Affinity-aware Dynamic Pinning Scheduling for Virtual Machines

Zhi Li, Yuebin Bai, Huiyong Zhang, Yao Ma
School of Computer Science, Beijing Key Laboratory of Network Technology, BeiHang University
lizhi@cse,buaa.edu.cn, byb@buaa.edu.cn, willzhy@cse.buaa.edu.cn, mayao1986@gmail.com

ABSTRACT

Virtualization provides an effective management in server consolidation. The transparence enables different kinds of servers running in the same platform, making full use of hardware resource. However, virtualization introduces two-level schedulers: one from Guest OS, where the tasks are scheduled to virtual CPUs (VCPUs); the other from the virtual machine monitor (VMM), where VCPUs are scheduled to CPUs. As a result, the lower level scheduler is ignorant of the task information so that it cannot allocate appropriate proportion of CPU resource for every Guest OS in some cases. This paper presents an affinity-aware Dynamic Pinning Scheduling scheduler (DP-Scheduling). We aim at two objects: Bridging the semantic gap between Guest OS and VMM, introducing an affinity-aware method and providing the tasks information about CPU affinity to VMM; Bringing up a novel scheduling, DP-Scheduling, so that VCPU can be pinned or unpinned on one CPU's running queue dynamically. For this purpose, we first get the Machine Address (MA) of process descriptor from the angle of VMM. The affinity information is also acquired before the task is enabled to run. To acknowledge the affinity information, DP-Scheduling calls an API provided by us. Depending on the affinity information, we put forward a series of measures to implement pinning dynamically as well as to keep workload balance. All implementation is confined to Xen VMM and Credit scheduler. Our experiments demonstrate that DP-Scheduling outperforms Credit scheduling by testing various indicators for CPU-bound tasks, without interfering the load balance.

1. Introduction

The development of multi-core technology enables server consolidation deployed in virtual machines (VM). The current mainstream in virtualization technology (such as Xen [5], KVM [25], VMware [29], etc.) contributes greatly to the isolation among VMs, the load balance for CPU resource, and the management for memory as well as hard disk. As a result, cloud computing [11] and High Performance Computing [12, 13] (HPC) gradually take the appropriate virtualization technology. Although virtualization allows the sharing of the underlying physical machine resources for different VMs, many thorny problems need to be solved (such as CPU usage, memory management, security, etc.). A hot issue on virtualization is how to improve the CPU utilization with the growing number of core, which is caused by the two-level independent schedulers and the perceived gap between them.

Our goal in this paper is to design a Symmetric Multi-Processors (SMP) scheduling, DP-Scheduling, to obtain better CPU utilization. Before scheduling, we capture the CPU affinity of the tasks in VMs and provide an API for bridging the perceived gap. Then, the DP-Scheduling calls the API before scheduling the VCPU where the aware task begins to run. Also, the DP-scheduling schedules VCPU mapping to CPU according to the affinity information. Besides, a policy is developed for keeping load balance, which means other kinds of tasks will not be penalized. Our experiments demonstrate that DP-Scheduling performs better than the original Credit scheduling by testing various indicators for CPU-bound tasks, without interfering with the load balance.

The rest of the paper is organized as follows. In section 2, we describe Xen, illustrate the Credit scheduler of Xen VMM and clarify the necessity of a new scheduling approach. In Section 3, we describe the affinity-aware mechanism which can infer the affinity information of tasks in VMs. Section 4 is the discussion of the design and implementation of DP-scheduling policy. Section 5 is the analysis of the experimental results from our prototype. We present related work in section 6 and conclusion in section 7.

2. Motivation

2.1 Xen: A Brief Overview

Xen, an open-source VMM, is in charge of hardware with the highest privilege [16]. There are two kinds of VMs: The privileged one, called domain0, is responsible for the management of other domains; the common one, called domainU, is the place where servers or applications are usually deployed.

Xen virtualizes the resources for domains such as VCPU. Having no right to access hardware directly, VMs use

978-0-7695-4302-4/10 $26.00 © 2010 IEEE
DOI 10.1109/CloudCom.2010.51

hypercalls to invoke functions in VMM. It means the guest OS has to be modified to fit the functionality, which is called para-virtualization. Besides, VMM proposes a highest ring, -1 ring, which makes guest OS can also be run in the ring of zero. We call this scheme hardware assisted virtualization which needs hardware supports, such as Intel-VT [14] and AMD-V [15] technology.

2.2 Credit Scheduler

Credit scheduler is the default scheduler in Xen designed to ensure proportionate sharing of CPUs. It sets the proportion of CPUs for VMs according to WEIGHT and CAP. WEIGHT stands for the share of the CPU time that the domain gets, which is a relative value. The CAP, an absolute value, determines the maximum CPU time for the domain. The VCPU priority is divided into four kinds: BOOST, UNDER, OVER and IDLE. Credit stands for time slices, whose value determines the priority. There are still some time slices remaining for VCPUs in state of UNDER, which is opposite to OVER. Those VCPU having nothing to do will back to IDLE. BOOST is a special priority for improving the I/O response. Only an IDLE VCPU waken up by an event can be set to BOOST, which means it will be scheduled firstly. The arrangement of VCPUs in CPUs run queue is merely ordered by priority but no direct relationship with the credit value.

