International Journal of Grid Distribution Computing
Vol.7, no.3 (2014), pp.123-134
http://dx.doi.org/10.14257/ijgdc.2014.7.3.12

An Energy-efficient Task Scheduler in Virtualized Cloud Platforms

Dongbo Liu and Ning Han

School of Computer and Communication, Hunan Institute of Engineering
liudongbo74@126.com

Abstract

In cloud platforms, virtualization technology has been widely applied for deploying large-scale IT-infrastructures due to its flexibility and extendibility. However, the extra software layer introduced by virtualization technology also raises many performance issues. One of them is the energy-efficiency losses when massive I/O-intensive tasks are running on virtualized servers. In this paper, we present a novel virtual machine scheduling approach, which the scheduler allows virtual machines to obtain extra CPU shares if they were frequently blocked by I/O interrupted recently. In this way, I/O-intensive tasks will have more chances of being scheduled so as to compensate their performance losses caused by I/O operations. Extensive experiments are conducted by using various benchmarks, and the results show that the proposed policy outperforms existing scheduling algorithm in the term of energy-efficiency especially when the virtualized system is in presence of intensive mixed-workloads.

Keywords: *Cloud Computing, Resource Virtualization, Task Scheduling, Virtual Machine*

1. Introduction

With the increasing number of large-scale datacenters, the energy consumption by operating them also dramatically increased in the past few years [1, 2]. Therefore, various solutions have been suggested to reduce the need for server hardware and energy consumption. Nowadays, service virtualization and consolidation, for example, are widely applied to datacenters, and also cloud computing has been introduced as an inherently energy-efficient datacenter architecture [3].

Generally speaking, virtualization technology provides two mechanisms (server consolidation and VM live-migration) which have been commonly considered as effective mechanisms for reducing the energy consumption in high-performance datacenters [5]. However, it also raises many challenging issues that need to be addressed. The most mentioned issue on virtualization technology is the performance degradation caused by I/O virtualization. Recently, many efforts have been taken into improving the performance of I/O virtualization [6-8]. However, most of these works focus on improving the performance-related metrics, *i.e.*, I/O latency, application responsive time, system throughput and *etc.*, while few of them take into consideration the energy-efficiency of I/O virtualization. As a result, some of the proposed techniques are very effective for improving I/O performance while ignoring the energy-efficiency losses caused by I/O virtualization.

In this work, we concentrate on improving the energy-efficiency of I/O virtualization. Before presenting any energy-efficiency polices for I/O virtualization, we need to know the key factors that influence the energy-efficiency of I/O virtualization. According to the existing studies, we summarize them as following:

ISSN: 2005-4262 IJGDC

- In cloud systems, workloads with different characteristics are often consolidated on a single server. As a result, an approach aiming at improving CPU energy-efficiency usually can not be adopted to improve I/O energy-efficiency [6, 9, 10].
- Traditional virtual machine (VM) schedulers focus on sharing CPU fairly among multiple VMs, while leaving the scheduling of I/O resources as a secondary concern. As a result, the I/O energy-efficiency in a virtualized system is often far from desirable [6, 11].
- When in presence of I/O-intensive workloads, the context-switching rate will be significantly increased, which in turn results in degradation of energy-efficiency for both CPU and I/O devices [7, 8, 12].

According to the above summary, it is clear that VM scheduling policy plays an important role for energy-efficiency in a virtualized system. Any energy-efficiency VM scheduling policies should take into account the characteristics of workloads, especially when both CPU-intensive and I/O-intensive applications are coexisting. In this work, we present a novel VM scheduling policy which is aiming at reducing the energy-efficiency losses caused by I/O virtualization mechanism, especially when the virtualization platform is in presence of intensive mixed-workloads.

The rest of this paper is organized as following. In Section 2, the related works are presented. In Section 3, we describe the background and motivation of our study. In Section 4, we present the scheduling policy and its implementation. In Section 5, experiments are performed and the results are carefully investigated and evaluated in terms of various metrics. Finally, Section 6 concludes the paper with a brief discussion of our future work.

2. Related Work

In the past few years, I/O scheduling in virtualization platform is one of the most mentioned topics, and plenty of techniques have been proposed accordingly. For example, Ongaro, *et al.*, studied the impacts of VM scheduler on I/O performance, and their experimental results indicated that the major reasons that lead to high I/O latency are hybrid workload [6]; In [13], the authors presented a task-aware VM scheduling mechanism for improving the responsiveness of I/O-intensive tasks; In [13], the authors proposed a multi-core I/O scheduling framework, in which processor cores are divided into three subsets with each employing different scheduling strategies for different workloads.

Recently, many researchers have noticed that a VM scheduler should be adaptive to workloads with various characteristics. For example, Weng, *et al.*, presented a hybrid scheduling framework, in which high-throughput and concurrent workloads are accommodated by different types of VM instances respectively [14]. In [15], the authors presented an implementation of group scheduler, namely MapReduce Group Scheduler (MRG), which has be incorporated with two mechanisms to improve the efficiency and fairness of the existing VM scheduler when scheduling I/O-intensive applications. In [16], the authors addressed the issue of scheduling latency-sensitive VMs and presented a vCPU scheduling policy called vSlicer.

