Drive-Thru: Fast, Accurate Evaluation of Storage Power Management

Daniel Peek and Jason Flinn
Department of Electrical Engineering and Computer Science
University of Michigan

Abstract

Running traces of realistic user activity is an important step in evaluating storage power management. Unfortunately, existing methodologies that replay traces as fast as possible on a live system cannot be used to evaluate timeout-based power management policies. Other methodologies that slow down replay to preserve the recorded delays between operations are too time-consuming. We propose a hybrid approach, called Drive-Thru, that provides both accuracy and speed of evaluation by separating time-dependent and time-independent activity. We first synchronously replay file system activity on the target platform to create a base trace *that captures the semantic relationship between file system activity and storage accesses. We then use the base trace as input to a simulator that can evaluate different disk, network, file cache, and file system power management policies. We use Drive-Thru to study the benefit of several recent proposals to reduce file system energy usage.*

1 Introduction

The battery capacity of small, mobile computers such as handhelds limits the amount of energy that can be expended to access data. Given that I/O devices are often power-hungry, it is essential that the storage hierarchy employ power management to extend battery lifetime. Consequently, power management has been a hotspot of recent research in storage and file systems.

Unfortunately, it is often difficult for storage researchers to evaluate new power management strategies. The reason is that many of the activities associated with power management are *time-dependent*: the execution of these activities is related to the amount of time that has passed since a prior event. For example, the disk [7] and network [15] may enter power-conserving states after they have been idle for a given amount of time, file systems may coalesce operations that occur within a given time interval to save power [19] and reduce network transmissions [16], and the file cache may delay flushing dirty blocks for some time to increase I/O burstiness [28]. Methodologies that ignore this time-dependent activity are highly inaccurate, while methodologies that capture it faithfully are often too slow.

Historically, *trace replay* on a live system has been one of the most popular methods of evaluating storage systems. Using this method, one first captures representative traces of user activity, then replays the traces to measure the performance impact of proposed modifications.

Usually, trace replay on a live system omits any idle time between activities in order to speed evaluation, especially when a large set of possible configurations is being tested. While this methodology accurately captures time-independent activity, the impact of time-dependent operations cannot be measured. For example, a power management algorithm that spins down the disk during idle periods is not invoked since replay does not pause between activities.

Based on the above observation, one might reasonably decide to preserve the recorded interarrival times. In our own experience [1, 19], we have found this technique to be highly accurate; however, we have also found it to be too time-consuming. The time to run a single experiment is essentially equivalent to the length of the original trace. Potentially, one might shave some time off replay by eliminating extremely long idle periods. However, we have found that the background processing on most modern computers ensures that no more than a few seconds passes without *some* file system activity—this makes the occurrence of long idle periods rare. Given that one will often want to test multiple design options and run multiple trials to demonstrate repeatability, preserving interarrival times is too slow for all but the shortest traces.

A method to preserve interarrival times while hastening evaluation is to decrease the length and number of traces. However, it is important to ensure that the remaining traces represent realistic user activity. These microbenchmarks can be a useful guide to optimizing a storage system, but over-reliance on one or two short traces can lead to erroneous conclusions, as we show in Section 4.

A final alternative is simulation. While discrete-event simulators [5, 30] are often used to model disks, the majority of power management algorithms are implemented in other layers of the storage hierarchy (e.g. in the device driver, the file cache, and the file system). Accurate simulation of storage power management requires that the simulator capture all layers. Thus, the complexity of developing an accurate storage hierarchy simulator is a significant drawback of this approach. For instance, the implementation of these layers (i.e. the VFS layer, ext2, and the IDE driver) in the Linux 2.4 kernel comprises over 50,000 lines of source code. Even after a simulator is developed, careful testing is required to ensure that it is bug-free and that it faithfully replicates the behavior of the simulated system. Finally, substantial simulator modifications may be needed in order to evaluate new operating systems or file systems; even minor version changes to these software layers may invalidate simulation results. One might possibly use a whole system simulator [23] that can run the entire operating system within the simulator and perform trace replay on top of it. However, substantial portability issues will still arise due to the diversity of hardware platforms seen in mobile computing today; one may potentially have to develop several complex simulators to examine behavior on all target platforms.

All of the above methodologies have different strengths, but no single methodology is fast, accurate, and portable. We therefore propose a new methodology, Drive-Thru, that combines the best features of trace replay on a live system and simulation by separating time-dependent and time-independent activity. Time-independent operations, such as the mapping of file system operations to disk accesses and the determination of which requests hit in the file cache, are captured through trace replay. The output of the replay is a *base trace* that captures the semantic relationship between file system operations and disk requests. The base trace is used as input to a simulator that models only time-dependent behavior such as disk spin-down and the delayed writes of dirty cache blocks. Our validation of Drive-Thru shows that it is over 40,000 times faster than a trace replay on a live system that preserves request interarrival times. At the same time, Drive-Thru's estimates for the ext2 file system are within 5% of the measured filesystem delay and within 3% of the measured file system energy consumption. Further, since our simulator need not model complex time-independent behavior, Drive-Thru is highly portable—currently, our simulator is only 914 lines of source code.

As we discuss further in Section 8, the success of this approach rests on two assumptions: first, that we can cleanly separate time-dependent behavior from time-independent behavior, and second, that most of the complexity of the storage hierarchy is time-independent.

We have used Drive-Thru to perform a detailed case study of storage power management policies. We concentrate on power management optimizations that require no application modification. For local file systems, we find that a policy that writes dirty data to disk before spin-down yields significant energy reductions, but that increasing the Linux `age_buffer` parameter beyond 30 seconds does not have a large enough benefit to offset the increased danger of data loss. We also note that operating systems should be careful not to set `age_buffer` to a value that is close to the spin-down time of the hard drive. We find substantially different behavior for a network file system. The policy of writing dirty data before using a network power saving mode *increases* energy usage. Further, writes need be delayed only 2 seconds for energy-efficiency; almost no additional energy reduction is realized by delaying writes further.

We begin with a description of the design and implementation of Drive-Thru. Section 3 evaluates the speed and accuracy of our methodology. Section 4 and Section 5 present case studies of power management strategies for a local and a network file system, respectively. Section 6 applies the results of these case studies to optimize an existing file system. After discussing related work and Drive-Thru's limitations, we conclude.

2 Drive-Thru: design and implementation

2.1 Overview

Figure 1 shows an overview of the Drive-Thru methodology. We begin with a pre-recorded trace of file system activity that captures POSIX operations such as `mkdir`, `open`, and `unlink`, as well as the time at which each operation occurred. Such traces may be found in the public domain [16]; alternatively, we have built a trace capture tool that allows us to record our own.

It is important to note that block-level traces that record disk accesses are insufficient for our purpose. A block-level trace does not capture enough semantic information to reason about how power management at higher layers of the system will affect performance and energy usage. Each disk trace captures one particular file system and cache behavior, so considering alternative behaviors is infeasible. For instance, if dirty blocks can reside in the file cache for up to 30 seconds, then two writes to the same block 20 seconds apart are coalesced into a single disk write. By examining only the disk trace, one cannot determine that two writes rather than one would occur if the write-back delay is reduced to 15 seconds.

