Considering the Energy Consumption of Mobile Storage Alternatives

Fengzhou Zheng* Nitin Garg* Sumeet Sobti* Chi Zhang* Russell E. Joseph†
Arvind Krishnamurthy‡ Randolph Y. Wang*

Abstract

This paper is motivated by a simple question: what are the energy consumption characteristics of mobile storage alternatives? To answer this question, we are faced with a design space of multiple dimensions. Two important dimensions are the type of storage technologies and the type of file systems. In this paper, we explore some options along each of these two dimensions. We have constructed a logical-disk system, which can be configured to run on different storage technologies and to emulate the behavior of different file systems. As we explore these configuration options, we find that the energy behavior is determined by a complex interaction of three factors: the power management mechanism of the storage device, the distribution of idleness in the workload, and the file system strategies that attempt to exploit and bridge these first two factors.

1. Introduction

Two important factors in determining the energy consumption characteristics of a mobile storage system are the type of storage technology and the type of file system. The storage technologies that we study are IBM Microdrives, flash PC cards, and wireless LAN cards (used for connecting to storage servers). These devices have different performance and power characteristics, and different built-in power-management mechanisms.

In terms of file system flavors, we begin by considering an update-in-place file system and a log-structured file system. The conventional wisdom is that log-structured file systems tend to be more efficient for many workloads; and increased performance efficiency often translates into lower energy consumption. A log-structured design, however, is an umbrella-term that encompasses a number of features. We isolate the impact of each of these features on the energy consumption of the storage system.

We have constructed a logical-disk system, which can be configured to run on different technologies and to emulate the behavior of different file systems. The system runs on both Linux laptops and iPaqs. By executing real I/Os, the system allows us to collect accurate energy consumption statistics. As we explore the various configurations, we study the interactions between device characteristics and file system techniques. Although it is true that a log-structured storage system can translate its performance benefits into energy savings under certain high I/O rates, we find that in less stressful workloads, the energy behavior is determined by a complex interaction of three factors: (1) the storage device's power management mechanism whose task is to detect idle periods, (2) the distribution of idleness in the workload, and (3) the file system strategies that attempt to exploit and bridge the above two factors. When the storage device lacks good power management, a log-structured design provides a robust way of introducing additional device burstiness to reduce its reliance on a sophisticated power management scheme. For less stressful I/O loads, there exist alternative hybrid designs that can behave just as well as a log-structured file system without incurring the overheads associated with storage garbage collection.

The remainder of this paper is organized as follows. Section 2 describes the performance and power characteristics of the mobile storage devices used in this study. Section 3 analyzes the file system alternatives. Section 4 describes the implementation of a logical disk system that allows us to explore the various options. Section 5 details the measurement apparatus and the workloads. Section 6 presents the experimental results. Section 7 describes some of the related work. Section 8 concludes.

2. Mobile Storage Alternatives

Although the IBM Microdrive has the form factor of compact flash, it has all the components of a magnetic hard disk. A flash PC card has relatively better performance and power characteristics than the Microdrive; but it has several

*Department of Computer Science, Princeton University, {zheng, nitin, sobti, chizhang, rywang}@cs.princeton.edu. Supported by NSF grant CCR-9984790.
†Department of Electrical Engineering, Princeton University, rjoseph@ee.princeton.edu.
‡Department of Computer Science, Yale University, arvind@cs.yale.edu. Supported by NSF grants CCR-9985304, ANI-0207399, and CCR-0209122.

Name	Read Latency (ms)	Write Latency (ms)	Read Bandwidth (MB/s)	Write Bandwidth (MB/s)
IBM 1GB Microdrive (MD)	21.7	23.3	3.0	1.9
Lucent 11 Mbps WaveLAN (WL)	102.7	102.7	0.1	0.3
SimpleTech 1GB Compact Flash (CF)	1.5	4.6	3.5	2.3

Table 1. Performance characteristics. Latency experiments measure the cost of random reads/writes of 4 KB blocks. Bandwidth experiments access data in 256 KB chunks.

Name	Operating Voltage (V)	"Idle" Power (mW)	Small Read Power (mW)	Small Write Power (mW)	Big Read Power (mW)	Big Write Power (mW)
MD	3.3	60	531	530	577	575
WL	5.0	66.1	220	171	444	650
CF	3.3	2.9	73.7	125	82	217

Table 2. Average power consumed during reads/writes. Small read and small write operations involve random access of 4 KB blocks. Big read and big write operations correspond to reading and writing 256 KB chunks.

disadvantages compared to the Microdrive. One of them is its worse durability: a data location on a flash card must be "erased" before new data can be written and the number of erasure cycles is typically limited to a few hundred thousand. The Microdrive, on the other hand, has a mean time to failure of about a million hours. Furthermore, the current technology trend is such that magnetic disk technology is likely to continue to enjoy lower cost per megabyte ratio [9]. We also study a wireless LAN card (which is typically not considered as a "storage technology"), since there are situations where the network storage is preferred due to its benefits of increased data reliability and ease of sharing.

2.1. Performance and Power Characteristics

Tables 1 and 2 show the performance and energy consumption characteristics of the mobile storage devices that we study in this paper. Although one may be tempted to compare the different technologies against each other throughout this paper, we do not intend to "pick sides." In fact, we believe that each technology has its own reason for existence. Our interest, instead, lies in understanding how the hardware characteristics of *each* of these technologies influence the software file system strategies in arriving at energy-efficient mobile storage solutions. When examining the hardware characteristics, we highlight the following.

Performance difference of reads and writes. Some of these technologies have noticeably lower performance and higher energy consumption for writes than reads. Asynchronous buffering of writes, which allows overwritten data to be deleted in main memory, could enhance file system performance. Buffering, however, may require the use of additional memory, which consumes additional energy.