As energy-efficiency problem become more and more important, many energy-efficient techniques have been proposed. For example, in [17], Koller, *et al.*, proposed an application-oriented power meter framework called WattApp, which uses simple linear functions to predict the power consumption of given applications. To reduce the energy consumption caused by resource over-provision, Rodero, *et al.*, proposed an integrated framework which uses clustering approach to implement a just-right VM provision mechanism [18]. In [19], the authors presented two energy-conscious task consolidation algorithms, called ECTC and

MaxUtil, which aim to maximize resource utilization without any performance degradation.

3. Research Background and Motivation

3.1. Virtual Machine Scheduling Framework

In virtualized platforms, CPU scheduling is implemented as VM scheduler which is responsible for assigning VM instances to physical processors in a time-sharing manner. To maximize the performance, the following issues should be taken into account when designing a VM scheduler: fairness, load balancing, CPU utilization, I/O performance, throughput and *etc*. According to these designing principles, a VM scheduler can be characterized by three features: *Proportional-Share* or *Fair-Share*, *Work-Conserving* or *Non-Work-Conserving*, and *Preemptive* or *Non-Preemptive*. Currently, the most mentioned VM scheduling policies includes Simple Earliest Deadline First (SEDF), Borrowed Virtual Time (BVT), and Credit Scheduling (CS).

In all these VM schedulers, they split the CPU capacity into a set of time slots as the basic scheduling units, and then assign these time slots to VMs according their own designing objectives as mentioned above. However, such a fixed-length time slot raises a challenging issue, that is, how long is the optimal length of time slot? A longer time slot is effective to improve the performance of computation-intensive workloads, while results in poor performance for I/O-intensive workloads; a shorter time slot can provide fine-grained scheduling results, but also lead to frequent context-switches which significantly degrades the energy-efficiency of a virtualized server.

Taking CS scheduler as an example, a fair-share of CPU can be guaranteed by this scheduler when multiple VMs run simultaneously; however, the CPU accessing latency tends to be a multiple of the default CPU time slice for each VM (*e.g.*, 30ms in Xen). As more and more VMs share the same CPU, the accessing latency experienced by each VM increases substantially. In addition, CS scheduler only attempts to fairly allocate processor resources among VMs in a long-term manner. As a result, it will result in unpredictable I/O latency among VM instances, because the I/O latency is determined by the position of a VM in the running queue. If a VM is close to the head of the queue, its I/O latency will be lower, otherwise it will be higher. In the worst case, if a VM is the only one that with I/O-intensive feature, it will frequently suffer from long I/O latency during its life cycle.

3.2. I/O Scheduling Framework

Currently, there are four I/O device virtualization approaches: VMM pass-through, device emulation, self-virtualized hardware, and parameter virtualization. Each method has corresponding advantages and disadvantages. Basically, the first three approaches are transparent to guest OS, while the final one needs to change the I/O control flow in guest OS. When implementing I/O virtualization, I/O interrupts generated by physical devices are firstly forward to VM hypervisor which is responsible for translating I/O interrupts into virtual I/O events; then, a special component (we called it as I/O Proxy) sends these virtual I/O events to VM's Event Handler, which will de-multiplex the pending events to the target applications. Such an I/O flowchart can be implemented in different approaches. For example, the I/O Proxy component can be designed as an isolated VM instance like the Xen platform, or it can be embedded into each VM instance which is often called as VM-bypass. Comparing with non-virtualized systems, I/O virtualization will introduce significant extra performance costs, which can be simply demonstrated by Figure 1.

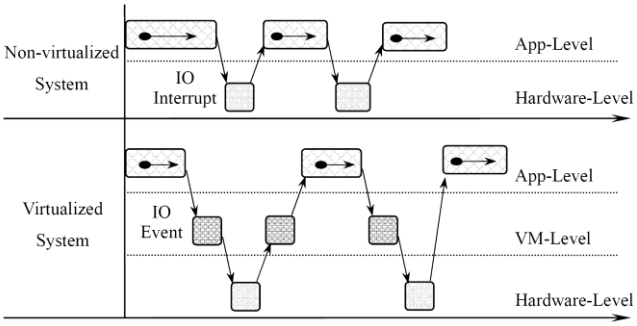


Figure 1. Comparison of I/O Handling in Non-Virtualized and Virtualization System

As shown in Figure 1, the reason of CPU accessing latency is that a VM with pending I/O events will have to wait for its turn to access CPU before processing its I/O events. The typical approach to reducing the I/O latency is to raise the priority of I/O-intensive VMs when I/O event is pending. For example, Xen's CS scheduler uses BOOST mechanism to shorten the I/O responsive time by temporarily boosting the I/O-intensive VMs. Such an approach works quite well when the workloads are pure I/O-intensive VMs. When in presence of mixed workloads, a VM just being scheduled by BOOST mechanism tends to suffer from longer scheduling latency for the rest of its scheduling rounds, since the VM scheduler has to maintain the proportional-fair principle.