As described in Section 2.2, we first replay each file system trace on the target platform, operating system, and

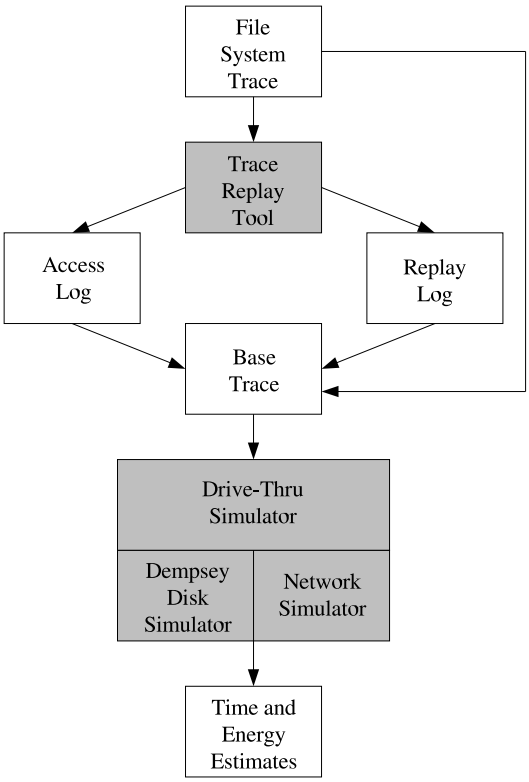


Figure 1. Drive-Thru overview

file system. The trace replay tool records the time when each file system operation begins and ends in the *replay log*. Simultaneously, we also generate a low-level *access log* that records each I/O access during the trace. After each POSIX operation completes, the trace replay tool ensures that all modifications have been reflected to storage (e.g. by using the `sync` system call) and then immediately begins the next operation. Reflecting modifications to storage immediately allows us to associate each I/O access with the POSIX operation that caused it. Because we omit recorded idle time, the base trace is generated in much less time than it took to record the original trace.

The next step, described in Section 2.3 is to merge the original trace, the access log, and the replay log into a *base trace* that captures the semantic relationship between high-level file system operations and low-level accesses. This relationship reveals time-independent behavior such as how a particular file system lays out its data on disk and which accesses hit in the file cache. The base trace is input to the Drive-Thru simulator, which models time-dependent behavior.

Drive-Thru simulates time-dependent activity at each layer of the hierarchy. For example, at the file system layer, it models queuing behavior in distributed file systems such as Coda [14] and BlueFS [19] where file operations are delayed before being sent over the network to the file server. At the file cache layer, the simulator models the coalescing of block writes due to delayed writeback of dirty blocks. At the device driver layer, the simulator models specific power management algorithms such as disk spin-down [7] or the use of 802.11b's Power Saving Mode (PSM) [11].

As described in Section 2.4, the Drive-Thru simulator delays, eliminates, and coalesces I/O accesses depending upon the specific power-management algorithms specified at each layer of the storage hierarchy. Each resulting I/O access is then passed to a device-specific simulator that models network or storage. For disk simulation, we use Dempsey [30], a modified version of DiskSim [5] that simulates both time and energy. For network simulation, we use a custom 802.11b simulation module. As output, Drive-Thru gives the expected time and energy to replay the trace on the target platform. By changing the parameters of the Drive-Thru simulator, one can quickly investigate alternative file system, cache, and device power management strategies.

2.2 Replay

The Drive-Thru file system trace replay tool runs on the target platform for evaluation. It replays operations recorded in the trace sequentially and forces all dirty data to storage before beginning each new operation. Since all file system operations issued by the replay tool are POSIX-compliant, the replay tool is portable; i.e., it can be run on any POSIX platform.

During replay, we record disk accesses using a modified `ide-disk` Linux kernel module. The module generates a disk access log that records the start time, completion time, starting sector, length, and type (read or write) of every access. The module writes this data to a 1 MB kernel buffer that is read by a user-level program through a pseudo-device after the trace is complete.

We employ a similar strategy to monitor network accesses made by distributed file systems. We modify the file system's remote procedure call (RPC) package to record the start time, completion time, size, and type (read or write) of each RPC. This information is written to a network access log—on the iPAQ platform, this log is stored in RAM.

2.3 Generating the base trace

As shown in Figure 2, we merge the original file system trace, the replay log, and the access log(s) to generate a *base trace*. The base trace preserves the interarrival times between file system operations that were captured in the

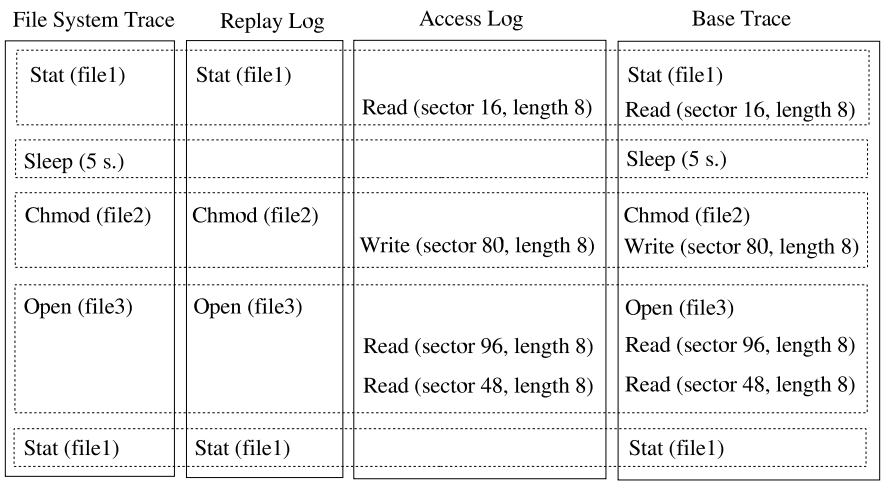


Figure 2. Example of generating a base trace

original file system trace—this is shown by the `sleep` record in the trace in Figure 2.

For each file system operation, the base trace shows the disk and/or network accesses generated by executing each operation on the target platform. Because the trace replay tool executes each operation sequentially and ensures that all dirty data is flushed before it begins the next operation, each storage access will overlap with exactly one file system operation (although a single file system operation may overlap several storage accesses). This methodology makes clear the semantic relationship between file system operations and disk and network accesses.

As shown in Figure 2, the base trace captures the relationship between the file system trace and the access trace—this shows time-independent behavior such as how the file system lays out its data on disk and which blocks hit in the file cache. Because the base trace already captures this behavior, the Drive-Thru simulator need only simulate time-dependent operations.

2.4 The Drive-Thru simulator

Using the base trace as input, the Drive-Thru simulator calculates the time and energy that it would take to run the file system trace on the target platform had interarrival times been preserved during trace replay. Drive-Thru uses discrete-event simulation to model three layers of the storage hierarchy: the file system, the file cache, and device power management. At each layer, the simulator modifies the stream of I/O accesses seen in the base trace. Based upon the time-dependent behavior specified, the simulator:

- *Delays I/O accesses.* Disk or network accesses are often delayed to save power. For example, on the Hitachi 1 GB microdrive, once the drive enters its standby mode, the next access is delayed approximately 800 ms [10]. Similarly, a transition between power modes on a 802.11b network interface delays accesses for several hundred milliseconds until the transition completes [1]. At the file system and cache layers, write accesses are often delayed to improve performance or save power. Based on the behavior specified, the Drive-Thru simulator may delay accesses at each layer of the storage hierarchy. This can result in reordering of accesses (e.g. if writes are delayed but reads are not).
- *Eliminates I/O accesses.* A delayed access may be obviated entirely by a subsequent file system operation, in which case the simulator eliminates it. For example, the `age_buffer` parameter specifies how long a dirty block may remain in the file cache until it is written to storage. If a `write` operation creates a dirty block that is overwritten by a subsequent `write` operation within `age_buffer` seconds, only one disk access will result. In this case, the simulator eliminates the first disk write upon seeing the second write. Similarly, the Coda file system delays sending operations to the file server for up to 300 seconds in write-disconnected mode [16]. If two operations that cancel each other are performed within this period, neither is sent over the network. The simulator would therefore eliminate all network accesses associated with the two operations.