The relationship between latency and bandwidth. For all of these technologies, when the total number of bytes transferred is the same, it is better to perform a single large I/O than to perform a number of small I/Os from the perspectives of both improving performance and saving energy. The latency/bandwidth relationships of the different technologies, of course, are quantitatively different and may dictate different file system strategies.

The relationship between latency and locality. Reading a flash card, for example, incurs roughly the same latency and energy regardless of the accessed location. This is not true for the Microdrive. Also, for the flash card, writing same amount of data with different levels of locality may incur different costs due to different number of erasure cycles.

Power management schemes. The firmware of a storage device may instruct the device to enter low-power modes during idle periods. The sophistication and effectiveness of such power management schemes will affect how a file system may exploit them. We next describe the power management schemes of these technologies.

2.2. Power Management Schemes

SimpleTech. The idle mode on this card consumes very little power, and the card goes back to the idle mode as soon as it finishes servicing an I/O request. One can view these cards as having "perfect" power management.

WaveLAN. The wireless network cards provide support for systems software to better manage their energy consumption. These cards can operate in a low power mode called "doze" or idle mode, which consumes a fraction of the power used when transmitting or receiving. The card enters low power mode adaptively, and the user can control the duration for which it stays in that mode before it comes back up to check for messages. Our experiments use a default value of 100 ms for this duration. The choice of a relatively long duration is a trade-off in favor of low power consumption resulting in relatively worse performance.

Microdrive. The Microdrive has the most sophisticated power management support called Adaptive Battery Life Extender-2 (ABLE-2) [1]. The Microdrive has two idle modes: "performance idle mode" and "low power idle mode." In performance idle mode, the drive is spinning and can immediately respond to a new command. The Microdrive consumes about 480 mW of power in this mode. In low power idle, the disk head is unloaded, and the disk incurs a delay of 300 ms to respond to a new command. The Microdrive consumes about 180 mW in this mode. The Microdrive also has a standby mode, which consumes about 60 mW and requires about 800 ms of wake-up time. The power consumed during wake-up periods is comparable to the power consumed during seeks. The Microdrive uses request history to transition between the various modes.

3. File System Alternatives

To understand the impact of file system issues on energy consumption, we begin by considering two well-known file systems: the Unix File System (UFS) [19] and the Log-structured File System (LFS) [21]. Under UFS, once disk blocks are allocated for a file, the disk locations of the file data do not move and updates are performed in place. Each of the small disk writes may incur a costly mechanical delay. Under LFS, file data does not necessarily reside at fixed locations—newly created data is accumulated in a large memory-resident buffer (called a *segment*) and filled buffers are appended to the end of a segmented disk-resident log. By replacing small disk writes of UFS with large disk appends, the LFS designers amortize disk latency over many writes to achieve better performance. As overwrites occur, however, "holes" are made in the log; and a compactor (called a *segment cleaner*) may need to run periodically to regenerate free disk segments.

The conventional wisdom is that LFS tends to be faster for many workloads; and faster programs tend to consume less energy. We would like to examine LFS closely and study its features that may impact energy consumption. Some of the features are desirable for saving energy while others are not; these features have different degrees of impact depending on the underlying device; and there may exist a hybrid design that selectively incorporates a subset of these features, without embracing the whole LFS doctrine.

3.1. Energy-Relevant Design Issues

Asynchrony. Asynchronous I/Os allow a greater degree of flexibility in scheduling. Asynchronous writes allow overwritten data to be deleted before they reach storage devices. This feature tends to both improve performance and lower energy consumption.

Burstiness. I/O burstiness may degrade response time as many requests contend for resources. On the other hand, increased burstiness can be a blessing for reducing energy consumption: as I/Os are closely clustered in time, there are more long idle periods that can be more readily detected and exploited by the power management schemes.

Layout. For storage devices that have non-uniform access times, layouts that increase locality tend to improve performance and lower energy consumption.

Background activities. Background activities, such as reorganizing data for improved read locality or compacting free space for improved write locality, can improve burst performance. However, whether it is worthwhile to consume "background energy" to save "foreground energy" is a trade-off that needs to be considered for different technologies and different workloads.

Protocol. For a device (such as the wireless LAN card) that exports an interface that is richer than the simple sector read/write interface of a disk, the protocol design raises additional questions regarding the granularity of communication and whether to transmit operations or data.

3.2. Hybrid Alternatives

For storage devices that export a disk interface, LFS can be considered as an extreme design point in terms of the first four of the above issues: LFS increases write asynchrony, write burstiness, write locality, and background activity (in the form of cleaning). UFS variants typically have less asynchrony and less burstiness with many UFS implementations performing synchronous metadata updates. However, there exist techniques that loosen this restriction and increase asynchrony [7]. To isolate and study the impact of these techniques, we consider several variants of UFS. We begin with a version of UFS that performs all writes synchronously. We refer to this basic implementation as *In-place File System* (IFS). We then consider a variant that buffers writes and performs delayed updates in a FIFO fashion as the memory buffer fills up. The memory buffer enables some of the blocks to be overwritten before it is presented to the storage system. We refer to this variant as *IFS with Delayed Writes* (IFS-DW). For the next file system variant, we increase burstiness by flushing all the writes in the memory buffer in a batched fashion when the buffer fills up. We refer to this variant as *IFS-Batch*. When IFS-Batch flushes its memory buffer, it may carefully schedule the I/Os to increase write locality. We refer to this variant as *IFS-Batch-Sched*. In the extreme case, we introduce data relocation and cleaning to reach the final design point: LFS.

