Energy-Efficient Storage in Virtual Machine Environments

Lei Ye, Gen Lu, Sushanth Kumar,
Chris Gniady, John H. Hartman

Department of Computer Science, University of Arizona
{leiy, genlu, sushanth, gniady, jhh}@cs.arizona.edu

Abstract

Current trends in increasing storage capacity and virtualization of resources combined with the need for energy efficiency put a challenging task in front of system designers. Previous studies have suggested many approaches to reduce hard disk energy dissipation in native OS environments; however, those mechanisms do not perform well in virtual machine environments because a virtual machine (VM) and the virtual machine monitor (VMM) that runs it have different semantic contexts. This paper explores the disk I/O activities between VMM and VMs using trace driven simulation to understand the I/O behavior of the VM system. Subsequently, this paper proposes three mechanisms to address the isolation between VMM and VMs, and increase the burstiness of hard disk accesses to increase energy efficiency of a hard disk. Compared to standard shutdown mechanisms, with eight VMs the proposed mechanisms reduce disk spin-ups, increase the disk sleep time, and reduce energy consumption by 14.8% with only 0.5% increase in execution time. We implemented the proposed mechanisms in Xen and validated our simulation results.

Categories and Subject Descriptors D.4 [*Operating Systems*]: Storage Management, Organization and Design, Performance

General Terms Design, Experimentation, Management, Measurement, Performance

Keywords Virtual Machine, Energy Management, Storage System

1. Introduction

The benefit of energy management has driven new ideas for designing and developing energy-efficient hard disk hardware and software. Typically, the OS is responsible for shutting down a disk based on an anticipated long idle period. The most common approach is based on a simple timeout that shuts down the disk when a pre-determined amount of time has passed since the last disk I/O. However, shutting the disk down and waking it up again is expensive in both time and energy, therefore it is critical to minimize the number of spin-ups and maximize the length of the idle periods to maximize energy savings and minimize delays. Virtual machines (VM) make the energy management process more challenging. It is difficult to use traditional techniques to reduce energy consumption for disks, because a VM interacts with a virtual machine monitor (VMM) and not the hardware directly. The guest OS in the VM accesses a virtual disk provided by the VMM; the VMM is responsible for multiplexing the underlying physical disk among the virtual disks used by the VMs. The guest OS therefore has no knowledge of the underlying physical disk, and the VMM has no information about the applications running on the guest OSes. There is no single entity that has the complete information necessary to perform efficient energy management.

This paper explores the disk I/O activities between the VMM and the VMs, and investigates the potential opportunities for reducing disk energy consumption in virtual desktops deployed on a VMM. Virtual desktops are used for interactive applications that often have long idle times that can be used successfully for energy management. The challenge grows with the number of VMs that are sharing the same hardware. More VMs implies more users, significantly reducing the number of long idle periods that can be efficiently used for energy management. Therefore the goal of energy management under multiple VMs is to maximize the length of idle periods by batching and shaping requests from multiple VMs before they are sent to the actual hardware.

In this paper, we propose and evaluate mechanisms for reducing disk energy consumption in virtual machine environments by shaping disk accesses. The approach focuses on integrating the VMM and the VMs in managing the energy of the physical disk. The studied mechanisms have been previously implemented in standalone operating systems that control the hardware directly, but haven't been investigated in a VM environment. First, we investigate a buffering approach that buffers write requests from multiple VMs before scheduling them on an actual physical disk. This is done at the VMM level with no changes to the VM. However, extended buffering may reduce the desired data reliability in case of system failure. Subsequently, we evaluate a second approach that maximizes the idle periods by requesting early flushes of dirty data from the buffer cache of each individual VM. This requires small modifications to the VM but does not reduce data reliability in case of system failure. Finally, we develop a trace-driven simulator to evaluate buffering, early flushes, and the interaction between them. We also implement a prototype of the studied mechanisms in the Xen virtual machine [1] platform to validate our simulation.

The structure of this paper is as follows. Section 2 presents the motivation and background for research on energy efficient hard disk accesses in virtual machine environment. Section 3 describes the proposed methods and design. Section 4 provides the implementation details in Xen. Section 5 discusses the methodology. Section 6 evaluates the studied mechanisms through simulation and

VEE'10, March 17–19, 2010, Pittsburgh, Pennsylvania, USA.
Copyright © 2010 ACM 978-1-60558-910-7/10/03. . . $10.00

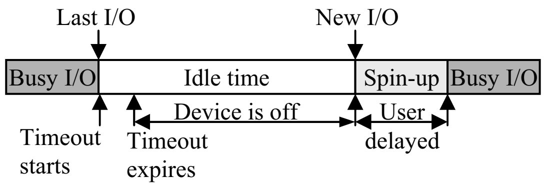

Figure 1. Disk states during a timeout based shutdown.

implementation. Section 7 gives related works on energy saving for hard disks. Finally, Section 8 concludes the paper.

2. Motivation

A VM environment allows a guest operating system and its applications to share resources by multiplexing the physical resources between VMs in a virtual machine monitor (VMM). Virtual machines offer many advantages over physical machines, including reduced cost-of-ownership (by reducing the number of physical machines required to run a collection of servers, for example), strict isolation between machines so that applications running in separate machines cannot interact with each other in unintended ways, increased availability (by allowing virtual machines to be migrated to handle physical machine failures), load balancing, and snapshot functionality that allows the state of a virtual machine to revert to a previous snapshot to "undo" administrative errors, intrusions, and software incompatibilities [6, 17].

The guest operating system running inside a VM interacts with virtual resources such as a network interface, a disk drive, and a processor. The characteristics of the virtual devices can be very generic and quite different from the underlying physical hardware. The virtual device abstractions remove dependencies on the physical hardware, allowing VMs to be more portable and allowing resource sharing, VM migration, and many other features. However, this abstraction also makes it difficult for the guest OS to manage the physical hardware since the underlying physical device may be shared by several virtual devices. For example, what does it mean for a guest OS to shut down a virtual disk if it is one of many virtual disks that share a physical disk? Furthermore, expecting the guest OS to manage energy without knowing the exact specifications of the physical devices may not only significantly reduce the performance, but can also increase energy consumption and running time as a result of poor energy management decisions.

For this work we investigated energy management strategies that allow the VMM to make intelligent decisions on when to shut down the physical disk. In current computer systems, hard disks support multiple power states to reduce the energy consumption. In the *active* mode, the hard disk is actively servicing I/O requests. This is the most energy-intensive mode – a typical hard drive consumes 10W to 12W during normal operations at full RPM [21], which is about 20% of the total PC power consumption. In the *idle* mode, the disk is not servicing requests, but the platters continue to spin. In the *sleep* mode, the platters cease to spin, but the electronics remain active and ready to respond to incoming I/O requests by spinning up the platters and returning to active mode.