In virtualized systems, I/O data need to be redundantly copied at virtualized level before sending to physical devices or applications. Unlike non-virtualized systems which often depends on OS for efficiently dealing with I/O buffer management, the I/O buffer mechanism in VM hypervisor is often more complicated and inefficient. For example, the Xen platform uses dom-0 to manage I/O buffers of all active VMs, as a result, dom-0 often becomes the I/O performance bottleneck when the number of I/O-intensive VMs is large. In addition, as the VM hypervisor should keep sufficient security for all VMs, when multiplexing/demultiplexing the data packets among guest VMs, dom-0 must have access to memory that belongs to different VMs, which will trigger cost-expensive memory-protection operations.

4. Scheduling Algorithm Design and Implementation

4.1. Definition and Problem Formulation

In a virtualized server, each VM instance can be characterized as $V_i = (w_i, d_i)$, where w_i is the proportional share of physical processor that assigned to V_i, d_i is its deadline. It is noteworthy that w_i is not the exact processor utilization obtained by V_i, because many factors have influences on VM scheduler which will make the exact processor utilization of V_i more/less than w_i. More importantly, the exact processor utilization of V_i heavily depends on the sampling time interval. Therefore, we note it as $u_i^{t_1 \to t_2}$ indicating the mean utilization of V_i during the time interval $[t_1, t_2]$. For a proper VM scheduler, it should try to keep $u_i^{t_1 \to t_2}$ close to w_i in long-term if it obeys the proportional-sharing principle. For each V_i, its j^{th} scheduling instance is noted as v_i^j, the start time of v_i^j is noted as a_i^j, the execution time of v_i^j is noted as c_i^j, and the deadline of v_i^j is noted as d_i^j.In most of existing VM schedulers,

the capability of CPU is divided into a series of time slots each with constant length noted as Δs.

For a CPU-intensive VM, its execution time in j^{th} scheduling instance often satisfies $c_i^j = d_i^j - a_i^j$. So, setting c_i^j as equal ad the length of time slot Δs is a good approach when scheduling multiple CPU-intensive VMs concurrently. However, for an I/O-intensive VM, its execution time in j^{th} scheduling instance often satisfies $c_i^j < d_i^j - a_i^j$, because any pending I/O event will trigger a context-switch so as to avoid CPU wasting. Therefore, constant length Δs is not suitable for scheduling I/O-intensive VMs. When in presence of mixed-workloads, how to configure the time slot length becomes a dilemma as mentioned in Section 2.1. The typical approach to dealing with this problem is using work-conserving mechanism, by which idling CPU can be assigned to any runnable VMs. However, it will significantly violate the proportional-sharing principle.

On the other side, if the scheduler aims at improving I/O responsiveness, it may adopt priority boosting mechanism, by which I/O-intensive VMs will obtain extra CPU shares when their I/O operations are finished. However, priority boosting mechanism might result in $u_i^{t_1 \to t_2} ? w_i$. More importantly, both mechanisms will introduce more context-switches, which inevitably result in some energy-efficiency losses. To deal with these problems, we design an energy-aware VM scheduling, namely *Share-Reclaiming Policy* which will be described in the next section.

4.2. Share-Reclaiming Scheduling Policy

To illustrate the idea of share-reclaiming mechanism, we present an example to show the potential improvements that can be obtained by taking advantages of those CPU shares donated by blocked VMs. Assuming that there are three VMs (v_1, v_2 and v_3) concurrently running on a virtualized server, where both v_1 and v_2 are I/O-intensive, while v_3 is CPU-intensive. When scheduling them by CS scheduler with work-conserving policy, the scheduling scheme may be as Figure 2.

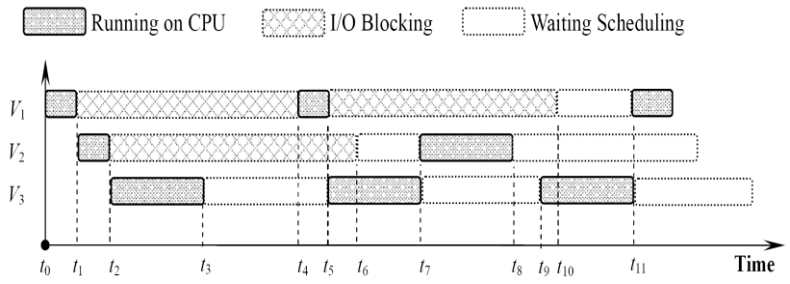


Figure 2. Scheduling Scheme by CS Policy with Work-Conserving Policy

Since we use the work-conserving policy, CS scheduler can assign CPU to any waiting VMs when the running VM are interrupted by I/O events. Therefore, it assigns CPU to v_2 and v_3 at time t_1 and t_2 accordingly. When v_3 runs out of its CPU share at t_3, CPU becomes idle until t_4 because both v_1 and v_2 are blocked by I/O waiting. During the interval $[t_8, t_9]$, CPU also be in idling state because both v_2 and v_3 are just running out their CPU shares, while v_1 is in I/O blocking state. In addition, we can see that v_2 has completed its I/O operation at time

t_6, however it has to be waiting during interval $[t_6, t_7]$ because V_3 is occupying the CPU. The same case occurs on V_1 during $[t_{10}, t_{11}]$. Such a waiting will have significant negative effects on the responsiveness of I/O-intensive applications.

