Write Off-Loading: Practical Power Management for Enterprise Storage

Dushyanth Narayanan Austin Donnelly Antony Rowstron
Microsoft Research Ltd.
{*dnarayan,austind,antr*}*@microsoft.com*

Abstract

In enterprise data centers power usage is a problem impacting server density and the total cost of ownership. Storage uses a significant fraction of the power budget and there are no widely deployed power-saving solutions for enterprise storage systems. The traditional view is that enterprise workloads make spinning disks down ineffective because idle periods are too short. We analyzed block-level traces from 36 volumes in an enterprise data center for one week and concluded that significant idle periods exist, and that they can be further increased by modifying the read/write patterns using *write off-loading*. Write off-loading allows write requests on spun-down disks to be temporarily redirected to persistent storage elsewhere in the data center.

The key challenge is doing this transparently and efficiently at the block level, without sacrificing consistency or failure resilience. We describe our write off-loading design and implementation that achieves these goals. We evaluate it by replaying portions of our traces on a rack-based testbed. Results show that just spinning disks down when idle saves 28–36% of energy, and write off-loading further increases the savings to 45–60%.

1 Introduction

Power consumption is a major problem for enterprise data centers, impacting the density of servers and the total cost of ownership. This is causing changes in data center configuration and management. Some components already support power management features: for example, server CPUs can use low-power states and dynamic clock and voltage scaling to reduce power consumption significantly during idle periods. Enterprise storage subsystems do not have such advanced power management and consume a significant amount of power in the data center [32]. An enterprise grade disk such as the Seagate Cheetah 15K.4 consumes 12 W even when idle [26], whereas a dual-core Intel Xeon processor consumes 24 W when idle [14]. Thus, an idle machine with one dual-core processor and two disks already spends as much power on disks as processors. For comparison, the 13 core servers in our building's data center have a total of 179 disks, more than 13 disks per machine on average.

Saving power in storage systems is difficult. Simply buying fewer disks is usually not an option, since this would reduce peak performance and/or capacity. The alternative is to spin down disks when they are not in use. The traditional view is that idle periods in server workloads are too short for this to be effective [5, 13, 32]. In this paper we present an analysis of block-level traces of storage volumes in an enterprise data center, which only partially supports this view. The traces are gathered from servers providing typical enterprise services, such as file servers, web servers, web caches, etc.

Previous work has suggested that main-memory caches are effective at absorbing reads but not writes [4]. Thus we would expect at the storage level to see periods where all the traffic is write traffic. Our analysis shows that this is indeed true, and that the request stream is write-dominated for a substantial fraction of time.

This analysis motivated a technique that we call *write off-loading*, which allows blocks written to one volume to be redirected to other storage elsewhere in the data center. During periods which are write-dominated, the disks are spun down and the writes are redirected, causing some of the volume's blocks to be off-loaded. Blocks are off-loaded temporarily, for a few minutes up to a few hours, and are reclaimed lazily in the background after the home volume's disks are spun up.

Write off-loading modifies the per-volume access patterns, creating idle periods during which all the volume's disks can be spun down. For our traces this causes volumes to be idle for 79% of the time on average. The cost of doing this is that when a read occurs for a non-off-loaded block, it incurs a significant latency while the disks spin up. However, our results show that this is rare.

Write off-loading is implemented at the block level and is transparent to file systems and applications running on the servers. Blocks can be off-loaded from any volume to any available persistent storage in the data center, either on the same machine or on a different one. The storage could be based on disks, NVRAM, or solid-state memory such as flash. Our current hardware does not have flash or other solid-state devices and hence we used a small partition at the end of each existing volume to host blocks off-loaded from other volumes.

Write off-loading is also applicable to a variety of storage architectures. Our trace analysis and evaluation are based on a Direct Attached Storage (DAS) model, where each server is attached directly to a set of disks, typically configured as one or more RAID arrays. DAS is typical for small data centers such as those serving a single office building. However, write off-loading can also be applied to network attached storage (NAS) and storage area networks (SANs).

A major challenge when off-loading writes is to ensure consistency. Each write request to any volume can be off-loaded to one of several other locations depending on a number of criteria, including the power state and the current load on the destination. This per-operation load balancing improves performance, but it means that successive writes of the same logical block could be off-loaded to different destinations. It is imperative that the consistency of the original volume is maintained even in the presence of failures. We achieve this by persisting sufficient metadata with each off-loaded write to reconstruct the latest version of each block after a failure.

This paper makes two main contributions. First, we show that contrary to conventional wisdom and current practice, idle periods in enterprise workloads can be exploited by spinning disks down, for power savings of 28–36%. Second, we present *write off-loading* as a generic and practical approach that allows further reduction of power consumption in storage systems and also eliminates the spin-up penalty for write requests. In our trace-based evaluation on a rack-mounted testbed, write off-loading enabled energy savings of 45–60%. The performance of all write requests, and 99% of read requests, was equivalent to that when not spinning disks down.

The rest of the paper is organized as follows. Section 2 presents an analysis of block-level traces from an enterprise data center, which motivates write off-loading. Section 3 describes the design and implementation of the write off-loading infrastructure. Section 4 presents an evaluation of write off-loading on a rack-based hardware testbed. Section 5 discusses related work, and Sections 6 and 7 conclude the paper.

Server	Function	#volumes
usr	User home directories	3
proj	Project directories	5
prn	Print server	2
hm	Hardware monitoring	2
rsrch	Research projects	3
prxy	Firewall/web proxy	2
src1	Source control	3
src2	Source control	3
stg	Web staging	2
ts	Terminal server	1
web	Web/SQL server	4
mds	Media server	2
wdev	Test web server	4

Table 1: Data center servers traced (13 servers, 36 volumes, 179 disks)

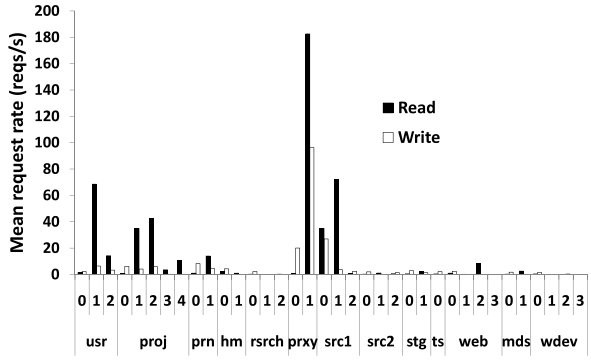
2 Volume Access Patterns

The traditional view is that spinning disks down does not work well for server workloads [5, 13, 32]. Gurumurthi et al. [13] show that for disk traffic patterns generated by the TPC-C and TPC-H benchmarks, spinning down disks is ineffectual: the periods of idleness are too short. Zhu et al. [32] also use the *cello* block-level volume traces [22] collected from a single file/compute server at HP Labs. These are not necessarily representative of all server workloads in enterprise data centers. Many enterprise servers are less I/O intensive than TPC benchmarks, which are specifically designed to stress the system under test. Enterprise workloads also show significant variation in usage over time, for example due to diurnal patterns.

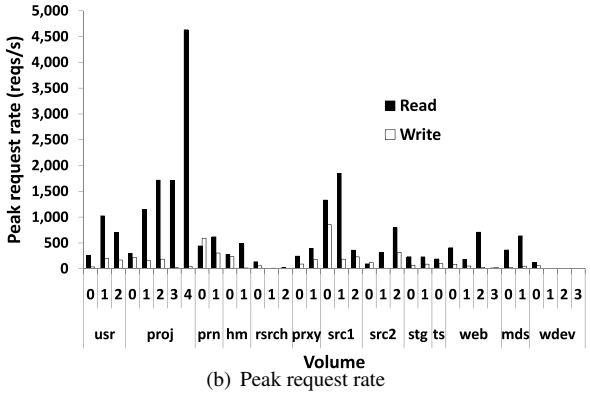
In order to understand better the I/O patterns generated by standard data center servers, we instrumented the core servers in our building's data center to generate per volume block-level traces for one week. Table 1 describes the servers that we traced: most of these are typical of any enterprise data center. In total, we traced 36 volumes containing 179 disks on 13 servers.

