Energy Reduction in Consolidated Servers through Memory-Aware Virtual Machine Scheduling

Jae-Wan Jang, Myeongjae Jeon, Hyo-Sil Kim, Heeseung Jo, Jin-Soo Kim, *Member*, *IEEE*, and Seungryoul Maeng

Abstract—Increasing energy consumption in server consolidation environments leads to high maintenance costs for data centers. Main memory, no less than processor, is a major energy consumer in this environment. This paper proposes a technique for reducing memory energy consumption using virtual machine scheduling in multicore systems. We devise several heuristic scheduling algorithms by using a memory power simulator, which we designed and implemented. We also implement the biggest cover set first (BCSF) scheduling algorithm in the working server system. Through extensive simulation and implementation experiments, we observe the effectiveness of the memory-aware virtual machine scheduling in saving memory energy. In addition, we find out that power-aware memory management is essential to reduce the memory energy consumption.

Index Terms—DRAM energy, virtual machine, scheduling, multicore processor.

1 Introduction

As the size of data centers continues to grow, it becomes important to address the ever-increasing energy consumption of computer systems. Not only does the high energy consumption cause environmental problems, but it also significantly increases the maintenance cost of data centers. Great efforts have been devoted to reducing the energy consumption by using various power management features available in computer components [1], [2], [3].

Recently, a large number of virtual machines have been consolidated in a single computer through virtualization for efficient utilization of computer resources and ease of maintenance. It is well known that most of the servers in data centers are often underutilized due to overprovisioning [4]. Virtualization allows these underutilized servers to be consolidated into a small number of physical servers. Moreover, the advent of new processors with the enhanced support for virtualization and advances in virtualization software enables more underutilized servers to be consolidated. This paper targets these consolidated underutilized servers.

- *J.-W. Jang, H.-S. Kim, and S. Maeng are with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 335 Gwahangno (373-1 Guseong-dong), Yuseong-gu, Daejeon 305-701, South Korea. E-mail: {jwjang, maeng}@calab.kaist.ac.kr, hyosil@jupiter.kaist.ac.kr.*
- *M. Jeon is with the Department of Computer Science, Rice University, PO Box 1892, MS-132, Houston, TX 77251-1892. E-mail: mjjeon@rice.edu.*
- *H. Jo is with the Department of Computer System Engineering, Chonbuk National University, 664-14 1-ga, Deokjin-dong, Jeonju, Jeonbuk, South Korea. E-mail: heeseung@jbnu.ac.kr.*
- *J.-S. Kim is with the School of Information and Communication Engineering, Sungkyunkwan University, 300 Cheoncheon-dong, Jangan-gu, Suwon 440-746, South Korea. E-mail: jinsookim@skku.edu.*

Manuscript received 29 June 2009; revised 10 Dec. 2009; accepted 11 Mar. 2010; published online 8 Apr. 2010.
Recommended for acceptance by S.H. Son.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-06-0308.
Digital Object Identifier no. 10.1109/TC.2010.82.

In order to provide sufficient memory for each virtual machine, the demand for a large amount of memory has increased. This leads to high energy consumption in the memory system. In server systems with a large amount of memory, the energy consumption in memory may exceed that in processors. For example, Lefurgy et al. reported that in a commercial server equipped with 16 processors and 128 GB main memory, the processors are responsible for only 28 percent of the entire energy consumption while the memory for 41 percent [1]. Thus, one of key steps in building energy-efficient data centers is to reduce memory energy consumption.

The fundamental requirement to save memory energy is to understand the characteristics of physical memory. Typically, the entire physical memory is divided into a number of memory blocks, each of which is the smallest power control unit. This unit is usually termed as *rank* in the SDRAM technology; in this paper, we call this unit a *memory node*. Each memory node in its entirety is allowed to be in one of power-saving modes in order to save energy. Compared to the normal operating mode, the power-saving mode enables memory nodes to dissipate less power without loss of stored data. When data in a memory node are accessed, its power mode has to be switched to the normal operating mode. In most cases, this power mode change incurs a substantial delay.

Using these characteristics of physical memory, the basic approach to reducing memory power dissipation is to put the memory nodes that are expected to experience relatively long periods of idleness into one of power-saving modes. In the server consolidation environment, a portion of entire physical memory is allocated to each virtual machine. When a virtual machine runs, only the memory nodes, which contain the memory pages allocated to it, are accessed. Meanwhile, other memory nodes are not accessed, and these can be put into one of power-saving modes to reduce memory power dissipation.

In traditional single processor systems, only a single virtual machine runs at a specific time; thus, the memory nodes used by the virtual machine are the major source of memory energy consumption. In multicore systems, however, several processor cores simultaneously access the memory since they run a number of different virtual machines at the same time. In these systems, the memory energy consumption dynamically varies, depending on the virtual machines that run together. In order to save memory energy, it is necessary to minimize the number of memory nodes accessed by these virtual machines through a sophisticated scheduling policy.

In this paper, we mainly focus on this scheduling issue in multicore-based computer systems. We present three features, which are essential for reducing memory energy consumption, and propose a memory power management architecture that supports these features. Based on this architecture, we devise several virtual machine scheduling algorithms, the goal of which is to minimize the energy consumption in the memory system. Through the simulation experiments, the proposed BCSF scheduling achieves memory energy savings of up to 57.4 percent compared to the conventional architecture that does not use memory power management techniques. We also implement the BCSF scheduling algorithm on top of the credit scheduler of Xen and present its impact on the performance of the entire system.

The rest of the paper is organized as follows: Section 2 presents background information. Section 3 describes the proposed memory power management architecture. In Section 4, the virtual machine scheduling problem is defined, and Section 5 explains several heuristic scheduling algorithms. The methodology to evaluate the devised heuristic scheduling algorithms is given in Section 6. Section 7 discusses the simulation results. We present the preliminary implementation results in Section 8. Section 9 summarizes the related work. In the final section, we present our conclusion and future work.

2 Background

2.1 DRAM Architecture

Typically, one, two, or four memory nodes are packaged as a DRAM module, which is called dual in-line memory module (DIMM). A memory node consists of a number of DRAM chips, which work synchronously. A DRAM chip, in turn, is composed of large arrays of capacitors and a number of subcomponents such as row/column decoders, sense amplifiers, etc. When all these subcomponents are enabled, the memory can handle read/write operations immediately. This mode is defined as a *precharge* state in the DDR3 SDRAM technology [5], which we call a *standby mode* in this paper.

Disabling some of subcomponents presents several operating and power modes such as *powerdown* and *self-refresh modes* in DDR3 SDRAM [5]. If a memory node is in one of these low-power modes, it dissipates less power while retaining the stored data. In order to service read/write operations, however, the memory node should transition to the standby mode, where all subcomponents are enabled. This power mode transition incurs a large transition delay and increases the memory access latency, resulting in significant performance degradation in the entire system. To avoid this performance degradation, the memory nodes that are expected to be accessed soon should be in the standby mode.

TABLE 1
Power Dissipation and Transition Delays of a 4 GB DDR3 Registered SDRAM Module Consisting of Two Memory Nodes [6]

Power dissipation (mW)			Delays (ns)
Standby (2 nodes)	Standby (1 node)	Power-saving (self-refresh)	(from self-refresh mode to precharge mode)
3819.6	2123.6	427.6	640

The actual power dissipation and transition delays of several power modes vary, depending on the size and operating clock of individual DRAM chips. Throughout this paper, we assume the use of 4 GB DDR3 registered DIMMs, which are widely used in server class computers. Note that they are equipped with a registering clock driver (RCD) [7] for high reliability. RCD continuously consumes substantial power even when DRAM chips enter one of low-power modes. Table 1 shows the power dissipation and transition delays of this DRAM module [6]. The *self-refresh* mode is used as the power-saving mode in this paper. In DDR3 DRAM, the least power is dissipated in this mode.