The simplest policy for determining when to shut down the disk is based on timeouts – after each I/O request the VMM starts a timer and the disk is shut down if the timer expires before another I/O request is received (Figure 1). When a new I/O request arrives, the disk returns to active mode to service the request. The VMM bases its decision to shut down the disk only on the time since the last I/O. This may not be indicative of the time until the next I/O arrives, so that the VMM may shut down the disk only to have to spin it up immediately to handle a request. This not only adds an unnecessary delay to the next I/O, but also may increase energy consumption. There is a break-even point on the length of sleep interval. Energy is consumed when shutting down and spinning the disk up again, so that if the idle period is too short it is more energy efficient to simply leave the disk spinning than to shut it down.

```
settimer(BUFFER_TIMEOUT, flush_buf)

flush_buf(buf):
    #to deliver all writes in buffer to disk
    for io in buf:
        execute(io);
io_process(io):
    if disk in active:      #process it immediately
        execute(io);
        return;
    If disk_io is READ:
        execute(io);
        flush_buf(buf);
    else:
        add_to_buf(io);
```

Figure 2. Algorithm for write buffering at the VMM level.

To increase the length of the idle periods we introduce two techniques for shaping disk requests: buffering and early flush. With buffering, disk writes that occur while the disk is in sleep mode are delayed for a finite interval before being delivered to the disk. With early flush, pending disk writes are performed prior to shutting the disk down. This may increase the total number of disk I/Os (some pending writes might otherwise have been cancelled before they occurred), but it increases the length of the idle period and therefore the amount of energy is saved. Our results show that both of these techniques are effective at saving energy while having a minimal impact on system response time.

3. Design

Our proposed I/O shaping techniques focus on write requests since they can be delayed or flushed earlier from buffers without impacting the performance of the application. Subsequently, we extend the idea of buffering and early flushes to the virtual machines.

3.1 Buffering in the VMM

The Linux kernel buffer cache stores dirty pages for up to 30 seconds before the *pdflush* daemon flushes them to disk. Longer times can be set by the user to minimize the number of disk spin-ups. However those flushes are not synchronized and can come at random times from multiple VMs. To limit the randomness of the write requests arriving at the physical drive, we implement another buffering mechanism for flushes from individual VMs. The basic algorithm is shown in Figure 2. The buffering is implemented in the VMM and buffers the buffer flushes from the VMs only when the disk is sleeping. The buffer is emptied when the disk becomes active due to a read request arriving at the queue or a preset timeout value expires.

Figure 3 shows an example of I/O requests from the VMs and the states of the physical disk. Before time t1, the disk is in the active or idle state continually. There are no more requests after time t1 so when the timeout expires at time t2 the disk is shut down.

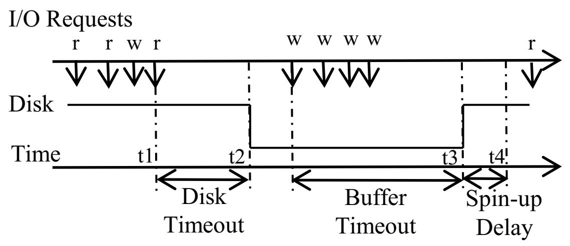

Figure 3. Disk behavior during sleep state with buffering at the VMM level.

Between time t2 and t3, all incoming writes requests are buffered. At time t3, the timeout that started at the first buffered write expires and the buffer is processed once the disk is spun up. This example illustrates that the writes will not be propagated to disk when the guest OS specifies, potentially affecting the data reliability in case of power failure. Data may be buffered in memory an additional 30 seconds after they are written by the guest OS. In addition, some of the writes are caused by `sync` or `fsync` calls and should be written to the disk immediately to preserve the semantics of the system calls. Since the VMM doesn't know what causes a write it treats them all as standard buffer flushes from VMs and are buffered in the VMM. This may affect the behavior of the system and therefore this approach may not be suitable for all applications.

3.2 Early Flush

An alternative approach that does not extend the write buffering time is to flush writes from the VMs early. Standard buffer flushes from VMs are handled on demand and can occur at any time. A write caused by such a buffer flush might arrive just after the disk was shut down, triggering a disk spin-up. To reduce the chance of this happening the VMM can force early flushes of the guest OS buffer caches early and avoid a subsequent disk spin-up, which in turn will prolong idle times and save energy. Figure 4 presents the basic mechanism involved in early flushes. First, the VMM notifies all VMs that the disk shutdown is imminent and they should immediately flush their dirty pages. Then, all VMs with dirty data perform buffer flushes. Finally, the VMM waits for the flushes to complete and shuts down the disk.

Early flushes do not change system call semantics. The only difference is a trigger to the VM to trigger flushes which is a common optimization before the disk is shut down in standalone systems. The reliability is not affected in case of system failures while the idle periods are extended and the disk can remain off longer. This approach does not increase the pressure on the I/O system since it only requests the buffer flushes from VMs when the disk has been idle for the timeout interval. In addition, early flushes do not affect the performance of read requests during the buffer flushes, because read requests are always given highest priority and scheduled first, and the buffer flushes are performed in the background. Accordingly, this approach can be implemented without any adverse effects on system reliability or performance to prolong the disk idle time.

3.3 Early Flushes with Buffering

The buffering and early flush mechanisms are two orthogonal approaches that attempt to maximize the idle time between write requests. The buffering mechanism buffers writes when disk is in sleep mode while the early flush prevents the dirty data from waking up the disk immediately after it is shut down. The early flush

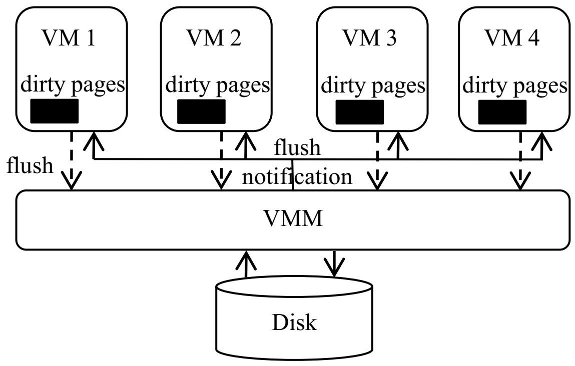

Figure 4. Early flush mechanisms communication between the VMM level and individual VMs.

mechanism can delay a disk spin-up by 30 seconds since this is the amount of time the dirty data is buffered in the cache. Buffering, on the other hand, can extend an idle period by 30 seconds. Therefore, even in a fairly active system the idle times for write activity can be extended to 60 seconds when both early flushes and buffering are combined, assuming `sync` or `fsync` calls in VMs are not present. If `sync` or `fsync` calls are present then buffering may not be feasible and early flushes may be less effective since the guest OS cannot buffer writes that are forced to disk because of a `sync` or `fsync`.

