the processor cores resources are not enough. At this time, assign theR_V processor cores toR_V nodes in accordance with the function$\mu\left(\lambda(V_i)\right)$, and detect dynamically if the task on the virtual machine completed or not. If finished, transfer to the other node at the same level that waiting for processing. When all the nodes at the same level have finished, then execute the tasks of each node of the next level. The events the total execution time $T(V)$ is as follow:$T(V) = \sum_{i=1}^{I} k_i \cdot T(V_i) + \sum_{i=1}^{I} \sum_j T\left(\mu_{V_{ij}}^{P_n}\right) = \sum_{i=1}^{I} \left\lceil \frac{j_i}{R_V} \right\rceil \cdot T(V_i) + \sum_{i=1}^{I} \sum_j T\left(\mu_{V_{ij}}^{P_n}\right)$; Where $k_i = \left\lceil \frac{j_i}{R_V} \right\rceil$, k_i denotes the task allocate to the $i-thlevel$, times of $T(V_i)$ time period on a processor cores is needed to complete all of the tasks in the $i-thlevel$. So the total execution time for such events is as follows:$\min T(V) = \min \sum_{i=1}^{I} \left\lceil \frac{j_i}{R_V} \right\rceil \cdot T(V_i) + \min \sum_{i=1}^{I} \sum_j T\left(\mu_{V_{ij}}^{P_n}\right)$.

Therefore, when the hybrid tasks request, the objective function is to minimize the execution time:

$$\min T = \max\left(\min T(V), \sum_{J} T(J_i) + \min \sum_{i=1}^{n} T\left(\mu_{J_i}^{P_n}\right)\right) . \tag{2}$$

Subject to:The time that task node V_{ij} obtained is decided by $\varpi(V_{ij})$, the $k - level$ task nodes scheduled before the $r - level$ task nodes; here $k < r$, and inequality$|W_R| \leq |D_R| \leq |P_R|$must be established.

3.4 The Algorithm Flow

The CON-Credit algorithm flow chart was shown as figure 5. In the multi-core processors virtual environment monitor system, the algorithm take VCPU for scheduling unit, each VCPU associated with the corresponding virtual machine. At the initial stage of this scheduling algorithm, it check the target VCPUs running state and the queue position, judge the request event type and obtain the proportion of the R value through the state decision module, and assign the core number of various types that needed according to the R value. During the operation, the state decision module monitor the ratio of the time required for each task type in dynamically, and generate the corresponding distribution value, reduce or increase the core number of the concurrent events needed according to the value to make more common tasks or concurrent events can be operated. In operation, if a new task request, the function$\mu\left(\lambda(W_i)\right)$ will generate a new allocated valueR_j, R_v, reallocate processor resources to each virtual CPU. Meanwhile, the scheduling decision module should judge the type of event and select scheduling strategy. For the concurrent event, the CON-Core is selected. Otherwise, the default COM-Core is assigned.

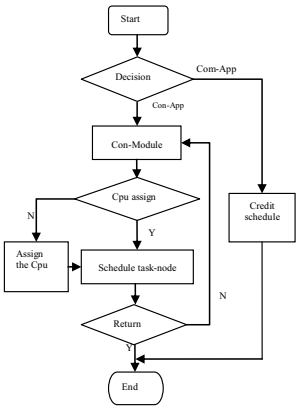

Fig. 5. The model flow chart

4 Performance Evaluation

4.1 Experiment Environment

The hardware configuration of the experimental platform is Intel eight-core Xeon 7550 processors, the Seagate 1TB IDE hard disk, DDRII-800 8GB memory, and the RTL 8139D 200Mbps Ethernet card. All experiments used Xen 4.1.2 on the Fedora16 operating system with Linux 64bit 3.10.17 kernel. All virtual machines are running with Fedora 16 with kernel 3.10.17. At the same time we build a executable parallel programming environment with the virtual machines. To study the performance comprehensively, we analyze the performance of CON-Credit scheduling algorithm in three ways:Efficiency: What is the performance profit CON-Credit algorithm can get comparing with the traditional Credit algorithms.Adaptability: How the CON-Credit algorithm work in the hybrid task mode.Scalability: How the number of virtual machine will give an influence on the CON-Credit scheduling performance.

4.2 Benchmarks

Now, there are four *MapReduce* benchmarks(*Dot product*, π *Computation*, *RC*4 *Key Search* and $N-$ *body problem*)[7]have been constructed to demonstrate the applicability of CON-Credit schedule framework.*Dot product*:We perform the multiplications in the Map function and additions in the Reduce function.π *Computation*: The classic Monte Carlo simulation is used to approximate the value of π. For N paths, the output of the Map function is a stream of binary values$\in 0, 1$. The Reduce function is the addition and π is computed by multiplying the reduced value by $\frac{4}{N}$ on the host computer.*RC*4 *Key Search*: The Map function input is an index indicating the position to start the search. The reduce function, implemented on the host, checks the return value and outputs the correct key if found.$N-$ *body*:In our test, the input to the Map function is the current information for the n particles and the particle index. It is the reduce functions responsibility to fresh each particle new state according to the formula associated[8].