2.2 Virtual Machine

Our memory power management architecture is based on the Xen virtual machine monitor (VMM) [8]. The basic function of Xen is to create several secure and isolated runtime environments on a single computer. When creating a new virtual machine, Xen makes virtual devices such as virtual processors, virtual disks, and virtual network interfaces.

At the same time, Xen allocates the entire requested memory to the virtual machine at once because Xen does not currently use a demand paging technique. Usually, this allocated memory cannot be used by other virtual machines, except when transferring data between virtual machines through temporary page sharing. Once a virtual machine is created, Xen schedules its virtual processor, which begins the boot-up procedure and eventually executes tasks inside the guest operating system.

A single virtual machine may have more than two virtual processors and run an SMP operating system to increase throughput. VMware employs a coscheduling technique to schedule such virtual processors [9]. Currently, Xen does not place any restrictions on scheduling more than two virtual processors that belong to a virtual machine. In this paper, we assume that all virtual machines use a single virtual processor except isolated driver domain (IDD). IDD is a special virtual machine that has a privilege to directly access physical hardware devices since other normal virtual machines are not allowed to access the physical hardware devices for security reasons. IDD provides hardware-related services on behalf of the normal virtual machines.

The latest version of Xen uses a *credit scheduler* [10], [11] to schedule virtual machines. The credit scheduler provides

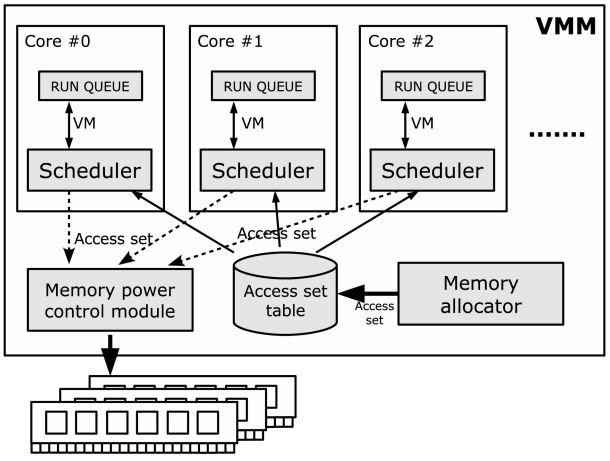

Fig. 1. Overall memory power management architecture.

fair sharing of CPU time among virtual machines based on credits. A credit indicates the relative amount of CPU time that a virtual machine can consume while guaranteeing fairness among virtual machines. Virtual machines consume credits during their execution, and the credits of all virtual machines are recharged periodically. Virtual machines that exhaust their credits are not executed until their credits are recharged. Note that Xen recently switches from a global request queue to a per-PCPU request queue, and thus, the credit scheduler uses a separate run queue for each processor core.

3 Memory Power Management Architecture

The proposed memory power management architecture is based on the current memory management and scheduling architecture of Xen. Fig. 1 shows the overall memory power management architecture proposed in this paper. This section presents three features required to save memory energy based on this architecture.

3.1 Memory Nodes Identification

The first feature is to identify a set of memory nodes accessed by each virtual machine, which is denoted as $M(v)$ or the *access set* of virtual machine v. Monitoring memory pages accessed by a virtual machine is one way to get the access set of the virtual machine. A nonintrusive way of tracking the accessed memory pages is to use hardware page faults as in [12]. In this technique, page table entries are deliberately invalidated; the pages that induce page faults are those used by the virtual machine running at that time. The access set of the virtual machine can be easily obtained from these accessed memory pages. However, it is necessary to periodically perform this task in order to maintain up-to-date information on the access sets of virtual machines. Due to the high overhead in the frequent use of page faults, this technique is inefficient to be used in the real world. Moreover, we cannot always guarantee that the recognized memory nodes will be accessed in the future since this technique relies only on the past behavior of virtual machines.

In our memory power management architecture, the set of memory nodes that contain memory pages allocated to a particular virtual machine is used as the access set of the virtual machine. When the **memory allocator** of the VMM allocates memory pages to a newly created virtual machine, the access set of the virtual machine is identified and recorded in the **access set table**. The access set acquired from this method may be larger than the access set identified from the previous technique, which uses hardware page faults. However, this method does not incur any runtime overhead for detecting access sets, and we can ensure that the virtual machine never accesses other memory nodes not in this access set.

3.2 Power Mode Transition

The second feature is to control memory power mode by using the identified access sets. As shown in Fig. 1, the **memory power control module** handles the power mode of each memory node based on currently used memory nodes. The currently used memory nodes at time t, which are denoted as $C(t)$, indicate a set of memory nodes accessed by virtual machines running on every processor core at time t. Since $C(t)$ is the union of the access sets of those virtual machines, it usually changes when one virtual machine finishes and another begins to run on any of the processor cores.

In order to reduce memory energy consumption without performance loss, the memory power control module puts memory nodes in $C(t)$ in the standby mode, and other memory nodes in the power-saving mode. Since $C(t)$ may change when the contexts of two virtual machines are switched, the virtual machine scheduler of each processor core informs the memory power control module of the access set of the newly scheduled virtual machine in advance of performing context switch between two virtual machines. When informed by the virtual machine scheduler, the memory power control module updates $C(t)$ and transitions the power mode of each memory node if necessary.

Transitioning power modes from the power-saving mode to the standby mode incurs a significant delay, as mentioned in Section 2.1. If the delay exceeds the time required for a context switch between two virtual machines, the next virtual machine may experience an unexpected delay when accessing memory node whose power mode transition is in progress. In order to avoid this performance degradation, the context switching latency should be larger than the transition delay from the power-saving mode to the standby mode.

We have measured virtual machine context switching latencies on the latest Intel Xeon E5405 processor running at 2 GHz. Fig. 2 shows the cumulative distribution function of the context switching latencies when eight virtual machines are executing various workloads on the same processor core. The figure illustrates that 98 percent of context switches take longer than the power mode transition delay, as shown in Table 1. Thus, we expect that these transition delays hardly influence the performance as long as we initiate the power mode transition before the actual context switch.

3.3 Memory-Aware Virtual Machine Scheduling

In multicore systems, the execution sequence of virtual machines affects memory energy consumption. We explain this issue with an example. Suppose that a computer system has two processor cores and four memory nodes. Fig. 3 shows two different virtual machine execution sequences and memory energy consumption in this system. In the figure, a shaded box indicates a running virtual machine;

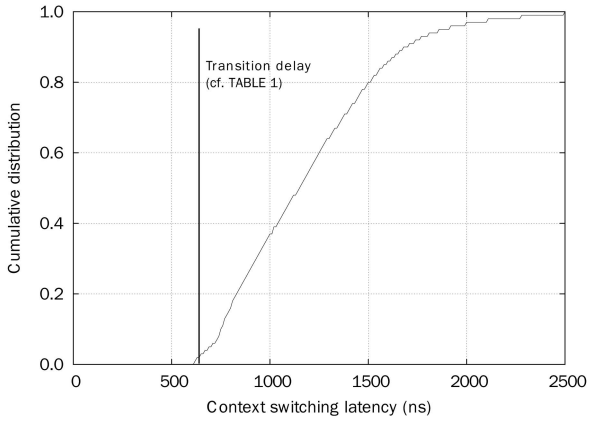

Fig. 2. The cumulative distribution function of the virtual machine context switching latencies.

we illustrate the virtual machine ID and its access set in the shaded box. In addition, $C(t)$ values at each unit time are shown. We assume that the memory energy consumption of a memory node in the standby mode for a unit time is one; we ignore the memory energy consumption in the power-saving mode. Based on this assumption, the memory energy consumption per unit time is also displayed.

Without any memory power management, the total memory energy consumption will be 32 since all four memory nodes should be in the standby mode for eight unit times. Using the features explained in Sections 3.1 and 3.2, we can reduce the total memory energy consumption. The actual amount of energy savings depends on the virtual machine execution order. When virtual machines are executed in the order as shown in Fig. 3a, the total memory energy consumption is reduced to 24.