4. Implementation

We implement the discussed mechanisms in Xen 3.3.1 running Linux 2.6.18.8 as the platform. Xen is an open-source virtual machine monitor(VMM) or "hypervisor" for the x86 architecture. Xen can support multiple virtual machines (domains) concurrently on a single physical machine. Domain 0 (Dom0) is privileged and has direct access to the physical devices. The guest OS in Dom0 contains the device drivers for these devices. The rest of the domains are unprivileged (DomU) and access virtual devices. The device drivers for these virtual devices communicate with Dom0, which maps accesses to the virtual devices into accesses to the appropriate physical devices. The device drivers communicate with Dom0 using ring buffers that are shared between DomU and Dom0 [3, 20].

4.1 Buffer Implementation

In normal operation, Dom0 continually removes I/O requests from the ring buffers and processes them. To improve disk scheduling, Dom0 uses a queue of I/O requests to optimize scheduling of disk I/O under heavy load. When an I/O request is received and the queue is empty, Dom0 does not process the request immediately. Instead, it sets a timer with a 4ms timeout. When the timer expires, it sorts the requests in the queue and begins to service them. The idea is that postponing the first request by 4ms allows multiple requests to build up in the queue, improving the effectiveness of sorting the requests. The first request is delayed by 4ms, but this increased latency is considered acceptable relative to the performance improvements gained by sorting the requests.

This behavior is implemented using a flag to indicate whether the queue is "plugged" or "unplugged" (which are equivalent to "deactivated" and "activated", respectively). If a request arrives when the queue is empty, then the queue is "plugged" by function blk_plug_device() and a 4ms timer is started. Any request that arrives before the timer expires is combined with the requests already in the queue, using one of the I/O scheduling algorithms im-

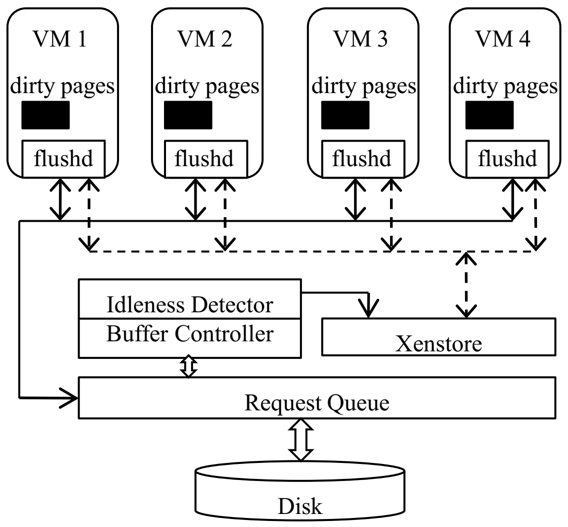

Figure 5. Implementation of buffering and early flushing.

plemented in Linux. The timer will eventually trigger the function blk_remove_plug(), which "unplugs" the queue, initiates the processing of requests in the sorted order, and clears the timer.

We modified the request queue to allow buffering of write requests when the disk is in sleep mode. The queue operates normally when the disk is active, but when the disk is in sleep mode the queue is "plugged", causing write requests to be buffered. If a write request is received while the queue is plugged a timer is started so that write requests are not buffered indefinitely. The queue is unplugged and the write requests are processed when the timer expires, or a read request is received, whichever comes first. This is accomplished through the function blk_disk_up() which invokes blk_remove_plug() and causes the buffered requests to be processed.

4.2 Shutting Down the Disk

In Linux, several utilities control the state of the disk, e.g. hdparm and noflushd, and can be used by normal users without any modification of the system. In general, these utilities will put the disk into a lower power mode if the idle time exceeds a specified threshold. Our idleness detector is similar to those utilities but it provides additional support for the buffering and early flush mechanisms. The disk idleness detector monitors the incoming requests from the block I/O layer in Dom0 when the disk is active. The detector sets a timer after each request is processed. The disk is declared idle if the timer expires before another request is received. The overhead of setting a timer for each request is negligible, as once the timer is active setting it again consists only of changing its timeout value. The overhead of this operation is insignificant compared to the overhead of performing a disk access. By default, the timeout is set to 19 seconds, which is the break-even time for the disk used in our experiments. When the timer expires, the disk is considered idle and put into the sleep mode after notifying the DomUs that the disk is about to be put to sleep so that they can perform early flush, as described in the next section.

4.3 Early Flush

During normal operation the Linux OSes running in the DomUs periodically flush their dirty blocks using the pdflush kernel thread.

VM ID	Number of Reads	Number of Writes	Number of Idle Periods	Total Idle Time [s]
1	62951	1339	255	40071
2	1935	15692	299	41138
3	11438	18754	283	40330
4	6726	23622	252	41136
5	40164	2153	226	40719
6	7055	20159	318	40275
7	15753	3008	155	42109
8	14027	19113	264	40449
2 VMs	64886	17031	435	37470
4 VMs	83050	59407	601	31263
8 VMs	160049	103840	566	20450

Table 1. Disk I/O trace statistics for arriving at the VMM level.

Pdflush is triggered by the kernel in two ways. First, it is triggered periodically (default 5 sec) by the kernel. Second, it is triggered when the ratio of dirty pages in the buffer cache exceeds some threshold (default 10%). Pdflush checks the dirty pages in the buffer cache and flushes those dirty pages that are older than a threshold (default 30 sec). These values are tunable in the proc file system under /proc/sys/vm. To trigger the buffer cache flushes from VMM, Dom0 has to notify each DomUs to invoke pdflush and flush all dirty pages immediately. This type of inter-domain communication in Xen can be accomplished using *xenbus*. Xenbus is typically used to exchange configuration information between domains. The configuration information is stored in a file-system-like database called *xenstore*. All domains can read and write xenstore, making it a natural place to implement the early flush mechanism.

We added a xenstore variable called flush_dirty_pages to let Dom0 notify each DomU that it should flush its dirty pages, and to notify Dom0 when the flushes are complete. We modified the Dom0 kernel to set flush_dirty_pages for each DomU before putting the disk to sleep. The kernel then waits for all the flush_dirty_pages variables to be cleared before putting the disk to sleep. Each DomU kernel monitors its flush_dirty_pages variable by polling xenstore at periodical time and invokes pdflush when it is set. Once the dirty pages have been flushed the kernel clears flush_dirty_pages, notifying Dom0 that the flush is complete, and resets the pdflush timeout.