The data center is air-conditioned and the servers are high-end rack-mounted machines. The default configuration is for each server to have two internal physical disks configured as a RAID-1 array, which is used as the boot volume. Each server is additionally configured with one or more RAID-5 arrays as data volumes: the storage for these is provided using rack-mounted co-located DAS. All the servers run the Windows Server 2003 SP2 operating system. Data on the volumes is stored through the NTFS file system and accessed by clients through a variety of interfaces including CIFS and HTTP.

We believe that the servers, data volumes, and their access patterns are representative of a large number of



(a) Mean request rate



(b) Peak request rate

Figure 1: Mean and peak request rates per volume over 7 days

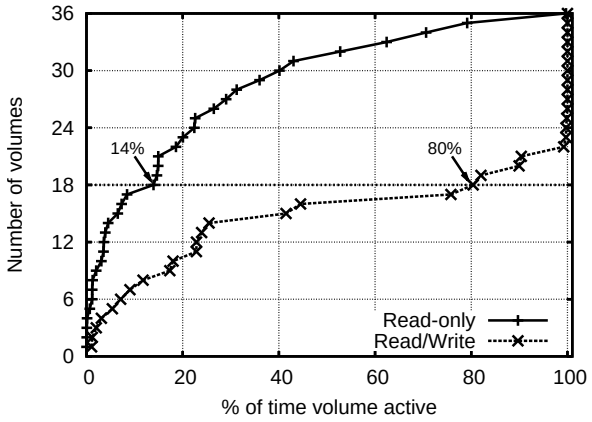


Figure 2: CDF of active time per volume

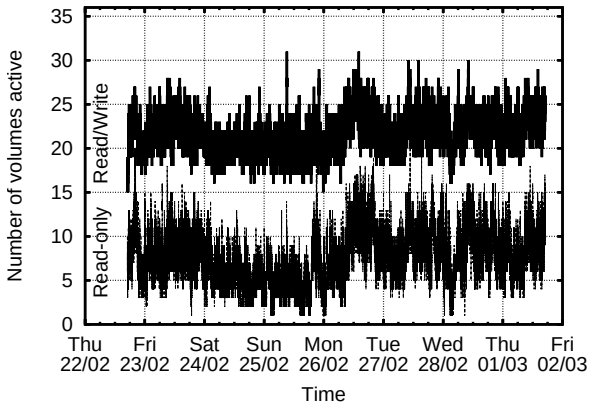


Figure 3: Number of active volumes over time

small to medium size enterprise data centers. Although access patterns for system volumes may be dependent on, for example, the server's operating system, we believe that for data volumes these differences will be small.

The traces are gathered per-volume below the file system cache and capture all block-level reads and writes performed on the 36 volumes traced. The traced period was 168 hours (1 week) starting from 5PM GMT on the 22nd February 2007. The traces are collected using Event Tracing For Windows (ETW) [17], and each event describes an I/O request seen by a Windows disk device (i.e., volume), including a timestamp, the disk number, the start logical block number, the number of blocks transferred, and the type (read or write). ETW has very low overhead, and the traces were written to a separate server not included in the trace: hence we do not believe that the tracing activity perturbed the traced access patterns. The total number of requests traced was 434 million, of which 70% were reads; the total size of the traces was 29 GB. A total of 8.5 TB was read and 2.3 TB written by the traced volumes during the trace period.

Figure 1(a) shows the average read and write request rate over the entire week for each of the volumes. There is significant variation across volumes, with volumes for the file servers, the source version control servers, and the web proxy all having significant read load. However, many of the volumes such as the research projects server and the test web server have low read and write load. Figure 1(b) shows the peak read and write rates, measured at a 60-second granularity for the 36 volumes. Peak loads are generally much higher than the mean load, indicating that while volumes may be provisioned for a high peak load, most of the bandwidth is unused most of the time.

Overall, the workload is read-dominated: the ratio of read to write requests is 2.37. However, 19 of the 36 volumes have read/write ratios below 1.0; for these volumes the overall read-write ratio is only 0.18. Further analysis shows that for most of the volumes, the read workload is bursty. Hence, intuitively, removing the writes from the workload could potentially yield significant idle periods.

Figure 2 confirms this intuition. It shows a cumulative distribution function across volumes of the number of volumes versus the percentage of time that the volume is active over the course of a week. We show the distribu-

	Mean	Median	99th pctile	Max
Read/ write	21.7 (60%)	22 (61%)	27 (75%)	31 (86%)
Read-only	7.6 (21%)	7 (19%)	15 (42%)	22 (61%)

Table 2: Number of concurrently active volumes: numbers in parentheses show the number of active volumes as a percentage of the total number of volumes (36)

tion both for the original trace (read/write) as well as the trace with the writes removed. In both cases we consider the volume to be idle (i.e., not active) when 60 seconds have elapsed since the last request.

Figure 2 shows that even without removing the writes, there is significant idle time for the volumes. As expected, the write workload has a large impact on the length of the idle periods. When the write load is removed, the mean amount of time a volume is active is only 21%. By contrast, the volume active time in the read/write case is 60% on average. Similarly, the median amount of time a volume is active drops from 80% to 14% when the write load is removed.

Finally, we measure the potential benefit in reducing the peak power consumption of the data center storage by examining the temporal relationship between volumes. Figure 3 shows the number of volumes active over time through the week. We see that removing the writes from the trace significantly reduces the number of concurrently active volumes.

Table 2 shows the mean, median, 99th percentile, and maximum number of volumes concurrently active during the week. These results indicate that simply spinning down when idle can reduce the peak power of the storage subsystem, and that creating longer idle periods by off-loading writes can reduce it even further. Note that the set of active volumes changes over time, and a rarely-active volume might still store a large amount of data or experience a high peak load. Thus we cannot simply save energy by using fewer disks per volume, since we must still provision the volumes for capacity and peak load.

This analysis indicates that there are significant potential power savings in spinning down enterprise data center disks when idle. Further, it shows that efficiently redirecting writes creates even longer periods of idleness leading to substantially higher power savings. This motivated the design of our write off-loading mechanisms.

3 Write Off-Loading

The goal of write off-loading is to utilize periods of write-dominated load to spin disks down and off-load write requests, reverting to normal operation during periods of read-dominated load. When writes are being off-loaded the aim is to achieve comparable write response times and throughput to using the local volume.

Each volume supporting off-loading has a dedicated *manager*. The manager is entirely responsible for the volume, which we refer to as its *home* volume: it decides when to spin the physical disks up or down, and also when and where to off-load writes. Off-loaded blocks are only temporarily off-loaded and the manager is also responsible for reclaiming blocks previously off-loaded. To achieve all these tasks, the manager needs to intercept all read and write requests to its home volume.

When a manager decides to off-load a block, it selects one or more *loggers* to store it temporarily. Each logger instance requires a small area of persistent storage, which is used exclusively to store off-loaded blocks and metadata until they are reclaimed by a manager or no longer required. The persistent storage could be a disk, NVRAM or solid-state memory such as flash, depending on what is available on each server; the logger's data layout should be optimized for the particular type of storage. Our current implementation uses only disk-based loggers.