- *Coalesces I/O accesses.* Two discrete I/O accesses may be merged into a single large access. For example, two writes to adjacent disk sectors can be merged into a single write to two sectors. Similarly, the Blue file system aggregates RPCs that modify data in order to save power [19]. Thus, for BlueFS, our simulator coalesces multiple small network accesses into a larger access whose size is the sum of the sizes of the smaller RPCs.

After the Drive-Thru simulator accounts for time-dependent behavior in the file system, file cache, and device power management layers, it hands the resulting accesses off to a device-specific simulation module to model the time and energy needed to perform the network and disk accesses. For disk simulation, we use the Dempsey simulator [30] developed by Zedlewski et al. For network simulation, we use our own custom module that models 802.11b behavior.

Dempsey is a discrete-event simulator based on DiskSim [5] that estimates the time and energy needed to perform disk accesses. Dempsey calculates when each disk access will complete based upon a detailed characterization of the drive. At the end of simulation, Dempsey reports the total energy used by the drive. Drive-Thru adds this value to its calculated energy usage for the network and the rest of the system to derive the total energy usage of the mobile computer.

The network simulator takes as input the measured latency and bandwidth between the mobile computer and file server, as well as a model of packet loss. The network simulator currently models three power management strategies: the Continuous Access Mode (CAM) and Power-Saving Mode (PSM) defined by the 802.11b standard [11], as well as Self-Tuning Power Management (STPM) [1], an adaptive strategy that toggles between CAM and PSM depending upon the observed network access patterns. The network module calculates the time and energy used to perform each RPC based upon a characterization of the network card that includes the power needed to transmit and receive data in each mode as well as the time and energy cost of switching between CAM and PSM. At the end of simulation, the module reports the total energy used by the network.

3 Validation

We validated Drive-Thru by comparing its time and energy results to those obtained through a trace replay on a live system that preserves request interarrival times.

3.1 Methodology

We evaluated Drive-Thru on an HP iPAQ 3870 handheld running the Linux 2.4.19-rmk6 kernel. The handheld has a 206 MHz StrongArm processor, 64 MB of DRAM and 32 MB of flash. We used a 1 GB Hitachi microdrive [10] and a 11 Mb/s Cisco 350 802.11b network card in our experiments. The Hitachi microdrive uses the adaptive ABLE-3 controller that spins down the disk to save power. Empirical observation of its behavior reveals that the microdrive spins down once it has been idle for approximately 2 seconds.

We measured operation times using the `gettimeofday` system call. Energy usage was measured by attaching the iPAQ to an Agilent 34401A digital multimeter. We removed all batteries from the handheld and sampled power drawn through its external power supply approximately 50 times per second. We calculated system power by multiplying each current sample by the mean voltage drawn by the mobile computer — separate voltage samples are not necessary since the variation in voltage drawn through the external power supply is very small. We calculated total energy usage by multiplying the average power drawn during trace replay by the time needed to complete the trace. The base power of the iPAQ when idle with no network or disk card inserted is 1.4 Watts. If we had turned off the screen, the iPAQ would have used only 0.9 Watts.

We used file system traces from two different sources. The first four in Figure 3 were collected by Mummert et al. at Carnegie Mellon University [17] from 1991 to 1993. The remaining traces are NFS network traces collected by Kim et al. at the University of Michigan in 2002 [13]. Because it would be too time consuming to replay each trace in its entirety, we selected segments approximately 45 minutes in length from a subset of these traces.

We first replayed each segment on the iPAQ, preserving operation interarrival times and measuring the time and energy used to complete each trace. We then compared the measured results with Drive-Thru's estimates. However, we note that the majority of time spent during trace replay is due to recreation of interarrival time rather than file system activity. A reasonable metric of accuracy may therefore be how well Drive-Thru estimates the non-idle time in the trace. We calculate this *file system time* by subtracting the idle time from the total trace time; this metric reflects the additional delay added by the file system. We also calculate *file system energy*, which is the amount of additional energy consumed by file system activity during trace replay. To generate this value, we subtract the product of the handheld's idle power and the idle time from the total energy used to replay the trace.

3.2 Local file system results

We began by evaluating how accurately Drive-Thru estimates the time and energy needed to replay traces on

Trace	Number of Ops.	Length (Hours)	Update Ops.	Working Set (MB)
Purcell	87739	27.66	6%	252
Messiaen	44027	21.27	2%	227
Robin	37504	15.46	7%	85
Berlioz	17917	7.85	8%	57
NFS2	39074	24.00	28%	21
NFS15	34188	24.00	16%	4

This figure shows the file system traces used in our evaluation. Update operations are those that modify data. The working set is the total amount of data accessed during a trace.

Figure 3. File traces used in evaluation

Linux's ext2 file system. Figures 4(a) and 4(b) compare the measured time and energy to replay traces on ext2 with Drive-Thru's estimates. For the Purcell trace, we vary Linux's file cache write-back delay (`age_buffer`) from 15 to 60 seconds; all other segments were replayed with the default 30 second value. As can be seen, Drive-Thru's estimates of total replay time and energy are extremely accurate—on average, they are within 0.10% of the measured time to execute each trace and within 0.21% of the measured energy.

As shown in Figures 4(c) and 4(d), Drive-Thru still maintains excellent accuracy when only file system time and energy are considered. Drive-Thru's estimates are, on average, within 5% of the measured file system time and within 3% of the measured file system energy usage.

3.3 Network file system results

We next validated Drive-Thru using a network file system in which data is stored on a remote server rather than local disk. For this purpose, we used the Blue file system [19], a distributed file system developed at the University of Michigan. One reason that we chose BlueFS for this study is that it has been designed from the start with energy-efficiency as a goal; consequently, BlueFS has been shown to use much less energy than other distributed file systems. BlueFS stores the primary replica of each file system object on a network file server. Further, it can optionally cache second-class replicas on local and portable storage devices. However, for this particular case study, we allow BlueFS to use only the remote file server so as to isolate the performance of Drive-Thru for network storage.

BlueFS issues remote procedure calls (RPCs) to the network server in a manner similar to NFS [18], e.g. each RPC roughly corresponds to an individual VFS operation. BlueFS queues RPCs that modify data for up to 30 seconds before sending them to the server. On the iPAQ, BlueFS also flushes the outgoing queue if the size of queued data exceeds 11.25 MB. If the queue size exceeds 15 MB, new file system operations are blocked until some operations are sent to the server. When sending queued operations to the server, BlueFS coalesces RPCs into aggregate RPCs of up to 1 MB. We modeled this time-dependent behavior in the Drive-Thru simulator with 44 lines of source code.

In Figures 4(e–h), we show the accuracy of Drive-Thru's estimates for the Purcell and NFS15 traces. For each trace, we examine behavior using 802.11b's CAM and PSM modes [11], as well as the adaptive STPM policy [1]. On average, Drive-Thru's estimates are within 0.86% of measured values for total time and within 0.90% for total energy. When we remove idle time from consideration, Drive-Thru's estimates are, on average, within 13% for file system time and within 7% for file system energy usage.