Figure 5 presents the entire implementation. The buffer controller is responsible for controlling and monitoring the request queue using the buffer timer and the disk shutdown timer. Before putting the disk to sleep the idleness detector will notify the VMs via xenstore. The designed flushd daemon in each VM kernel space flushes all dirty pages in its buffer cache through the normal I/O path.

5. Methodology

In order to analyze and evaluate the proposed mechanisms, we collected disk I/O traces on a modified Xen-enabled Linux kernel. The measurement platform was Xen 3.3.1 and Xen Linux kernel 2.6.18.8 running on an AMD Phenom II X4 940 with 4GB RAM. To avoid recording traces to the local disk and disturbing the I/O behavior of the system we recorded I/O traces on a remote machine. Each VM is traced individually and the system records the following information about each I/O operation: timestamp, access type, access size, domain identifier, process identifier, process name, and file inode. To simulate a remote desktop environment we deployed 8 VMs and installed the X window system in each VM along

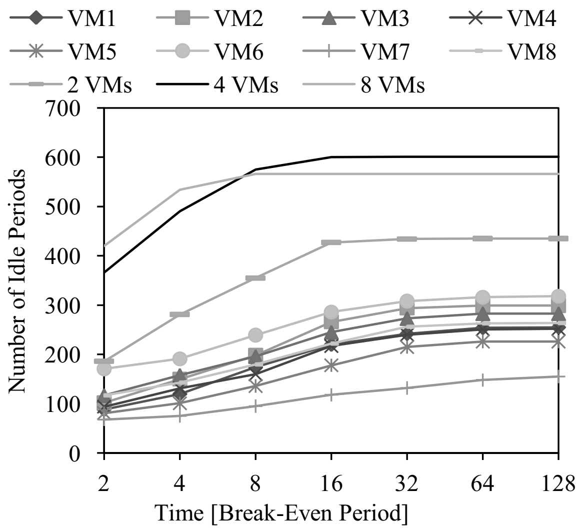

Figure 6. Distribution of Idle Periods that are presented at the VMM level.

with several common desktop applications: firefox, evince, gedit, gthumb, calc, xemacs, and evolution. Firefox is a web browser. Evince is a document viewer supporting multiple document formats. XEmacs and gedit are powerful general purpose text editors. Gthumb is an image viewer and browser for the GNOME desktop. Calc is a small GUI calculator, and Evolution provides integrated mail, address book, and calendar functionality to users.

We selected eight users to interact with eight individual VMs remotely via VNC connections. We traced the eight users concurrently for 12 hours in several separate sessions. The users performed typical daily computing activities utilizing our server and the trace statistics are listed in Table 1. The table lists the details of the trace from each VM, such as the number of read operations, the number of write operations, the number of idle periods greater than the break-even time, and the total idle time greater than the break-even time. It also presents the statistics of trace in multiple VMs environment. The two VMs scenario contains VM1 and VM2; the four VMs scenario contains VM1, VM2, VM3 and VM4; and the eight VMs scenario contains all the eight VMs. As the number of VMs increases the idle periods become shorter, since each additional VM generates disk requests. The number of idle periods initially increases with the number of VMs because longer idle periods are split into multiple shorter idle periods, but once there are eight VMs the number of idle periods decreases because of the volume of disk requests.

Figure 6 shows the cumulative distribution of the number of idle periods greater than the break-even time (BE) for each individual domain. The x-axis indicates the length of the idle period in BE units, and the y-axis indicates the cumulative number of idle periods. For all VMs, most of the idle periods are between 1 BE and 16 BEs. Combination of the traces will subsequently result in some subset of the original idle periods. As the number of concurrent VMs increases, the longer idle periods are converted into shorter periods resulting in fewer energy saving opportunities.

We developed a trace-driven simulator to simulate multiple VMs and generate the final I/O traffic at the VMM. We used the simulator to compare the mechanisms and evaluate the energy efficiency of each mechanism. To obtain the encountered delays and energy calculations from the simulator we used specifications for

State	Power
Read/Write	10.6W
Seek	13.3W
Idle	10.0W
Standby	1.8W
State Transition	**Energy**
Spin-up	148.5J
Shutdown	6.4J
State Transition	**Time**
Spin-up	9.0 sec.
Shutdown	4.0 sec.

Table 2. Energy consumption and delay specifications for WD2500JD.

Western Digital Drive Model WD2500JD as shown in Table 2 [7]. The disk has significant energy consumption and latency associated with the spin-up after sleep, which is common for high performance desktop drives. The corresponding break-even time, is approximately 19 seconds and we set the timeout before putting the disk to sleep to the break-even time. The number of traces supplied to the simulator can be varied and the simulator emulates the I/O traffic generated by the specified number of domains.

6. Evaluation

In this section, we examine how the performance and energy efficiency of the studied mechanisms changes when we increase the number of virtual machines.

6.1 Reducing Spin-ups

Figure 7 presents the distributions of causes for disk spin-ups for different mechanisms and different number of concurrent virtual machines. Three factors can cause a disk spin-up: (1) disk read will result in immediate disk spin-up in all mechanisms; (2) disk write will result in disk spin-up when the VMM is not buffering writes; (3) buffer flush will spin-up the disk when a write has been buffered for the maximum buffering time. Therefore, we have only two causes for disk spin-ups in the studied mechanisms. The case of one VM is comparable to the studied mechanisms running in a standalone operating system. In this case, the idle times are long and the number of spin-ups is lower. The majority of the spin-ups are due to reads since most of the writes that follow previous I/O activity are already flushed. In this scenario, the buffer cache flush time in the guest operating system is set to 30 seconds. Some flushes may still occur if they are not performed before the disk is shut down. Buffering does not offer much benefit since it can at most prolong the pending write by additional 30 seconds and the idle times in the case of only one VM are significantly longer. As a result, almost all write flushes from the VM result in a buffer flush in the VMM. Early flushes eliminate most disk spin-ups that were generated by the pending flushes from the VM, eliminating a significant fraction of spin-ups. The combination of the two techniques flushes pending writes before shutting down the disk, and delays the disk spin-up by up to 30 seconds by buffering any writes that occur while the disk is in the sleep mode.