The network device allows even more flexibility. One approach is to use the wireless LAN card as if it were a disk – we read and write blocks to the network and a remote "storage server" performs the actual I/Os to a server disk. We will call this the *network disk* approach. We can run any one of the file system variants on top of this network disk. The other approach is to use a higher-level file system protocol such as NFS. Of course, alternative protocols exist; and the network disk and NFS provide different functionality (such as support for sharing). Nevertheless, we believe it is interesting to understand the energy consumption be-

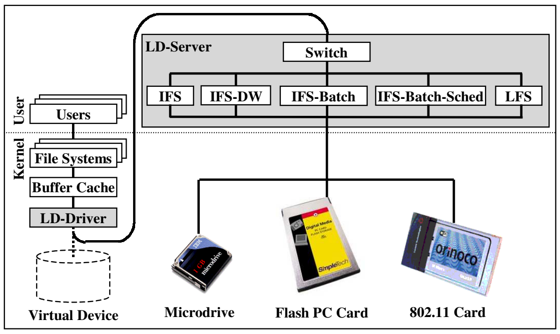

Figure 1. Implementation of multiple file system alternatives in a logical disk. The shaded parts, the LD-Driver and the LD-Server, are the software components that we have implemented.

havior of these options for several reasons. First, they are simple practical options that a PDA user may use *today* and we would like to compare their energy consumption. (Despite its lack of support for sharing, the network disk is perfectly adequate for scenarios that do not involve concurrent write sharing.) Second, these two options use two different styles of communication. When operating on the network disk, the local file system caches and communicates both data and metadata at the granularity of blocks. Under NFS, on the other hand, the client caches metadata differently from file data blocks and transmits operations (such as a "mkdir" operation) instead of blocks (such as a directory block). By doing so, NFS may transmit fewer bytes but more messages. The impact of such protocol differences on energy consumption is what interests us.

4. Implementation using Logical Disks

We would like to implement a range of file system alternatives and evaluate their energy consumption on different underlying storage technologies. The implementation of a file system involves many additional degrees of freedom such as the way it manages the buffer cache and its metadata cache. When evaluating the different alternatives, we would like to keep most of the tangential issues constant and focus only on the central issues that we have discussed in the last section. Modifying or building several file system variants that happen to share common mechanisms such as buffer cache management can be a complex undertaking.

To maximally leverage the existing file system work, and maintain a consistent framework for all implementations, we adopt a design based on a *logical disk* [2, 23]. A logical disk appears to the native in-kernel file systems as a normal disk: it allows existing file systems to read and write *logical disk addresses*. A particular implementation of a logical disk can map these logical addresses to physical addresses in any way that it chooses; and the I/Os to the underlying device can be scheduled in a flexible way. This degree of flexibility turns out to be sufficient for our study.

Figure 1 illustrates the components of our Linux-based implementation. The system consists of two main components: the logical disk driver (LD-Driver) and the logical disk server (LD-Server). The LD-Driver is a pseudo block device driver that exports the interface of a disk. Upon receiving I/O requests, the LD-Driver forwards them via upcalls to the LD-Server, a user-space process. The LD-Server can be configured to behave as any one of the five local file system variants discussed in Section 3.2: IFS, IFS-DW, IFS-Batch, IFS-Batch-Sched, or LFS. The existing kernel services, such as the file systems and the buffer cache, work unmodified and provide a consistent framework for comparison. This system allows us to run existing applications.

The IFS module in the LD-Server implements the simplest logical-to-physical address mapping: an identity mapping. Scheduling in the IFS module is also simple: it performs an immediate read or write request upon the receipt of an upcall. The IFS-DW module adds a write buffer and delays the writes till the buffer is full or dirty data has been present in the buffer for over 30 seconds. When the buffer becomes full, the writes are performed gradually to create space for subsequent writes. The write buffer decreases the number of disk writes by absorbing block overwrites. The IFS-Batch flushes the write buffer in a single burst instead of draining it gradually. It increases burstiness to allow the underlying device to be idle for longer durations thereby triggering its power management mechanism. IFS-Batch-Sched module sorts the buffered writes by their addresses and performs the writes in the sorted order. In cases of crashes, this crude scheduling algorithm may compromise the integrity of the underlying file system as it aggressively reorders writes. Unlike the other flavors, IFS-Batch-Sched is not meant to be a practical solution. Moreover, sorting writes by their addresses may not be the most effective scheduling algorithm. More sophisticated and "correct" scheduling alternatives exist [7], but we would like to gain some insights into the energy consumption implications before we attempt these more complex implementations.

The LFS module implements a *log-structured logical disk* [2]. Logical addresses of writes that are performed next to each other in time are mapped to contiguous physical addresses. These writes are buffered in memory-resident segments, which are then appended to a device-resident log. The LFS module includes a segment cleaner. When the cleaner runs, it cleans the least utilized segments first. The cleaner can be configured to run in one of two possible modes. The *lazy-cleaner* runs only when the number of free segments falls below a minimum value, which is twice the number of memory-resident segments in our implementation. The *auto-cleaner*, on the other hand, *actively* initiates cleaning when the LD-Server has been idle for more than 5 seconds. An active-cleaning phase ends either when the auto-cleaner has cleaned all the partially-filled segments, or

Processor	Intel StrongARM SA-1110 206 MHz
Memory	16 MB ROM, 32 MB SDRAM
Battery	950 mAh Lithium Polymer
OS	Linux 2.4.7-rmk3-np1

Table 3. Characteristics of the iPaq Pocket PC H3635.

when a new I/O request is received. If there is no idle time, then the behavior of the auto-cleaner degenerates to that of the lazy-cleaner. Although the lazy-cleaner may cause poor response time due to on-demand cleaning, it may be preferable for reducing energy consumption because it keeps the amount of cleaning to a minimum and it also increases the burstiness of the cleaning traffic.