When using share-reclaiming mechanism, all the unused CPU slices that donated by I/O blocked VMs are stored in the global queue *qSR*, and the scheduling scheme can be optimized as Figure 3.

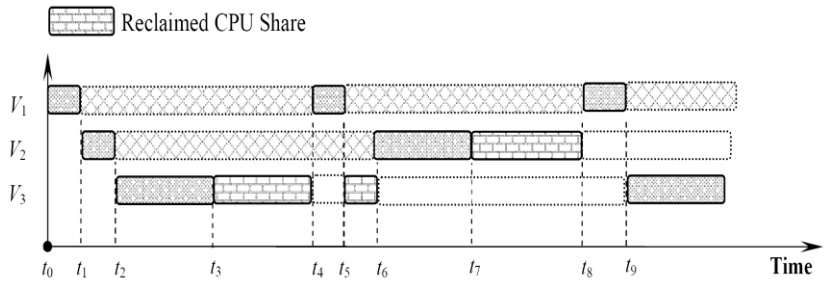


Figure 3. Scheduling scheme by share-reclaiming policy

So, we can use *qSR* to reclaim extra CPU shares to the currently running VM. Therefore, extra CPU shares are assigned to V_3 during $[t_3, t_4]$ and $[t_5, t_6]$, and such a reclaiming also happens on V_2 during $[t_7, t_8]$. In this way, share-reclaiming mechanism works like back-filling mechanism, which is very effective to increase the CPU utilization as well as its energy-efficiency. The key difference between share-reclaiming and work-conserving is that, when using work-conserving mechanism the assigned CPU-share is concrete and can not be preempted; while our share-reclaiming mechanism allows the scheduler preempts the VM when it is using the reclaimed CPU shares. Therefore, share-reclaiming mechanism will not postpone the responsiveness of I/O-intensive VMs. For instance, in Figure 2, V_1 and V_2 are postponed during time interval $[t_6, t_7]$ and $[t_{10}, t_{11}]$ respectively, because V_3 occupies the CPU; while in Figure 3, both of them are not postponed since the reclaimed CPU share can be preempted. So, the other benefit of the share-reclaiming mechanism is to improve responsiveness of I/O-intensive VMs. In addition, we can see that share-reclaiming will not introduce extra context-switches, since we always try to reclaim CPU shares for the current running VM.

4.3. Scheduling Algorithm Implementation

The formal description of VM scheduling algorithm based on Share-Reclaiming Policy (SRP) is shown in the following.

SRP: *VM scheduling algorithm based on Share-Reclaiming Policy*
Begin
1. **if** V_{cur} is blocked by pending I/O event **then**
2. **if** $context == IO\_BLOCK$ **then**
3. Find the first V_i with pending I/O events in the $qBatch$, and assign CPU to it;


```
4.    else
5.      Assign CPU to the first V_i in the qBatch;
6.    end if
7.  end if
8.  if V_cur runs out its CPU share then
9.    if context == IO_BLOCK then
10.     Sort (π_io, π_1, L, π_n) as (π_k1, π_k2, L, π_ks) in
        ascendant order of (ω_ki − b_ki) / ψ_ki;
11.     if π_k1 == π_io then
12.       Assign CPU to the first V_i in the
          qBatch;
13.     else
14.       Assign CPU to π_k1;
15.     end if
16.   else //context == SHARE_OUT
17.     if all VMs in qBatch are I/O blocked
        then
18.       Find the first ξ_SR that satisfying
          d_curr < d_SR in the queue qSR;
19.       Adjust the V_cur 's wrapper π_cur as
          following: d_curr := d_SR , b_cur := b_SR ;
20.       Remove ξ_SR from the queue qSR;
21.       Assign CPU to V_cur ;
22.     end if
23. end if
End.
```

In SRP algorithm, we introduced a set of scheduling wrappers noted as $\pi_i = (\omega_i, \psi_i, d_i, b_i)$ each representing the scheduling state of V_i, where ω_i is the maximum budgets of V_i, ψ_i is the scheduling period time, d_i is the deadline of next scheduling, b_i is the remaining budgets of V_i. It is clear that π_i has all the scheduling-related information of V_i. To implement the share-reclaiming mechanism, we introduce an ordered global queue called *qSR*, in which the unused CPU slices are ordered in the ascendant of deadline. Each element in the queue *qSR* can be considered as an available CPU share donated by blocked VM instances, and it is characterized as $\xi_{SR} = (d_{SR}, \omega_{SR}, \psi_{SR}, b_{SR})$, where d_{SR} is the deadline of the CPU share, ω_{SR}, ψ_{SR} and b_{SR} are inherited from the scheduling wrapper $\pi_i = (\omega_i, \psi_i, d_i, b_i)$ that donates this CPU share. More details of the collective I/O and share-reclaiming mechanisms will be described in the next section.