The I/O activity in the VMM increases as the number of VMs increases. There are two potential outcomes: (1) the idle periods are shorter due to interleaving of I/O from different VMs, which results in more spin-ups; or (2) the I/O activity and idle times have some overlap that can result in longer active times and as a result fewer idle periods that are longer than the break-even time. It is important

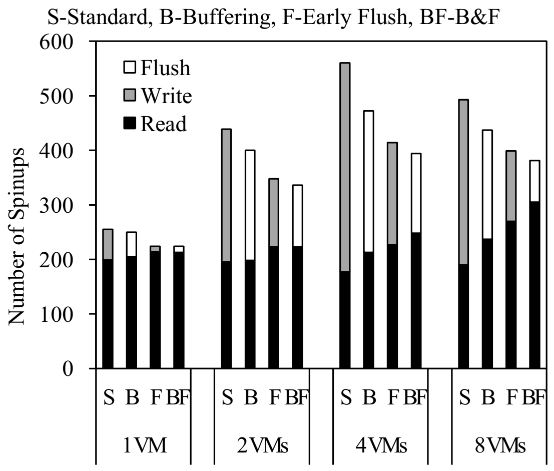

Figure 7. Breakdown of the disk spin-ups triggers for studied mechanisms.

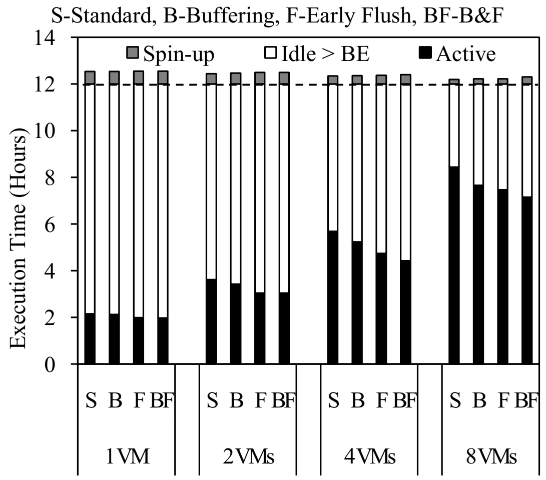

Figure 8. Breakdowns of execution time for studied mechanisms.

to note that results from the single VM cannot be extrapolated to other experiments as we include more user traces that have different behaviors. Therefore, we can only make general comparisons between results for different number of VMs and detailed comparisons between the mechanisms for the given number of VMs.

When two VMs run concurrently, there is a significant increase in spin-ups due to write activity, for the reasons noted above. Buffering can help in reducing spin-ups and early flushes offer a larger benefit. In this case, early flush is preferred since it reduces the spin-ups and does not delay writes, which may impact reliability in case of failures. The trends are similar for four VMs. In this case, buffering reduces larger fraction of spin-ups. The reason is that the delayed writes resulted in some aggregation of I/O activity. There is also a small increase in number of spin-ups due to reads. This is expected since buffering and early writes cause fewer writes to spin-up the disk. Therefore, the side effect is a potential increase in read latencies. However, these latencies are spread across all the users, so when there are four VMs each user will experience on average 1/4 of the delays. Finally, with eight VMs the number of spin-ups decreases since increased rate of I/O requests keeps the disk active longer resulting in fewer shutdowns.

6.2 Execution Time

Figure 8 presents the breakdown of total execution time in different scenarios. The execution time is composed of the time the disk is servicing I/O requests, idle time (less than break-even time) when the disk is waiting for I/Os to arrive during timeouts, the spin-up time required to return the disk to active mode after it has been in sleep mode, and disk idle time that is greater than break-even time and will result in a shutdown. The time spent servicing I/O requests is a relatively small portion of the overall execution time so we combined it with the disk idle time and refer the combination as the "disk active time". The shutdown time is part of idle time greater than break-even time since in the majority of cases it is hidden from the execution. In some cases, the shutdown results in delays because the disk has to be completely shut down before it can spin up again. Therefore, a read request that arrives during shutdown must wait for the shutdown to complete followed by a spin-up before it can be serviced and include these exposed delays in the spin-up time.

When only one VM is running, the majority of the execution time is occupied by the long periods of idleness. There are 199 spin-ups caused by reads, each of which is delayed nine seconds while the disk spins up. These delays increase the nominal 12 hour execution time by 31 minutes, or less than 5%. We will consider delay-hiding techniques in our future work with the hope of significantly reducing or eliminating these delays. As the number of VMs increases the total execution time decreases. This is because the disk is in the active mode more, which reduces the number of spin-ups. Energy savings correspondingly decrease.

Write buffering is effective at increasing long idle times as the number of VMs increases. Write buffering can extend idle periods by up to 30 seconds, unless a read request arrives, which can potentially translate into longer sleep times and higher energy savings. Early flushes show better improvement in extending the length of idle periods by significantly reducing the number of required spin-ups. There is a clear benefit when buffering and early flushes are combined. With eight VMs the combined mechanism extends the sleep time by 6.8% as compare to early flushes alone, and by 36.3% when compare to standard shutdown mechanisms in the VMM.

6.3 Energy Consumption

Figure 9 presents the breakdown of total energy consumption in different scenarios. The energy consumption is composed of the energy consumed while servicing I/Os, energy consumed in the active mode while waiting for I/Os to arrive, the power-cycle energy to spin down and spin up the disk, and the disk sleeping energy. The energy consumed to perform I/Os is relatively small in the studied applications and similarly to I/O time we include it as a part of idle-active energy and call the combined value "disk active energy". The energy closely follows the execution time breakdown as shown in Figure 8. The longer the disk sleeps, the larger the energy savings. Furthermore, fewer disk spin-ups reduce the power-cycle energy.

Similarly to the improvements in idle time that is greater than break-even time, the best energy efficiency is presented by a combination of buffering and early flushes. With four VMs buffering and early flushes reduce energy consumption by 18.3% as compared to the standard shutdown mechanism in VMM and by 4.2% when only early flushes are present. With eight VMs buffering and

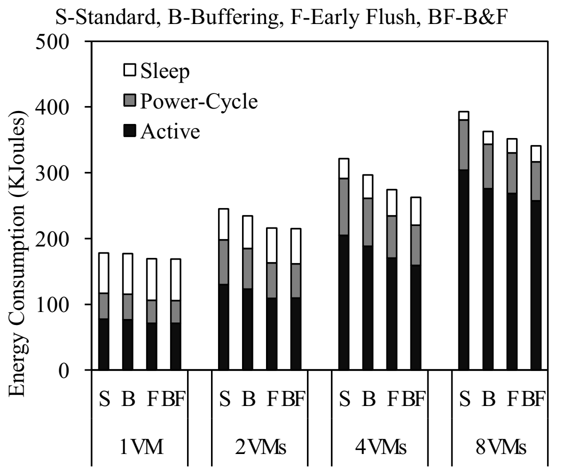

Figure 9. Breakdowns of energy consumption for studied mechanisms.