Our work is motivated by the fact that in multicore systems, the total energy consumption in memory can be reduced further by changing the schedule of the given virtual machines. For example, if the execution sequence of the same virtual machines is changed as depicted in Fig. 3b, the total energy consumption is decreased to 16. Note that the energy reduction stems from the decrease in the number of memory nodes that are put into the standby mode by rearranging the execution order of virtual machines.

Therefore, it is necessary to develop a new scheduling policy that reorders the execution sequence of virtual machines in order to save additional memory energy. The fundamental goal of this *memory-aware virtual machine scheduling* is to minimize the number of memory nodes in the standby mode. In the following sections, we mainly focus on this issue.

When devising a virtual machine scheduler, one of the important considerations is to ensure fairness among virtual machines. In our memory power management architecture, we insert memory-aware scheduling features into the credit scheduler of Xen instead of making a new scheduler from scratch. Thus, our scheduler selects a virtual machine among the virtual machines that have credits and reside in the run queue. This temporarily reorders the execution sequence of virtual machines in the run queue, and thus, ensures that the memory-aware scheduler provides the same level of fairness compared to the credit scheduler of Xen.

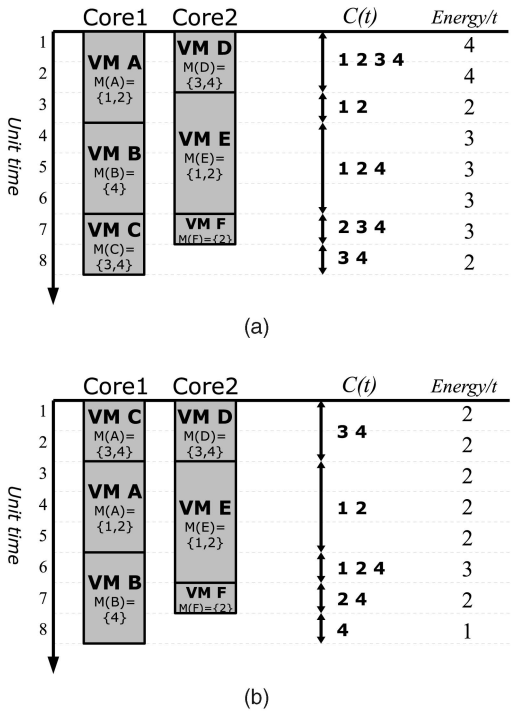

Fig. 3. Memory energy consumption of various virtual machine execution sequences. (a) Scenario 1. (b) Scenario 2.

4 Memory-Aware Virtual Machine Scheduling Problem

In this section, we formally define the memory-aware virtual machine scheduling problem. The given computer system consists of a set of memory nodes $\mathbb{M}$ and $\mathcal{C}$ processor cores, where $\mathcal{C} \geq 2$. The system has a set of virtual machines and schedules them based on their credits as explained in Section 2.2.

Let us denote V as the set of all virtual machines and V_i as the set of virtual machines that are executed on the ith processor core. v_i^t indicates the particular virtual machine that runs on the ith processor at time t. Then, $C(t)$ can be defined as

$$C(t) = \bigcup_{i=1}^{\mathcal{C}} M(v_i^t),$$

where $M(v)$ is the access set of the virtual machine v.

The power dissipation in memory at time t, denoted as $p(t)$, depends on the number of memory nodes and their operating modes. More specifically, $p(t)$ can be defined as

$$p(t) = e \cdot |C(t)| + \overline{e} \cdot |\mathbb{M} - C(t)|,$$

where e and $\overline{e}$ are the power dissipation of a memory node in the standby mode and the power-saving mode, respectively.

Therefore, the total energy consumption in memory, E, from 0 to a fixed time T is

$$E = \int_0^T p(t)\mathrm{d}t,$$

and our goal is to minimize it. That is given by

$$\begin{aligned}\min E &= \min \int_0^T p(t)\mathrm{d}t \\ &= \min \int_0^T (e \cdot |C(t)| + \overline{e} \cdot |\mathbb{M} - C(t)|)\mathrm{d}t \\ &= \min\left\{(e-\overline{e}) \int_0^T |C(t)|\mathrm{d}t + \overline{e} \int_0^T |\mathbb{M}|\mathrm{d}t\right\}.\end{aligned}$$

Since e, $\overline{e}$, and $|\mathbb{M}|$ are the invariant with respect to time, minimizing E is equivalent to

$$\min \int_0^T |C(t)|\mathrm{d}t.$$

Here, the memory-aware virtual machine scheduling problem is to find a schedule $S = \{S_1, S_2, \ldots, S_c\}$ that minimizes $\int_0^T |C(t)|\mathrm{d}t$, where S_i is a virtual machine execution sequence on the ith processor core such that $S_i = (\tau_1, \tau_2, \ldots, \tau_k, \ldots)$ and $\tau_k \in V_i$.

The restricted version of this problem is proven to be NP-COMPLETE.[1] Therefore, we devise several heuristic virtual machine scheduling algorithms that aim to minimize memory energy consumption.

5 Heuristic Scheduling Algorithms

In this paper, we present three different heuristic virtual machine scheduling algorithms: BCSF, BNF, and COMB. Each scheduling algorithm makes a scheduling decision based on $C(t)$ and the access set of the virtual machines in run queues. This section illustrates the key idea and pseudocode of each algorithm. The following notations are used in pseudocodes to describe scheduling algorithms:

- $\mathbb{R}_i$ represents the ordered set of virtual machines waiting in the run queue of the ith processor core.
- τ denotes the next virtual machine to run as a result of the scheduling decision.

Each scheduling algorithm is invoked when the VMM needs to schedule a virtual machine in the ith processor core.

5.1 FIFO

First In, First Out (FIFO) does not use any sophisticated scheduling policies. It simply returns the virtual machine at the head of $\mathbb{R}_i$ as the next virtual machine to run. FIFO shows the energy saving performance when the execution order of virtual machines is not modified. We consider the memory energy consumption of FIFO as the *base memory energy consumption*, denoted as E_{base}.

5.2 BCSF

Biggest Cover Set First (BCSF) utilizes $C(t)$ to schedule virtual machines. From the access sets of virtual machines in $\mathbb{R}_i$, BCSF tries to find the biggest access set that is completely covered by $C(t)$ and schedules the corresponding virtual machine. If such a virtual machine is not found, BCSF looks for other virtual machine whose access set is overlapped with $C(t)$ as many as possible. The main idea of BCSF is to suppress the increase in $C(t)$ whenever it schedules a new virtual machine. The pseudocode of BCSF is described in Algorithm 1.

Algorithm 1. BCSF

```
1: τ ← φ
2: for j ← 1, |R_i| do
3:    v ← the jth virtual machine in R_i
4:    if M(v) ⊂ C(t) AND |M(τ)| < |M(v)| then
5:       τ ← v
6:    end if
7: end for
8: if τ = φ then
9:    shared ← 0
10:   for j ← 1, |R_i| do
11:      v ← the jth virtual machine in R_i
12:      if |M(v) ∩ C(t)| > shared then
13:         τ ← v
14:         shared ← |M(v) ∩ C(t)|
15:      end if
16:   end for
17: end if
```

5.3 BNF

Biggest memory Node First (BNF) schedules virtual machines based on the popularity of individual memory nodes. The popularity of a certain memory node is defined as the number of virtual machines that use the memory node in the entire computer system. The higher the popularity of a memory node is, the more virtual machines use it.

The pseudocode of BNF is displayed in Algorithm 2. BNF uses two global variables: ρ and r. ρ is a set of memory nodes, and r is used in selecting memory nodes for ρ. These values are globally shared among all processor cores.