Why is Drive-Thru less accurate for network file systems than for local disk file systems? Partially, this is because network accesses are inherently more variable (as can be seen by the larger error bars in the measured results for network accesses in Figure 4). Also, we do not yet model the storage hierarchy of the network file server. Finally, we might be able to improve Drive-Thru's estimates by replacing our simple network simulator with one that is more accurate [12, 26].

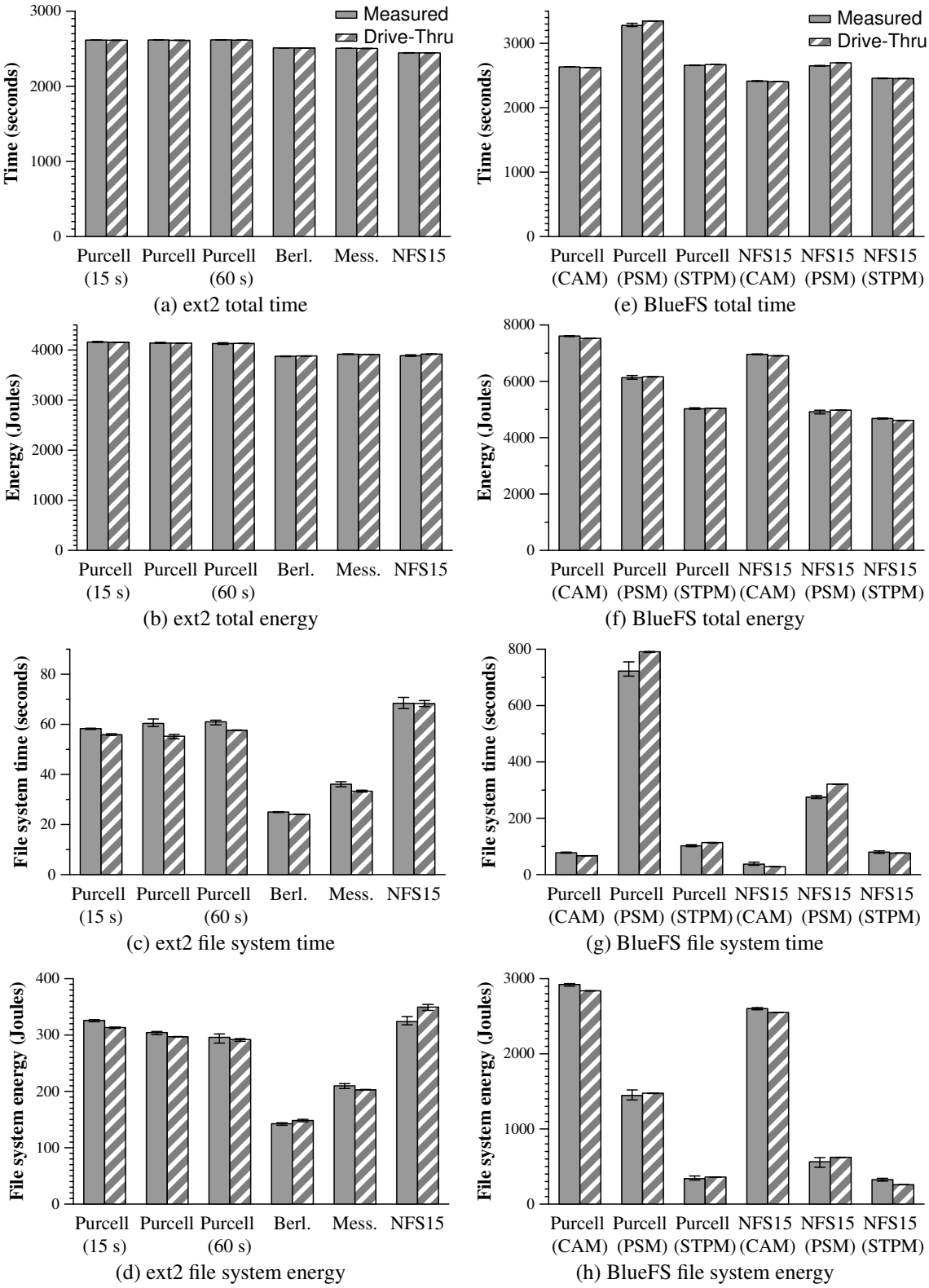
4 Case study: local file system

4.1 Methodology

Using Drive-Thru, we quantified the benefit of several recent proposals that modify file cache behavior in order to reduce energy consumption. We selected for our study only proposed modifications that could easily be implemented in current commodity operating systems. As a consequence, none of the proposals we selected require changes to existing application source code.

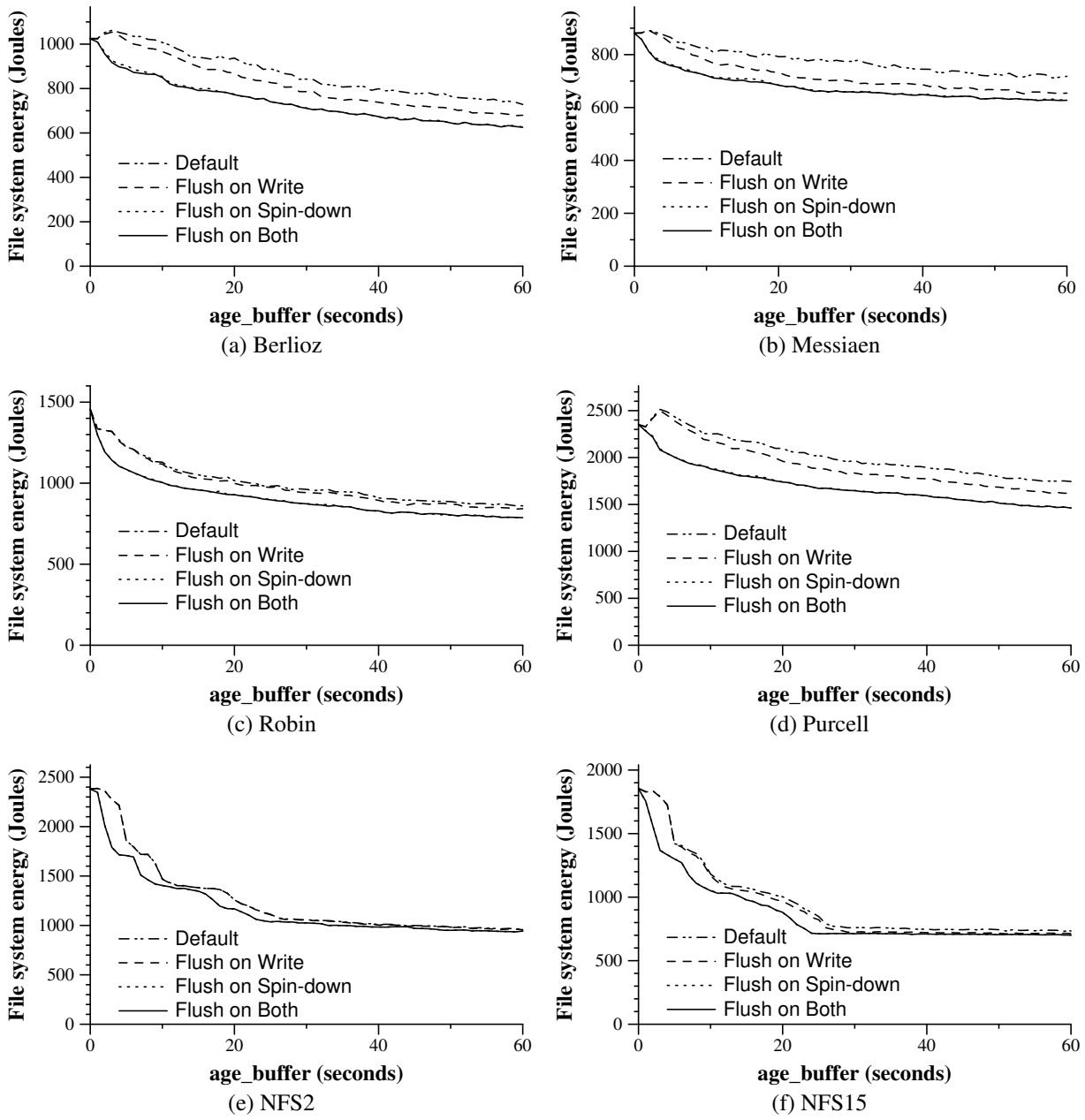
The three modifications that we studied were:

- *Flush on write.* Whenever any single dirty cache block is written to disk, write all dirty cache blocks. This modification was proposed by Weissel et al. [28], who referred to it as the "write back all buffers" strategy. Papathanasiou and Scott also propose to write all dirty blocks in their energy-efficient prefetching work [21].
- *Flush on spin-down.* Before the disk is spun down to save power, write all dirty blocks in the cache to disk. Weissel [28] first proposed this strategy, referring to it as "Update on shutdown".
- *Increasing age_buffer.* This allows a dirty block to reside in the cache for a longer period of time. The default Linux `age_buffer` value is 30 seconds. Weissel et al. [28] increased this value to



The left column validates Drive-Thru for a local file system by comparing its time and energy estimates with measured results for ext2. The right column validates a network file system by comparing results for BlueFS. File system time and energy are metrics that reflect the overhead of the file system. Each bar is the mean of three trials, with error bars giving the value of the lowest and highest trial. Note that the scales are different between the two columns due to the larger overhead of network file systems.

Figure 4. Drive-Thru validation



Each graph shows the results of implementing the three cache management policies described in Section 4.1 for the ext2 file system. We show only energy results since the policies studied had negligible performance impact. Note that the scales are different in each graph. In most graphs, the "Flush on Spin-down" and "Flush on Both" lines overlap and thus appear to be only a single line. Similarly, the "Default" and "Flush on Write" lines overlap in some graphs such as 5(e).

Figure 5. Results from local file system case study

60 seconds in their work. Papathanasiou [21, 22] also increased this value to 60 seconds (or longer in some cases). In our own work [1], we have proposed a similar strategy that delays writes as long as possible for data that is replicated elsewhere.

We used Drive-Thru to assess the potential benefit of each strategy for the ext2 file system on the iPAQ described in Section 3. We used the six full-length file traces shown in Figure 3. We examined the four combinations of enabling or disabling *flush on spin-down* and *flush on write*, while varying `age_buffer` from 0–60 seconds. Note that `age_buffer` only controls the eligibility of dirty blocks to be written from the Linux cache; the `updated` thread that actually writes the data wakes up

every `interval` seconds. By default, `age_buffer` is set to 30 seconds and `interval` is set to 5 seconds (meaning that a dirty block may actually reside in the cache for up to 35 seconds before flushing starts). As we varied `age_buffer`, we also varied `interval` to preserve the default 6:1 ratio.

4.2 Results

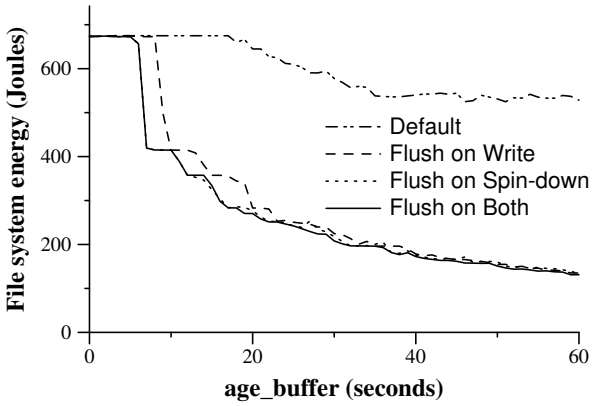
As can be seen in Figure 5, both the *flush on spin-down* and the *flush on write* policies are effective in reducing energy usage for almost every trace. However, the *flush on write* policy is less effective; in the NFS2 trace, for example, its energy usage is essentially equivalent to the default Linux cache management policy. Interestingly, for all six traces, the *flush on spin-down* policy alone saves almost the same amount of energy as the combination of the two policies. Examination of the detailed results shows that if the cache is flushed on spin-down, the *flush on write* mechanism is almost never used as long as the value of `age_buffer` is larger than a few seconds. Essentially, it is rare for read operations to constantly keep the disk spinning long enough for the *flush on write* policy to take effect.

Across all six traces, *flush on spin-down* reduces file system energy usage by an average of 11%. Since *flush on spin-down* is relatively easy to implement and because it requires no application modification to be effective, we recommend that the policy be added to commodity operating systems such as Linux.

We were surprised to see that increasing `age_buffer` beyond the Linux default of 30 seconds has little effect on energy consumption. On average, increasing `age_buffer` from 30 to 60 seconds reduces file system energy usage by only 8% (assuming *flush on spin-down* is implemented). Increasing the value to more than 60 seconds does not significantly decrease energy usage further. Intuitively, the effect is minimal because the distribution of disk accesses is such that few interarrival times fall between 30 and 60 seconds. Unlike the other two policies, increasing `age_buffer` has a substantial downside because it increases the window during which data may be lost due to system crash. Therefore, we recommend that `age_buffer` be left unchanged unless storage energy usage is an overriding concern.

Close examination of the trace results revealed a feature that we did not predict. In Figure 5(d), the default Linux cache policy experiences a peak in energy consumption when `age_buffer` is set to 2 seconds. This peak exists in all other traces, although it is not particularly pronounced in some. In addition to using Drive-Thru, we have also confirmed the existence of this peak experimentally.

By studying Drive-Thru trace results, we determined that this peak occurs when `age_buffer` is set to a value



This shows the results of implementing the three cache management policies described in Section 4.1 on the energy needed to download a 37 MB file and store it on disk.

Figure 6. Results from download microbenchmark

that closely approximates the disk spin-down timeout. In all of our traces, reads and writes tend to be clustered together due to the bursty nature of file system accesses [27]. For each cluster, the disk spins up to service the read requests, then spins down immediately before `updated` writes back dirty cache blocks modified by operations in the cluster. The disk must then spin up to service `updated` writes; it also remains in a high-power state for another two seconds before spinning down again. Note that use of the *flush on spin-down* avoids this peak because it writes dirty blocks to the cache before the first spin-down operation. To avoid this behavior, we recommend that the operating system set the disk spin-down threshold to a value different from `age_buffer` (or else, implement *flush on spin-down*).

Calculating time and energy estimates using Drive-Thru requires two steps: the generation of the base trace, followed by the execution of the simulator. When comparing the speed of Drive-Thru to traditional trace replay, the worst-case comparison is when Drive-Thru is only used to generate a single data point. For the six traces that we studied, Drive-Thru was 158–614 times faster than trace replay for this task. Further, for the entire case study, we only needed to generate the base trace once for each file system trace. This means that the effort to generate the base trace is amortized over many runs of the simulator. In fact, if we were to generate the results in this study using trace replay, it would have taken over 40,000 times longer than the time required using Drive-Thru. Put in raw terms, we can generate all the results in this case study in less than 45 minutes using Drive-Thru; the same results would take over 3 years to generate using trace replay on a live system.