2.3 A Scheduling Crisis: Cache Miss

In non-virtualization environment, scheduler from OS, acquainted with tasks attributes, can schedule them to some CPUs directly. OS can be evenly distributed to each thread on CPUs for multi-threads task, which makes full use of CPU resources in most cases. OS would not like to migrate the CPU-bound tasks except for loading balance, which avoids many L2 cache misses. However, there is a change in virtualization. Tasks can miss L2 cache even when Guest OS does not migrate them into other VCPU. We run the STREAM benchmark with different array sizes (1MBytes, 3MBytes, and 5MBytes) in both non-virtualization and virtualization environment with the same number of CPU/VCPU (8) and memory (1GBytes) as well as the same hardware platform. Detailed descriptions of our experimental test bed are given in Section 5. We run same tasks both on non-virtualized host and Xen-based system. We use Oprofile and Xenoprof [27] to capture the cache miss counts per 10000 counts. The result is shown in Figure 1. The number of threads has few effects on cache miss counts, but virtualization leads to more L2 cache miss counts, which means more memory access times and lower CPU usage.

2.4 Analysis of the Issue

The two-level schedulers, both from Guest OS and VMM, produce two kinds of load balance for SMP: The average task location for VCPU and the average VCPU location for CPU. Therefore, CPU will be fully used, which is Credit Scheduler's intention. However, it cannot distinguish VCPUs from different VMs, resulting in many VCPUs from a VM running on one CPU sometimes. The more VMs, the more frequent situation in this kind. With the increasing number of cores, the negative impact of this situation becomes more serious.

Although Guest OS strives to make threads run in different VCPUs, Credit scheduler can put some VCPUs in this kind to the same CPU run queue arbitrarily. Hence, two or more CPU-bound threads take turns to run, which makes L2 cache refresh frequently. In addition, Credit scheduler will seek another runnable VCPU with higher priority when the local CPU run queue includes no BOOST or UNDER VCPU. This migration will be more frequently when total number of VCPU is more than CPU . Once this kind VCPU is migrated to other CPU run queue, L2 cache will be missed completely.

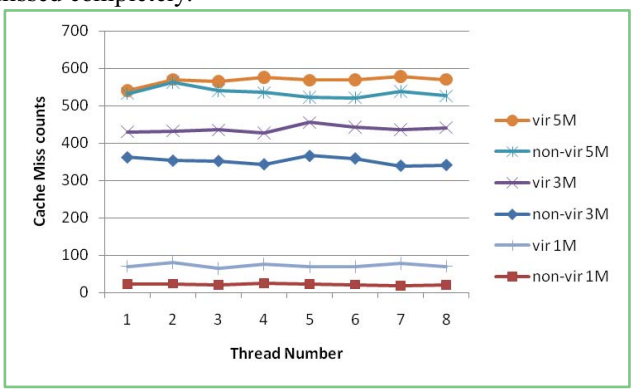

Figure 1. L2 Cache misses on both non-virtualization and virtualization

To bridge the semantic gap between Guest OS and VMM, our new scheduler, affinity-aware DP-Scheduling, introduces an affinity-aware method and provides the task affinity information to VMM. Then, it brings up a novel scheduling, DP-Scheduling, for implementing that VCPU can be pinned or unpinned on one CPU's running queue dynamically.

3. CPU Affinity-aware Method

3.1 Timing Control

In the x86 architecture, the structure of virtual main memory is a two-level Page Table (PT). Each process is assigned to a page table. Page table is a tree, whose root is a page of 4KBytes, called the Page Directory (PD). Each page directory entry points to 4KBytes pages respectively. When a processor switches memory address, operating system will notify the Memory Management Unit (MMU) by writing this address to the control register CR3. Meanwhile, if no such item in Translation Lookaside Buffer (TLB), the processor will also write this address into TLB.

In virtualization, Guest OS has no such authority for this form of write operations, so hypercall is invoked to inform VMM. We do not care the content of CR3 wrote by VMM, but the time when the content is to be changed. This means a new process will be run, which is the best opportunity for capturing the process information. Therefore, the scheduler can acquaint with the process scheduled immediately, ensuring the perception in a real-time. If the task was run before, it is the best time to make statistics. If not, the hard affinity information (detailed in Section 3.2) can also be captured.

3.2 Methodology for Capture

Virtualization has brought the concept of Pseudo-physical Address (PA), regarded as the "continuous" memory address by Guest OS. Virtual Address (VA) provided for applications in Guest OS can be mapped to PA. The real continuous memory, called Machine Address (MA), is divided into several parts for Guest OS by VMM. There are two tables for translation between MA and PA: Machine-to-Physical Translation Table (M2P) and Physical-to-Machine Translation Table (P2M). This makes the convenient address translation between Guest OS and VMM. The page table virtualization, such as MMU para-virtualization and shadow page, can implement the translation between VA and MA.

Firstly, we get VA from the register ESP, and compute VA of the kernel stack pointer. As the pointer of the process descriptor lies in the bottom of the kernel stack, we can get the VA of the process descriptor. Then, the VA of affinity information can be captured easily according to the offset. Finally, we translate the VA to MA, and convert the content of MA to the kind that we want.