The LD-Server can be configured to manage one of three types of devices: a Microdrive, a flash PC card, or a wireless LAN card. The Microdrive and the flash card already present a disk interface and the interaction with them is simple. To avoid unwanted pollution of the buffer cache, the LD-Server interacts with them using raw I/O through the Linux `/dev/raw/raw*` interface. When the LD-Server runs on a wireless LAN card, a *storage server* runs on a remote machine to accept the physical I/Os performed by the LD-Server via a TCP socket. If the LD-Server is configured to run in the IFS modes, individual requests are forwarded as separate network requests. If the LD-Server is configured to run in the LFS mode, the segment that is being flushed is transmitted as a single TCP message. Because locality is not an issue with the network device itself, the distinction between IFS-Batch and IFS-Batch-Sched, is no longer relevant. Although it is possible to flush the IFS-Batch write buffer as a single TCP message, we choose to send separate messages for the buffered writes (in a way that is similar to how the write buffer is flushed when IFS-Batch is running on a Microdrive or a flash card) to isolate the energy consumption impact of the number of messages.

While interactions with devices allow us to run real applications and gather accurate energy consumption numbers, physically performing all I/Os can be slow. Some phases of our experiments (such as the warm-up periods) do not require us to physically perform the I/Os. In these situations, the system bypasses the interactions with storage devices, effectively turning itself into a functional file system simulator that runs at a much faster speed.

5. Measurement Apparatus and Workloads

The measurement setup is pictured in Figure 2. It includes a PC card extender (1) attached to a shunt resistor (2) in series with the card's power supply. Following the general approach taken in [6, 12], we measure the voltage across the shunt resistor. The card extender houses a mobile storage device (4) and is inserted into a PCMCIA sleeve of an iPaq (3). A digital multimeter (5) is then used to mea-

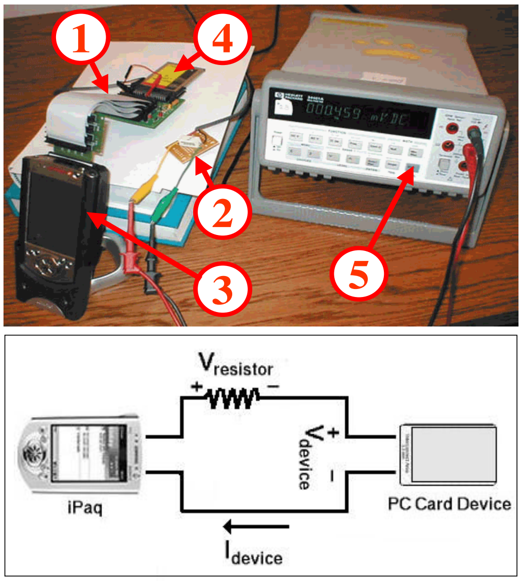

Figure 2. Measurement apparatus and its schematic diagram. (1) PC card extender. (2) Shunt resistor. (3) iPaq with a PCMCIA sleeve. (4) Flash PC card. (5) Digital multimeter.

sure the voltage and integrate it over a sampling interval to compute the average voltage measurement for that interval. These measurements are logged via a serial link to an independent computer. After independently determining the resistance of the shunt resistor and voltage of the power supply, we deduce the current delivered to the device using Ohm's Law ($V_{resistor} = I_{device} R_{resistor}$). We then compute the average power consumed by the device using $P_{device} = V_{device} I_{device}$.

One conscious decision that we made in constructing the measurement apparatus is to measure the energy consumption of the mobile storage device alone instead of including the energy consumption of the whole iPaq. We choose the iPaq because it appears to be a reasonable representative of a PDA in terms of its CPU speed and the amount of memory present, both of which can influence the energy consumption of the storage device. This study, however, is foremost a study of the energy behavior of the storage system. We are unwilling to allow the energy consumption artifacts of the host device to unnecessarily complicate the analysis of the results. Some example artifacts include: the energy consumption of the host CPU, its memory, and the extra devices — one may choose to run the storage system under a processor architecture that has comparable speed but different energy characteristics, or with a memory management system that manages power differently, or with no extra devices. Indeed, we have reasons to believe that the energy consumed by memory on the iPaq is far from optimal. We therefore believe that it is undesirable to burden the analysis with these tangential issues.

In choosing workloads for our experiments, we decided not to confine ourselves to the applications that are currently

	Duration (hours)	Number of I/Os	Number of reads	Number of writes	Percentage of Idle Periods				
					$<$ 0.5 secs	$<$ 1 sec	$<$ 5 secs	$<$ 10 secs	$<$ 60 secs
Trace 1	3	38,081	12,463	25,618	61.7	71.5	93.3	96.4	99.2
Trace 2	3	202,088	106,727	95,361	72.8	87.1	99.9	99.9	100.0
Trace 3	5	37,416	6,445	30,971	83.0	87.8	94.8	96.8	99.9
Trace 4	1	56,638	14,113	42,525	93.2	95.2	99.9	100.0	100.0

Table 4. Trace characteristics.

available on mobile devices. The applications, the operating system and hardware support have not matured for mobile computers; therefore, we do not believe that the limited set of applications that are currently available for mobile devices constitutes a representative set.

Having refused to confine ourselves to existing applications, we believe that it is in fact not as undesirable as it may first appear to instead use some workloads collected on desktop computers. We believe that there is very little reason why most of these desktop applications should not be made available to a mobile user. Although we make no claim about the representativeness of the chosen workloads, we believe that the measurements obtained for these traces still represent interesting data points.