In our SRP algorithm, when a VM is blocked by pending I/O event, the first VM in the *qBatch* is scheduled. Since I/O-intensive VMs have been stored in *qBatch*, that is saying, SRP algorithm always tries to collectively scheduling I/O-intensive VMs. Second, if the current VM uses out its CPU shares, we call the algorithm will try to reclaim CPU slices from *qSR*

list. Before doing this, we should first figure out the current scheduling context. Specifically, if the scheduling context is IO_BLOCK, we select the wrapper with $\min\{(\omega_{k_n} - b_{k_n}) / \psi_{k_n}\}$ as the scheduling candidate; otherwise, the CPU slices in *qSR* will be allocated.

5. Experiments and Performance Comparison

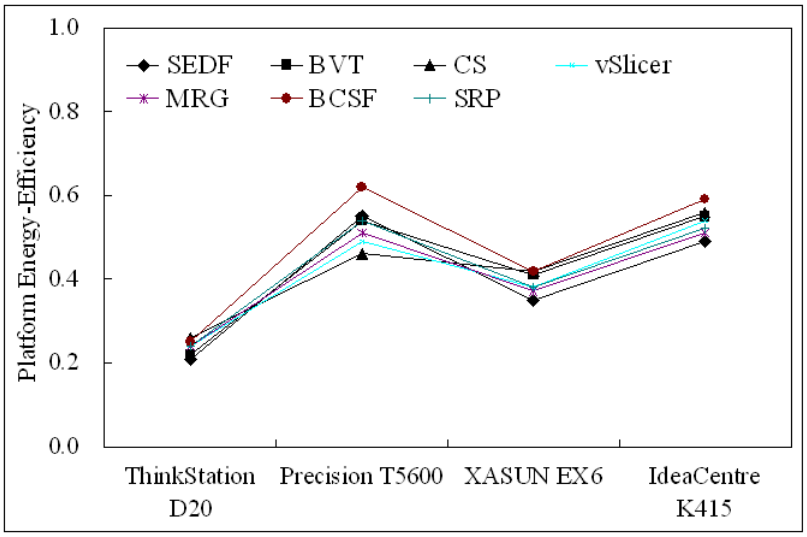
4.1 Experimental Settings

In the experiments, we use four platforms from different vendors as the experimental platforms, including ThinkStation D20, XASUN EX6, Precision T6500 and IdeaCentre K415. The VM hypervisor used in the experiments is Xen with version 4.1.2 and the underlying operation system is Linux with kernel version 2.6.2. To examine the performance of SRP scheduler, various benchmarks are used as workloads in the experiments. In these benchmarks, bzip2 and mcf are from the SPECcpu2006 benchmark suite; TPC-W is a representative benchmark for testing the performance of web servers; Cachebench is a benchmark for testing the performance of cache subsystem; IOZone is a benchmark which issues intensive file-accessing operations during its runtime. It is clear that these benchmarks can be classified into two types: CPU-intensive and I/O-intensive. Besides the three classical VM schedulers (SEDF, BVT, CS), we also select other three VM schedulers, including vSlicer [16], MRG [15] and BCSF [20], to compare the performance with our SRC-I/O scheduler. The above schedulers have been implemented in the Xen platform and widely used in practical systems.