We propose the Affinity Coefficient (AC) to quantify CPU Affinity. If the tasks are pinned on VCPU by users from Guest OS, the value of AC will be the highest. We can capture this information by reading the affinity bits called hard affinity in task struct. In addition, the value of AC will be increased by the operation of tasks in a period. Times of memory read/write can make AC rise oppositely to the counts of I/O access. We make a region for the value of AC. It can be set from 0 to 100. The initial value is 50, which can be reduced when I/O access happens and added when reading or writing memory. Once the hard affinity is detected, it will be set to 100.

3.3 API: Bridging the Semantic Gap

We now get the real-time CPU affinity information by the methods mentioned in Section 3.1 and 3.2. In order to inform scheduler timely, we provide an API, which can be called during scheduling. First, a new struct, Process Information (PI), is produced. We then add a member of PI to the VCPU struct. The information captured, AC for instance, will be written to PI. The API is responsible for informing information about CPU affinity to DP-scheduler. Every time the API called, PI will be refreshed to the VCPU corresponding.

4. The Design of Scheduler

4.1 Scheduling Framework

As Process Information (PI) is the premise of scheduling, the scheduler should call the API to capture the CPU affinity of the next running task. In Figure 2, every time the content of CR3 changes, PI Manager will catch the virtual address of PI. The part of affinity-aware detector is responsible for acquiring the CPU affinity information by translating VA to MA, detailed in Section 3. In the scheduling period, VCPU monitor first provides VCPU information modified by API to scheduler. Then, DP-Scheduling decides whether this VCPU needs pinning.

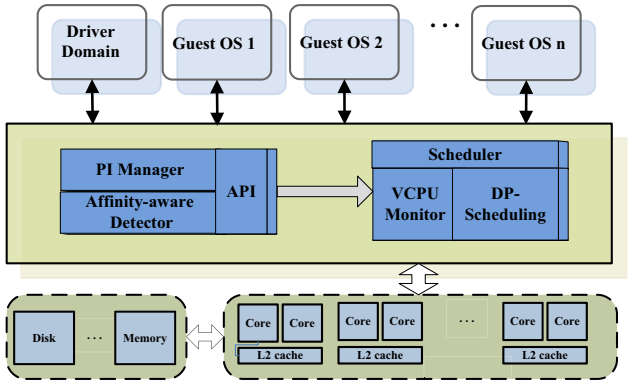

Figure 2. **The architecture of scheduling framework**

4.2 DP-Scheduling Model

In this section, we prove that DP-Scheduling outperforms Credit scheduling by formalizing the model. The theoretical result also shows the necessity of DP-Scheduling.

First, we make the following assumptions.

a). The total number of VCPUs from domains is more than CPU . Actually, it is the most common scenarios.

b).The number of VCPU pinned by DP-Scheduling is less than CPU at any time. If not, two or more VCPUs can be pinned on one same CPU in some cases. Cache miss will happen repetitively according to the principle of rotation, which leads to bad efficiency

$V_i = \{V_{i1}, V_{i2}, \ldots, V_{i|V_i|}\}$ denotes the set of VCPU from CPU_i, where $|V_i|$ stands for the number of VCPUs. In order to keep load balance, two cases may happen for CPU_i: A VCPU from V_i, called $VCPU\_MI_{ip}$, may migrate to other CPU; A VCPU from V_j, called $VCPU\_MI_{jq}$, may be migrated to CPU_i. Therefore, if all of VCPU in V_i have a time slice to execute within a period of time T, the overhead is as follow:

$$\Gamma_i = |V_i| * \text{Slot} + \sum_{j,q} (\text{VCPU}_{\text{MI}_{jq}} * C_{jq}) + \sum_{j,q} \text{VCPU}_{\text{MI}_{jp}} * \text{Slot} - \sum_{p} \text{VCPU}_{\text{MI}_{ip}} * \text{Slot} \quad \text{(i)}$$

We define Slot as one time slice. $C_j = \{C_{j1}, C_{j2}, \ldots, C_{j|V_j|}\}$ denotes the cost of migrating VCPU_MI_{jq} to CPU_i, such as cache failure, etc. j can be the index of all CPUs except i. $\text{CPU} = \{\text{CPU}_1, \text{CPU}_2, \ldots, \text{CPU}_{|\text{CPU}|}\}$ indicates the set of CPU, where |CPU| is the number of CPU. The set of $\sum_p \text{VCPU}_{\text{MI}_{ip}}$ will be executed on other CPUs. This is, $\forall x, p,\ \text{CPU}_x \in \text{CPU}, \text{VCPU}_{xp} \in V_x, \exists y, q, \text{CPU}_y \in \text{CPU}, \text{VCPU}_{yq} \in V_y, \text{VCPU}_{\text{MI}_{xp}} * \text{Slot} = \text{VCPU}_{\text{MI}_{yq}} * \text{Slot}$. This presents neutralization. Therefore, the overhead of all CPUs in Credit Scheduler within the period of time T is :

$$\Gamma^c = \sum_i \Gamma_i = \sum_i |V_i| * \text{Slot} + N^c * AVG^c(C) \quad \text{(ii)}$$

, where

$$N^c * AVG^c(C) = \sum_i \sum_{j,q} (\text{VCPU}_{\text{MI}_{jq}} * C_{jq}) \quad \text{(iii)}$$

N^c denotes the times of migration in T, and $AVG^c(C)$ is the average cost of migration.