Algorithm 2. BNF

```
1: if R_i = φ or beginning of the schedule then
2:    r ← 1
3:    ρ ← φ
4: end if
5: repeat
6:    ρ ← ρ ∪ the rth popular memory node in M
7:    τ ← φ
8:    for j ← 1, |R_i| do
9:       v ← the jth virtual machine in R_i
10:      if M(v) ⊂ ρ AND |M(τ)| < |M(v)| then
11:         τ ← v
12:      end if
13:   end for
14:   if τ = φ then
15:      r ← r + 1
16:   end if
17: until τ ≠ φ
```

In BNF, each processor core tries to select the access set that is completely covered by ρ and schedules the corresponding virtual machine. If a processor core cannot find such an access set, ρ is expanded and the scheduler repeats finding a suitable access set with it. The expansion of ρ enables the scheduler to select virtual machines that have bigger access sets. This also increases the size of $C(t)$ and memory energy consumption. Thus, in order to save memory energy, it is necessary to keep the size of ρ as small as possible.

1. The complete proof of the NP-Completeness of the memory-aware virtual machine scheduling problem can be found in [13].

TABLE 2
Memory Node Combinations of a Computer System with Four Memory Nodes

n	n-combinations
1	$(1)_1, (2)_2, (3)_3, (4)_4$
2	$(1,2)_5, (1,3)_6, (1,4)_7, (2,3)_8, (2,4)_9, (3,4)_{10}$
3	$(1,2,3)_{11}, (1,2,4)_{12}, (1,3,4)_{13}, (2,3,4)_{14}$
4	$(1,2,3,4)_{15}$

TABLE 3
Worst Case Time Complexity of the Heuristic Scheduling Algorithms

Algorithm	Time complexity
FIFO	$\mathcal{O}(1)$
BCSF	$\mathcal{O}(\nu)$
BNF	$\mathcal{O}(\nu \cdot \lvert\mathbb{M}\rvert)$
COMB	$\mathcal{O}(\nu \cdot 2^{\lvert\mathbb{M}\rvert})$

In order to curb the growth of ρ, BNF exploits the popularity of memory nodes. At first, ρ contains only the memory node with the highest popularity. When ρ is expanded, the memory node with the next highest popularity is inserted to ρ. Initially, ρ consists of the memory nodes with relatively high popularity. In this case, the probability of finding the access set that is completely covered by ρ is high in each processor core, avoiding the expansion of ρ substantially. Moreover, the small access set size of the selected virtual machine (due to the small ρ) leads to significant memory energy savings in BNF.

5.4 COMB

COMB is a variant of BNF, which is enhanced with the fine-grained control over ρ. Whereas BNF exploits the popularity of memory nodes to schedule virtual machines, COMB uses memory node combinations. In this paper, memory node combinations of the given $\mathbb{M}$ indicate all the possible combinations from 1-combinations to $|\mathbb{M}|$-combinations. These are partially ordered from the smallest combination to the largest. We present an example of memory node combinations for $|\mathbb{M}| = 4$ in Table 2. The subscript of each combination shown in the table stands for the index of the partially ordered combinations. Note that the number of possible memory node combinations for a given $\mathbb{M}$ is $2^{|\mathbb{M}|} - 1$.

Algorithm 3 presents the pseudocode of COMB. COMB schedules virtual machines through the same method used in BNF, with the exception of constructing ρ. ρ is initially the memory node combination with the smallest index. If a processor core cannot find the access set that is entirely covered by ρ, ρ is replaced with the memory node combination of the next index.

Algorithm 3. COMB
1: **if** $\mathbb{R}_i = \phi$ or beginning of the schedule **then**
2: $r \leftarrow 1$
3: $\rho \leftarrow \phi$
4: **end if**
5: **repeat**
6: $\rho \leftarrow$ the rth memory node combination from $\mathbb{M}$
7: $\tau \leftarrow \phi$
8: **for** $j \leftarrow 1, |\mathbb{R}_i|$ **do**
9: $v \leftarrow$ the jth virtual machine in $\mathbb{R}_i$
10: **if** $M(v) \subset \rho$ AND $|M(\tau)| < |M(v)|$ **then**
11: $\tau \leftarrow v$
12: **end if**
13: **end for**
14: **if** $\tau = \phi$ **then**
15: $r \leftarrow r + 1$
16: **end if**
17: **until** $\tau \neq \phi$

Compared to BNF, COMB further reduces the growth of ρ by using memory node combinations in organizing ρ. In BNF, whenever no access set that is completely covered by ρ is found, ρ has to be expanded by adding another memory node, increasing the size of ρ. In contrast, in the same situation, COMB does not immediately expand ρ but replaces it with the memory node combination of the next index, which is usually the same size as the previous one. This gives COMB additional chances to find a suitable access set without increasing the size of ρ, leading to additional energy savings.

5.5 Time Complexity

The time complexity of scheduling algorithms is important since scheduling is one of the most frequently-invoked services in VMM. Thus, its overhead should be kept small to avoid performance degradation. Table 3 summarizes the time complexity of each algorithm, where ν stands for the average number of virtual machines in a run queue of each processor core.

BCSF requires the least computation since it just checks all virtual machines in the run queue. BNF is more complex than BCSF as it has to consider the popularity of memory nodes. COMB shows the highest time complexity since it searches for all the combinations of memory nodes, which requires $\mathcal{O}(2^{|\mathbb{M}|})$. Thus, the use of COMB requires careful considerations in a computer system with a large number of memory nodes.

6 Evaluation Methodology

We have developed a memory power simulator, called MPSim, to evaluate the scheduling algorithms described in the previous section.

6.1 Simulation Parameters

6.1.1 Size of Simulated Computer System

One of the basic simulation parameters is the number of processor cores and the amount of RAM in the system under evaluation. Given the memory nodes, the *maximum memory energy consumption*, denoted as E_{max}, indicates the memory energy consumption assuming that no memory power management technique is used, and thus, every memory node is in the standby mode. Note that we do not consider the memory power consumption during read or write operations in E_{max} since it highly depends on the workloads. Table 4 shows the various systems used in the experiments.

6.1.2 Consolidation Ratio

Another simulation parameter is the *consolidation ratio*, which represents the average number of virtual machines

TABLE 4
System Size Configurations

	Configuration
System A	4 processor cores, 16GB RAM (8 memory nodes)
System B	8 processor cores, 32GB RAM (16 memory nodes)
System C	12 processor cores, 48GB RAM (24 memory nodes)
System B1	8 processor cores, 16GB RAM (8 memory nodes)
System B2	8 processor cores, 24GB RAM (12 memory nodes)
System B3	8 processor cores, 32GB RAM (16 memory nodes)
System B4	8 processor cores, 40GB RAM (20 memory nodes)
System B5	8 processor cores, 48GB RAM (24 memory nodes)

that run on a single processor core. This paper utilizes three different consolidation ratios: 4, 8, and 12 virtual machines per processor core. The total number of virtual machines of a system is given by the number of processor cores of the system multiplied by the consolidation ratio.

6.1.3 CPU and I/O Burst Time Trace

In order to reflect the execution characteristics of virtual machines in the simulator more realistically, statistics on the CPU and I/O burst time are used. The CPU burst time means the time duration that a virtual machine runs before it is switched out due to I/O events or the expiration of a given time slice. The I/O burst time represents the time duration that a virtual machine waits for the completion of I/O events. The CPU burst time of a virtual machine, as well as its I/O burst time, depends on the workload characteristics of individual virtual machines.

We have collected CPU and I/O burst time traces from virtual machines running on a real platform using Xen. We have instrumented the Xen scheduler so that we can record the actual CPU and I/O burst time of the given workload. The traces are gathered from three different workloads: Linux kernel compilation (KRNL), RUBiS auction site benchmark (RUB) [14], and Web browsing through VNC (VNC) [15]. These three workloads exhibit different execution characteristics, which are summarized below.

- *KRNL.* This workload shows a long CPU burst time and almost no I/O burst time.
- *RUB.* Only a relatively small amount of CPU time is necessary to process HTTP requests. Thus, the CPU burst time is short, while the I/O burst time is relatively long.
- *VNC.* This workload shows a mixture of CPU and I/O intensive characteristics since rendering Web pages requires significant CPU time, while the VNC server and client send/receive data over the network.