4.3 The danger of microbenchmarks

Seemingly, our results for ext2 contradict previous results reported in the literature that show greater benefit for these cache management strategies. We believe that both measurements are accurate, but that the difference between the two results shows the danger of over-reliance on microbenchmarks. We tried to confirm this by replicating one experiment [8] where data is downloaded to a mobile computer and stored on local disk. In our experiment, we download a 37 MB file over the iPAQ serial port and write it to the microdrive. The download rate over the serial link is approximately 8 KB/s.

We first generated a base trace by running the download application. We then used Drive-Thru to produce the results shown in Figure 6. Note that both *flush on spin-down* and *flush on write* produce substantially larger energy savings for this microbenchmark (61% and 62% respectively). Once these policies are implemented, increasing `age_buffer` from 30 to 60 seconds yields a 37% reduction in energy usage.

Although these results are substantially more impressive than those that are generated when we apply the policies to our file traces, we believe that the two sets of results are consistent. Some applications captured in the file system trace may well have access patterns that yield benefits similar to the download microbenchmark. However, there may be many other applications using the file system for which the policies under study are considerably less effective.

We also believe that these results demonstrate the primary benefit of Drive-Thru: because evaluation is dramatically faster using our methodology, it is possible to evaluate new storage power management policies on a much wider set of data. As the scope of data used for validation increases, the danger that results reflect a particular feature of the workload is diminished.

5 Case study: network file system

5.1 Methodology

We next examined how well the policies studied in the previous section translate to a network file system. In this study, we use BlueFS, described previously in Section 3.3. As before, we do not allow BlueFS to use local storage so as to isolate the effect of network storage.

Recall that 802.11b defines a high-performance Continuous Access Mode (CAM) and a low-power Power-Saving Mode (PSM). The STPM policy used by BlueFS adaptively switches between these two modes depending upon observed network traffic [1]. In these experiments, it switches to CAM before beginning the third consecutive foreground RPC received within a short time window, and it switches to PSM after the network interface is idle for 300 ms. In the context of BlueFS and STPM, we therefore define the three policies studied as:

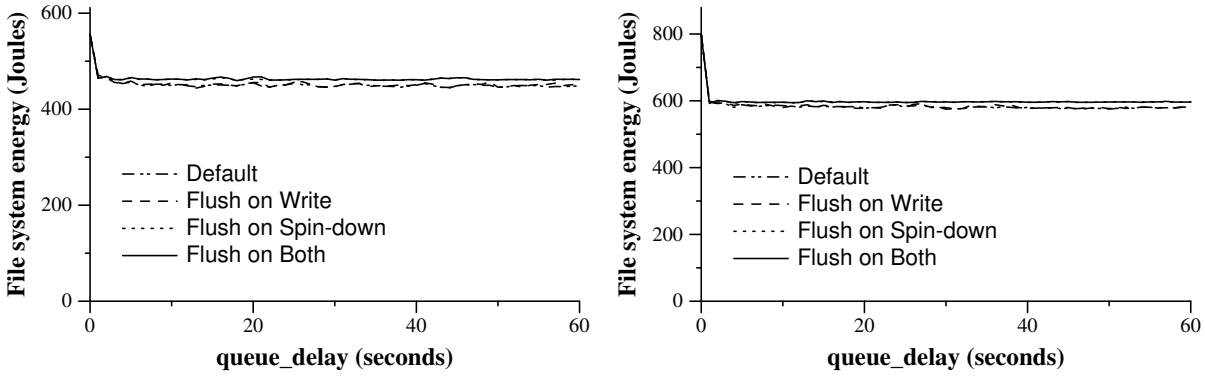
- *Flush on write.* Whenever any single RPC containing a modification is sent to the server, all queued modifications are also sent to the server.
- *Flush on PSM.* Before the network card is placed in PSM to save power, all queued modifications are sent to the server. This is the network equivalent of *flush on spin-down*.
- *Increasing queue_delay.* This allows a modification to be queued by the file system for a longer period of time. This is the BlueFS equivalent of `age_buffer`. By default, BlueFS sets `queue_delay` equivalent to Linux's default `age_buffer` of 30 seconds.

We used Drive-Thru to assess the potential benefit of each strategy for BlueFS running on the iPAQ described in Section 3. We again used the six full-length file traces shown in Figure 3. However, since the results of the study for all six traces were essentially equivalent, we show results for only two traces in Figure 7.

5.2 Results

Comparing Figures 5 and 7, it is immediately apparent that the three policies under study have a markedly different effect on the network file system than they do on the local disk file system. The *flush on PSM* policy appears to have a slight negative effect, increasing file system energy usage for all values of `queue_delay` greater than a few seconds. The *flush on write* policy has no noticeable impact, using essentially the same energy as the default policy. We have also observed that neither policy has a significant effect on trace replay time.

We believe that the observed difference in effectiveness is due to the difference between disk and network power management. While a disk is spun-down to save power, no data may be read from the drive; it must therefore spin up to service any write request that it receives. In contrast, when the network interface is in PSM, it can still transmit and receive data. While the latency of each RPC increases in PSM, the energy usage does not. Since write RPCs are asynchronous (i.e. they are sent by a background thread), STPM can (and does) leave the network interface in PSM while they are sent and received. Because write RPCs can efficiently be serviced in PSM, it is not necessary to flush them from the queue before transitioning the network card.



Each graph shows the results of implementing the three cache management policies described in Section 5.1 for BlueFS. We show only energy results since the policies studied had negligible performance impact. Note that the scales are different in each graph.

Figure 7. Results from network file system case study

The effect of `queue_delay` in Figure 7 is also striking. While all six traces show considerable benefit from setting `queue_delay` greater than zero, no trace showed more than a trifling benefit from increasing `queue_delay` beyond two seconds. This came as a complete surprise to us (the creators of BlueFS), since we expected the effect of `queue_delay` to mirror that of `age_buffer` for ext2.

In BlueFS, `queue_delay` directly affects the durability of file data, since writes are only guaranteed to be safe from client crash once the server replies to a RPC. Since there is almost no performance or energy cost, this study made a convincing case to us that we should decrease `queue_delay` from 30 to 2 seconds for the BlueFS server write queue—we explore this further in the next section.

The speedup obtained by using Drive-Thru is somewhat less for network file systems than for local file systems. The base trace takes longer to generate since operations must be performed synchronously and each operation requires a network round-trip. However, even for the generation of a single data point, Drive-Thru was 38–116 times faster than trace replay for the file system traces we studied. In total, we were able to generate the results in this case study over 13,000 times faster using Drive-Thru than we would have been able to generate them them if we had used trace replay on a live system.

6 Optimizing the Blue file system

We are applying the lessons from these case studies to BlueFS. Since BlueFS layers its client disk cache on top of ext2, the results of the first case study apply to its local cache, whereas the results of the second case study apply to its network communication.

The first lesson is that different storage devices require different policies. Policies such as *flush on spin-down* that benefit disk storage are detrimental to network storage. We therefore changed BlueFS to allow these policies to be selectively applied to each network or local storage device attached to the mobile computer. Note that BlueFS already allowed each network or local storage device to select its own `queue_delay`.