4.3 Performance Measurement and Analysis

Now, we take the above 4 typical*MapReduce* programs and serial program (such as *Rank* sorting algorithm and *gcc* compile test in SPEC CPU2000) as the example to verify our proposed virtual machine scheduling algorithm and compare its performance with traditional one.

The first experiment tests per the formance of the distributed *MapReduce* applications which ran in eight virtual machines. Each domain is configured with 4 VCPUs and only one *MapReduce* task runs on each of them. The result is depicted in Figure 6. Compare with the traditional scheduler the total execution time of four benchmarks reduced by 28.17%, 25.86%, 30.47%, 30.39% respectively under the CON-Credit algorithm.

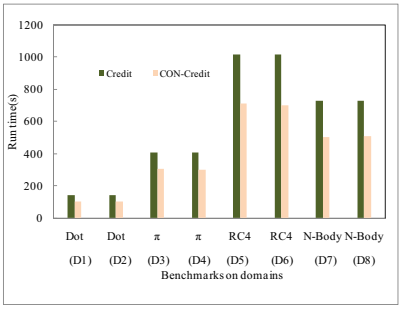

Fig. 6. The *MapReduce* benchmarks run simultaneously on 8 domains, respectively

The second experiment measures the overall performance of CON-Credit with hybrid workload, server the parallel and serial task at the same time. In this test, eight domains are created on XEN, and each domain is configured with 4 VCPUs. The figure 7 show the case then 4 domains run the *MapReduce* programs while the others involve only serial programs. According to Figure 9, the proposed CON-Credit algorithm show better performance than traditional algorithm, the execution time of dot product is decreased by 25.61%,π *Computation* is decreased by 23.06%,$RC4$ *Key Search* is decreased by 28.60%, $N-body$ is decreased by 26.27%.However, the execution time of serial programs *Rank* is increased by 5.33%, and 176.*gcc* is increased by 9.07%. Because the improved VMM scheduler has added a scheduling decision module, all the tasks need the scheduling decision module to allocate physical processor resources. That leads to the time of serial tasks has a slight increase. This is acceptable in the large environment.

The figure 8 show the another case, up to 6 domains are configured to run *MapReduce* programs and the other 2 domains severed for the serial programs. The overall execution time of the benchmarks with two schedulers is depicted. According to Figure 10, the CON-Credit algorithm speeds up the *dot product*,

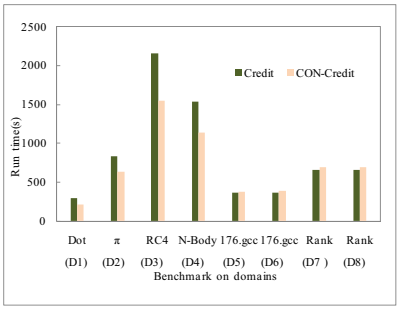

Fig. 7. Benchmarks run simultaneously on 8*domains* respectively, 4*domains* are used to *MapReduce* programs and the others are used to serial programs, and the iterations are set to 10^9

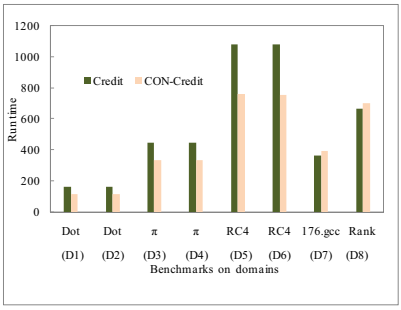

Fig. 8. The benchmarks run simultaneously on 8*domains* respectively, 6*domains* are used to *MapReduce* programs and the other 2*domains* are used to serial programs, and the iterations are set to 10^9

π *Computation*, and *RC4 Key Search* by 26.23%, 24.94%, 29.70% respectively. However, the execution time of *Rank* serial programs is increased by 5.25%, and 176.*gcc* is increased by 6.20%. It is also acceptable in the large environment as the same reason of before said.

In the 3rd experiment, we create several virtual machines to stimulate the cluster environment which be used to test the hybrid scheduling performance, and the *MapReduce* parallel tasks and ordinary serial tasks are scheduled simultaneously. In order to find out how the number of domains influences the overall performance of scheduling algorithm, we test the result with different cores and domains. Each domain is equipped two VCPU, and the test cases are set to the 4 *MapReduce* benchmarks and two serial tasks. In this test, the CON-Credit algorithm show better performance than the traditional scheduling algorithms for parallel tasks, and adapt to the hybrid mode which involves both the parallel and serial tasks.