Also, the overhead in DP-Scheduling in the same situation is:

$$\Gamma^D = \sum_i |V_i| * \text{Slot} + N^D * AVG^D(C) + C(\text{CPU}^{\text{idle}}) \quad \text{(iv)}$$

N^D denotes the times of migration in T, and $AVG^D(C)$ is the average cost of migration in DP-Scheduling. As DP-Scheduling pins VCPU to CPU in some cases (Section 4.3), $\text{CPU}^{\text{idle}} = \{\text{CPU}_1^{\text{idle}}, \text{CPU}_2^{\text{idle}}, \ldots, \text{CPU}_{|\text{CPU}|}^{\text{idle}}\}$ indicates the cost of CPUs going to IDLE when load balance is broken. $C(CPU^{idle})$ stands for the time wasted by IDLE CPUs when more than one VCPU are waiting in other CPUs.

Therefore, the performance comparison between Credit and DP-Scheduling is:

$$\theta = \Gamma^c - \Gamma^D = N^c * AVG^c(C) - N^D * AVG^D(C) - C(CPU^{idle}) \quad \text{(v)}$$

As DP-Scheduling avoids migration in some cases, the migration is less than Credit Scheduling. That is, $N^c > N^D$. Meanwhile, DP-Scheduling pins the VCPU whose migration can result in serious cache failure. This cost is over the average. That is, $AVG^c(C) > AVG^D(C)$. As assumption (b) indicates the number of VCPUs pinned is less than CPUs and assumption (a) indicated the VCPU number is more than CPU, there always exists one or more VCPU(s) for migrating when a CPU has no BOOST and UNDER VCPUs except for no runnable VCPU in other CPUs. Therefore, $C(CPU^{idle}) = 0$. Now, we prove that:

$$\theta > 0 \quad \text{(vi)}$$

Therefore, the theoretical result presents that DP-Scheduling has lower overhead than Credit. Also, with the increasing of difference between $AVG^c(C)$ and $AVG^D(C)$, the performance gap between the two schedulers will be further enlarged. Besides, the result shows that a new scheduler reducing VCPUs' migrating cost is needed.

4.3 Basic Scheduling: S'

We present the basic Scheduling, S', for implementing dynamic pinning. S' first calls the API for capturing the task affinity information from the VCPU next to be run. Also, we make a threshold of AC (Affinity Coefficient) for deciding the degree of affinity information. Once AC is over this threshold, S' will make the VCPU corresponding non-migrating, VCPU Y.1 in Figure 3 (B) for instance. Also, S' will unpin a VCPU pinned before if the degree of affinity information is not enough, which means migration will not bring too many cache misses.

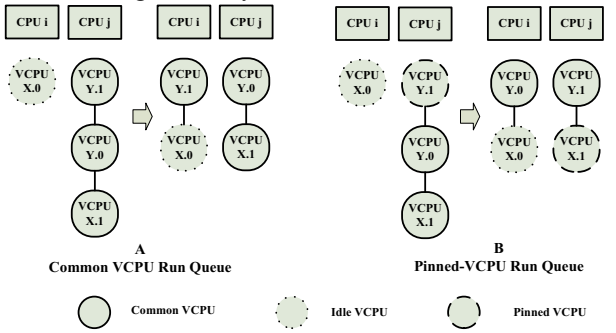

Figure 3. **Comparison between common and pinned VCPU run queue.**

We compare the situation of VCPU migration between Credit scheduler and S' as shown in Figure 3.There is no runnable VCPU in the local CPU run queue when VCPU X.0 gets back to the state of Idle. For keeping the load balance, Credit scheduler (A) migrates VCPU Y.1 from the head of CPU j's running queue, whereas S' (B) migrates VCPU Y.0 for VCPU Y.1 was pinned. As VCPU Y.1 is to run a task with high CPU affinity, S' will avoid cache miss, especially when CPU i and CPU j are not in one CPU socket.

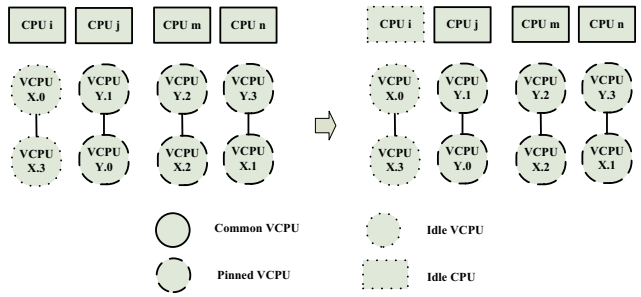

Figure 4. **Situation of load balance broken**

However, if too more VCPUs is pinned at one time, the starvation of some CPUs may occur as shown in Figure 4. As CPU i has no runnable VCPU, VCPUs from other CPUs cannot be migrated. CPU i has to turn to IDLE in spite of some VCPUs probably waiting in other CPU run queue. Therefore, load balance on SMP is broken, which may result in lower CPU utilization. S' solves this problem by making a policy: VCPUs pinned at one time must be less

than CPU, which is also our assumption (b) in theory model. If the number of VCPU pinned is $|CPU| - 1$, VCPU will be not pinned even when high CPU affinity tasks runs. Another reason for this policy is that two high affinity tasks taking turns to execute will bring serious cache misses. However, it is not sufficient, but just an essential condition. This is why the basic scheduler S' needs modification as below.