Our second lesson is that we should implement *flush on spin-down* for local disk-based storage. We modified the disk power management module used by BlueFS to call the Linux kernel routine that flushes dirty cache blocks for a particular device (`fsync_dev`) before the module spins down the disk. This required only 20 lines of source code.

We ran the 45-minute Purcell trace used in Section 3 on BlueFS with and without our modification. Our experimental setup was the same as before, except that we allowed BlueFS to use both the network server and a local cache on the microdrive's ext2 file system. As shown in Figure 8, using *flush on spin-down* for the local cache reduces file system energy usage by 12.4%. Note that this improvement is for a file system that has already been substantially optimized to reduce energy usage. Given that this change also reduces trace execution time, it seems a nice improvement. Although total energy savings for the entire system is only 1.4%, our experiment assumes that the handheld does not hibernate or employ similar system-wide power-saving modes during periods of inactivity. We make this conservative assumption because our file system traces do not record sufficient information to predict when the system would hibernate.

Finally, we decreased the value of `queue_delay` for the server write queue from 30 seconds to 2 seconds. Using the same 45-minute Purcell trace, we evaluated the

	Original BlueFS	BlueFS with flush on spin-down	Percentage improvement
Time (s)	2674 (2671–2677)	2663 (2661–2665)	0.4%
Total energy (J)	5482 (5465-5506)	5408 (5394–5423)	1.4%
File system time (s)	118 (115–121)	105 (105–107)	11.0%
File system energy (J)	597 (580–621)	523 (509–537)	12.4%

This shows the results of implementing the flush on spin-down policy in BlueFS. Each value reports measured replay time or energy usage for an approximately 45 minute segment of the Purcell file system trace. Each value shows the mean of three trials with the minimum and maximum given in parentheses.

Figure 8. Flush on spin-down in BlueFS

behavior of these two settings for the network-only configuration of BlueFS (omitting the local disk cache). Our results showed that the difference in energy usage between these two values was within experimental error of the 6 Joule difference predicted by Drive-Thru. Further, the time to replay the trace was equivalent within experimental error for the two settings. The benefit of less data loss after a crash seems well worth this cost.

7 Related Work

To the best of our knowledge, Drive-Thru is the first system to quickly and accurately evaluate storage power management by separating out time-dependent and time-independent behavior. The foundations of our methodology lie in trace replay, which has long been used as an evaluation technique in the storage research community [5, 16, 20, 29]. Recent efforts to improve the accuracy of trace replay [3] could benefit our methodology. However, our key observation is that trace replay that does not preserve operation interarrival times is too inaccurate to model storage power management, while trace replay that preserves interarrival times is too slow. Thus, while trace replay has been used previously to evaluate storage power management [1, 2, 19], the traces used in those studies have been, of necessity, quite short.

Drive-Thru's process of generating the base trace can be viewed as instance of a gray-box system [4]. Both Drive-Thru and the gray-box File Cache Content Detector (FCCD) use a real file cache to estimate cache contents. In contrast to our work, FCCD provides on-line estimates of cache contents to applications. Drive-Thru's cache content prediction is fundamentally off-line. Our current approach for evaluating each storage hierarchy is to modify the Drive-Thru simulator parameters in order to reflect the file system, cache, and device power management strategies for the system under study. Potentially, it may be feasible to automatically extract Drive-Thru parameters using techniques such as those outlined for semantically-smart disk systems [25].

Discrete event simulation of disk [5] and network [12, 26] behavior is not a contribution of this work. In fact, we use the Dempsey disk simulator directly in our methodology. Similarly, we could potentially use a network simulator such as QualNet [26] or NS-2 [12] to improve Drive-Thru's estimates for network file systems. The focus of our work is above the physical device in the storage hierarchy. We concentrate on the file system, file cache, and device driver where storage power management algorithms are typically implemented.

Similarly, all of the power management strategies that we investigate in our case study have been previously proposed by the research community. The contribution of our work is that we perform the first systematic study of these proposals that uses substantial traces of actual user activity to measure impact on typical file system workloads. Given the several surprising results that we generated through our case study, we believe that this type of comprehensive evaluation is of considerable merit.

A very large number of fixed and adaptive strategies have been proposed for controlling the power state of hard disks [7, 6, 9, 24]. This decision can be augmented by considering additional information provided by applications [2, 8, 28] or based on an OS-level energy budget [31]. It has also been observed that since disk power management algorithms are limited by the lengths of continuous idle time, it might be beneficial to rearrange or prevent disk accesses so that idle times could become longer [8, 22, 28].

8 Discussion and Future Work

We believe that the results of our case studies demonstrate the effectiveness of Drive-Thru's approach to power management evaluation. Because Drive-Thru generates accurate results quickly, we were able to look at a much larger set of data than previous researchers have used in their evaluations. This led to a number of surprising results, three of which we implemented in the Blue file system that we are currently developing.

At the same time, Drive-Thru has several limitations that must be acknowledged. First, our methodology relies on the ability to separate time-dependent and time-

independent behavior in storage hierarchies. This separation is not always clear. For instance, dirty blocks may be prematurely evicted from the Linux file cache due to memory pressure. Since the trace replay tool generates little memory pressure of its own, we have never seen this happen. However, Drive-Thru may not be able to handle extremely write-intensive workloads that write blocks at a faster rate than `updated` writes them to disk. We note that even on an iPAQ with an extremely limited memory capacity (64 MB), our results remain quite accurate.

A second limitation of Drive-Thru is that the accuracy of its evaluation results depend upon how well the particular traces used in the evaluation reflect the actual behavior of the system under study. Good traces of file system activity are a rare commodity. Thus, it may be hard to find a precisely representative sample. Certainly, the traces we used in our study may not be representative of the workload one might see on a Linux handheld. However, we note that this is a limitation of trace replay in general, and is not specific to Drive-Thru. Further, we believe that file system traces, even those not specifically collected on a target platform, are often more representative than application microbenchmarks. In the future, we plan to help remedy this limitation by using the trace capture tool that we have developed to generate traces of file system activity from handhelds and other small, mobile devices.

A final limitation of the Drive-Thru methodology is that we assume that the time-dependent component of storage system behavior is easy to model. This assumption has held for all the systems that we have modeled so far; we have been able to specify time-dependent behavior in a few hundred lines of source code or less. However, as storage systems continue to evolve and become more complex, it is possible that this assumption will no longer hold. One remedy, mentioned in the previous section, may be to use semantically-smart techniques to automatically extract simulation parameters.

9 Conclusion

We believe that the speed, accuracy, and portability of Drive-Thru make it an extremely valuable tool for the evaluation of storage power management policies. For our study of ext2, we were able to generate results over 40,000 times faster than trace replay on a live system that preserves operation interarrival times. At the same time, our results are within 3-5% of the values that trace replay generates. Further, since our trace replay tool is POSIX-compliant and our simulator is less than 1,000 lines of source code, we believe that Drive-Thru will port relatively easily to new file systems and platforms.