The set of loggers that a manager uses is configurable. It is important that the loggers used by a manager offer the same or better failure properties as the home volume. It is also possible to configure the manager so that it will only off-load blocks to loggers residing on the same server as itself, in the same rack, or across the entire data center. We have evaluated write off-loading at both a server and rack granularity. Current off-the-shelf gigabit networking makes the rack granularity feasible, with low network overhead and good performance. Server-granularity off-loading is feasible at any network speed since off-load traffic does not go over the network.

In the rest of this paper, we refer to a volume as being *active* if its disks are spinning and I/O operations are being performed on it. If the disks are spinning but no I/O operations are being performed, we refer to the volume as being *idle*: in this state the disk spindles continue to use energy even though they are doing no work. Finally, if the volume's disks are spun down, we refer to the volume as being in the *standby* state. We assume that all the disks belonging to a volume are always in the same state, since all the power management strategies considered in this paper operate on entire volumes at a time.

When we refer to a manager or logger component being in standby, we mean that the volume used by that component has transitioned to the standby state. When a manager goes into standby, it will force loggers sharing the same physical disks to go into the standby state. The manager will then off-load writes to loggers that are not in the standby state. Note that loggers using solid-state memory or NVRAM would never enter the standby state.

3.1 Detailed Design

Loggers. Conceptually the logger's role is simple: it temporarily stores blocks. Loggers support the following remote operations: *write*, *read*, *invalidate*, and *reclaim*. A write consists of persisting the provided blocks and metadata. The metadata consists of the source manager identity, a range of logical block numbers (LBNs), and a version number. A read returns the latest stored versions of the requested blocks. An invalidate request specifies a set of blocks and versions that are no longer required. To ensure consistency, the invalidate request explicitly includes version information, and the logger marks the corresponding versions as invalid. The logger can then lazily garbage collect the space used to store the invalidated data and metadata. A reclaim request is like a read, except that no block range is specified: the logger can return any valid block range it is holding for the requesting manager. Invalidates and reclaims are non-latency-critical operations; reads and writes are latency-critical but reads are expected to be rare. Hence loggers are optimized for the performance of writes.

Our current implementation uses a log-based on-disk layout. This means that writes have good locality; both data and metadata are written with a single I/O to the current head of the log. Log compaction and other maintenance tasks are done in the background with low priority. Metadata about the valid blocks stored for each manager, their versions, and their location in the log are cached in main memory for fast access.

Each logger uses a small partition at the end of an existing volume to persist data and metadata. This avoids the need to dedicate additional storage for off-loading. The remainder of the volume functions as before, and could have an associated manager to enable off-loading. In general a volume might host zero or more managers and zero or more loggers, on distinct partitions but on the same set of physical disks. In our evaluation we run with a typical configuration for a data volume: one manager and one logger with the latter using a small partition at the end.

Managers. The manager controls the off-loading of blocks, deciding when to off-load blocks and when to reclaim them. It is also responsible for ensuring consistency and performing failure recovery. To achieve this, each manager maintains persistently the identities of a set of loggers with which it interacts, referred to as the *logger view*. It also maintains two in-memory data structures, as shown in Figure 4. The *redirect cache* stores, for each block off-loaded, the block's LBN, the identity of the logger storing the current data for the block and the corresponding version number. Version numbers are unique monotonically increasing 64-bit quantities, which

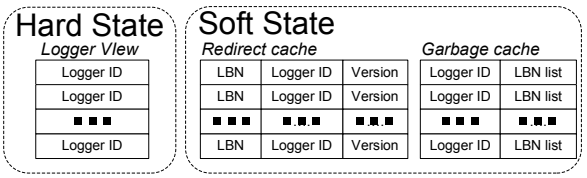


Figure 4: Manager data structures.

ensure that the manager can identify the last written version of any block during failure recovery. The *garbage cache* stores the location of old versions of blocks. In the background, the manager sends invalidation requests for these versions; when these are committed by the logger they are removed from the garbage cache.

The manager intercepts all read and write requests sent to the home volume. For a read request, it first checks the redirect cache for existing logged versions. If none is found, the read is serviced locally from the home volume, causing it to transition from standby to active if necessary. Otherwise the request is dispatched to the logger identified as having the latest version of the block. Multi-block reads are split as required, to fetch data from the home volume and/or one or more loggers.

For a write request, the manager off-loads the write to a logger if the home volume is in standby. It also off-loads the write if there are currently logged versions of any of the blocks, to ensure that the new version is persistently recorded as the latest version. Writes that are not off-loaded are sent directly to the home volume.

To off-load a write, the manager probes the loggers in its logger view: this is currently done using subnet broadcast for efficiency. Each logger replies with a set of metrics including the power state of the logger's volume, its queue length, the amount of available space, etc. The manager ranks the loggers using these metrics and selects one to off-load the write to. When the write is committed and acknowledged by the logger, the manager updates its redirect cache with the latest version and moves any older versions to the garbage cache.

When the home volume is idle, the manager reclaims off-loaded blocks from loggers in the background and writes them to the home volume. After the reclaimed blocks are written to disk, the manager sends invalidation requests to the appropriate loggers. To ensure correct failure recovery, the latest version of a block is invalidated only after all older versions have been invalidated. The background reclaim and invalidation ensure that all blocks will eventually be restored to the home volume and that logger space will eventually be freed.

Finally, the manager controls state transitions to and from standby for the home volume. The manager monitors the elapsed time since the last read and the last write; if both of these have passed a certain threshold,

it spins the volume down and off-loads all subsequent writes. The volume spins up again when there is a read on a non-off-loaded block, or when the number of off-loaded blocks reaches a limit (to avoid off-loading very large amounts of data). Before putting the volume into standby, the manager first ensures that there is at least one logger in its logger view that is using a set of disks different from its own and that is not currently in standby. This ensures that any future writes to the home volume can be off-loaded by the manager without waiting for disks to spin up. If there are no such loggers, then the manager does not spin down, but periodically probes its logger set for any change in their status.

This design is optimized for the common case: during periods of intense activity, the home volumes will be in the active state, and all I/Os will be local, except for a small number of requests on blocks that are currently off-loaded. During periods of low, write-dominated load, we expect that the home volume will be in standby and writes will be successfully off-loaded to a logger.

Uncommon cases are handled through fall-back strategies. For example, if the manager cannot find any available loggers, it spins up the home volume in the background, and retries the request until a logger is found or the home volume is spun up. If a volume needs to be taken off-line (say for maintenance) then the manager spins it up, as well as all volumes that it depends on or that depend on it. It then forces blocks to be reclaimed until the volume has all its own blocks and none of any other's, i.e., its state is restored as if no off-loading had occurred.

Write off-loading can mask the performance impact of spinning up disks for write requests. For read requests on spun-down disks we cannot mask the spin-up delay. For some applications this large delay (10–15 seconds) will be unacceptable even if rare: write off-loading should not be enabled on the volumes that these applications use.

3.2 Failure Resilience

Enterprise storage is expected to provide consistency and durability despite transient failures such as reboots as well as single-disk permanent failures. At the volume level, the failure resilience with off-loading is the same as that without. However, off-loading can create failure dependencies between managers and loggers. With off-loading at the rack or data center level, a manager on machine A could off-load blocks to a logger on machine B: if machine B suffers a failure, then the off-loaded blocks would become unavailable on machine A until machine B was brought on-line again.

This problem can be solved by off-loading each block to multiple independent loggers. With k-way logging, a manager can tolerate up to $k-1$ failures in its logger view. Given the high availability and reliability of enterprise servers, we do not think k-way logging would be required in most cases.