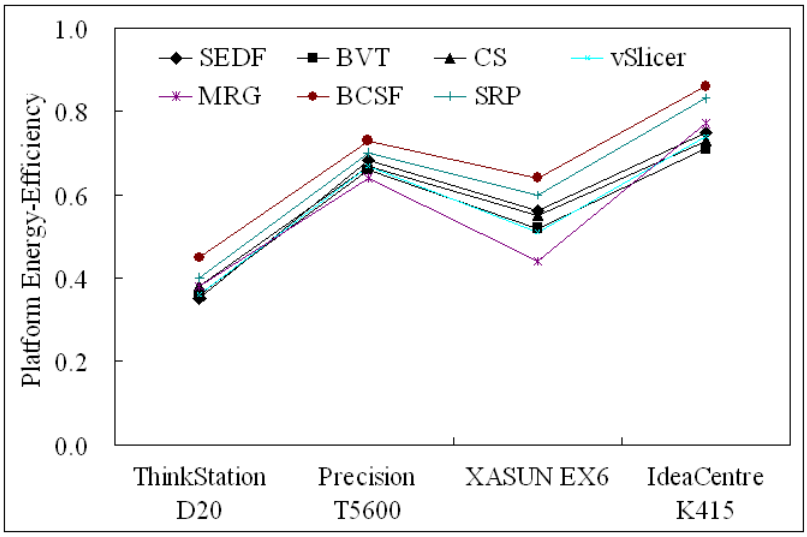
3.2 Comparison of energy-efficiency

In the experiments, we use the benchmarks to build four sets of workloads (noted as workload#1 ~ workload#4). The workload#1 consists of ten VM instances, in which five of them are running bzip2 and the others are running mfc benchmark; the workload#2 consists of fifteen VMs, in which ten of them are running bzip2 and mfc just like workload#1, and the other five VMs are running Cachebench; in the workload#3, we add TPC-W and IOZone benchmarks on the basis of workload#2, and the total number of VMs is twenty-five; in the workload#4, we strengthen the workload#3 by doubling the number of VM instances for each benchmark. From workload#1 to workload#4, it is clear that the workload characteristic is gradually changed from CPU-intensive to I/O-intensive. In addition, workload#1 and workload#2 can be considered as pure CPU-intensive, while workload#3 and workload#4 are typical mixed-workloads.

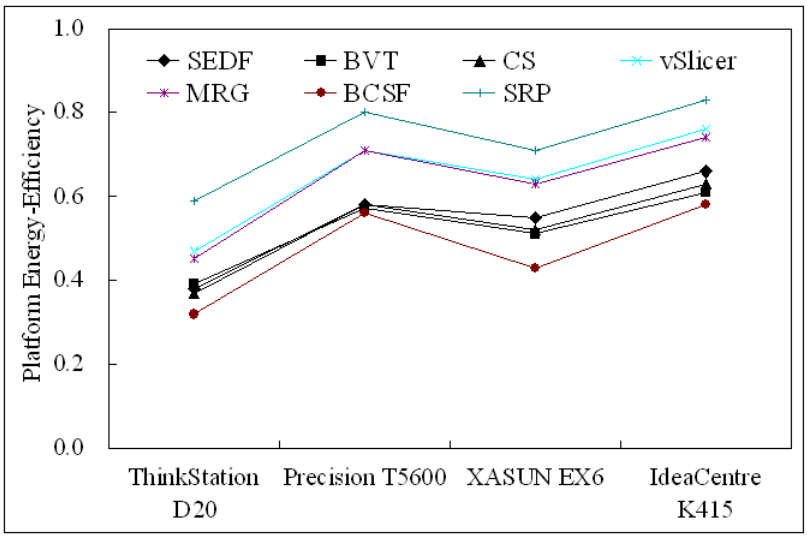
The first metric we measured is the Platform Energy-Efficiency (PEE), which is often used to evaluate that how much energy is spent on real works and how much is wasted when the machine is in idle state. The experimental results are shown in Figure 4. The first thing we noticed is that the PEE metrics of Precision T5600 and Idea Centre K415 are generally higher that the other two platforms in most of the cases, which is especially true when the testing workloads are workload#1 and workload#2. The reason is that the latter two platforms are high-performance workstations. So, when the testing workloads are not very intensive, their resource utilizations are relative lower that those of desktop PCs. When we gradually increase the workload intensiveness, such a difference becomes small as shown in Figure 4(c) and (d).



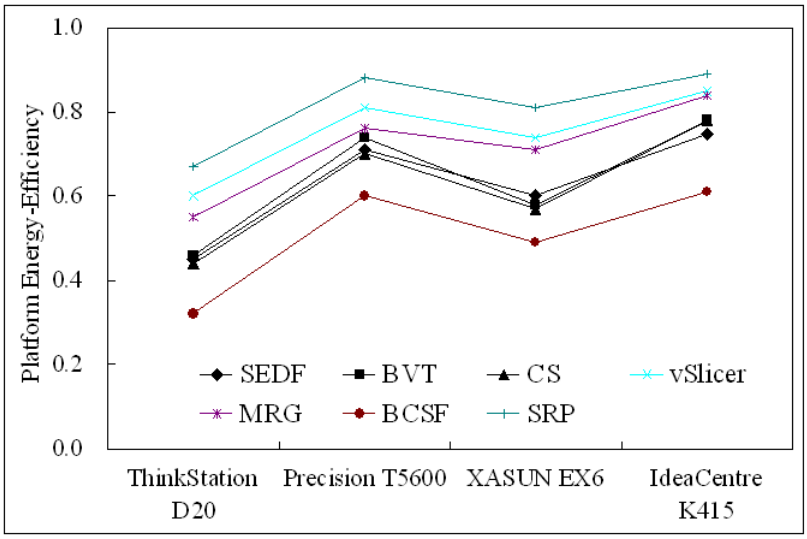
(a) workload#1



(b) workload#2



(c) workload#3



(d) workload#4

Figure 4. PEE Comparison on Different Platforms

When the workloads are pure CPU-intensive (workload#1 and workload#2), the PEE metric of BCSF is the highest in all tested schedulers regardless of the underlying platforms. By monitoring the running procedure, we found that the energy-efficiency improvement of BCSF mainly comes from memory subsystems. More specifically, BCSF divides the memory subsystem into a set of modules; by monitoring VM's memory-accessing operations, it maintains a set of active memory modules which is capable of accommodating those runnable VMs and power-off those inactive memory modules. As a result, the memory energy consumption can be significantly reduced which in turn improves the PEE metric. The policy of BCSF is more effective if the memory-related energy consumption contributes more to the total energy consumption. This is the reason that the PEE metrics of desktop platforms are higher than that of workstations when using BCSF, since the energy consumption of two