4.4 DP-Scheduling Algorithm

The basic Scheduling, S', mainly accomplishes the dynamic pinning, which makes the migration sequence of VCPU change. Although it can avoid some negative effects as shown in Figure 4, load balance cannot be ensured. Besides, the case in Figure 5 shows that VCPU Y.1 and VCPU X.0 will take turns to execute, leading to serious cache failure. So, the measure we adopt should pin VCPU dispersedly.

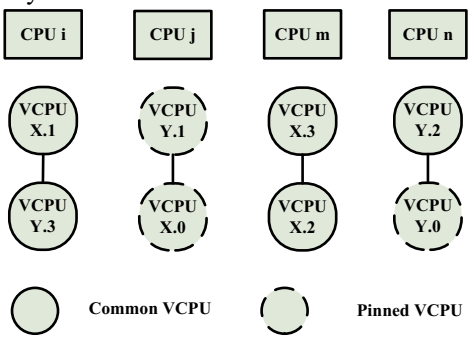

Figure 5. The defect of S'

Therefore, we propose a series of strategies as follow:

(a).pin VCPU to the CPU with no pinned VCPU at this time.

(b).pin VCPU to the CPU with lower workload.

(c).pin VCPU to the local CPU if both (i) and (ii) do not happen.

(d).do not migrate the VCPU actively when it is unpinned.

(e).unpin the VCPU with the lowest value of AC when the number of CPU equals $|CPU|$.

(f).unpin the VCPU pinned before if it goes to the state of OVER or IDLE.

(a), (b), (c), and (e) are used for keeping load balance, which eradicates the case that a CPU goes to IDLE when many runnable VCPUs are still waiting. (d) aims at reducing the cost of unnecessary migration. We apply (f) to reduce the effects on VCPUs need pinning following up for the total pinning number as shown in (e) cannot be too much. Besides, this kind of VCPUs can also be pinned according to its value of AC, when they get back to UNDER or BOOST.

We implement (b) by computing the average running time of CPUs respectively in the recent periods. VCPU will be pinned to the CPU with the lowest value, which means this CPU goes to the state of IDLE more often. Other methods in estimating CPU load can also be available: average or moving average of the amount of one CPU's running time, for instance. The policy we selected works well.

Let Φ stand for VCPU set from all domains, that is, $\Phi = \{VCPU_1, VCPU_2, \dots, VCPU_{|\Phi|}\}$. Let φ_p be the set of VCPUs that were pinned, whereas φ_{up} is the set of unpinning VCPUs. Hence, $\Phi = \varphi_p \cup \varphi_{up}$, and $|\Phi| = |\varphi_p| + |\varphi_{up}|$. Besides, let $\Upsilon = \{\gamma_1, \gamma_2, \dots, \gamma_{|\Phi|}\}$, where γ_i is the value of AC for $VCPU_i$.

```
For each VCPU_i ∈ Φ {
  CPU_x = VCPU_i ->processor;
  /*VCPU_i has been pinned by its high value of AC*/
  If (γ_i ≥ THRESHOLD and VCPU_i ∈ φ_p) Return;
  /*VCPU_i with its high value of AC was not pinned */
  Else If (γ_i ≥ THRESHOLD and VCPU_i ∈ φ_up
  and VCPU_i ->pri > OVER){
  peer_cpu = select_peer(VCPU_i);
  set_unmigrateable(peer_cpu, vcpu);
  del(φ_up, VCPU_i);
  add(φ_p, VCPU_i);
  }
  /*Pinned VCPU_i contains low value of AC currently
  *according to strategy (d)*/
  Else If (γ_i < THRESHOLD and VCPU_i ∈ φ_p){
  set_migrateable(CPU_x, VCPU_i);
  del(φ_p, VCPU_i);
  add(φ_up, VCPU_i);
  }
  /*The total number of VCPUs should not over
  *CPU number -1*/
  If (|φ_p| == |CPU|)
  set_migrateable_by_ac(φ_p, φ_up);
}
```

In the pseudo-code above, select_peer is a function to achieve (a), (b) and (c), and set_migrateable_by_ac is to achieve (e). set_unmigrateable and set_migrateable implement pinning and unpinning vcpu dynamicly. Threshold is set to 75 through repeated experiments.

DP-Scheduling scheduler is based on Xen VMM, which provides an interface to schedulers. There is a structure including pointers to functions that is used to implement the scheduler [16]. We add DP-Scheduling to the scheduler set by defining the sched_dps_def as follows.,

```
struct scheduler sched_dps_def = {
.name = "DP-Scheduling Scheduler",
```

```
.opt_name = "dp-scheduling",
.sched_id = XEN_SCHEDULER_DPS,
…
.do_schedule = dpssched_schedule.
…
}
```

do_schedule is responsible for scheduling VCPUs. The codes we wrote are just in do_schedule. Also, we modify the function do_domctl (in domctl.c) for adding the DP-Scheduling strategies.