The case studies reported in this paper show the value of our tool. By studying a more comprehensive set of workloads than those studied in the past, we have been able to gain the following insights about the effect of cache behavior on storage power management:

- For the ext2 file system, a *flush on spin-down* policy reduces file system energy usage by 11% with no performance cost and without the need to modify applications. Further, if *flush on spin-down* is implemented, a *flush on write* policy realizes little additional benefit.
- Conversely, for the BlueFS network file system, *flush on spin-down* increases energy usage, while *flush on write* has no significant effect.
- Increasing the Linux `age_buffer` parameter to a value greater than 30 seconds yields little energy savings (8%) to offset the additional danger of data loss during system crash.
- The BlueFS `queue_delay` parameter for network RPCs can be reduced to 2 seconds to improve consistency without substantially sacrificing performance or energy usage.
- Operating systems should not set `age_buffer` and the disk spin-down time to the same value.

The last two results came as complete surprises to us. This demonstrates an additional benefit of Drive-Thru: because we are able to evaluate a large parameter space quickly, we have the opportunity to mine for insights that would not otherwise be apparent. We are currently using Drive-Thru in our ongoing development of the Blue file system. Our efforts have already yielded the three enhancements described in Section 6. Based on these results, we are confident that Drive-Thru can be of considerable use in developing energy-efficient storage.

Acknowledgments

We thank Fengzhou Zheng and the Dempsey team for making their tool available to us. We would also like to thank Minkyong Kim, Jay Lorch, and Lily Mummert for providing file system traces. Manish Anand, Edmund B. Nightingale, Ya-Yunn Su, the anonymous reviewers, and our shepherd, Carla Ellis, made many suggestions that improved the quality of this paper. The work is supported by the National Science Foundation under award CCR-0306251, and by an equipment grant from Intel Corporation. Jason Flinn is supported by NSF CAREER award CNS-0346686. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Intel, the University of Michigan, or the U.S. government.

References

[1] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Self-tuning wireless network power management. In *Proceedings of the 9th Annual Conference on Mobile Computing and Networking* (San Diego, CA, September 2003), pp. 176–189.

[2] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Ghosts in the machine: Interfaces for better power management. In *Proceedings of the 2nd Annual Conference on Mobile Computing*

Systems, Applications and Services (Boston, MA, June 2004), pp. 23–35.

[3] ANDERSON, E., KALLAHALLA, M., UYSAL, M., AND SWAMINATHAN, R. Buttress: A toolkit for flexible and high fidelity I/O benchmarking. In *Proceedings of the USENIX FAST '04 Conference on File and Storage Technologies* (San Francisco, CA, March 2004), Hewlett-Packard Laboratories, USENIX Association, pp. 45–58.

[4] ARPACI-DUSSEAU, A., AND ARPACI-DUSSEAU, R. Information and control in gray-box systems. In *Procedings of the 18th ACM Symp. on Operating Systems Principles* (Banff, Canada, October 2001), pp. 43–56.

[5] BUCY, J. S., GANGER, G. R., AND CONTRIBUTORS. The DiskSim simulation environment version 3.0 reference manual. Tech. Rep. CMU-CS-03-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 2003.

[6] DOUGLIS, F., KRISHNAN, P., AND BERSHAD, B. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, MI, April 1995), pp. 121–137.

[7] DOUGLIS, F., KRISHNAN, P., AND MARSH, B. Thwarting the power-hungry disk. In *Proceedings of 1994 Winter USENIX Conference* (San Francisco, CA, January 1994), pp. 292–306.

[8] HEATH, T., PINHEIRO, E., AND BIANCHINI, R. Application-supported device management for energy and performance. In *Proceedings of the 2002 Workshop on Power-Aware Computer Systems* (February 2002), pp. 114–123.

[9] HELMBOLD, D. P., LONG, D. D. E., AND SHERROD, B. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking* (1996), pp. 130–142.

[10] HITACHI GLOBAL STORAGE TECHNOLOGIES. *Hitachi Microdrive Hard Disk Drive Specifications*, January 2003.

[11] IEEE LOCAL AND METROPOLITAN AREA NETWORK STANDARDS COMMITTEE. *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications.* New York, New York, 1997. IEEE Std 802.11-1997.

[12] INFORMATION SCIENCES INSTITUTE. NS-2 network simulator, 2003. http://www.isi.edu/nsnam/ns/.

[13] KIM, M., NOBLE, B., CHEN, X., AND TILBURY, D. Data propagation in a distributed file system, July 2004.

[14] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[15] KRASHINSKY, R., AND BALAKRISHNAN, H. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)* (Atlanta, GA, July 2002).

[16] MUMMERT, L., EBLING, M., AND SATYANARAYANAN, M. Exploiting weak connectivity in mobile file access. In *Proceedings of the 15th ACM Symp. on Op. Syst. Principles* (Copper Mountain, CO, Dec. 1995).

[17] MUMMERT, L., AND SATYANARAYANAN, M. Long term distributed file reference tracing: Implementation and experience. *Software Practice and Experience 26*, 6 (1996), 705–736.

[18] NETWORK WORKING GROUP. *NFS: Network File System protocol specification*, March 1989. RFC 1094.

[19] NIGHTINGALE, E. B., AND FLINN, J. Energy-efficiency and storage flexibility in the Blue File System. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004), pp. 363–378.

[20] OUSTERHOUT, J. K., COSTA, H. D., HARRISON, D., KUNZE, J. A., KUFER, M., AND THOMPSON, J. G. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the 10th ACM Symp. on Op. Syst. Principles* (Orcas Island, WA, Dec. 1985), pp. 15–24.

[21] PAPATHANASIOU, A. E., AND SCOTT, M. L. Increasing disk burstiness through energy efficiency. Tech. Rep. 792, Computer Science Department, University of Rochester, November 2002.

[22] PAPATHANASIOU, A. E., AND SCOTT, M. L. Energy efficiency through burstiness. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, October 2003), pp. 444–53.

[23] ROSENBLUM, M., BUGNION, E., HERROD, S. A., WITCHEL, E., AND GUPTA, A. The impact of architectural trends on operating system performance. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)* (Copper Mountain, CO, December 1995), pp. 285–298.

[24] SIMUNIC, T., BENINI, L., GLYNN, P., AND MICHELI, G. D. Dynamic power management for portable systems. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00)* (Boston, MA, August 2000), pp. 11–19.

[25] SIVATHANU, M., PRABHAKARAN, V., POPOVICI, F. I., DENEHY, T. E., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Semantically-smart disk systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technology* (San Francisco, CA, March/April 2003), pp. 72–88.

[26] TECHNOLOGIES, S. N. QualNet Simulator. http://www.scalable-networks.com.

[27] VOGELS, W. File system usage in Windows NT 4.0. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Kiawah Island, SC, December 1999), pp. 93–109.

[28] WEISSEL, A., BEUTEL, B., AND BELLOSA, F. Cooperative I/O: A novel I/O semanatics for energy-aware applications. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 117–129.

[29] WILKES, J., GOLDING, R., STAELIN, C., AND SULLIVAN, T. The HP AutoRAID hierarchical storage system. In *Proceedings of the 15th ACM Symp. on Op. Syst. Principles* (Copper Mountain, CO, Dec. 1995), pp. 96–108.

[30] ZEDLEWSKI, J., SOBTI, S., GARG, N., ZHENG, F., KRISHNAMURTHY, A., AND WANG, R. Modeling hard-disk power consumption. In *Proceedings of the 2nd USENIX Conference on File and Storage Technology* (San Francisco, CA, March/April 2003), pp. 217–230.

[31] ZENG, H., ELLIS, C. S., LEBECK, A. R., AND VAHDAT, A. Currentcy: A unifying abstraction for expressing energy management policies. In *Proceedings of the 2003 USENIX Annual Technical Conference* (San Antonio, TX, June 2003), pp. 43–56.