We construct a different mixture of these workloads for each consolidation ratio, as summarized in Table 5. Since the actual traces are affected by the consolidation ratio and workload combinations, we have gathered the traces separately for each configuration shown in Table 5 and used in the comparable simulation experiments.

6.1.4 Access Set Configuration

The final simulation input is the access set of each virtual machine. We actually identify the access sets of all virtual machines from the working system. Note that we evenly distribute the entire physical RAM to virtual machines. In System B, for example, each virtual machine uses 490 MB for main memory if the consolidation ratio is 8 VMs/core.

TABLE 5
Consolidation Ratios and Workload Mixtures

Consolidation ratio	KRNL	RUB	VNC	IDD
4 VMs/core	1	1	1	1
8 VMs/core	2	3	2	1
12 VMs/core	3	5	3	1

Creating and destroying virtual machines repeatedly lead to memory fragmentation and increase the access sets of the newly created virtual machines. In order to see this impact on memory energy consumption, we use three different access set configurations captured as follows: The MAS1 configuration is taken from the access sets when Xen initially creates all virtual machines. The MAS2 configuration is obtained after half of the total virtual machines are destroyed and recreated; the MAS3 configuration is obtained after two and half of the total virtual machines are destroyed and recreated.

The actual property of access set configurations is affected by the size of the system and consolidation ratios. As an example, we show the property of the access set configurations actually identified from System B under the consolidation ratio of 8 VMs/core in Table 6. Xen currently tries to allocate a contiguous memory region to a newly created virtual machine. Thus, MAS1 shows smaller memory nodes than other configurations.

6.2 Simulation Architecture

MPSim simulates virtual machine executions over a number of processor cores and keeps track of $C(t)$ at every time t for a given simulation duration. The series of $C(t)$ is used to estimate the total memory energy consumption.

For each processor core, MPSim classifies all simulated virtual machines into the following four states:

- *EXECUTE* for the virtual machine that is currently running,
- *RUNNABLE* for the virtual machine that is waiting for the CPU with remaining credits,
- *EXPIRED* for the virtual machine that has exhausted its credit, and
- *WAIT* for the virtual machine that waits for the completion of I/O event.

MPSim maintains a separate queue for each state, and virtual machines in the same state are kept in the same queue. Especially, the queue for the RUNNABLE state is

TABLE 6
Property of the Access Set Configurations Obtained from System B under 8 VMs/Core

Conf.	Memory nodes			
	Avg.	Var.	Min.	Max.
MAS1	1.80	1.15	1	4
MAS2	2.60	2.30	1	7
MAS3	4.11	3.25	1	8

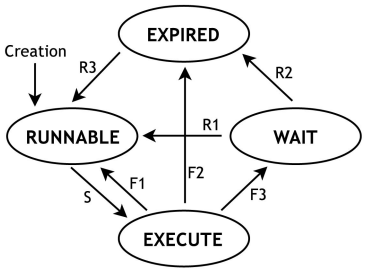

Fig. 4. States transitions of virtual machines.

simply called the *run queue*. Fig. 4 shows these states and possible state transitions.

Each simulated processor core in MPSim repeats the following procedures for the specified simulation duration:

1. **Virtual machine scheduling.** MPSim selects one of virtual machines in the run queue based on a scheduling algorithm. $C(t)$ is updated using the access set of the selected virtual machine if necessary.
2. **Execution time generation.** The execution time of the selected virtual machine is decided using the *CPU burst time trace*, one of the simulation parameters explained in Section 6.1.3. Then, the state of the virtual machine is changed to `EXECUTE` (transition S).
3. **Execution.** If the execution time of the virtual machine is equal to or greater than the *time slice*, which is, by default, 30 ms in the credit scheduler, the virtual machine is preempted. Otherwise, the virtual machine voluntarily relinquishes the CPU after consuming its CPU burst time. In both cases, MPSim charges 100 credits of the running virtual machine for every 10 ms as the current credit scheduler of Xen does.
4. **Postprocessing.** If the virtual machine voluntarily relinquishes the CPU, the state of the virtual machine is changed to `WAIT` (transition F3). For other virtual machines, which consume the entire time slice, the next state is decided based on the remaining credits. If the virtual machine still has some credits, its state transitions again to `RUNNABLE` (transition F1); otherwise, to `EXPIRED` (transition F2).

After the postprocessing procedure, MPSim moves to the first procedure in order to pick the next virtual machine to run.

A virtual machine in the `WAIT` state waits for a specific amount of time, which simulates the waiting time for I/O events. In MPSim, the waiting time of the virtual machine is determined by using the *I/O burst time trace* (cf. Section 6.1.3). After the decided I/O burst time has expired, the state of the virtual machine is changed to `RUNNABLE` if it has remaining credits (transition R1); otherwise, to `EXPIRED` (transition R2).

Like the credit scheduler of Xen, MPSim recharges the credits of all virtual machines every 30 ms [10], [11]. Once the credits are recharged, virtual machines in the `EXPIRED` state advance to the `RUNNABLE` state (transition R3).

7 Simulation Results

This section presents the simulation results. All the measurements are taken from the average of 50 simulations, each running for 10,000 ms of simulated time.

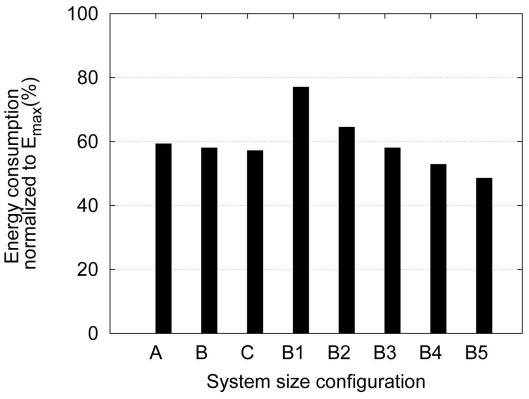

Fig. 5. Normalized E_{base} under various system size configurations.

7.1 Base Memory Energy Consumption

7.1.1 System Size

This section examines the impact of the system size on E_{base} relative to E_{max}. E_{max} of a given system is directly affected by the number of memory nodes installed in the system; we denote this number as $|\mathbb{M}|$. However, E_{base} depends on $C(t)$, as explained in Section 4. Since $C(t)$ represents the set of memory nodes used by virtual machines running at the same time, it is proportional to both: 1) the average number of memory nodes used by a virtual machine, denoted as N_{mem}, and 2) the number of processor cores of the system, denoted as $N_{\mathcal{C}}$. Thus,

$$\frac{E_{base}}{E_{max}} \propto \frac{N_{\mathcal{C}} \times N_{mem}}{|\mathbb{M}|}.$$

Fig. 5 plots E_{base} normalized to E_{max} of the various computer systems listed in Table 4. We assume the access set configuration of `MAS1` and the consolidation ratio of 8 VMs/core. The figure shows that even without any scheduling algorithm, our memory power management architecture successfully saves memory energy.

The relative amount of energy savings in each system is not the same. In this experiment, the same access set configuration is used, and thus, N_{mem} is identical in all cases. Then,

$$\frac{E_{base}}{E_{max}} \propto \frac{N_{\mathcal{C}}}{|\mathbb{M}|}.$$

We denote the ratio of $N_{\mathcal{C}}$ to $|\mathbb{M}|$ simply as *C/M ratio*. When the *C/M ratio* is identical as in `System A`, `B`, and `C`, we observe that the relative memory energy consumption is similar to about 60 percent of E_{max}, regardless of the actual size of each system. However, when we decrease the *C/M ratio* as shown in the second half of Table 4 (`System B1` to `B5`), the memory energy consumption reduces. Especially in `System B5`, which has the smallest *C/M ratio* among all the simulated systems, the memory energy consumption is reduced to about 48.5 percent of E_{max}.

As the degree of server consolidation grows, the demand for a large amount of memory is increasing to provide sufficient memory to each virtual machine. Thus, the *C/M ratio* is also expected to decrease. In such an environment, our result shows that the proposed memory power management architecture effectively reduces the memory energy consumption.