We use five workloads. The first two are disk traces collected on Dell Dimension 8100 desktops running Windows 2000. "Trace 1" was collected on a desktop with 256 MB of main memory. We use a three-hour period comprising of I/O requests accessing about 149 MB of data. "Trace 2" was collected on a desktop with 128 MB of main memory, and we again use a three-hour snippet of I/O accessing a total of 789 MB of data. During the monitoring process, the users were performing activities that are typical to personal computer users, such as reading email, web browsing, document editing, and playing multimedia files. We used some simple tools (such as `tracelog` and `tracedmp`) from Microsoft's Windows 2000 resource kits to collect these traces.

Our third workload is a disk trace gathered on a file server at HP Labs in 1992 [22]. One of the characteristics of this workload that intrigued us is the remarkable fact that the amount of computing power, in terms of CPU power, the amount of memory, and the capacity of the storage devices, available on that server then is roughly equivalent to what is available on an iPaq today! We use a five-hour period that contains accesses to 146 MB of data. We will refer to this workload as "trace 3." The fourth workload is an updated version of the third: a trace collected on an upgraded version of the same server at HP Labs in 1999. This trace has a higher I/O rate: during a one-hour measurement period, the trace contains accesses to a total of 226 MB of data. We will refer to this workload as "trace 4."

The characteristics of the four traces are summarized in Table 4. Notice that traces 1 and 3 have a low I/O rate and perform more writes. In contrast, traces 2 and 4 are I/O intensive and trace 2 performs more reads. In our measurements, we play each of the I/Os contained in the trace synchronously from a user level process running on the iPaq. The logical disk implementation performs physical I/Os to the underlying mobile storage devices, allowing real energy consumption statistics to be collected.

The above workloads are disk traces. They cannot be played to an NFS client, which is an option that we would like to evaluate. For this purpose, we use a fifth benchmark, a synthetic one, called "MMAB". MMAB or "Modified MAB" is an enhanced version of the "Modified Andrew Benchmark" [20]. MMAB has four steps, each exhibiting a different mix of file system operations. The first step is metadata intensive and creates a directory tree of 50,000 directories, in which every non-leaf directory has ten subdirectories. The second step is data-write intensive and creates one large 256 MB file and many small 4 KB files. It creates five small files in each of the directories in the first five levels of the directory tree, resulting in a total of about 55,000 small files. The third step performs file attribute operations. It first performs a recursive `touch` on all the directories and files; it then computes disk usage of the directory tree by invoking `du`. The fourth and final step reads files. It first performs a `grep` on each file; it then reads all the files again by performing a `wc` on each file.

6. Experimental Results

We begin by examining the device-independent characteristics of the I/O generated by the different file systems. Table 5 summarizes the number of I/O operations and the distribution of the idle periods. As expected, IFS-Batch and LFS increase burstiness and a greater portion of the time is spent in longer idle periods.

In the rest of this section, we study the energy consumed by the file systems running on different devices for different workloads. We also evaluate the impact of the utilization of the storage device on the energy consumed by LFS and compare different cleaning strategies. Finally, for the network device, we study the energy consumption of two network storage protocols: NFS3 and a network disk.

6.1. Zero Idle Time

We now study the energy consumed by the devices when we play the traces with all of their idle periods eliminated. These measurements quantify the energy consumed by the devices in the simplest setting where the power manage-

		Number of I/Os	Number of reads	Number of writes	Percentage duration of the experiment spent in idle periods of length					
					< 0.5 secs	0.5 – 1 sec	1 – 5 secs	5 – 10 secs	10 – 60 secs	> 60 secs
Trace 1	IFS	38,081	12,463	25,618	1.4	3.1	14.8	8.4	22.8	49.5
	IFS-DW	31,315	12,453	18,862	1.0	2.0	11.3	6.1	21.1	58.5
	IFS-Batch	32,063	12,453	19,610	0.4	0.1	1.7	2.0	12.2	83.6
	LFS	32,319	12,453	19,866	0.4	0.1	1.7	2.0	12.2	83.6
Trace 2	IFS	202,088	106,727	95,361	12.0	39.7	46.8	1.3	0.2	0.0
	IFS-DW	187,448	106,535	80,913	10.0	38.8	49.5	1.0	0.7	0.0
	IFS-Batch	188,368	106,555	81,813	5.2	23.5	33.6	9.9	27.8	0.0
	LFS	189,115	106,555	82,560	5.2	23.5	33.6	9.9	27.8	0.0
Trace 3	IFS	37,416	6,445	30,971	3.8	3.4	14.8	11.8	65.8	0.4
	IFS-DW	18,264	6,117	12,147	1.8	0.4	3.4	4.9	35.8	53.7
	IFS-Batch	19,174	6,135	13,039	0.8	0.3	2.2	3.0	22.3	71.4
	LFS	19,293	6,135	13,158	0.8	0.3	2.2	3.0	22.3	71.4
Trace 4	IFS	56,638	14,113	42,525	23.6	8.9	62.7	4.8	0.0	0.0
	IFS-DW	49,347	13,775	35,572	23.2	8.7	62.7	5.4	0.0	0.0
	IFS-Batch	50,239	13,804	36,435	17.3	6.1	46.1	20.9	9.6	0.0
	LFS	50,569	13,804	36,765	17.3	6.1	46.1	20.9	9.6	0.0

Table 5. Characteristics of I/O operations generated by the file systems for the different traces.