5. Performance Evaluation

5.1 Experimental Platform

Our experimental hardware platform is the IBM Bladecenter HS21, with two quad-core of Intel Xeon 5405 at 2.0GHZ. Every two cores share 6MB L2 cache. The server has 4G of RAM.

The server runs the CentOS-5.4 in Domain 0 with Xen 3.4.3 Hypervisor[1]. The Guest OS in Domain U is also CentOS-5.4. Each DomainU is allocated 8 VCPUs, 1G RAM, 8GB Disk and adopts the bridge network interface for the interconnection with Dom0 and DomainU.

5.2 Benchmarks

We select three different types of current mainstream benchmarks listed in Table 1 to ensure the experiment results accuracy and convincing. In addition, the MPI library used is mpich2-1.0 [10].

Table 1. Benchmarks Overview

Benchmark Category	Code Name	Variable	Measurement
HPCC	STREAM	Array size	Memory Bandwidth
EPCC	OpenMP Micro-benchmark suite	Thread Number	Time
IMB	Sendrecv	Message Size	Transfer Speed
	Exchange		

High Performance Computing Challenge (HPCC) [7] is a standard set of testing system performance in kinds of aspects, including CPU speed, network bandwidth, memory read speed, and network delay, etc. We use STREAM benchmark to test the speed of memory read. The results are under different conditions by the size of arrays.

Edinburgh Parallel Computing Centre (EPCC) OpenMP micro-benchmarks [8] are to test the synchronization and the operations on cycles as well as arrays. It contains a series of OpenMP test programs, such as testcrit, testlock, testorder, etc. We integrate them together to test the overall performance.

Intel MPI benchmark (IMB) [9] is a set used to test some important MPI functions. In our experiment, we use Sendrecv and Exchange in IMB to evaluate the transfer speed with different schedulers.

5.3 Results and Analysis

We compare the Credit scheduler with DP-Scheduling scheduler in our experiments. We will run every benchmark in VM with each scheduler respectively on our platform, where other configurations are all the same. Firstly, we run scripts at least 1000 times in each case though some benchmarks are also executed repeatedly. We then compute the average value except for some few abnormal points. The results are as follows.

We run 8 OpenMP threads of STEAM with different array sizes to fetch the memory bandwidth by adopting Credit scheduler and DP-Scheduling scheduler respectively. Each thread works with non-synchronization. The result is in Figure 6. When the size of arrays is small, total arrays can be cached in L2 cache. As DP-Scheduling pins the VCPU running this task to CPUs by strategies proposed in Section 4.4, it largely guarantees the L2 cache hits. However, Credit Scheduler migrates VCPUs among CPUs frequently for keeping load balance, which leads to L2 cache updated constantly. With the increase of arrays size, the two results become close. But the performance of DP-Scheduling scheduler is still not below the one of Credit Scheduler as it also has the load balancing policy.

OpenMP micro-benchmark contains a suite of high synchronous tasks, where the contention on the critical resource among threads will happen frequently. As shown in Figure 7, the performance gap between the two schedulers is growing with the increase of threads' number. On one time, a blind migration can also result in terrible L2 cache failure. On the other time, DP-Scheduling scheduler reduces the time of waiting for locks obtained by other thread for it makes VCPUs in warm CPU affinity pin on different CPU run queues. This will be more obvious when the number of threads is closed to the total number of VCPU in VM.

Sendrecv and Exchange, the parallel transfer benchmarks in IMB, are two different kinds of testing the transfer speed among threads. Any thread can receive messages from its left thread and send messages to its right neighbor in Sendrecv, whereas Exchange makes all threads send and receive messages from both sides. We create 8 threads as same as VCPU number in VM for both Sendrecv and Exchange, as shown in Figure 8 and Figure 9. When Message size is too small, the bottleneck lies in the sending and receiving message for too many messages are waiting.

When the message size is too large, the size of data buffer pools is the main factor that determines the transfer speed. DP-Scheduling scheduler significantly improves the

[1] The reason we do not choose newest version, Xen 4.0.0, is that no stable Xenoprof patch is supported until now. Besides, the default scheduler in Xen 3.3.x is as same as in Xen 4.0.0.

speed in the range of 128 Bytes to 131072 Bytes, where the CPU speed plays a decisive role. For Credit scheduler, operations on sending and receiving make VCPUs switch frequently if two VCPUs running the neighbor threads lie in same CPU run queue, while DP-Scheduling scheduler can guarantee the real-time interaction among threads as well as load balance.