TABLE 7
Normalized E_{base} of System B under Different Access Set Configurations and Consolidation ratio of 8 VMs/Core

Access set conf.	E_{base} (relative to E_{max})
MAS1	57.98 %
MAS2	72.67 %
MAS3	88.75 %

7.1.2 Access Set Configuration

We measure E_{base} of System B under various access set configurations when the consolidation ratio is 8 VMs/core. The results are displayed in Table 7. We observe that E_{base} is strongly influenced by the access set configuration. As the access set configuration changes from MAS1 to MAS3, the access set size of each virtual machine increases and so does $C(t)$. This eventually raises the memory energy consumption.

7.2 Energy Savings of Scheduling Algorithms

The following sections discuss the additional memory energy savings relative to E_{base} via memory-aware scheduling algorithms. For each scheduling algorithm, we measure the energy consumption of System B under various consolidation ratios and access set configurations. Fig. 6 shows the memory energy savings relative to E_{base}.

7.2.1 Impact of Consolidation Ratio and Access Set Configuration

In any access set configuration, we notice that the memory energy savings tend to be improved as the consolidation ratio increases. Enlarging the consolidation ratio increases the number of virtual machines in the run queue. This allows each algorithm to find a more suitable virtual machine to schedule, leading to better energy savings.

Fig. 6 also presents the memory savings for various access set configurations. We find that the proposed scheduling algorithms result in more energy savings with respect to E_{base} for all the access set configurations. The largest energy savings have been achieved when the access set configuration is MAS1. However, as the access set configuration varies from MAS1 to MAS3, the amount of energy savings decreases accordingly. To achieve more energy savings, this indicates that the memory power-aware scheduling algorithms should be accompanied by power-aware memory management that can shrink the access set size of each virtual machine.

7.2.2 Energy Saving Performance of Individual Scheduling Algorithms

First of all, COMB shows substantial energy saving performance when the consolidation ratio is small, while the energy savings are not greatly improved as the consolidation ratio increases. As explained in Section 5.4, COMB consumes substantial CPU time to suppress the increase in $C(t)$ through fine-grained control over ρ. In real-world workloads, however, it turns out that this additional effort does not lead to significant energy savings.

BCSF exhibits the largest energy savings when 12 virtual machines per processor core are consolidated under the MAS1 access set configuration. In this configuration, BCSF results in energy savings of up to 22.3 percent relative to E_{base}, which is 57.4 percent relative to E_{max} of the given system. BCSF shows superior energy saving performance in MAS1. This is because $C(t)$ is not drastically increased since BCSF usually selects the access set that is completely covered by the current $C(t)$. If BCSF cannot find such access sets, $C(t)$ is expanded to include the access set of the newly scheduled virtual machine. In spite of this, $C(t)$ does not grow much since the access set size of each virtual machine is relatively small in MAS1.

In MAS2 and MAS3, the amount of memory energy saved using BCSF considerably decreases due to relatively large access set size. In BCSF, the presence of even one big access set can be a significant obstacle in keeping the size of $C(t)$ small. When the large access set is selected and the corresponding virtual machine is scheduled, $C(t)$ is expanded. The increased $C(t)$ hardly shrinks although virtual machines with small access sets are scheduled later. In BCSF, each processor core chooses the access set overlapped with $C(t)$ as many as possible but does not consider overlapping with the access sets selected in other processor cores. Thus, the selected access sets usually do not contain many same memory nodes. These access sets form $C(t)$ again, and the size of $C(t)$ is not easily reduced. This leads to the small memory energy savings.

BNF saves much smaller energy than BCSF. Initially, ρ is small. As time advances, ρ has to increase in order to schedule virtual machines whose access sets are not covered by ρ. As BNF schedules virtual machines with increased ρ, it has a similar problem as BCSF. Each processor core chooses the access set overlapped with ρ as many as possible but does not reflect the access sets of the

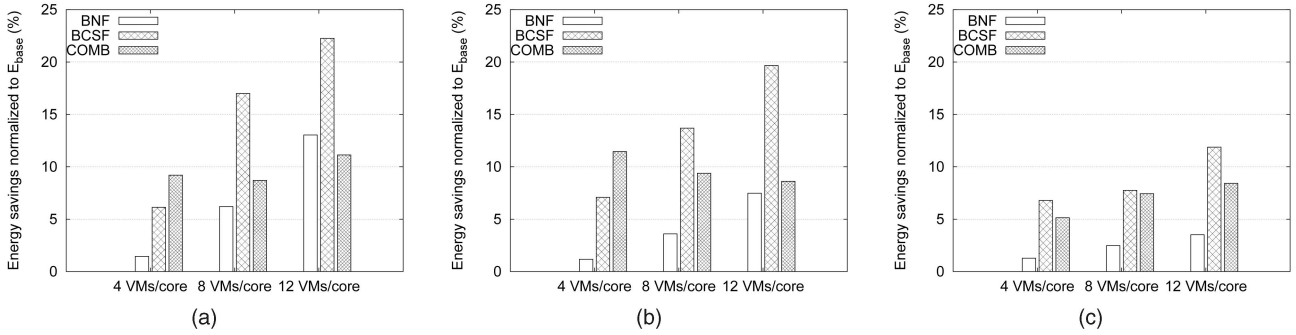

Fig. 6. Energy savings of BCSF, BNF, and COMB for various consolidation ratios and access set configurations. (a) *MAS1*. (b) *MAS2*. (c) *MAS3*.

TABLE 8
Average Elapsed Time to Schedule a Virtual Machine

	Duration (us)
No memory-aware scheduling	11.49
With BCSF algorithm	12.45

TABLE 9
Average Elapsed Time for Kernel Compilation

	Average (s)	Standard deviation (s)
No memory-aware scheduling	983.89	35.6
With BCSF algorithm	980.76	26.3

virtual machines scheduled on other processor cores. Then, the selected access sets usually do not include many same memory nodes. Thus, in this case, BNF does not efficiently save memory energy.

8 Implementation

We implement the BCSF algorithm on top of the credit scheduler of Xen 3.4 since it shows the best energy saving performance in most of the simulation results. This section presents the actual energy savings and the impact of memory-aware scheduling on the existing scheduling architecture.

For measurements, we use a server equipped with two Xeon E5405 (quad-core) processors and 32 GB RAM. Throughout the experiments in this section, we run two virtual machines for KRNL, three for RUB, two for VNC, and one for IDD over each processor core (i.e., 8 VMs/core) unless otherwise stated. All the measurements are taken while running this workload for 300 seconds.

In order to compare the simulation results with the implementation results under different access set status, `MAS1` to `MAS3` are also used in the implementation experiments. For `MAS1`, the experiments are performed when Xen initially creates all virtual machines. For `MAS2` and `MAS3`, we perform the experiments after half of all virtual machines and two and half of all virtual machines are destroyed and recreated, respectively.

Unfortunately, the memory controller of our server does not provide any direct interfaces for controlling the memory power mode although DRAM itself has several low-power modes. As a result, we log the scheduling time of each virtual machine and its memory node status during the execution of workloads and indirectly calculate the energy consumption.

8.1 Impact of Memory-Aware Scheduling on the Conventional Architecture

In this section, we use the `MAS1` access set configuration for all experiments.

8.1.1 Scheduling Overhead

The BCSF scheduling algorithm adds additional computation time when scheduling virtual machines. Scheduling is one of the most frequently invoked services in the VMM and needs to be lightweight not to affect system performance. We measure the average elapsed time to schedule a virtual machine without memory-aware scheduling and with the BCSF algorithm. Table 8 shows the measured time. When using BCSF, additional $1\ \mu s$, which is about 8.3 percent of the conventional scheduling time, is spent to select a virtual machine that can reduce memory energy consumption. When we run an experiment for 300 seconds, however, less than 0.1 second is consumed in scheduling virtual machines, and thus, additional $1\ \mu s$ for memory-aware scheduling is negligible and hardly affects the overall system performance.