	Trace 1				Trace 2			
	IFS	IFS-DW	IFS-Batch	LFS	IFS	IFS-DW	IFS-Batch	LFS
MD	209	229	214	48	1109	1134	1066	434
WL	598	579	455	431	3457	3648	2830	2613
CF	13.0	13.0	12.6	8.4	65.9	65.5	62.3	44.0
	Trace 3				Trace 4			
	IFS	IFS-DW	IFS-Batch	LFS	IFS	IFS-DW	IFS-Batch	LFS
MD	136	139	137	63.4	404	434	416	97.7
WL	277	268	224	183	874	846	899	747
CF	6.9	6.1	6.8	4.9	21.3	21.2	20.5	17.1

Table 6. Energy measurements (Joules) with all idle periods eliminated.

ment mechanisms of the devices are not triggered. We also keep the storage utilization sufficiently low so that the cleaner is not invoked under LFS.

The results of this experiment are summarized in Table 6. The devices operate in high-power modes throughout the duration of the experiment. For this workload, performance gains translate almost directly into energy savings. LFS benefits from write locality and therefore performs better than the other file system variants. What is interesting is that IFS-Batch, the IFS variant that attempts to increase burstiness to exploit the power management mechanisms of the devices, is counter-productive in certain instances—under high I/O load, the smoother traffic presented by IFS results in more efficient I/O than the bursty traffic of IFS-Batch that results in greater queueing delays.

6.2. Ample Idle Time

We next study the behavior of traces with low I/O rate. We play traces 1 and 3 at their original speeds, restoring the ample amount of idle time present in the traces. The results of these experiments are summarized in Table 7. Whereas the bursty traffic produced by IFS-Batch had provided no gain in Table 6, IFS-Batch now delivers various degrees of energy savings. The exact amount of gain depends on the power management schemes of the underlying device.

	Trace 1				Trace 3			
	IFS	IFS-DW	IFS-Batch	LFS	IFS	IFS-DW	IFS-Batch	LFS
MD	1871	1883	1158	1105	2865	2110	1659	1616
WL	1154	1131	1115	1093	1415	1398	1374	1366
CF	46.4	45.5	42.0	41.3	59.9	57.2	57.6	55.4

Table 7. Energy measurements (Joules) of the traces 1 and 3.

	Trace 1			Trace 3		
Energy	IFS	IFS-Batch	LFS	IFS	IFS-Batch	LFS
Active	1430	625	566	2061	666	619
Idle	441	533	539	804	993	997
Total	1871	1158	1105	2865	1659	1616

Table 8. Breakdown of energy (Joules) spent by the Microdrive in different power modes for executing traces 1 and 3.

Now let us examine in turn the effectiveness of the different power management mechanisms.

We first study the Microdrive. Table 8 shows the amount of energy spent under the different power modes of the drive. We see that IFS-Batch and LFS share similar behavior, while IFS spends most of its energy in the high-power mode of the drive. The energy that IFS spends in the high-power mode can be potentially attributed to two different sources: (1) individual I/Os (especially writes) may cost more time and energy than their LFS counterparts due to the different layout and locality characteristics of the two file systems; and (2) idle times become more fragmented under IFS as they are more frequently interrupted by I/Os so the drive more frequently remains in high-power mode even during periods when I/Os are absent. Data from the previous section allows us to determine which of these two sources dominates: Table 6 shows that when there is no idle time, the energy difference between IFS and LFS, which can be almost exclusively attributed to the difference in I/O costs, is significantly lower than the active energy difference between IFS and LFS shown in Table 8. It is therefore clear that the second source of extra IFS active energy dominates: when the idle times are fragmented, fewer of the idle time periods cause the drive to enter the low-power mode.

	Trace 2				Trace 4			
	IFS	IFS-DW	IFS-Batch	LFS	IFS	IFS-DW	IFS-Batch	LFS
MD	4661	4997	4587	4633	1939	2065	1714	1345
WL	3742	3931	3011	2980	1251	1221	1270	859
CF	94.2	94.7	76.0	73.6	34.1	33.7	33.5	28.8

Table 9. Energy measurements (Joules) for traces 2 and 4.

ABLE-2 is more effective under IFS-Batch and LFS, as the file system lends a "helping hand" to the underlying power management mechanism as it lengthens the idle periods.

We now study the WaveLAN. In Table 6, we have seen that the WaveLAN consumes about $3\times$ as much energy as the Microdrive does when there is no idle time. In Table 7, on the other hand, the WaveLAN does better. This data suggests that the power management mechanism on the WaveLAN is more successful at coping with certain idle periods than the Microdrive. The Microdrive typically remains in high-power mode for the first 2.5 s of an idle period. It then transitions into a low-power mode and reverts back to high-power mode when the idle period ends. The transitions between modes consume both time and energy. The behavior of the WaveLAN is different. During idle periods, the card transitions to low-power mode almost immediately and repeatedly transitions between idle and active modes at regular intervals to check for communication activities. The average power consumed by the card over a sequence of such transitions is 145 mW, a value that falls in between the low-power and high-power modes of the Microdrive. The WaveLAN therefore consumes less energy than the Microdrive for short idle periods and performs better than the Microdrive for certain workloads.

Finally, the differences in the energy consumed by the different file systems are least dramatic on the SimpleTech, thanks to this card's almost perfect power management.

6.3. Varying Idle Time

So far, we have seen two extreme cases in terms of the amount of idleness in the system. We now vary the rate at which a trace is played. For example, when we use a "speed-up factor" of $2\times$, we cut all the inter-arrival times by half. Figure 3 shows that the behavior of the file systems indeed forms a predictable and smooth continuum as we vary the rate at which trace 3 is played; the performance of IFS-Batch approaches that of LFS as the rate decreases, and approaches that of IFS as the rate increases. The general relationships among the file system variants not only hold for different rates of trace 3, but as Table 9 shows, similar trends hold for traces 2 and 4 as well. When we play traces 2 and 4 at their original speeds, the resulting behaviors lie in between the two extremes that we have examined so far.