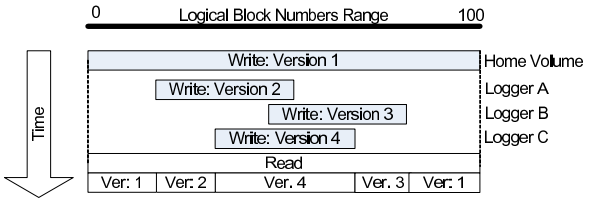


Figure 5: Consistency across loggers example

Write off-loading guarantees both consistency and durability across failures. We achieve durability by acknowledging writes only when both data and metadata have been reliably persisted, i.e., we do not employ write-back caching of any form. Consistency is achieved by using versioned metadata to mark the latest version of a block. When a read is performed for a range of blocks, it is quite possible that the required blocks are distributed over multiple loggers as well as the home volume, as shown in Figure 5. The manager uses the version information to ensure that the applications using the volume see a consistent view of the stored data. We also add a checksum to the metadata to ensure that partial writes are correctly detected on failure recovery.

If one or more machines reboot due to, say, a power failure, all the loggers recover concurrently by scanning their persistent logs to reconstruct their soft state. Each manager can be brought on-line when all the loggers in its logger view are on-line. A manager recovers its soft state (the redirect cache and garbage cache) by requesting information about all blocks stored for it from each logger in its logger view. To optimize the common case of a clean shutdown/reboot of a server, the manager writes the soft state to a small metadata partition during shutdown; this allows managers to restart after a clean shutdown without any network communication.

It is important that a manager's logger view be restricted to loggers which have the same or higher failure resilience as the home volume. Otherwise, when blocks are off-loaded, they will not have the same failure resilience as non-off-loaded blocks. If the storage uses standard solutions such as RAID-1 or RAID-5 for all volumes, then this property will be ensured, and off-loading will provide the same resilience to single disk failures as standard RAID solutions.

When a logger experiences a single-disk failure, it pushes all off-loaded blocks to other loggers or the appropriate manager, which should typically take seconds to minutes. This reduces the risk of losing off-loaded blocks due to multiple disk failures; the risk can be further reduced if desired by using k-way logging.

4 Evaluation

Section 2 presented a trace-driven analysis showing the potential benefits of write off-loading. This analysis was based on block-level traces of enterprise data center workloads. In this section we evaluate write off-loading using a real testbed and these workload traces.

4.1 Experimental Setup

The experiments were all run using a testbed consisting of four standard HP servers, each with a dual-core Intel Xeon processor and an HP SmartArray 6400 controller connected to a rack-mounted disk enclosure with a SCSI backplane. All the servers were running Windows Server 2003 SP2. For the purposes of trace replay, data volumes were accessed as raw block devices rather than as file systems, since our traces are at the block level.

The disk enclosures were populated with 56 Seagate Cheetah 15,000 RPM disks: 28 of size 36 GB and 28 of size 146 GB. The servers were connected via a switched 1 Gbps Ethernet. The device driver for the SmartArray 6400 does not support physical spin-down and spin-up of the disks, highlighting the fact that disk spin-down is not standard practice in enterprise storage systems today. We did not have access to the driver source code and hence were forced to emulate the power state of each volume in a software layer. This layer delays requests on a spun-down volume until the volume is spun up; it also models the power consumed by each volume based on the number and type of disks and the emulated spin state.

The parameters for emulating the power state of the disks used in the testbed are shown in Table 3. These parameters were derived manually from the voltage/current profiles given in the Seagate Cheetah 15K.4 SCSI product manual [26] (Section 6, Figures 4 and 5). The steady-state power consumption when spun up is based on the current draw of both the 12 V input line (which powers the motor) and the 5 V line (which powers the electronics); the power consumption when spun down is based on the 5 V current only. The energy cost of spinning up is defined as the difference between the total energy used while the disk was spinning up, and that used if the disk were idle and spinning for that duration. We do not model the energy cost of doing I/O over and above that of keeping the disk electronics powered and the platter spinning; in general, this is difficult to model for an arbitrary workload and is also relatively small.

To drive the experiments we used real-time replay of the data center traces that we analyzed in Section 2. Using all the trace data would have required one week per experimental run. To make this tractable, each trace was split into seven one-day (24-hour) traces. These traces were statically analyzed to find the "least idle" and the "most idle" days. Averaged across all volumes, the least idle day provides the smallest potential amount of idle time for write off-loading, whereas the most idle day provides the largest. The least idle day ran from midnight on Monday 26th February 2007 to midnight on the following day; it had 35 million requests with 73% reads. The most idle day ran from midnight on Sunday 25th February to midnight on the following day; it had 21 million requests with 70% reads. These two days represent the worst and the best case for energy savings using write off-loading, and hence our evaluation is based on them.

Time to spin up (36 GB disk)	10 s
Time to spin up (146 GB disk)	15 s
Energy cost of spinning up	20 J
Power when spun up	12 W
Power when spun down	2.6 W

Table 3: Energy parameters for Seagate Cheetah 15K.4

Rack	Server	Function	#volumes
1	usr	User files	3
	mds	Media server	2
	prn	Print server	2
	hm	H/w monitoring	2
2	src2	Source control	3
	proj	Project files	5
	wdev	Test web server	4
3	rsrch	Research projects	3
	prxy	Firewall/web proxy	1
	src1	Source control	2
	stg	Web staging	2
	ts	Terminal server	1
	web	Web/SQL server	4

Table 4: Servers grouped by rack

To emulate the traced data center volumes, sets of volumes were mapped on to the testbed. The entire testbed's disk capacity is much smaller than the original traced servers. Therefore the traced servers were divided into three sets or "racks" (Table 4). Experiments were run for each rack independently; all the volumes in a single rack were simultaneously emulated on the testbed. two of the 36 volumes (**prxy/1** and **src1/0**) could not be accommodated on the testbed with enough disks to sustain the offered load, so they were omitted. Due to the physical limitations of the testbed, the mapping does not keep volumes from the same original server on the same testbed server, or vice versa. However, our results show that the peak network load imposed by write off-loading is less than 7% of the network capacity: hence remapping does not significantly affect the system's performance. Each volume was mapped to a RAID-1 or RAID-5 array with sufficient capacity to replay the volume trace.

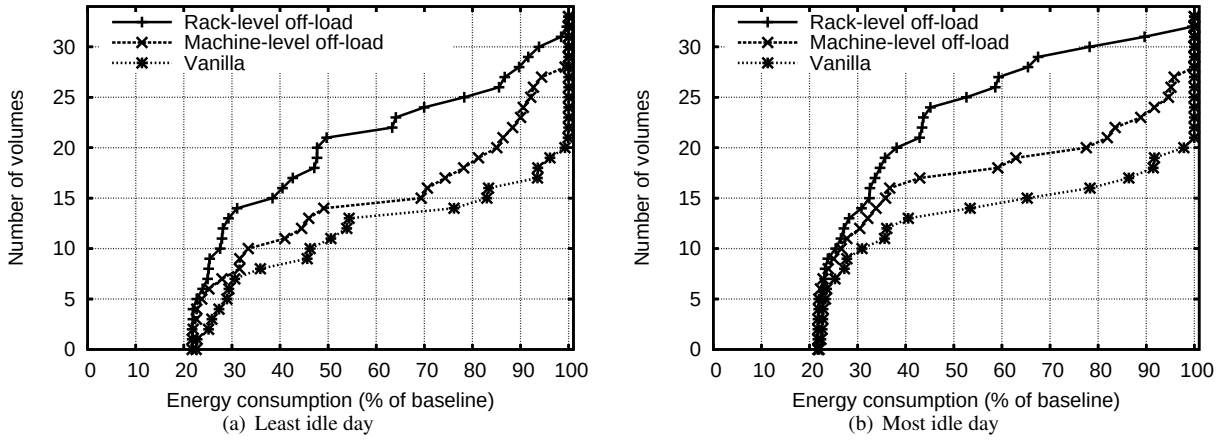


Figure 6: CDF of energy consumed as a percentage of baseline

A per-server trace replay component was used to replay the traces and gather performance metrics. The start of trace replay was synchronized across all servers. Each trace event was then converted to an I/O request sent to the corresponding emulated volume at the time specified by the timestamp: i.e., the traces were replayed "open-loop" in real time. This is necessary because the block-level traces do not capture higher-level dependencies between requests. However, requests which accessed overlapping block ranges were serialized, under the assumption that such requests would not be issued concurrently.