8.1.2 Fairness

As stated in Section 3.3, the memory-aware virtual machine scheduling is based on the credit scheduler of Xen and has the same level of fairness. In order to see the effect of memory-aware virtual machine scheduling on fairness, we run eight virtual machines on each processor core and make them compile the Linux kernel. Then, we measure the elapsed time and display the results in Table 9. As expected, memory-aware virtual machine scheduling distributes CPU time quite evenly over all the virtual machines.

8.1.3 Run Queue Waiting Time

Although memory-aware virtual machine scheduling reveals the same level of fairness compared to the credit scheduling, reordering virtual machine execution sequences temporarily affects the waiting time in the run queue. This waiting time influences the responsiveness of virtual machines. We measure the average waiting time of virtual machines in the run queue under the `MAS1` access set configurations, as shown in Table 10. Using the BCSF algorithm, about 37 percent of waiting time is added compared to the results without memory-aware virtual machine scheduling. This shows the trade-off between energy savings of the entire system and responsiveness of individual virtual machines.

The average waiting time of IDD is shorter than that of other virtual machines. The major reason is that IDD frequently runs for relatively short time since its task is usually to handle interrupts of physical hardware devices. Thus, IDD does not exhaust its credit and remains in the run queue for most of the experiment time, leading to the higher probability to be scheduled. Also, the increased average waiting time of IDD using BCSF is smaller than that of other virtual machines. One of the reasons is that IDD has a bigger access set than other normal virtual machine due to the memory allocation policy of Xen. In BCSF, there can be several access sets covered by $C(t)$ when $C(t)$ is large, but the biggest one is selected. Thus, virtual machines of big access sets have a higher probability to be scheduled when $C(t)$ is large. Likewise, the big access set of IDD leads to more chances to be scheduled when $C(t)$ is large.

TABLE 10
Average Waiting Time in Run Queue

	Average of all VMs (ms)	IDD (ms)
No memory-aware scheduling	49.13	31.68
With BCSF algorithm	67.69	36.66

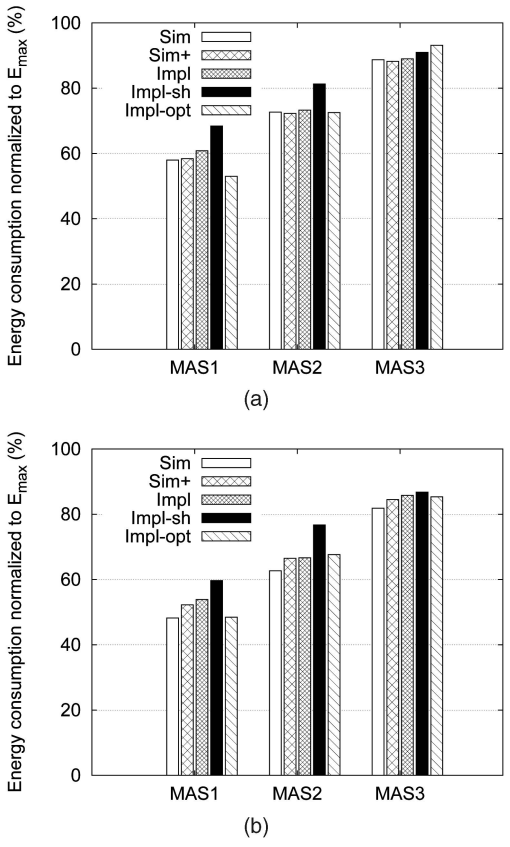

Fig. 7. Energy consumption normalized to E_{max}. (a) Base energy consumption. (b) Energy consumption when the BCSF algorithm is used.

8.2 Energy Savings of BCSF Scheduling

We display the energy consumption measured through simulation and implementation experiments under various access set configurations together in Fig. 7. In the figure, *Sim* stands for the simulation results and *Impl* for the implementation results. For E_{base}, the implementation results present energy consumption similar to the simulation results, as shown in Fig. 7a. However, when BCSF is used, we observe that simulation results and implementation results have a non-negligible difference, as depicted in Fig. 7b.

One of the major reasons for this gap is the increased waiting time of IDD in the run queue, as shown in Table 10. The increased waiting time of IDD leads to the deferred interrupt processing, which increases the I/O burst time of other virtual machines. This bias is not considered in the simulation experiments with BCSF since we use the same I/O burst time traces for all algorithms under evaluation in the previous section. Because of the increased I/O burst time, the number of virtual machines in the run queue decreases and BCSF has a reduced chance to find a more suitable virtual machine, which might reduce energy savings. Thus, in implementation experiments, BCSF consumes more energy than that in the simulation.

We apply the bias in the simulation experiments with BCSF by adding the increased I/O burst time into the I/O burst time traces and measure the energy consumption again. The simulation results, denoted as *Sim+*, are also shown in Fig. 7. By doing this, the energy difference between the simulation and implementation results is decreased within 1.6 percent of E_{max} from 5.7 percent of E_{max}.

8.3 Shared Memory Impact on Saving Memory Energy

Xen exploits shared memory to move data, such as disk blocks and network packets, between virtual machines. The previous simulation experiments cannot estimate the impact of shared memory since shared memory is dynamically created and removed. Due to the implementation, we can actually evaluate such impact.

First, we adjust the access set of a virtual machine when it uses the shared memory that is located out of the memory nodes it currently uses. After the virtual machine releases the shared memory, the access set of the virtual machine is restored. The modified access sets are used when scheduling virtual machines in BCSF and updating $C(t)$.

We display the results, denoted as *Impl-sh*, in Fig. 7. When the shared memory is considered, the access sets of virtual machines expand and the memory energy consumption accordingly increases. The increase of the energy consumption is great especially under the `MAS1` and `MAS2` access set configurations. In `MAS3`, the increased access sets do not affect the memory energy consumption since the access sets of all virtual machines are substantially large already. However, regardless of the access set configurations, BCSF reduces the energy consumption by 4.63 percent (`MAS3`) to 12.8 percent (`MAS1`) compared to E_{base}.

When we inspect the usage of shared memory during the experiment, most of shared memory is created between IDD and other virtual machines. When IDD moves data from physical devices to virtual machines and vice versa, IDD maps the memory pages of the target virtual machines in its address space and accesses those pages. When the use of the pages finishes, IDD immediately unmaps the shared memory and the access set of IDD is restored to the previous access sets. Since all the virtual machines need the help of IDD for I/O services, the access set of IDD is frequently expanded, which leads to the increased memory energy consumption.

One of the ways to avoid the increased access set of IDD due to shared memory is to use the memory pages from IDD instead of that from the target virtual machines when creating shared memory. In this way, the target virtual machine maps the memory pages from IDD in its address space to access the shared memory pages. Then, the access set of IDD does not expand due to the shared memory although the access set of the target virtual machines may increase. Note that this method is used in [16] to improve I/O performance in Xen.

Assuming that this method is applied to the memory management of Xen, we measure energy consumption over the same experimental environment. We show the results, denoted as *Impl-opt*, in Fig. 7. The memory energy consumption is dramatically reduced compared to the results of *Impl-sh* when the access set configuration is relatively small such as in `MAS1` and `MAS2`. This shows that reducing the access set of IDD is important in order to save the memory energy when other access sets are small.

9 Related Work

Traditionally, the energy consumption in memory has been reduced using the information inside the operating system since it controls the execution of applications and various

system services. Delaluz et al. manipulated power mode transitions within the process scheduler of the operating system when switching contexts between processes in conjunction with hardware-directed memory power management [12]. Huang et al. exploited page aggregation and migration techniques in order to reduce the number of memory nodes allocated to a process, which considerably reduces memory power [19]. Lee et al. saved additional energy by considering buffer caches of the operating system in memory power management [20].

All these researches are based on a single-processor computer system. They only focused on minimizing the number of memory nodes used by a process. In this paper, we mainly consider the scheduling problem of memory power management in multicore systems. However, all previous techniques can be orthogonally used to reduce the number of memory nodes used by a virtual machine since this could save substantial energy as shown in the simulation and implementation results.