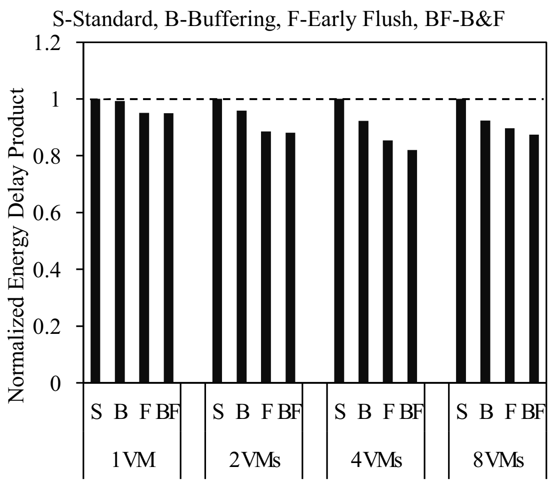

Figure 10. Normalized Energy-Delay Product for studied mechanisms.

early flushes reduce energy consumption by 13.3% as compared to the standard shutdown mechanism in the VMM and by 3.1% when only early flushes are present. While buffering offers some additional energy savings to early flushes, the additional benefit may be offset by reduced reliability in case of crashes. Therefore, using early flushes may be the best option that does not sacrifice reliability and still offers high energy savings.

6.4 Energy Delay Product

Energy saving and performance are competing optimizations. Shutting down the disk saves energy but at the same time it introduces spin-up delays that increase the execution time. One way of quantifying both the performance and energy impact is the *energy-delay product* [10]. A lower energy-delay product indicates better combination of performance and energy optimizations. Figure 10 presents the energy-delay product for the system configurations we studied. The results are normalized to the standard timeout mechanism for each configuration. Figure 8 shows a limited impact on the execution time; as a result, improvements in energy savings dominate the energy-delay product and the trends in Figure 10 are similar to the energy saving trends in Figure 9. The combination of buffering and early flushes has the lowest energy-delay product. When extension in buffering times is problematic due to impact on reliability, the early flush mechanism is a promising alternative.

6.5 Optimizing Buffer Flush Time

The results so far use a uniform timeout period before the disk is shut down which does not consider the source of the I/Os. In this section we do not wait for a timeout before putting the disk to sleep after a buffer flush; instead, the disk is put to sleep immediately. Other spin-ups due to reads from the application indicate application I/O activity and the disk is kept active for a full timeout interval. This change only impacts shutdowns in those configurations that use buffering. The early flush mechanism is not affected since there is no additional buffering in the VMM.

Figure 11 compares the disk spin-ups for the regular timeout(T) and immediate shutdown(I) for the buffering mechanisms. In general, there is very little change in number of spin-ups due to reads when the timeout period after a disk flush is eliminated. This leads to several conclusions. First, disk flushes have low correlation to the actual disk I/O activity. Second, the users will experience similar numbers of spin-up delays. Finally, there is an increase (max 10.4%) in the number of spin-ups due to writes when using the buffering mechanism only. When buffering is combined with early flushes the number of spin-ups is almost the same. Furthermore, the additional write spin-ups will not result in any increases in application execution time; however the additional power-cycles may increase energy consumption.

Figure 12 compares the execution time breakdown for the regular timeout(T) and immediate shutdown(I) for the buffering mechanisms. As expected, the minor differences in the number of spin-ups caused by reads in Figure 11 have limited impact on the delays due to spin-ups, with 5.5% in case of two VMs and negligible differences for four and eight VMs. The elimination of timeout interval for shutdown after the buffer flush results in shorter active times and longer idle times, which translates into higher energy savings.

Finally, Figure 13 shows the energy consumption breakdown for the regular timeout(T) and immediate shutdown(I) for the buffering mechanisms. The increase in idle time available for shutdowns results in longer sleep times and higher energy savings. However, Figure 11 shows more disk spin-ups, mainly due to reads, that will translate into increase in energy consumption to perform the additional power-cycles. On average, immediate shutdown during buffer flushes in the VMM results in a 3.3% improvement in energy consumption for buffering only, and a 2.8% improvement for combined buffering and early flushes. Therefore, if buffering is done in the VMM, the disk should be shut down immediately following the buffer flushes since this improves in energy savings without significantly increasing delay.

6.6 Writes in Buffering and Early Flush

Buffering of writes in the VMM can result in additional delays before the writes are committed to the disk. This in turn may affect reliability of the system in case of power failures or system crashes. The longer the write resides the buffer, the higher the impact on reliability. Table 3 presents the average number of writes and their delay in the buffer before the buffer flush. The average varies between 10.6 to 15.5 seconds. The average number of writes operations in the buffer is relatively small varying between 3.5 for one VM and 16.1 for eight VMs. With eight VMs there are

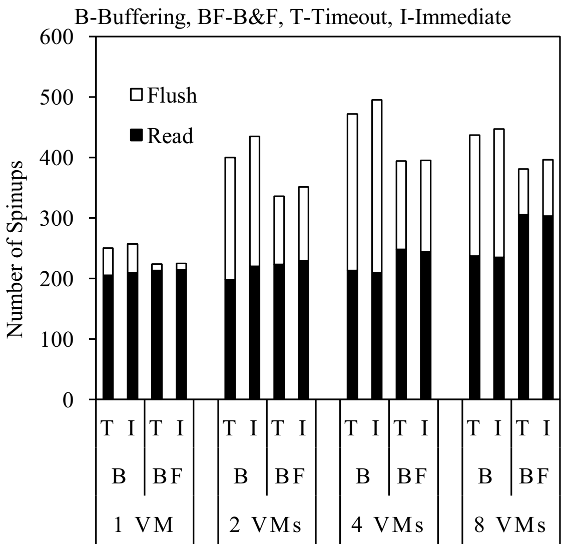

Figure 11. Comparison of disk spin-ups under immediate shutdown following the buffer flush.

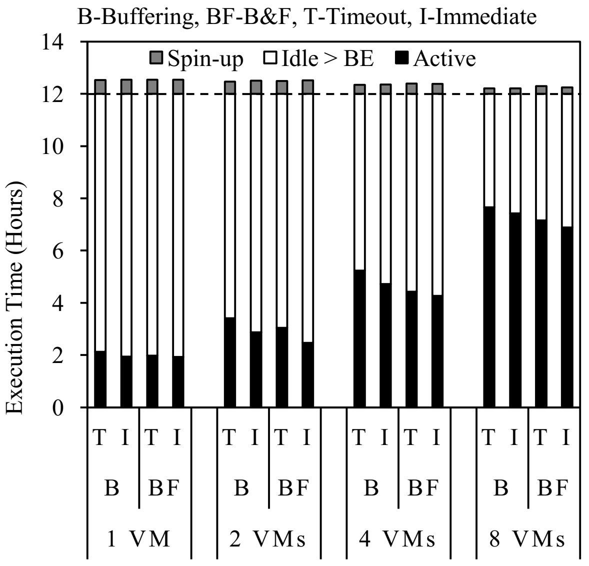

Figure 12. Comparison of execution time breakdown under immediate shutdown following the buffer flush.

slightly more than two pending writes per VM. Therefore, very few writes would be affected by a crash, and the impact on reliability is limited. In addition, Table 3 shows the number of VM that are involved in buffer cache flushes triggered by early flushes from the VMM, which is little more than one. This is not a surprising result since the timeout mechanism will wait for the last VM to finish the I/O before waiting an additional timeout period and shutting down the disk. During that time, other VMs have finished I/O activity earlier and flushed the buffer cache; therefore, most of the time only the last VM that has I/O activity has to be flushed.