When configured for write off-loading each emulated volume was assigned both a manager and a logger. The logger used a 4 GB partition at the end of the volume; on hardware with flash or other solid-state devices the logger would run on the solid-state component instead. All manager and logger components on each server were linked together into a single user-level process along with the trace replay component. This component opened each server volume as a raw Windows block device; trace I/Os were then converted into read and write requests on these devices. Communication between managers and loggers is in-process if on the same physical server; otherwise we use UDP for broadcast and TCP for unicast.

In the experiments we evaluated four configurations:

- *baseline*: Volumes are never spun down. This gives no energy savings and no performance overhead.
- *vanilla*: Volumes spin down when idle, and spin up again on the next request, whether read or write.
- *machine-level off-load*: Write off-loading is enabled but managers can only off-load writes to loggers running on the same server: here the "server" is the original traced server, not the testbed replay server.
- *rack-level off-load*: Managers can off-load writes to any logger in the rack.

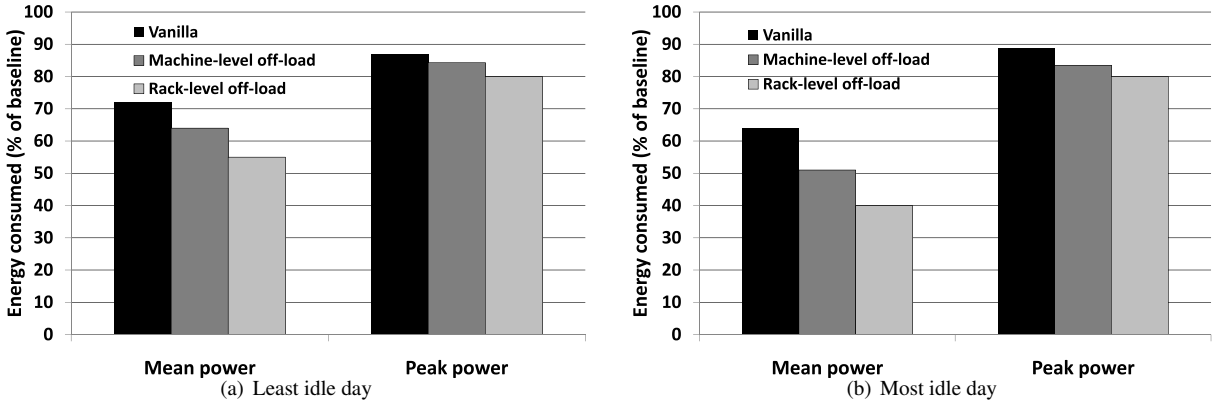
The configurations which place volumes in standby require an idle period before initiating standby. In the vanilla configuration we use an idle period of 60 seconds. For the two off-load configurations, a volume is placed in standby after 60 seconds of no reads and 10 seconds of no writes. Each off-load manager also limits the amount of off-loaded data to 1 GB: on reaching this limit, the manager spins up the volume in the background.

In the remainder of this section we present the summarized results of our experimental runs. Each result is presented both for the "most idle" day and the "least idle" day, and for each of the four configurations. For a given day and configuration, results are aggregated across racks; although the experiments were run sequentially, this emulates all three racks running concurrently with off-loading (if enabled) happening at the rack or machine level depending on the configuration.

4.2 Energy Savings

Figures 6(a) and 6(b) show the CDFs of energy consumed per volume for the least idle day and most idle day, respectively. Obviously, in the baseline case, all disks would always be spun up, and volumes would always be at their maximum power level. Hence we normalize the power consumption of each volume in the other three configurations by the corresponding baseline value. All three configurations on both days save significant amounts of energy compared to the baseline, as seen by the number of volumes that use 50% or lower energy compared to baseline. Figure 7 summarizes these results showing the mean and peak power consumption across all volumes, again as a percentage of the baseline value.

For the least idle day, of the three non-baseline configurations, the vanilla configuration consumes the most energy: 72% of baseline. This is because it does not utilize write off-loading to lengthen the idle periods. Machine-level off-loading is able to do this, and hence uses less energy: 64% of the baseline. However, the energy savings



(a) Least idle day (b) Most idle day

Figure 7: Total power consumption as percentage of baseline

are limited by the need to keep one volume spinning per machine to absorb the off-loaded writes. This means that at all times at least 13 of 34 volumes were kept spinning. Rack-level off-loading uses the least energy of all (55% of baseline) since it does not suffer from the limitations of the other two approaches. With rack-level off-loading, potentially a single spun-up volume could absorb the off-loads for the entire rack.

For the most idle day, all three non-baseline configurations improve their energy savings by exploiting the increased amount of idleness. As on the least idle day, vanilla uses significantly less energy (64%) than baseline; machine-level off-load does better (51%) than vanilla; and rack-level off-load does even better (40%).

The peak power results in Figure 7 show that vanilla reduces peak power to 87–89% of the baseline; machine-level off-load to 83–84%, and rack-level off-load to 80%. Unlike the mean power usage these results show that there is not a significant difference between the most and least idle days. This is because the power usage varies considerably over the time scale of minutes, as shown in Figure 3, and hence even on a day with a low mean power usage, the peak could be quite high. However, there is still a significant difference between the off-loading and non-off-loading configurations.

4.3 Performance Impact

We now evaluate the performance impact of spinning disks down, both with and without off-loading. We measured the response time of each read and write request in each configuration, on each of the test days. Figure 8 shows the response time distributions for reads and writes, for the two days, aggregated over all volumes. Since most requests have the same response time in all configurations, we put the y-axis on a log scale to highlight the differences between the configurations. The x-axis is also on a log scale, since response times vary from tens of milliseconds in the common case to over 15 seconds in the worst case. For example, the baseline curve in Figure 8(a) shows that 0.01 (1%) of the read requests in the baseline configuration on the least idle day had a response time of more than 100 ms.

We see that for the majority of requests, the performance was identical in all configurations. For a small fraction of the requests, we see a long tail in some cases, going out to over 10 seconds. This tail represents requests that needed to wait for a disk to spin up. In the vanilla configuration both reads and writes are impacted by this effect. For the machine-level and rack-level off-load only reads are affected: for write requests they track baseline performance up to and beyond the 0.0001 point, i.e., for 99.99% or more of the requests. For a very small number of requests (fewer than 0.01%) on the least idle day, the machine-level off-load does worse than the baseline or rack-level off-load; this is because in the case of a heavy burst of write requests, the machine-level off-load cannot spread the load across loggers on multiple servers, whereas the rack-level off-load can do this.

These results confirm our expectation that spinning disks down causes a large penalty for a small number of requests. This penalty occurs for both reads and writes if we do not off-load writes, and only for reads if we do. It is worth noting that both the length and thickness of this tail are increased by an artifact of our experimental setup. Since we replay the traces open-loop, all requests that arrive while a disk is spinning up will be queued, and simultaneously released to the disk when it spins up. Since the spin-up period is relatively long (10–15 seconds), a large number of requests could arrive during this period, which cause an additional large queuing delay even after the disk is spun up. For example, the tail for the vanilla configuration in Figure 8(c) goes out to 56 seconds: this is due to a single episode in which 5,000 requests were queued while waiting for a volume to spin up, and 22 of these requests suffered extremely high delays as a result.