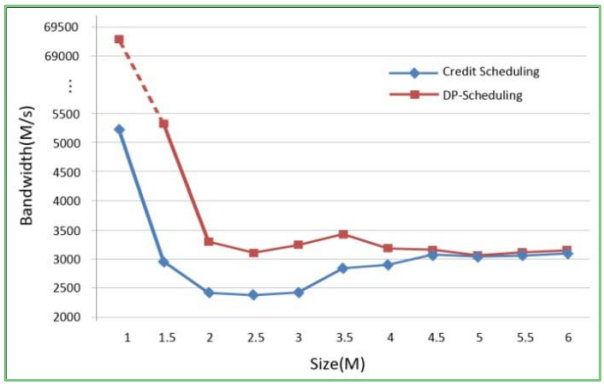

Figure 6. **Bandwidth in STREAM benchmark**

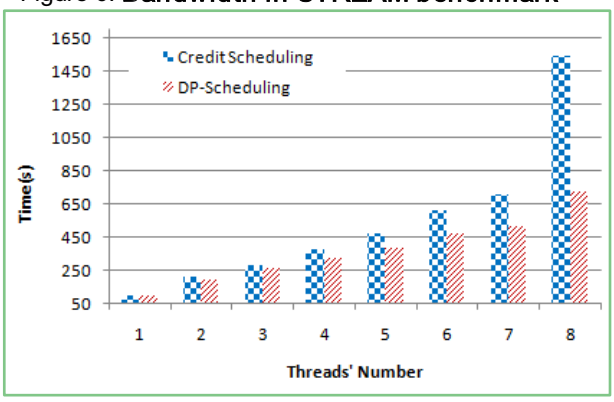

Figure 7. **Running time of OpenMP Micro-benchmarks**

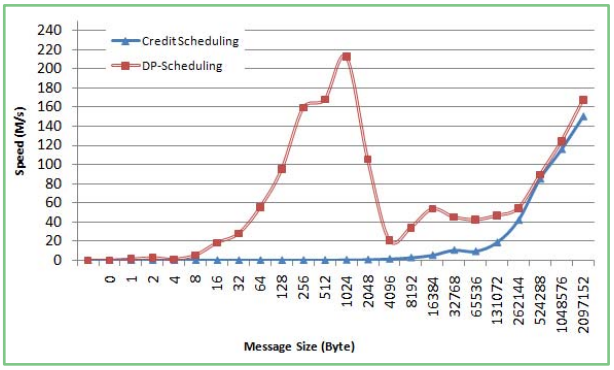

Figure 8. **Sendrecv of IMB benchmark**

Our benchmarks have a wide coverage among CPU-bound applications. STREAM is a typical example of asynchronous multi-thread, Openmp test suite has authority in high synchronization, and the two benchmarks from IMB are common use in parallel transfer. All in all, the experimental results indicate that DP-Scheduling strategy performs well in numerous CPU-bound tasks. Either synchronous or asynchronous, DP-Scheduling can effectively increase CPU utilization by reducing unnecessary migration.

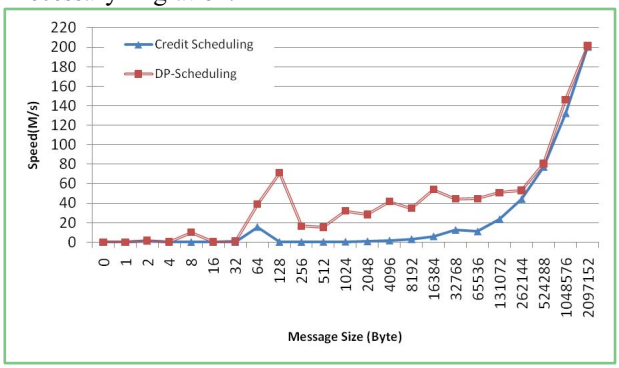

Figure 9. **Exchange of IMB benchmark**

6. Related Work

CPU scheduling policy plays a crucial role in accelerating the performance of VM in virtualization. The traditional schedulers, such as Credit scheduler and SEDF, are universal for all kinds of applications. [21] evaluated the two schedulers with different configurations. In future, Credit 2 [22] will be added to the Xen tree. It will focus on the fairness among VM, working well for latency-sensitive workloads, hyperthreads, and cores power.

There are several classifications for applications, such as single-thread/multi-threads, synchronization/asynchroniza-tion, etc. One significant way is to classify them by the operations both on CPU and I/O, that is, CPU-bound and I/O-bound applications. Therefore, many schedulers based on them aim at improving such special applications. The work in [23] proposed an efficient I/O virtualization for high end systems, by offloading the virtualization functionality from the Guest OS onto device. [1] proposed a communication-aware CPU scheduler in Xen by modifying the SEDF scheduler to improve the throughput for I/O-bound applications. [24] provided a policy, called VMM-bypass I/O, for improving I/O performance. It reduces the I/O response time by carrying I/O operations from VMM to Guest OS. [6] provided a task-aware virtual machine scheduling mechanism, where partial boosting is introduced to thin the task-level granularity. [20] presented a task-aware based co-scheduling scheduler to meet the need of concurrent application in virtual SMP system. A hybrid scheduling framework [3] was presented to reduce the CPU time when the system workload was the concurrent applications. [2] made Xen friendlier to the soft real-time tasks by modifying Credit scheduler, which makes soft real-time tasks run well in non-preemptive scheduler. This method provides a novel thinking in supporting soft real-time tasks in VMM.

There are also some performance monitors [26, 27, 28],

which configure the schedulers to reach high performance by the information captured. [4] analyzed the performance of a server consolidation benchmark on a multi-core platform of Xen virtual environment. CPI (Cycles Per Instruction) and L2 cache misses per instruction are studied to prove the advantage of larger caches.