high-performance workstations mainly spent on their CPUs. On the other side, when we are running mixed-workloads (workload#3 and workload#4), the PEE metric of BCSF decreased significantly as shown in Figure 4(c) and (d). In fact, its PEE is the lowest among all tested schedulers. The reason is that BCSF scheduler is the only one which makes a trade off between performance and energy consumption in all the seven schedulers. When the workloads are low-intensive, this trade off is effective for improving PEE; when the workloads become more and more intensive, BCSF itself brings about more extra costs by switching off the memory's power. For example, when TPC-W and IOZone benchmarks are waiting for I/O completion events, BCSF will ignore their memory requirements by performing its policy; therefore, if any I/O event comes TPC-W and IOZone have to wait some extra time for BCSF powering on some memory modules.

As to the SRP scheduler, its PEE metric is the highest when running workload#3 and workload#4. The results in Figure 4 also show that the PEE metrics of vSlicer and MRG are very similar with SRP. As mentioned in Section 4.1, both vSlicer and SRP allow malleable CPU slices at runtime. Meanwhile, both MRG and SRP are incorporated with I/O batching scheduling mechanism. In order to figure out that how these two mechanisms work for improving PEE, we carefully analyze the intermediate logs of the experiments. In Figure 5, we present the CPU utilization statistics of each benchmark when running workload#4 on ThinkStation D20.

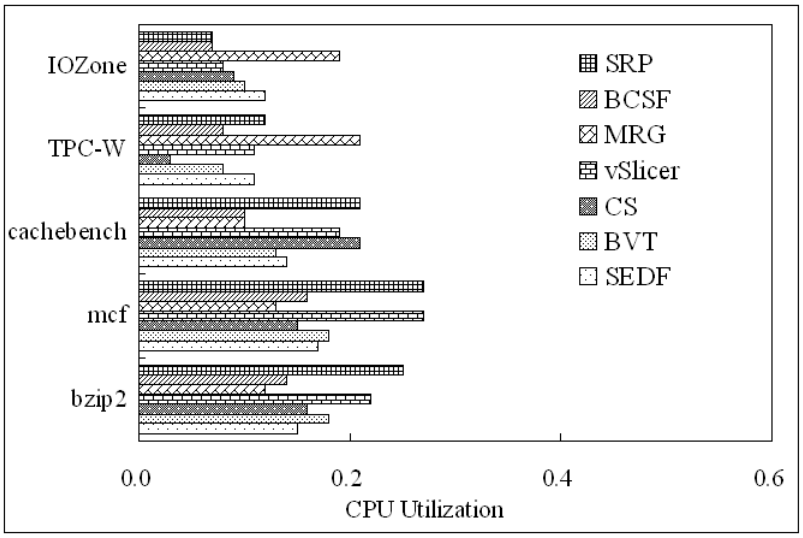


Figure 5. CPU Utilization Statistics of Each Benchmark

In the experiments, we have set that all the VMs have the same weights of sharing CPUs. As shown in Figure 5, it is clear that the practical CPU utilizations of each benchmark are quiet different from the predefined weights. For example, when using CS scheduler, the CPU utilizations of TPC-W and IOZone benchmarks are only about 3% and 9%, which are far less that the predefined value 20%. This is caused by the work-conserving policy as mentioned in Section 2.1. Even so, we also notice that the total CPU utilization of all benchmarks is only about 64% when using CS scheduler, which means more than 30% CPU time are wasted even the test workload#4 are very intensive. This also happens on SEDF, BVT, MRG and BCSF schedulers. On the other side, the total CPU utilizations when using vSlicer and SRP scheduler are about 87% and 92% respectively. As to the results obtained from other server

platforms, they exhibit the similar findings. This result indicates that malleable CPU slice mechanism is effective to increase the CPU utilization, which in turn improves the CPU energy-efficiency.

6. Conclusion

In this work, we present a novel VM scheduling policy which is aiming at reducing the energy-efficiency losses caused by I/O virtualization mechanism, especially when the virtualization platform is in presence of intensive mixed-workloads. To achieve this goal, we design a novel mechanism called Share-Reclaiming policy, and integrate them into the VM scheduler, which is currently implemented in Xen platform. Extensive experiments with various benchmarks are conducted on different platforms. In the experiments, we use platform energy-efficiency metric to evaluate the performance of the proposed scheduler. By comparing with other schedulers, the proposed SRP outperforms other scheduler in the term of energy-efficiency significantly. In the future, we plan to implement SRP scheduler in other VM hyperviors. In addition, we are taking efforts on incorporating some policies for real-time workloads in our SRP scheduler.