6.7 Implementation Measurement

In order to evaluate our implementation we replayed the traces used in the simulations on a Xen platform with a kernel modified to buffer writes in the VMM and to flush the guest OS buffer caches early. A replay driver in each guest OS reads the entire trace into the memory and replays the I/O operations at the proper times. The entire trace is read into memory to eliminate any additional I/O operations due to reading the trace during the replay. Due to time constraints, we omitted buffering only implementation and only replayed first six hours of the traces for four concurrent VMs.

Figure 14 presents the time the disk spends in three states: active, power-cycle, and sleep. This is not the same as the application execution time shown in Figure 8, but the actual time the disk spends in each state. In the case of standard shutdown mechanism, the disk spent 35% of the total disk time in the sleep state. When early flushes are introduced into the execution the sleep time increases to 45% of total disk time. Finally, when we combine early flushes with buffering the sleep time increases to 56% of the total disk time . The power-cycle time is significant, however this time is not directly exposed to the applications and only a few spin-ups result in delays as shown in Figure 8. Finally, the disk time from Figure 14 can be directly translated to energy consumption by multiplying the time in each state by the corresponding power demand.

7. Related Work

Energy management for hard disks has been extensively studied on standalone platforms. There have been considerable research achievements in saving energy for disks within the context of operating systems and applications. Zedlewski et al. [22] model the hard disk power consumption, and propose an approach to accurately estimate the energy consumed by each of several sub-components of a disk request. They point out the analysis of the power mode transition is important. In our simulation experiment, we also consider different power modes and mode transitions, like spin-up and spin-down. Li et al. perform a quantitative analysis of the pros and cons of spinning down disks, and address when the disk should be put to sleep to minimize energy consumption [12]. They find the optimal spin-down delay time is 2 seconds. This result is very similar to our experiment on Section 6.5, in which immediate shutting down improves energy saving. Chung et al. [4] propose adaptive approach based on sliding windows and two-dimensional linear interpolation to address the problem of power managing a single disk.

Zhu et al. propose many power-aware storage cache management policies which can significantly reduce disk energy consumption [23]. Their approach focuses on cache layer, and studies offline power-aware greedy (OPG) algorithm, PA-LRU, and different cache write policies on disk energy consumption. Our proposed mechanisms also improve energy dissipation of disk accesses by controlling buffer cache flush. Early flush is useful to save disk energy in that it signals each VM to trigger buffer cache flush before shutting down disk. Buffering saves disk energy by holding flushed dirty pages for a preset timeout. The idea of buffering writes and performing periodic updates has already been studied [2, 14] and this approach has been implemented in many operating systems. The operating system could defer writing dirty pages to disk within the predefined time period and put disk into low power mode, thus reducing disk traffic and overhead and reducing disk energy consumption.

In addition, the timeout mechanism has gained wide popularity due to its simplicity. After the last request, a timer is started and the

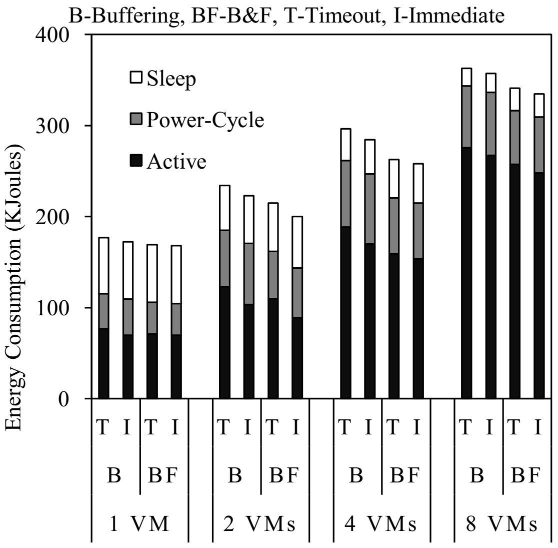

Figure 13. Comparison of energy consumption breakdown under immediate shutdown following the buffer flush.

VMs	Policy	Average Time of Writes in Buffering	Average Number of Writes in Buffering	Average Number of VMs in Early Flush
1	B	14.5s	3.5	0.0
	F	0.0s	0.0	1.0
	BF	10.6s	1.9	1.0
2	B	13.8s	8.3	0.0
	F	0.0s	0.0	1.1
	BF	14.0s	9.2	1.0
4	B	13.8s	11.3	0.0
	F	0.0s	0.0	1.2
	BF	15.5s	9.8	1.2
8	B	13.9s	14.1	0.0
	F	0.0s	0.0	1.4
	BF	13.2s	16.1	1.5

Table 3. Detailed statistics for buffering and early flush mechanisms.

hard disk is shut down once the timer expires unless additional disk activity occurs during the timeout period. Once the new request arrives, the disk platters have to be accelerated before the request is served. Douglis et al. [8] describe a method for varying the spin-down threshold dynamically by adapting to the user's access patterns, which results in up to a 50% reduction in disk spin-ups. Golding et al. [9] show that there are many opportunities to power down the disk during idle periods. They construct several idle-time detection algorithms, which can be used to decrease disk energy consumption. They point out that power consumption will decrease if the savings from the powered-down mode outweigh the power cost of spinning it up again. Similarly, we use the break-even time as the uniform timeout period to determine whether or not to shut down the disk in our experiment. The break-even time decreases unnecessary power transitions and ensures that spinning the disk down will benefit energy saving.

