# 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 large design space of multiple dimensions. Two important dimensions are the type of storage technologies and the type of file systems. In this paper, we systematically explore this large design space. We have constructed a logical disk system that can be easily configured to run on different storage technologies and to emulate the behavior of different file systems. As we explore the large number of configuration options made possible by this system, we gain many insights about the interaction between the device power management scheme and a log-structured design. This interaction is more complex than the conventional wisdom of equating performance and energy improvements. Instead, the most critical issue in determining the energy behavior is a complex interaction of three factors: the power management mechanism of the storage device, the distribution of the idleness in the workload, and the file system strategies that attempt to exploit and bridge these first two factors. As an example of the energy performance achieved on our implementation, a Microdrive running under a log-structured design consumes only 56% of the energy used by a comparable conventional system.

## 1 Introduction

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 large design space of multiple dimensions. Two important dimensions are the type of storage technologies and the type of file systems. In this paper, we systematically explore this large design space.

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, different power characteristics, and different built-in power-management mechanisms.

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, more often than not, translates into increased energy efficiency. A log-structured design, however, is an umbrella-term that encompasses a number of issues. We systematically isolate the impact of each of these features on the energy consumption of the storage system. For the network devices, we also study the energy impact of two network storage protocols: NFS3 and a network disk.

We have constructed a logical disk system that can be easily configured to run on different storage technologies and to emulate the behavior of different file systems. This flexible infrastructure provides a consistent framework for conveniently evaluating a large design space of mobile storage alternatives. By executing real I/Os, the system allows us to collect accurate energy consumption statistics. The system runs on both Linux laptops and iPaqs.

As we explore the large number of mobile storage configuration options made possible by this system, we are starting to gain many insights. For example, we are able to paint a much more complete picture of the energy consumption behavior of a log-structured design than that is implied by the conventional wisdom of equating performance and energy improvements. Although it is true that an LFS can translate its performance advantage into energy advantage under certain heavy workloads, we

*Department of Computer Science, Princeton University, {zheng, nitin, sobti, chizhang, rywang}@cs.princeton.edu.
†Department of Electrical Engineering, Princeton University, rjoseph@ee.princeton.edu
‡Department of Computer Science, Yale University, arvind@cs.yale.edu.

have learned that in many less stressful workloads, the most critical factor in determining the energy behavior is a complex interaction of three entities: (1) the power management mechanism of the storage device whose task is to detect idle periods so it can instruct the device to enter lower-power modes, (2) the distribution of the 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 a good power management scheme, a log-structured design provides a robust way of introducing additional device burstiness to reduce one's reliance on a sophisticated power management scheme. This aspect of LFS, more than anything else, seems to be responsible for its desirable energy profile. For these less stressful I/O loads, there also exist alternative hybrid designs that can behave just as well as LFS without risking incurring the high energy cost of storage garbage collection. Such conclusions, on the other hand, can be highly technology-sensitive, and different file system strategies are called for on different devices if we were to achieve the best energy profile.

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

# 2 Mobile Storage Alternatives

Although it has the form factor of a compact flash, the IBM Microdrive 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 disadvantages compared to the Microdrive. One of them is its poorer durability: 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, whereas the mean time to failure of a Microdrive is about a million hours. Furthermore, the current technology trend is such that magnetic disk technology is likely to continue to enjoy a more favorable cost per megabyte ratio in the near future [10]. Although the third technology possibility, a wireless LAN card, is not typically considered as a "storage technology", there are environments where the network storage provided by a wireless network is preferred to local storage due to benefits such as increased data reliability and ease of sharing.

## 2.1 Performance and Power Characteristics

Table 1 lists the mobile storage devices that we study in this paper. Table 2 and Table 3 show their performance and energy consumption characteristics, respectively. Although one may be tempted to compare the different technologies against each other throughout this paper, we do not intend to "pick sides" in this paper. 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 a energy-efficient mobile storage solution. When examining the hardware characteristics in Tables 2 and 3, we highlight the following.

**Performance difference of reads and writes.** Some of these technologies (such as the flash cards) have noticeably poorer performance and energy consumption for writes than reads. Asynchronous buffering of writes that allows data to be deleted in main memory plays a more crucial role. Increasing buffering, however, may require the use of additional main memory, which itself consumes energy. This tradeoff is an example where the correlation between performance and energy is more complex than it may appear.

**The relationship between latency and bandwidth.** For all 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 stand points 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 the address of the I/O. This is not true for accessing the Microdrive. And writing a flash card may trigger different number of erasure cycles, resulting in different energy consumption.