Acknowledgements

This work is supported by the Hunan Provincial Natural Science Foundation of China (14JJ7071).

References

[1] A. Kipp, T. Jiang, M. Fugini, *et al.*, "Layered Green Performance Indicators", Future Generation Computer Systems, vol. 28, no. 2, **(2012),** pp. 478-489.
[2] B. Dougherty, J. White and D. C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure", Future Generation Computer Systems, vol. 28, no. 2, **(2012),** pp. 371-378.
[3] S.-Y. Jing, S. Ali, K. She, *et al.*, "State-of-the-art research study for green cloud computing", Journal of Supercomputing, vol. 65, no. 1, **(2013)** Jul, pp. 445-468.
[4] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and ARM architectures power efficiency", Journal of Parallel and Distributed Computing, vol. 72, no. 12, **(2012),** pp. 1770-1780.
[5] S. Ruth, "Green IT - More Than a Three Percent Solution?", IEEE Internet Computing, vol. 13, no. 4, **(2009),** pp. 74-78.
[6] D. Ongaro, A.L. Cox and S. Rixner, "Scheduling I/O in virtual machine monitors", Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual execution Environments, Seattle, Washington, USA, **(2008),** pp. 1-10.
[7] A. Gordon, N. Amit, N. Har'El, *et al.*, "ELI: bare-metal performance for I/O virtualization", Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, London, England, **(2012),** pp. 411-422.
[8] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A System for Energy-Efficient Management of Virtual Machines", Acm Transactions on Design Automation of Electronic Systems, vol. 16, no. 1, Nov, **(2010),** pp. 1-45.
[9] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and ARM architectures power efficiency", Journal of Parallel and Distributed Computing, vol. 72, no. 12, **(2012),** pp. 1770-1780.
[10] D. Kliazovich, P. Bouvry and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers", Journal of Supercomputing, vol. 62, no. 3, **(2012),** pp. 1263-1283.
[11] R.-C. Leou, "An economic analysis model for the energy storage system applied to a distribution substation", International Journal of Electrical Power & Energy Systems, vol. 34, no. 1, **(2012),** pp. 132-137.
[12] H. Kim, H. Lim, J. Jeong, H. Jo and J. Lee. "Task-aware virtual machine scheduling for I/O performance", Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual execution Environments, Washington DC, USA, **(2009),** pp.101-110.

[13] Y. Hu, X. Long, J. Zhang, *et al.*, "I/O scheduling model of virtual machine based on multi-core dynamic partitioning", Proceedings of ACM International Symposium on High Performance Distributed Computing, Chicago Illinois, USA, **(2010),** pp. 142-154.
[14] C. Weng, Z. Wang, M. Li and X. Lu, "The hybrid scheduling framework for virtual machine systems", Proceeding of ACM SIGPLAN/SIGOPS International Conference on Virtual execution Environments, Washington DC, USA, **(2009),** pp. 111-120.
[15] H. Kang, Y. Chen, J.L. Wong, *et al.*, "Enhancement of xen's scheduler for mapreduce workloads", Proceedings of International Symposium on High Performance Distributed Computing, San Jose, CA, USA, **(2011),** pp. 251-262.
[16] C. Xu, S. Gamage, P.N. Rao, *et al.*, "vSlicer: latency-aware virtual machine scheduling via differentiated-frequency CPU slicing", Proceedings of International Symposium on High-Performance Parallel and Distributed Computing, Delft, Netherlands, **(2012),** pp.3-14.
[17] R. Koller, A. Verma and A. Neogi. "WattApp: An application aware power meter for shared data centers", Proceedings of International Conference on Autonomic Computing, Reston, VA, USA, **(2010),** pp. 31-40.
[18] I. Rodero, J. Jaramillo and A. Quiroz, "Towards energy-aware autonomic provisioning for virtualized environments," Proceedings of International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA, **(2010),** pp. 320-323.
[19] Y.C. Lee and A.Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems", Journal of Supercomputing, vol. 60, no. 2, **(2012),** pp. 268-280.
[20] J.W. Jang, M. Jeon, H.S. Kim, *et al.*, "Energy reduction in consolidated servers through memory-aware virtual machine scheduling", IEEE Transactions on Computers, vol. 60, no. 4, **(2011),** pp. 552-564.

Authors

Dongbo Liu, he received his doctor degree in Hunan University in 2012. Now he works in Hunan Institute of Engineering as an associate professor. His research interests include distributed system, cloud computing, high-performance application. He is now a member of CCF in China, and worked as Senior Engineer in HP High-performance Lab of Hunan Institute of Engineering.

Ning Han, he received his bachelor degree in Beijing University of Science and Technology, and now persuading master degree in Xiangtan University. Currently, he works in the HP High Performance Lab of Hunan Institute of Engineering as a senior networking engineer. His research interests include complex networking deployment, distributed computing, information security technology, fault-tolerance in distributed systems.