The results of the experiment show that the total execution time changed with the growing number of domains, and the average time of all tasks with the corresponding number of domains are all shown on figure 9. The experiment demonstrates the relative advantage of CON-Credit algorithm for *MapRrduce* tasks than the default scheduling algorithm. With the number of virtual machines increased, the execution time of *dot product* is decreased by9.18%, 18.03%, 13.62%, 12.75%, 13.08%, 14.81%; the execution time of π *Computation* is decreased by 9.31%, 16.82%, 9.98%, 12.18%, 13.43%, 15.43%; the execution time of *RC4 Key Search* is decreased by 10.08%, 13.55%, 10.74%, 9.81%, 10.71%, 11.99%; the execution time of $N-body$ is decreased by 9.68%, 14.01%, 9.81%, 9.79%, 10.55%, 11.40%. However, the execution time of *rank serial programs* increased by 1.19%, 5.45%, 4.42%, 3.62%, 3,37%, 2.94% in proper order, and 176.*gcc* increased by 2.79%,1.97%,2.43%,3.60%,5.18%,7.91% in proper order.

As the Figure 9 shows the whole system get the maximum performance when the number of domains is configured to 4. The reason is there are only 8 physical cores in the testbed, when 4 domains are running on top of the systemthere are eight VCPUs on XEN in total because two VCPUs are set to each domain.

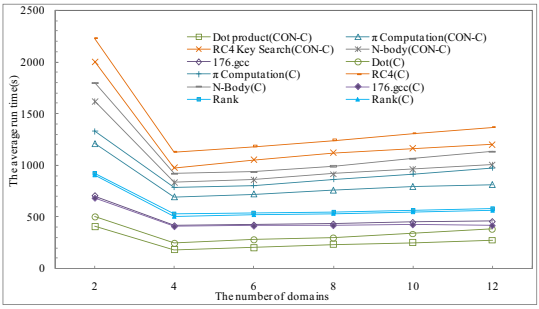

Fig. 9. 4 *MapReduce* benchmarks and 2 common tasks are run simultaneously on domains, and the *MapReduce* benchmarks iterations set to 10^9

The communication overhead among domains in the Reduce process is smaller relative to more domains. When the number of VCPUs is equal to the number of physical processor cores, the execution time of Map process is reduced, and system reach a balance point. When the number of VCPUs is great than 8, the extra communication overhead will increase dynamically and fade out the benefit of more VCPUs. Besides, with the increase of domains, the increase velocity of the time of CON-Credit is gentle and trends to a steady value, but the default algorithm still keep in an increasing trend. For serial task, when four VCPUs in two virtual machines are created, the efficiency gets its peak value. However, when the number of domains increases, the time spend in scheduling make decision module also grow. The scalability experiment shown on the figure 9 shows that even take this element into account, the CON-Credit algorithm still exhibit good scalability in large scale system.

5 Related Work

In this section, we present the related work on scheduling in the XEN VMM. Several extensions to Xens Credit scheduler are proposed to improve I/O performance, by adding a highest priority status named BOOST. The BOOST related credit scheduler sort the RUN queue based on their remaining credits, and tickling the scheduler when events are sent. The [9] has largely focused on improving the efficiency of I/O operations and has not explored the impact of the scheduler on I/O unrelated task. Scheduler improvements for I/O are likely to also benefit these innovative I/O designs. Kim et al[10] proposed a task-aware virtual machine scheduling mechanism based on inference techniques using gray-box knowledge. The proposed mechanism infers the I/O-boundless of guest-level tasks and correlates incoming events with I/O-bound tasks. Chuliang Weng et al[11] analyzed the CPU scheduling problem and presented a hybrid scheduling framework for the CPU scheduling in the virtual machine monitor. Lee et al[12] have identified the area of soft real-time application domains and the performance problems they encounter in virtualized environments. L Shi et al [13]

proposed vCUDA, a general-purpose graphics processing unit computing solution for virtual machines. vCUDA allows applications executing within VMs to leverage hardware acceleration, which can be beneficial to the performance of a class of HPC applications. Philip M. Wells et al. [14] proposed a simple hardware technique to detect when a VCPU is spinning, without requiring any software modification, and preempt that VCPU in favor of one which is making forward progress. Hui Kang et al. [15] designed and implemented the MRG scheduler, a new Xen scheduler for VMs running MapReduce workloads. The scheduler facilities MapReduce job fairness by introducing a two-level group credit based scheduling policy. Chuliang Weng et al [16] proposed an adaptive dynamic co scheduling method to mitigate the problem, while avoiding unnecessary overhead for co scheduling, and implement a prototype ASMan. Neither of above work combines the strength of MapReduce framework with the flexibility of virtual machine technology. We propose a schedule algorithm, CON-Credit , which is a fundamental approach for adapting to the diversity of VMs in the cloud platform, to decide when and how to map VCPUs to cores wisely. Moreover, the hybrid scheduling framework supports distributing core resources among VCPUs based on demand, as well as distributing equally.