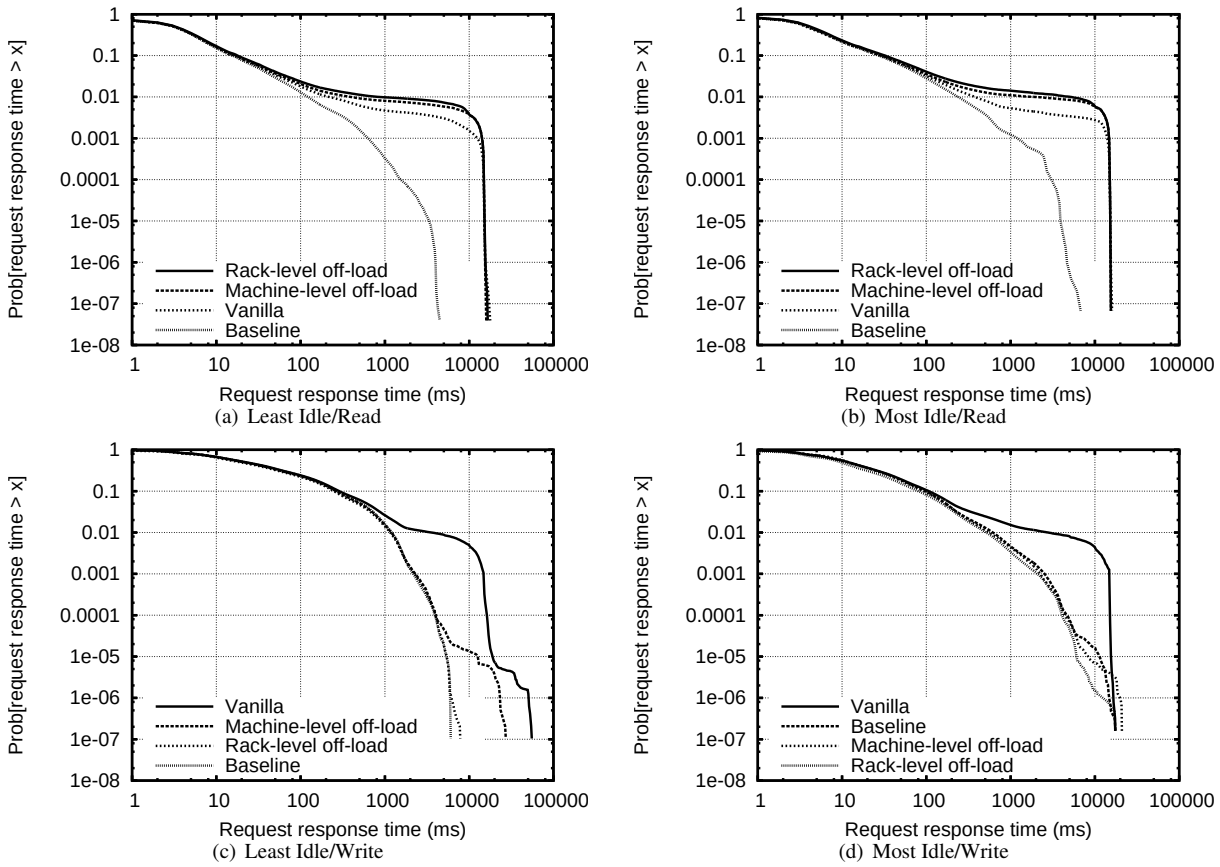


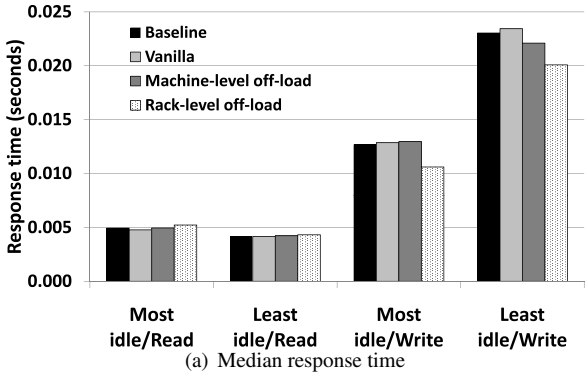
Figure 8: Response time distributions: the x-axis shows response times on a log scale, and the y-axis shows the fraction (also on a log scale) of requests with response times higher than some value

In reality, many of the requests in a single such burst would have been issued by the application in a closed loop, i.e., serialized one after another. Hence, delaying the first request would have prevented additional requests from being issued, avoiding the large queues and queuing delay. Further, if the request burst was created by a non-interactive application, for example a backup, then an initial delay of 10–15 seconds is acceptable. For interactive applications, of course, this first-byte delay will be visible to the user; if a volume supports interactive applications that cannot tolerate this delay even infrequently, then write off-loading or indeed any kind of spin-down should not be enabled for that volume.

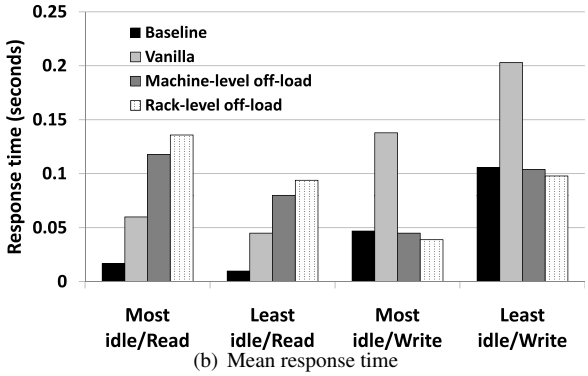
We now present some summary statistics on the response time distributions. Figure 9(a) shows the median response time; as expected, there is no penalty for spinning disks down or off-loading. Figure 9(b) shows the mean response time. Here we see the effect of the skewed response time distribution, i.e., the long thin tail, which causes a small number of requests to increase the mean significantly. For reads, all the non-baseline configurations have a high mean response time. The off-load configurations do worse than vanilla, because they spin down more often (and save more energy): hence a burst of read requests is more likely to hit a spun-down volume in these cases. For the same reason the rack-level off-load has a higher mean response time than the machine-level off-load.

In the case of writes, the mean is high for vanilla, but the off-load configurations do slightly better than the baseline case, with the rack-level off-load having the best performance of all. This is because logger writes have good locality, since they use a log structure, even if the original access pattern does not have good locality. Further, during bursts of high write load, rack-level off-load is able to load-balance across multiple loggers on multiple servers, and hence delivers the best performance.

Figure 10 shows the percentage of requests incurring a spin-up delay in each case. Obviously, the baseline configuration has no spin-up delays, and the off-load configurations do not have spin-up delays on write requests. Reads for rack-level/machine-level off-load, and all requests for vanilla, have significant (up to 1.2%) numbers of spin-up delays, which skews the mean response time



(a) Median response time



(b) Mean response time

Figure 9: Median and mean response times

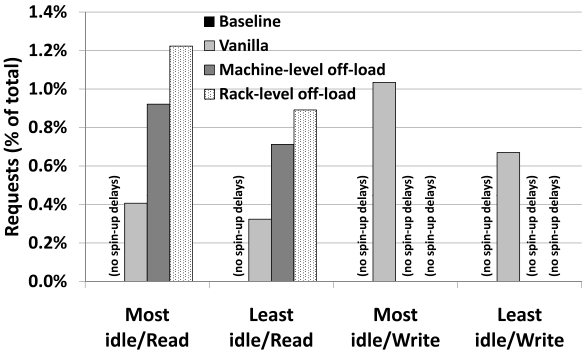


Figure 10: Percentage of requests incurring spin-up delays

for these cases. However, as remarked previously, some of this effect is an artifact of open-loop replay.