Additional characteristics about application execution can lead to more aggressive dynamic predictors, which shut down the hard disks much earlier than the timeout mechanisms. Dynamic predictors observe recent history of application behavior and use that information to predict the length of the idle period and shut down the disk accordingly [5, 11, 18]. Papathanasiou et al. describe an automatic system that monitors the past application behavior in order to generate appropriate prefetching hints, and a general system of kernel enhancements that coordinate I/O activity across all running applications. They report that this method could save disk energy about 60-80% [16]. A detailed study and evaluation of predictors is presented by Lu et al. [13] who come to the following conclusions: (1) Timeout predictors offer good accuracy, but waiting for the timeout to expire consumes energy; (2) Dynamic predictors shut down the device immediately, but until recently, they have had much lower accuracies than the simple timeout prediction; (3) Stochastic methods usually require off-line preprocessing, which are more difficult to implement, and may run into problems when the workload changes.

Recent research on energy management for hard disks has focused on the virtualized computing platforms. Stoess et al. [19] designed a hierarchical framework to partition energy and account energy-aware resources. Their disk energy model distinguishes two different power states, namely active and idle, and their prototype is capable of enforcing power limits for guest OSes. Nathuji et al. implement VirtualPower Management for Xen hypervisor to support online power management, which provides up to 34% improvements in power consumption [15]. Compared with their works, our proposed mechanisms explore the existing buffer flush approaches to reduce the energy consumption for hard disks in virtual machine environments.

8. Conclusions

Energy management in virtualized computing environment requires the cooperation between VMM and VMs. VMs reflect the usage information of virtual devices and VMM keeps a global view to control the physical hardware. In this paper, we have presented an energy-efficient design for accessing a hard disk in multi-VMs environment. The energy-efficient storage maximizes the length of disk idle periods through batching and shaping requests from multiple VMs, effectively reducing energy consumption.

We have explored techniques for reducing the energy dissipation of a hard disk and proposed three mechanisms to increase the burstiness of hard disk accesses which increases the length of disk idle periods and therefore decreases the energy consumption. These mechanisms include: (1) a buffering mechanism in the VMM that buffers writes from the VMs to improve scheduling and enhance burstiness; (2) an early flush mechanism that flushes the dirty pages from the guest OS buffer caches prior to putting the disk to sleep; and (3) a combined buffering and early flush mechanism that maximizes burstiness and increases the length of idle times.

Our evaluation of these mechanisms, using desktop interactive applications, showed significant improvements in the energy efficiency. These gains come from prolonging the disk sleeping time, which can be translated into less energy consumption. We also showed that these mechanisms can perform very well with the increasing number of VMs. By using our developed trace-driven simulator, different mechanisms under different configurations have been compared. With eight VMs, the combined buffering and early flushes mechanism, based on uniform timeout shutdown, improves energy savings by 13.3% compared with standard disk shutdown techniques. With further optimization provided by immediate shutdown, the combined mechanism improves energy savings by 14.8%

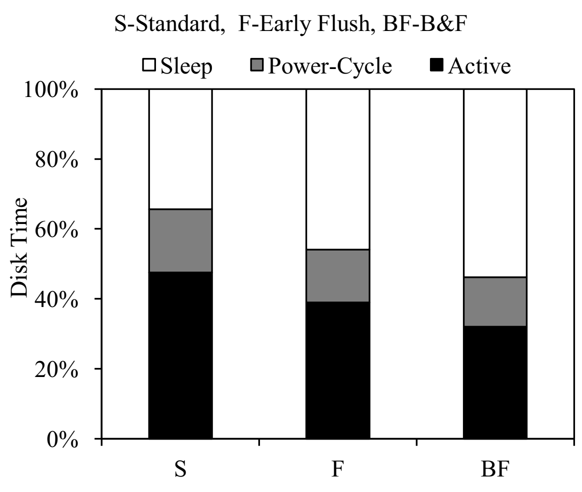

Figure 14. Disk time distribution measurements in the implementation.

and reduces the number of spin-ups by 19.7% as compared to standard disk shutdown techniques in case of eight VMs. The prototype implementation on Xen shows the effectiveness of our proposed mechanisms.

Future research will consider the use of sophisticated prediction techniques to improve the accuracy and timeliness of shutdown prediction in virtual machine environments. Our research will also focus on multiple disks used by multiple virtual machines at the same time. The different energy saving policies and algorithms would be studied and implemented. We also plan to research energy saving for hard disks in IO-intensive scenarios, such as database systems running in VMs, and SANs shared by VMs.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments, which help us to improve the final version of this paper. This research was funded by the National Science Foundation under Grant No. 0834179.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[2] S. D. Carson and S. Setia. Analysis of the periodic update write policy for disk cache. *IEEE Transactions on Software Engineering*, 18(1):44–54, 1992.

[3] D. Chisnall. *The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[4] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli. Dynamic power management for nonstationary service requests. *IEEE Transactions on Computuing*, 51(11):1345–1361, 2002.

[5] E.-Y. Chung, L. Benini, and G. De Micheli. Dynamic power management using adaptive learning tree. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design*, pages 274–279, Piscataway, NJ, USA, 1999. IEEE Press.

[6] I. D. Craig. *Virtual Machines*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[7] I. Crk and C. Gniady. Network-aware program-counter-based disk energy management. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SCI 209*, pages 1860–949X. Springer Berlin, 2009.

[8] F. Douglis, P. Krishnan, and B. N. Bershad. Adaptive disk spin-down policies for mobile computers. In *MLICS '95: Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, pages 121–137, Berkeley, CA, USA, 1995. USENIX Association.

[9] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *UWTC' 95: Proceedings of the USENIX Winter Technical Conference*, pages 201–212, 1995.

[10] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, 1996.

[11] C.-H. Hwang and A. C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):226–241, 2000.

[12] K. Li, R. Kumpf, P. Horton, and T. Anderson. A quantitative analysis of disk drive power management in portable computers. In *WTEC'94: Proceedings of the USENIX Winter Technical Conference*, pages 279–291, 1994.

[13] Y.-H. Lu, E.-Y. Chung, T. Šimunić, L. Benini, and G. De Micheli. Quantitative comparison of power management algorithms. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 20–26, New York, NY, USA, 2000. ACM.

[14] J. C. Mogul. A better update policy. In *USTC '94: Proceedings of the USENIX Summer Technical Conference*, pages 99–111, 1994.

[15] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *SOSP '07: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pages 265–278, New York, NY, USA, 2007. ACM.

[16] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *ATC '04:Proceedings of the USENIX Annual Technical Conference*, pages 255–268, 2004.

[17] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[18] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on Very Large Scale Integration Systems*, 4(1):42–55, 1996.

[19] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2007. USENIX Association.

[20] W. von Hagen. *Professional Xen Virtualization*. Wrox Press Ltd., Birmingham, UK, 2008.

[21] William Van Winkle. Reseller advocate magazine issue 84. `http://www.reselleradvocate.com/public/ram/eram/84/feature1.html`, 2009.

[22] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[23] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pages 118–129, Washington, DC, USA, 2004. IEEE Computer Society.