6 Conclusion

More and more parallel computing tasks have been deployed in the cloud computing and virtualization platform. In this paper, we designed and implemented a scheduling algorithm named CON-Credit that make full use of the inherent characteristics of multi-core architecture, since the traditional virtual machine scheduling algorithm does not adapt to the parallel task scheduling. The CON-Credit achieved the dynamic classification and matching of the physical CPUs, and improved the execution efficiency of the *MapReduce* task through cooperative scheduling between the ordinary tasks and parallel tasks. The experiments prove that the algorithm worked well for the parallel task scheduling and mixed task scheduling, and have strong adaptability and scalability. For now the CON-Credit algorithm is only implemented in XEN virtual machine platform. In future it will be ported to other VMM platform like KVM and Hyper-V. In addition, the cloud computing platform may involved a large number of heterogeneous cores, that is a good candidate to exhibit the flexibility and scalability of CON-Credit. How to combine the parallel computing tasks and heterogeneous virtual computing environment, and design a more adaptable virtual machine scheduling algorithm is still a topic worth further studying.

References

1. Chisnall, D.: The definitive guide to the Xen hypervisor, pp. 222–224 (November 2007)
2. Credit Scheduler, `http://wiki.xensource.com/xenwiki/creditscheduler`

3. Ibrahim, S., Jin, H., Lu, L., Qi, L., Wu, S., Shi, X.: Evaluating MapReduce on Virtual Machines: The Hadoop Case. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 519–528. Springer, Heidelberg (2009)
4. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: SOSP, pp. 29–43 (2003)
5. Matsunaga, A., Tsugawa, M., Fortes, J.: CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In: Proceedings of the 2008 Fourth IEEE International Conference on eScience, pp. 222–229 (2008)
6. Ibrahim, S., Jin, H., Cheng, B., Cao, H., Wu, S., Qi, L.: CLOUDLET: towards MapReduce implementation on virtual machines. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, pp. 65–66 (2009)
7. Yeung, J.H.C., Tsang, C.C., Tsoi, K.H., et al.: Map-reduce as a Programming Model for Custom Computing Machines. In: 16th International Symposium on Field-Programmable Custom Computing Machines, pp. 149–159 (2008)
8. Tsoi, K.H., Ho, C.H., Yeung, H.C., Leong, P.H.W.: An arithmetic library and its application to the n-body problem. In: Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 68–78 (2004)
9. Kim, H., Lim, H., Jeong, J., Jo, H., et al.: Task-aware Virtual Machine Scheduling for I/O Performance. In: Proceedings of the 4th International Conference on Virtual Execution Environments (VEE), pp. 101–111 (2009)
10. Weng, C., Wang, Z., Li, M., Lu, X.: The Hybrid Scheduling Framework for Virtual Machine Systems. In: Proceedings of the 4th International Conference on Virtual Execution Environments (VEE), pp. 111–120 (2009)
11. Ongaro, D., Cox, A., Rixner, S.: Scheduling I/O in virtual machine monitors. In: Proceedings of the 4th International Conference on Virtual Execution Environments (VEE), pp. 1–10 (2008)
12. Lee, M., Krishnakumar, A., Krishnan, P., Singh, N., Yajnik, S.: Supporting soft real-time tasks in the XEN hypervisor. In: Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 97–108. ACM (2010)
13. Shi, L., Chen, H., Sun, J.H., Li, K.L.: vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. IEEE Transaction on Computers, doi:10.1109/TC.2011.112
14. Wells, P.M., Chakraborty, K., Sohi, G.S.: Hardware support for spin management in overcommitted virtual machines. In: Proc. of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT 2006), Seattle, Washington, USA, September 16-20 (2006)
15. Kang, H., Chen, Y., Wong, J.L., Wu, J., Sion, R.: Enhancement of Xen's Scheduler for MapReduce Workloads. In: HPDC 2011, San Jose, California, USA, June 8-11 (2011)
16. Weng, C., Liu, Q., Yu, L., et al.: Dynamic Adaptive Scheduling for Virtual Machines. In: HPDC 2011, San Jose, California, USA, June 8-11 (2011)