From these experiments, we see that the energy consumption advantage of LFS is most pronounced in some but not all situations. First, when the storage device lacks a

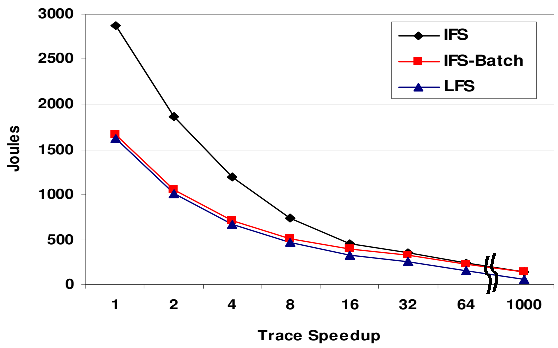

Figure 3. Energy measurements (Joules) of the Microdrive under trace 3 as we vary the trace playing rate.

good power management scheme in dealing with idle time, LFS provides a robust way of introducing additional device burstiness to reduce the reliance on a sophisticated power management scheme. Second, when the I/O load is very high, the increased write locality of LFS translates its superior performance into superior energy consumption behavior. If the I/O load is low and the device lacks a good power management mechanism, IFS-Batch can improve the effectiveness of the power management scheme in a way that is similar to LFS without risking paying the energy cost of segment cleaning which, as we shall see next, can be significant in some extreme conditions. If the I/O load is low and the device has good power management, at least for the devices that we have studied, such as the WaveLAN and the SimpleTech, the differences in energy consumption among the file system variants appear to be small.

6.4. Effect of Scheduling Writes

We also examined the effect of sorting blocks by addresses before scheduling writes on the Microdrive. IFS-Batch-Sched was not significantly better than IFS-Batch for the different workloads. The increase in write locality obtained by sorting the blocks inside the LD-Server does not translate into significant differences in performance and energy consumption. We thus omit the detailed data.

6.5. Impact of Segment Cleaning

We now study the impact of invoking the segment cleaner on overall energy consumption. We initially run the logical disk in its "functional simulator mode" (as described in Section 4) with ten days' worth of I/O operations, which quickly warms up the file system state and fragments free space, without performing physical device I/Os. We then make our measurements using trace 3 and vary the utilization by limiting the total capacity of the storage device. Table 10 gives the Microdrive and SimpleTech energy measurements for an LFS file system that performs garbage collection using a lazy-cleaner. As the results indicate, the

	IFS	IFS Batch	LFS 25%	50%	75%	99%
MD	2865	1659	1616	1639	1652	2097
CF	59.9	57.6	55.4	56.6	57.8	121.1

Table 10. Energy measurements (Joules) for trace 3 played at $\times 1$ on IFS, IFS-Batch, and LFS under different utilizations.

	25%	50%	75%	99%
Lazy Cleaner	1616	1639	1652	2097
Auto Cleaner	3153	3118	3101	3031

Table 11. Energy measurements (Joules) for trace 3 run at different utilizations on the Microdrive.

cleaner overheads are substantial only at very high utilizations. For utilizations of 75% and lower, the effect of the cleaner on energy consumption is tolerable. Interestingly, more detailed statistics reveal that at 99% utilization, LFS generates about $5\times$ more write traffic than it does at 25% utilization. The corresponding rise in energy consumption of the Microdrive is much less. This is mostly because the cleaning I/O is highly clustered in time and causes little disturbance to the idle time distribution, which is one of the major determining factors of overall energy consumption.

The data presented above is for a lazy-cleaner. We now compare the lazy-cleaner with the auto-cleaner, which performs garbage collection when it recognizes that the system is not performing user I/O. The lazy cleaner, which runs only when there are very few free segments, may cause poor response time as it might be required to perform garbage collection during user I/O. However, in its attempt to clean segments eagerly, the auto-cleaner has the detrimental effect of interrupting the idle periods of the storage device. As we have seen before, the power management schemes are more likely to perform poorly when the idle periods are fragmented, and this effect is precisely what we observe in Table 11. It is therefore interesting to note that the auto-cleaner is trading-off energy consumption for better response times, while the lazy-cleaner allows the power management scheme to be more effective. In fact, there are instances where a less utilized disk provides the auto-cleaner with more opportunities to be active and thereby disrupt the power management scheme to a greater extent.

6.6. Comparison with NFS

We compare NFS3 with an LD-Server that is configured to run over WaveLAN in IFS-Batch mode. (The latter alternative is explained in Section 4.) We report results for the MMAB workload, since the four disk traces cannot be played to an NFS client. Table 12 presents results for the four steps of MMAB. Each of the two "Ratio" columns gives the ratios of the values in its two preceding columns. The two ratio columns are almost identical. In other words, the energy consumed is proportional to the time taken to finish the workload. This happens because MMAB contains little idle time, and the WaveLAN never enters the doze mode. The time and energy cost of NFS is mainly due to its synchronous data and metadata writes, while IFS-Batch running on the network-backed logical-disk allows more asynchronous operations. Note that we are not making simplistic conclusions such as "IFS-Batch is better" here, since the two options offer different functionalities. Both options, however, are practical alternatives available to a mobile storage user today, and the data quantifies the energy impact of a buffered block interface.

	Energy (Joules)			Time (seconds)		
	NFS3	IFS-Batch	Ratio	NFS3	IFS-Batch	Ratio
Step1	4505	994	4.53	5400	1183	4.56
Step2	4647	2660	1.75	5196	3085	1.68
Step3	10611	2174	4.88	12721	2799	4.54
Step4	1688	1663	1.02	1959	1975	0.99

Table 12. Energy and time measurements for the MMAB workload. The columns labeled as "IFS-Batch" contain measurements of a logical disk that is backed by the WaveLAN card.