**Power management schemes.** The firmware or device driver of a storage device may instruct the device to enter low-power modes during idle periods. The sophistication and effectiveness of such power

| Name | Maker/Model | Capacity (GB) | Purchase Price |
|---|---|---|---|
| Microdrive | IBM Microdrive | 1 | $320 |
| Viking | Viking ATA Flash Type II PC Card | 1 | $627 |
| SimpleTech | Hitachi, Simple Tech ATA Flash Type I PC Card | 1 | $858 |
| WaveLAN | Lucent, Orinoco 11 Mbps Silver Wireless Lan Card | – | $86 |

*Table 1: Devices studied.*

| Name | Read Latency (m*s*) | Write Latency (m*s*) | Read Bandwidth (MB/s) | Write Bandwidth (MB/s) |
|---|---|---|---|---|
| Microdrive | 21.7 | 23.3 | 3.0 | 1.9 |
| Viking | 4.5 | 17.3 | 1.1 | 0.7 |
| SimpleTech | 1.5 | 4.6 | 3.5 | 2.3 |
| WaveLAN | 2.3 | 2.3 | 0.6 | 0.6 |

*Table 2: Performance characteristics.*

management schemes will affect how a file system may exploit them. Next, we describe the differences in the power management schemes of these technologies in greater detail.

### 2.2 Power Management Schemes

**SimpleTech.** The idle mode on these cards consumes very little power and the card goes back to the idle mode as soon as it finishes servicing an I/O request. Since these cards have an ATA interface, the sleep mode is entered only after prolonged idle interval. Hence there is neither need nor availability of sophisticated power management schemes for these cards. One can view these cards as having "perfect" power management.

**WaveLAN.** The wireless network cards provide simple 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 when transmitting or receiving. The card enters low power mode adaptively, and the user can control the duration for which it will stay in that mode before it comes back up to check for incoming messages. A user can also put the card into idle mode and some cards also allow the radio transmission power to be set. Different systems may choose to leverage these features differently.

**Microdrive.** The Microdrive has the most sophisticated built-in power management support; it is called the Adaptive Battery LifeExtender-2 (ABLE-2) [1]. This feature has three idle modes: "performance idle mode", "active idle mode", and "low power idle mode." In performance idle mode, the drive is spinning and can respond to a new command immediately but some of the electronics are powered down. In low power idle, the drive is spinning but heads are unloaded from the disk. It can respond to a new command within 300 m*s*. The active idle mode is not available on the Microdrive. The Microdrive monitors the commands which are sent from the host to detect patterns and adapts to them. The standard ATA standby and sleep modes are also present and are controlled by the host.

## 3 File System Alternatives

### 3.1 Existing Alternatives

To understand the impact of file system design choices on energy consumption, let us begin with two well-known local file systems: the Unix File System (UFS) [22] and the Log-structured File System (LFS) [25]. Under UFS, once disk blocks are allocated for a file, the disk locations of the file data do not move and updates are always performed to the same locations. Each of the small disk writes may incur a costly mechanical delay. Under LFS, file data does not necessarily reside at fixed disk 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 hope to amortize the cost of 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, into which more new data can be written.

The conventional wisdom is that LFS tends to be faster for many workloads; and faster programs, more often than not, tend to consume less energy.

| Name | Operating Voltage (V) | "Idle" Power (m$W$) | Small Read Power (m$W$) | Small Write Power (m$W$) | Big Read Power (m$W$) | Big Write Power (m$W$) |
|---|---|---|---|---|---|---|
| Microdrive | 3.3 | 60.5 | 531.1 | 530.6 | 577.1 | 575.7 |
| Viking | 3.3 | 2.8 | 56.8 | 82.5 | 78.7 | 109.3 |
| SimpleTech | 3.3 | 2.9 | 73.7 | 125.0 | 82.5 | 217.4 |
| WaveLAN | 5.0 | 66.1 | 897.7 | 1086.5 | 916.5 | 1114.9 |

*Table 3: Power characteristics.*

Before we are willing to accept this conventional wisdom and declare LFS the local file system of choice for low-power situations, we need to dissect it more closely.

For providing storage to the PDA over the wireless LAN card, an obvious existing option is NFS. We would like to examine its energy consumption behavior too and compare it against other alternatives.

### 3.2 Energy-Relevant Design Issues

An LFS encompasses a number of features that impact energy consumption. Some of the features are desirable for saving energy while others are not; these features have different degree of impact depending on the underlying storage technology; and there may exist a hybrid file system design that selectively incorporates a subset of these features without embracing the whole LFS doctrine to have the best energy behavior. For devices like the wireless LAN card, the amount of flexibility is even greater. We enumerate some of the following energy-relevant 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 improve both performance and energy consumption.