Figure 11(a) shows the 95th percentile response times respectively for the different cases. There are only minor differences between the different configurations, since much fewer than 5% of requests see spin-up delays in any configurations. Finally, Figure 11(b) shows the maximum response time for each case. For reads, the non-baseline configurations have similar worst-case performance to each other, although worse than the baseline: this is due to spin-up delays. For writes, all configurations have similar performance on the most idle day. On the least idle day, there is a large penalty for vanilla due to a combination of spin-up delay and queuing delay for a large write burst as previously discussed. Machine-level off-load also has a significant though lower penalty, due to the lack of load balancing on a heavy write burst. Rack-level off-load is able to load-balance such bursts and hence has a much better worst-case performance.

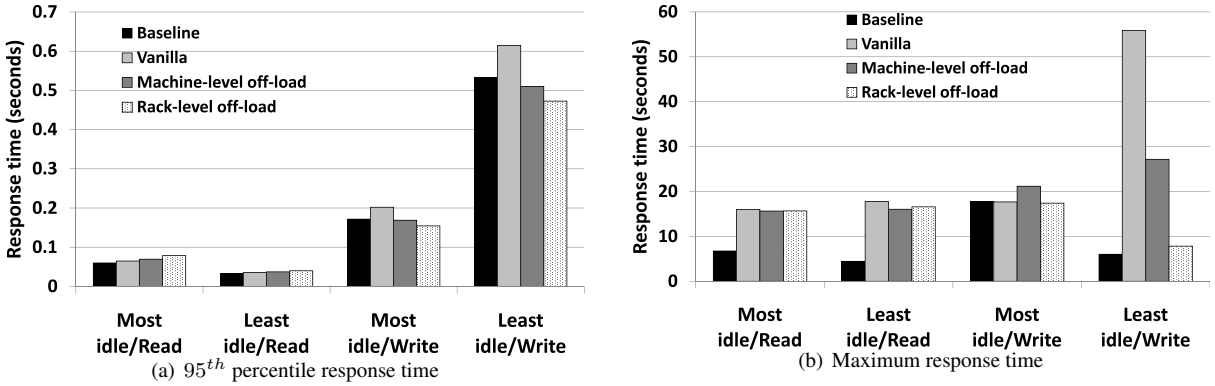
In summary, all configurations have comparable performance to the baseline case for a majority of requests; however, reads in the off-load configurations and both reads and writes in the vanilla configuration have a long thin tail, which is unavoidable. Finally, rack-level off-load consistently outperforms machine-level off-load on both energy savings and write performance, but has worse read performance and adds network traffic to the system. Administrators can configure the logger views on a per-manager basis to provide the appropriate trade-off between these metrics.

4.4 Network Usage

We also measured the network overheads of rack-level off-loading across both test days combined. These are summarized in Table 5. Note that the bandwidth usage is based on communications between managers and loggers that belong to different servers in the original trace. In other words, this measures the network overheads if write off-loading had been run on the original traced servers rather than the testbed servers. Thus there are no network overheads for machine-level off-load and of course none for the baseline or vanilla configurations.

The average bandwidth usage is low compared to the bandwidth available in a typical data center. The peak bandwidth usage can easily be supported by gigabit networks, which are widespread in enterprise data centers. The mean RPC round-trip latency is the amount of additional latency incurred by requests due to manager-logger communication.

The last two entries in Table 5 show that a substantial fraction of write requests were off-loaded, but only a very small fraction of reads were remote. A remote read is one that required the manager to interact with a logger because some or all of the requested blocks had been off-loaded. This is the expected behavior: due to main-memory buffer caches, very few recently-written blocks are read back from the storage layer. In other words, most off-loaded blocks will be overwritten or reclaimed before they are read again. The small fraction of remote reads also justifies our decision to optimize the loggers for write rather than read requests. Machine-



(a) 95^{th} percentile response time

(b) Maximum response time

Figure 11: 95th percentile and maximum (worst-case) response times

Total network transfers over 2 days	46.5 GB
Average network bandwidth usage	2.31 Mbps
Peak bandwidth usage over 60 s	68.8 Mbps
Mean RPC round-trip latency	2.6 ms
Fraction of writes off-loaded	27%
Fraction of remote reads	3.8%

Table 5: Network overheads for rack-level off-loading

level off-loading gives similar results to rack-level off-loading, with 8.3% of writes off-loaded but only 0.78% of reads being remote.

5 Related Work

There has been considerable research in power management for laptop and mobile device storage [9, 18, 30] and also on high-level power management for data centers as a whole [6]. We focus on related work in power management for enterprise storage systems.

The closest related work is Massive Arrays of Idle Disks (MAID) [7] which has been proposed for replacing tape libraries as tertiary storage environments for very large-scale storage. MAIDs are storage components composed of 1,000s of disks holding hundreds of terabytes of storage. A subset of the disks are kept spinning, acting as a cache, while the rest are spun down. For standard RAID-based enterprise primary storage, this requires a minimum of two additional disks per volume. For non-archival storage this is an unacceptable overhead in terms of both energy and cost. In contrast, write off-loading does not require additional dedicated disks per volume or new hardware: we can opportunistically use any unused storage in the data center to store blocks.

Power-aware cache management [33] optimizes cache replacement for idle times rather than miss ratios, and shows power savings for OLTP, *cello* [22], and synthetic traces. This is orthogonal to our approach: any increase in the inter-read time will result in increased energy savings with write off-loading. However, we observe that the enterprise workloads that we traced already have large inter-read times that we exploit using write off-loading.

DRPM [12] and Hibernator [32] are recently proposed approaches to save energy by using multi-speed disks (standard enterprise disks spin at a fixed rate of 10,000 or 15,000 rpm). They propose using lower spin speeds when load is low, which decreases power consumption while increasing access latency. However, multi-speed disks are not widely deployed today in the enterprise, and we do not believe their use is likely to become widespread in the near future.

Several approaches [16, 28, 29] have proposed power management schemes for RAID arrays at the RAID controller level or lower, which are orthogonal to write off-loading which works above the block device level. In particular, PARAID [28] uses different numbers of disks in the array for different load levels, and provides good power savings for read-mostly workloads. Hence it is complementary to write off-loading, which can convert a mixed workload to a read-only one.

Popular Data Concentration (PDC) [19] migrates data between disks to concentrate hot data on a few disks, allowing the remainder to spin down. This could be done within the disks on a volume: in this case it would be transparent and orthogonal to volume-level off-loading. PDC could potentially be done across volumes as well: however, this is not practical for most enterprise solutions since volumes will no longer be isolated from each other for performance, cannot be tuned individually, and acquire long-term data dependencies on each other.

Serverless file systems, such as xFS [1], attempt to distribute block storage and management across multiple networked machines, and use co-operative caching [8] to improve performance. By contrast, write off-loading is designed to save energy rather than reduce latency or in-

crease throughput. It also works at the block level, and rather than storing data remotely for days or weeks, a relatively small number of blocks are temporarily hosted on remote machines.

The log structure used to store off-loaded data and meta data is similar to those used in log-structured file systems [11, 21, 27]. However, log-structured file systems store all written data for the long term, whereas our logs store only off-loaded data, and temporarily.

Finally, various approaches to storage workload tracing and trace replay have been proposed in the research literature [2, 10, 15, 31]. We decided to use ETW for tracing since it is already supported on our traced servers, and it provides the functionality we needed (tracing block-level I/O requests) with low overhead.