7. Conclusion

In this paper, we develop a novel approach for improving the utilization of CPU resource by reducing both cache miss for CPU-bound tasks and waiting time for synchronous threads. The solution lies in the hypervisor scheduler. Firstly, we compare the cache miss between non-virtualization and virtualization, and do analysis on it. Then, we propose a method of capturing CPU affinity to bridge the gap between Guest OS and VMM, and provide an API for the new scheduler, called DP-Scheduling scheduler based on Xen Credit Scheduler. Depending on the information from API, DP-Scheduling scheduler will pin or unpin the VCPUs from CPU dynamically as well as keeping load balance. We apply a series of strategies for this scheduler in order to make VCPU run in CPU dispersedly, which can bring high cache hits. The experiment designed contains plenty kinds of CPU-bound applications. And the results demonstrate that affinity-aware DP-Scheduling scheduler can promote the performance of the virtual machine system efficiently.

8. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation of China under Grant No. 61073076, and the National High Technology Development 863 Program of China under Grant No. 2007AA01Z118.

9. References

[1] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and Co.: Communication-aware cpu scheduling for consolidated Xen-based hosting platforms. *Proceedings of the 3rd international conference on Virtual execution environments (VEE)*, 2007, pages 126-136

[2] Min Lee, A. S. Krishnakumar, P. Krishnan, Navjot Singh, Shalini Yajnik. Supporting Soft Real-Time Tasks in the Xen Hypervisor. *VEE'09*, 2010, pages 97-108.

[3] Chuliang Weng, Zhigang Wang, Minglu Li, and Xinda Lu.The Hybrid Scheduling Framework for Virtual Machine Systems. *VEE'09*, 2009, pages 111-120.

[4] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T.Adelmeyer. Characterization & analysis of a server consolidation benchmark.*VEE '08*, 2008, pages 21-30.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *SOSP'2003*, 2003, pages 164-177.

[6] Hwanju Kim Hyeontaek Lim Jinkyu Jeong Heeseung Jo Joowon Lee1. Task-aware Virtual Machine Scheduling for I/O Performance. *VEE '09*, 2009, pages 101-110.

[7] http://icl.cs.utk.edu/hpcc/.

[8] http://www.epcc.ed.ac.uk/research/openmpbench.

[9] Intel Corporation Document. Intel MPI Benchmarks: User Guide and Methodology Description.

[10] MPICH2, ttp://www.mcs.anl.gov/research/projects/mpich2/.

[11] Amazon Elastic Compute Cloud (EC2). ws.amazon.com/ec2.

[12] A Gavrilovska, S Kumar, H Raj, K Schwan, V Gupta, R Nathuji, R Niranjan, A Ranadive, P Saraiya. High-Performance Hypervisor Architectures:Virtualization in HPC Systems. *HPCVirt'07*, 2007.

[13] Adit Ranadive, Mukil Kesavan, Ada Gavrilovska, Karsten Schwan. Performance Implications of Virtualizing Multicore Cluster Machines. *HPCVirt'08*, 2008.

[14] Darren Abramson, Jeff Jackson. Intel Virtualization Technology for Directed I/O. Intel Technology Journal. 2006.

[15] AMD Virtualization Website: Introducing AMD virtualization, 2006. http://www.amd.com/virtualization.

[16] David Chisnall. The Definitive Guide to the Xen Hypervisor. Prentice hall, 2007.

[17] S. T. Jones. Implicit operating system awareness in a virtual machine monitor. PhD thesis, Madison, WI, USA, 2007.

[18] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. *Proc. USENIX Annual Technical Conference*, 2006.

[19] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Geiger:Monitoring the buffer cache in a virtual machine environment. *Proc. ASPLOS-XII,* 2006, pages 14-24.

[20] Yuebin Bai, Cong Xu, Zhi Li. Task-aware based co-scheduling for virtual machine system. *The 26th Symposium On Applied Computing*, 2010, pages 181-188.

[21] D. Ongaro, A. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. *VEE'08*, 2008, pages 1-10.

[22] George W. Dunlap. Scheduler development update.

[23] G. Liao, D. Guo, L. Bhuyan, and S. R. King. Software techniques to improve virtualized IO performance on multi-core. *ACM/IEEE Symposium on Architectures for Networking and Communications Systems,* 2008, pages 161-170.

[24] J. Liu, W. Huang, B. Abali, and D. Panda. High performance vmm-bypass I/O in virtual machines. *Proceedings of USENIX '06 Annual Technical Conference*, 2006.

[25] Himanshu Raj, Karsten Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. *HPDC'07*, 2007, pages 179-188.

[26] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. *Proceedings of the USENIX Annual Technical Conference*, 2005.

[27] A. Menon, J. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing performance: Overheads in the Xen virtual machine environment. *VEE'05*, 2005, pages 13-23.

[28] D. Gupta, R. Gardner, and L. Cherkasovah. XenMon: QoSmonitoring and performance profiling tool. Technical Report HPL-2005-187, HP Labs, 2005

[29] http://www.vmware.com/products/esx/