The nonvirtualized computing environment based on multicore systems has a similar scheduling problem in reducing memory energy consumption. Our scheduling algorithms are also applicable in this environment if we consider virtual machines as processes, and VMM as an operating system. However, this paper targets the virtualized computing environment since it is more attractive than the nonvirtualized environment for efficiently utilizing the increasing performance capabilities of computing resources.

Memory is one of major bottlenecks in achieving a high degree of server consolidation. Several researches used the memory page sharing technique to consolidate more virtual machines on the same computer system. The VMware ESX server used content-based page sharing and hot I/O page remapping to share memory page frames [17]. Gupta et al. devised subpage level sharing and in-core memory compression, which leads to additional reduction in memory use [18]. As the number of shared memory pages increases, the access set of a virtual machine is also expected to increase, and thus, the memory allocation status gradually seems to transition to the memory allocation status similar to MAS3. Based on the simulation results shown in Section 7.2, we expect that our memory management architecture saves additional memory energy even in such environments.

10 Concluding Remarks and Future Work

This paper identifies that virtual machine scheduling affects the memory energy consumption in the server consolidation environment based on multicore systems. We propose a memory power management architecture by adding memory power-aware features to Xen. In order to examine the memory energy consumption in various systems, a memory power simulator, MPSim, is designed and implemented. On top of MPSim, we devise heuristic scheduling algorithms: BCSF, BNF, and COMB.

Through extensive simulation results, we present that our memory power management architecture effectively reduces the memory energy consumption and our scheduling algorithms make additional energy savings. Especially, BCSF saves memory energy by up to 57.4 percent compared to the conventional system that does not exploit memory power management features. As the degree of server consolidation increases, energy saving performance of the scheduling algorithms further improves. We also find that power-aware memory management is necessary in order to achieve more energy savings by reducing the number of memory nodes used by a virtual machine.

In order to see whether the simulation results can be replicated on real systems, we implement the BCSF scheduling algorithm in the credit scheduler of Xen. Simulation and implementation results are compared and similar results are obtained. Based on the implementation, we discover that shared memory between virtual machines significantly influences the memory energy consumption. We present that the use of memory in IDD in creating shared memory can relieve such problems.

Our simulator and implementation currently do not support virtual machine migration between processor cores since Xen usually restrains virtual machine migrations in order to benefit from the processor caches. We plan to support virtual machine migration and devise various migration policies to save the memory energy further.

Acknowledgments

This work was supported by Mid-career Researcher Program through National Research Foundation (NRF) grant funded by the MEST (No. 2010-0000114).

References

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller, "Energy Management for Commercial Servers," *Computer,* vol. 36, no. 12, pp. 39-48, Dec. 2003.

[2] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems," *Computer,* vol. 37, no. 11, pp. 68-74, Nov. 2004.

[3] D. Meisner, B.T. Gold, and T.F. Wenisch, "PowerNap: Eliminating Server Idle Power," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 205-216, 2009.

[4] VMware, "Server Consolidation Overview, Building a Virtual Infrastructure," http://www.vmware.com/solutions/consolidation/ consolidate.html, 2009.

[5] Micron Technology, Inc., DDR3 SDRAM, http://www.micron.com, 2009.

[6] "DDR3 SDRAM MT41J256M4—32 Meg × 4 × 8 Banks," Micron Technology, Inc., 2006.

[7] "1.35V/1.5V Registering Clock Driver with Parity Test and Quad Chip Select (SSTE32882HLB)," Integrated Device Technology, 2006.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe Hardware Access with the Xen Virtual Machine Monitor," *Proc. First Workshop Operating System and Architectural Support for the On Demand IT InfraStructure,* 2004.

[9] VMware, "ESX Server—Best Practices Using VMware Virtual SMP," white paper, http://www.vmware.com/pdf/vsmp_best_practices.pdf, 2005.

[10] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 35, no. 2, pp. 42-51, 2007.

[11] D. Ongaro, A.L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," *Proc. Conf. Virtual Execution Environments (VEE),* pp. 1-10, 2008.

[12] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "Scheduler-Based DRAM Energy Management," *Proc. 39th Ann. Design Automation Conf. (DAC),* pp. 697-702, 2002.

[13] J.-W. Jang, M. Jeon, H.-S. Kim, H. Jo, J.-S. Kim, and S. Maeng, "NP-Completeness of Memory-Aware Virtual Machine Scheduling Problem," Technical Report CS-TR-2009-321, Computer Science Dept., KAIST, 2009.

[14] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and Implementation of Dynamic Web Site Benchmarks," *Proc. IEEE Workshop Workload Characterization (WWC)*, pp. 3-13, 2002.
[15] T. Richardson, Q. Stafford-fraser, K.R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33-38, Jan./Feb. 1998.
[16] J.R. Santos, Y. Turner, G.J. Janakiraman, and I. Pratt, "Bridging the Gap between Software and Hardware Techniques for I/O Virtualization," *Proc. USENIX Ann. Technical Conf.*, pp. 29-42, 2008.
[17] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," *ACM SIGOPS Operating Systems Rev.*, vol. 36, pp. 181-194, 2002.
[18] D. Gupta, S. Lee, M. Vrable, S. Savage, A.C. Snoeren, G. Varghese, G.M. Voelker, and A. Vahdat, "Difference Engine: Harnessing Memory Redundancy in Virtual Machines," *Proc. In 8th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 2008.
[19] H. Huang, P. Pillai, and K.G. Shin, "Design and Implementation of Power-Aware Virtual Memory," *Proc. USENIX Ann. Technical Conf.*, 2003.
[20] M. Lee, E. Seo, J. Lee, and J.-S. Kim, "PABC: Power-Aware Buffer Cache Management for Low Power Consumption," *IEEE Trans. Computers*, vol. 56, no. 4, pp. 488-501, Apr. 2007.

Jae-Wan Jang received the BS degree in computer engineering from Kyungpook National University in 2002, and the MS and PhD degrees in computer science in 2004 and 2010, respectively, from Korea Advanced Institute of Science and Technology (KAIST). He is currently a performance engineer in NHN Corporation. His research interests include distributed system, networked system, and virtualization.

Myeongjae Jeon received the BE degree in computer engineering from Kwangwoon University in 2005, and the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2009. He is currently working toward the PhD degree in computer science at Rice University. His research interests include machine virtualization, distributed systems, and storage systems.

Hyo-Sil Kim received the BS degree in computer engineering from Kyungpook National University in 2005, and the MS degree in computer science in 2007 from Korea Advanced Institute of Science and Technology (KAIST), where she is currently working toward the PhD degree. She is interested in the theoretical problems on the discrete and computational geometry.

Heeseung Jo received the BS degree in computer science from Sogang University, Korea, in 2000, and the PhD degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), in 2010. Currently, he is the faculty of the Chonbuk National University. His research interests include computer hardware and system software.

Jin-Soo Kim received the BS, MS, and PhD degrees in computer engineering from Seoul National University, Korea, in 1991, 1993, and 1999, respectively. He is currently an associate professor in Sungkyunkwan University. Before joining Sungkyunkwan University, he was an associate professor at Korea Advanced Institute of Science and Technology (KAIST) from 2002 to 2008. He was also with the Electronics and Telecommunications Research Institute (ETRI) from 1999 to 2002, as a senior member of research staff, and with the IBM T.J. Watson Research Center as an academic visitor from 1998 to 1999. His research interests include embedded systems, storage systems, and operating systems. He is a member of the IEEE and the IEEE Computer Society.

Seungryoul Maeng received the BS degree in electronics engineering from Seoul National University (SNU), Korea, in 1977, and the MS and PhD degrees in computer science in 1979 and 1984, respectively, from Korea Advanced Institute of Science and Technology (KAIST), where he has been a faculty member in the Department of Computer Science since 1984. From 1988 to 1989, he was with the University of Pennsylvania as a visiting scholar. His research interests include microarchitecture, parallel computer architecture, cluster computing, and embedded systems.