6 Discussion

Hardware trends. Recently, there has been considerable interest in solid-state drives (SSD) for mobile devices and laptops [23, 24]. These drives currently vary in size from 4–32 GB, and use less power than disks. While SSD-based storage is likely to become widely used in laptops over the next 2–3 years, it is unlikely to replace disks in the enterprise in the foreseeable future due to the high per-GB costs and performance characteristics.

However, it is likely that solid-state memory (flash) will become common, either in hybrid drives or as a small generic block-level storage device on motherboards. Hybrid drives include a small amount of flash within the disk. This allows the drive to spin the physical disk down and use the flash as a persistent buffer cache. This is very similar to the idea of using battery-backed NVRAM as a buffer cache to reduce disk traffic [3].

Thus, if and when enterprise storage becomes fully SSD-based, write off-loading will offer few advantages. However, for the next decade or so we expect that server systems will continue to have disk-based systems, increasingly augmented with solid-state memory: by running loggers on the solid-state devices and using them for write off-loading, the power savings of write off-loading can be further increased.

Traditionally, spinning a disk up and down is viewed as increasing the stress on the drive and reducing the mean time to failure (MTTF). For the state-of-the-art enterprise class Seagate Cheetah 15K.4 SCSI drives, the MTTF calculations assume 200 power cycles per year. Recent research has re-examined some of the assumptions about factors that impact disk lifetime [20, 25] but has not examined the effect of spinning disks down: we see it as an open question what impact spinning up and down will have on enterprise disks.

Configuration and management. Write off-loading requires some level of administrator configuration and management. For example, an administrator might wish to disable write off-loading for some period of time on some set of disks, say a RAID array, hosting both managers and loggers. When this is desired, all data on loggers hosted on those disks must be reclaimed by their home volumes. Similarly, all data off-loaded by managers on those disks must be reclaimed and invalidated on the loggers that were storing them. This would be the procedure, for example, for decommissioning a volume that currently has write off-loading enabled.

System boot volumes typically should not have an off-load manager enabled (although they can certainly support a logger). This avoids off-loading blocks that are required for the system to boot.

7 Conclusion

In this paper we propose a technique called write off-loading to save energy in enterprise storage. It allows blocks written to one volume to be temporarily redirected to persistent storage elsewhere in an enterprise data center. This alters the I/O access pattern to the volume, generating significant idle periods during which the volume's disks can be spun down, thereby saving energy.

We analyzed the potential savings using real-world traces gathered for a week from the 13 servers in our building's data center. Our analysis shows that simply spinning disks down when idle saves significant energy. Further, write off-loading enables potentially much larger savings by creating longer idle periods. To validate the analysis we implemented write off-loading and measured its performance on a hardware testbed. The evaluation confirms the analysis results: just spinning disks down when idle reduces their energy consumption by 28–36%, and write off-loading increases the savings to 45–60%.

We believe that write off-loading is a viable technique for saving energy in enterprise storage. In order to use write off-loading, a system administrator needs to manage the trade-off between energy and performance. We are designing tools to help administrators decide how to save the most energy with the least performance impact.

Acknowledgements

We would like to thank our IT support staff at Microsoft Research Cambridge, particularly Nathan Jones for helping us to trace our data center servers. We would also like to thank our anonymous reviewers and our shepherd Erez Zadok.

References

[1] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'95)*, Copper Mountain, CO, Dec. 1995.

[2] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: A file system to trace them all. In *Proc. USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, Mar.–Apr. 2004.

[3] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'92)*, Boston, MA, Oct. 1992.

[4] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'91)*, Pacific Grove, CA, Oct. 1991.

[5] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *Proc. ACM International Conference on Supercomputing (ICS'03)*, San Francisco, CA, June 2003.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[7] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proc. ACM/IEEE Conference on Supercomputing*, Baltimore, MD, Nov. 2002.

[8] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. Symposium on Operating Systems Design and Implementation (OSDI'94)*, Monterey, CA, Nov. 1994.

[9] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *Proc. USENIX Winter 1994 Technical Conference*, San Francisco, CA, Jan. 1994.

[10] D. Ellard, J. Ledlie, P. Malkani, and M. I. Seltzer. Passive NFS tracing of email and research workloads. In *Proc. USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Mar. 2003.

[11] L. Ganesh, H. Weatherspoon, M. Balakrishnan, and K. Birman. Optimizing power consumption in large scale storage systems. In *Proc. Workshop on Hot Topics in Operating Systems (HotOS'07)*, San Diego, CA, May 2007.

[12] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic speed control for power management in server class disks. In *Proc. International Symposium on Computer Architecture (ISCA'03)*, San Diego, CA, June 2003.

[13] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M. Irwin. Interplay of energy and performance for disk arrays running transaction processing workloads. In *Proc. International Symposium on Performance Analysis of Systems and Software (ISPASS'03)*, Austin, TX, Mar. 2003.

[14] Intel Corporation. *Dual-Core Intel® Xeon® Processor 5100 Series Datasheet*, Nov. 2006. Reference Number: 313355-002.

[15] N. Joukov, T. Wong, and E. Zadok. Accurate and efficient replaying of file system traces. In *Proc. USENIX Conference on File and Storage Technologies (FAST'05)*, San Francisco, CA, Dec. 2005.

[16] D. Li and J. Wang. EERAID: Energy efficient redundant and inexpensive disk arrays. In *Proc. 11th ACM SIGOPS European Workshop (SIGOPS-EW'04)*, Leuven, Belgium, Sept. 2004.

[17] Microsoft. Event tracing. `http://msdn.microsoft.com/library/`, 2002. Platform SDK: Performance Monitoring, Event Tracing.

[18] E. B. Nightingale and J. Flinn. Energy-efficiency and storage flexibility in the Blue file system. In *Proc. Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, CA, Dec. 2004.

[19] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proc. Annual International Conference on Supercomputing (ICS'04)*, June 2004.

[20] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, Feb. 2007.

[21] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'91)*, Pacific Grove, CA, Oct. 1991.

[22] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proc. USENIX Winter 1993 Technical Conference*, San Diego, CA, Jan. 1993.

[23] Samsung. NAND flash-based solid state disk product data sheet, Jan. 2007.

[24] SanDisk. SSD UATA 5000 1.8" data sheet. Document No. 80-11-00001, Feb. 2007.

[25] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, Feb. 2007.

[26] Seagate Technology LLC, 920 Disc Drive, Scotts Valley, CA 95066-4544, USA. *Cheetah 15K.4 SCSI Product Manual*, Rev. D edition, May 2005. Publication number: 100220456.

[27] M. I. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin. An implementation of a log-structured file system for UNIX. In *Proc. USENIX Winter 1993 Conference*, San Diego, CA, Jan. 1993.

[28] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning. PARAID: The gear-shifting power-aware RAID. In *Proc. USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, Feb. 2007.

[29] X. Yao and J. Wang. Rimac: A novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. In *Proc. EuroSys Conference*, Leuven, Belgium, Apr. 2006.

[30] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *Proc. USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, Mar. 2003.

[31] N. Zhu, J. Chen, and T. Chiueh. TBBT: Scalable and accurate trace replay for file server evaluation. In *Proc. USENIX Conference on File and Storage Technologies (FAST'05)*, San Francisco, CA, Dec. 2005.

[32] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proc. ACM Symposium on Operating Systems Principles (SOSP'05)*, Brighton, United Kingdom, Oct. 2005.

[33] Q. Zhu and Y. Zhou. Power-aware storage cache management. *IEEE Trans. Computers*, 54(5):587–602, 2005.