7. Related Work

Many papers have studied the effects of aggressively spinning down disks when the time since last I/O request exceeds some threshold [5, 17]. Policies for dynamically varying the spin-down threshold in response to changing user behavior and priorities have also been considered in the literature [4, 10, 15, 8]. Li et al. [17] also examine the effects of caching and delayed writes on energy consumption. Our study takes existing power management schemes of various devices and quantifies how file systems may influence the idle time distribution seen by a device to influence its power management decisions.

Similar optimizations have been proposed for wireless network cards. For example, energy can be saved by putting such cards to sleep when they are neither transmitting data nor expecting to receive data. To achieve this, several energy-efficient transport-level protocols have been proposed [14, 24, 11]. Active involvement of popular applications like email and web browsing in managing power has also been explored, but developing energy-efficient file-systems has not been considered.

Several file systems have been designed to address the idiosyncrasies of Flash memory [13, 16], many of which are log-structured [21]. Energy characteristics of some flash-based storage systems have been studied in [3, 18], where they are observed to be more energy-efficient than disk-based systems. We have found that due to its easier power management problem, Flash memory appears to be least sensitive to the file system techniques that we have studied, especially under low I/O rates.

Many previous studies have relied solely on trace-driven simulations. Our results are derived from actual energy measurements of real implementations running on real devices. An accurate power simulator for hard-disks has emerged only recently and it appears to be a non-trivial endeavor [25]. In retrospect, we believe that the quantitative results of this study are likely to be more reliable than what we could obtain using simple simulators.

8. Conclusion

We have constructed a logical-disk system that can be configured to run on different storage technologies and to emulate the behavior of different file systems. This system allows us to quantify the energy impact of several file system decisions on three different kinds of hardware technologies. The interplay among the idle time distribution, the file systems' ability to manipulate the idle time seen by the device, and the power management scheme of the device appears to be the most important factor. For high I/O rates and low device utilizations, a log-structured file system translates its performance benefits into energy savings. For less stressful I/O loads, we demonstrate the feasibility of alternative hybrid designs that avoid incurring the overheads associated with storage garbage collection and still reduce the file system's reliance on sophisticated power management schemes. By quantifying and thus prioritizing the many factors that may impact the overall energy consumption, we see this study as a first step toward the construction of an energy-efficient mobile storage system. This study also allows us to better understand the strengths and weaknesses of the different technologies. Instead of pitting the devices against each other, we conjecture that a judicious *combination* of the different technologies may play an important role in the mobile storage system that we build in the aftermath of this study.

References

[1] Adaptive battery life extender. http://www.storage.ibm.com/-hdd/library/able.htm, 1999.

[2] W. de Jonge, M. F. Kaashoek, and W. C. Hsieh. The Logical Disk: A New Approach to Improving File Systems. In *Proc. of ACM Symposium on Operating Systems Principles*, 1993.

[3] F. Douglis, F. Kaashoek, K. Li, R. Cceres, B. Marsh, and J. A. Tauber. Storage alternatives for mobile computers. In *Proc. of Operating Systems Design and Implementation*, 1994.

[4] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing*, pages 121–137, April 1995.

[5] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[6] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J.-A. M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proc. of SIGMETRICS*, pages 252–263, 2000.

[7] G. R. Ganger and Y. N. Patt. Metadata Update Performance in File Systems. In *Proc. of Operating Systems Design and Implementation*, pages 49–60, 1994.

[8] R. A. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *USENIX Winter*, 1995.

[9] E. Growchowski. Emerging Trends in Data Storage on Magnetic Hard Disk Drives. In *Datatech*, pages 11–16. ICG Publishing, September 1998.

[10] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.

[11] T. Imielinski, M. Gupta, and S. Peyyeti. Energy efficient data filtering and communications in mobile wireless computing. In *Proceedings of the Usenix Symposium on Location Dependent Computing*, April 1995.

[12] R. Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, Aug. 2001.

[13] A. Kawaguchi, S. Nishioka, and H. Motoda. A flash-memory based file system. In *USENIX Technical Conference*, pages 155–164, 1995.

[14] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *International Conference on Mobile Computing and Networking*, pages 157–168, 1998.

[15] P. Krishnan, P. M. Long, and J. S. Vitter. Adaptive disk spin-down via optimal rent-to-buy in probabilistic environments. In *Proceedings of the Twelfth International Conference on Machine Learning (ML95)*, pages 322–330, 1995.

[16] M. Levy. Interfacing microsoft's flash file system. In *Memory Products, Intel Corp.*, pages 4–318–4–325, 1993.

[17] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proc. of Winter 1994 USENIX Conference*, pages 279–292, January 1994.

[18] B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. In *Proceedings of Hawaii International Conference on System Science*, 1994.

[19] M. McKusick, W. Joy, S. Leffler, and R. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1984.

[20] J. Ousterhout. Why Aren't Operating Systems Getting Faster As Fast As Hardware? In *Proc. of Summer USENIX*, 1990.

[21] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. In *Proc. of Symposium on Operating Systems Principles*, 1991.

[22] C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In *Proc. of the Winter 1993 USENIX*, pages 405–420, San Diego, CA, Jan. 1993. Usenix Association.

[23] S. Sobti, N. Garg, C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang. PersonalRAID: Mobile Storage for Distributed and Disconnected Computers. In *Proc. of First Conference on File and Storage Technologies*, January 2002.

[24] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8), 1997.

[25] J. Zedlewski, S. Sobti, N. Garg, A. Krishnamurthy, and R. Y. Wang. Modeling Hard-Disk Power Consumption. In *Proc. Second Conference on File and Storage Technologies*, 2003.