**Burstiness.** Increased burstiness refers to closer grouping of I/Os in time. Increased 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 and longer idle periods that can be more readily detected by power management mechanisms so the storage devices can be instructed to enter lower-power modes more frequently.

**Layout.** Layouts that increase locality tend to improve both performance and energy consumption for storage devices that have non-uniform access times.

**Background activities.** Background activities such as reorganizing data for improved read locality [21] or compacting free space for improved write locality can improve burst performance. However, these activities themselves consume energy. 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 used by the file system to interact with the device impacts not only the above issues, it also introduces additional questions such as the granularity of communication and whether to transmit operations or data.

### 3.3 Hybrid Alternatives

For storage devices that export a disk interface, UFS and LFS can be considered as two extreme design points in terms of the first four of the above issues: LFS increases write asynchrony, burstiness of write traffic, write locality, and background activity (in the form of cleaning). There may exist other design points between these two extremes; and the exact choice of which design point to employ will depend on the underlying mobile storage technology on which the file system runs. For example, if the storage technology is such that it allows for a "perfect" power management mechanism that can exploit even the most minuscule idle period, then the file system's ability to increase burstiness is no longer crucial.

We now consider several hybrid file system alternatives that incorporate elements of both UFS and LFS. By default, UFS at least performs metadata updates synchronously to the storage device. There are existing techniques that loosen this restriction and increase the asynchrony of the file system [8]. The memory that temporarily buffers these writes serves a role that is analogous to that of a "leaky bucket": writes trickle out to the storage device gradually. To increase the burstiness to allow

for more effective power management, we do not perform the buffered writes gradually—instead, we flush the buffer in a batched fashion. We will refer to this simple variant of UFS as *UFS-Batch*. To increase the write locality, when UFS-Batch flushes its memory buffer, it may carefully schedule the I/Os to further increase locality. We will refer to this variant of UFS as *UFS-Batch-Schedule*. In the extreme case, we introduce background cleaning and data relocation to reach the final design point: LFS.

We do not pretend that any of the above design points by themselves are some sort of great revelation. Instead, our interest lies in understanding the energy consumption behavior of these alternatives for the different mobile storage technologies.

The network device can be used to provide storage in even more flexible ways. 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. Under this approach, we can use any one of the local file system variants discussed above. 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 (in terms of, for example, their ability to support sharing or the lack of it). Nevertheless, we believe it is interesting to understand the energy consumption behavior of these options for several reasons. First, they are simple practical options that a PDA user may use *today*. (Despite its lack of support for sharing, the network disk is perfectly adequate for usage scenarios that do not involve concurrent write sharing.) We would like to compare the energy consumption of these two practical options. Second, for designers of a low-energy network storage protocol, we hope that our experiences with the network disk and NFS may provide insight into the impact of the protocol on energy consumption. 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.

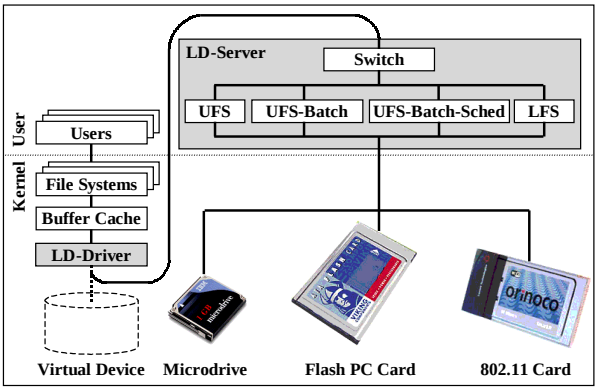


*Figure 1: Implementation of multiple file system alternatives in a logical disk. The shaded parts, the LD-Driver and the LD-Server, are the software components that we have implemented.*

# 4 Implementation

## 4.1 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 (including a UFS and an LFS) which happen to share common mechanisms such as buffer cache management can be a complex undertaking.

To simplify our task, maximally leverage the existing file system work, and maintain a consistent framework for all implementations, we adopt a design based on a *logical disk* [3, 28]. 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 implementer of a logical disk can map these logical addresses to physical addresses in a way that she chooses; and the I/Os to the underlying storage devices can be scheduled in a flexible way. Interestingly enough, this freedom offered by a logical disk design is sufficient for our purpose of evaluating the different design alternatives.

## 4.2 Components

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 four local file system variants discussed in Section 3.3: UFS, UFS-Batch, UFS-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. The user-level implementation of the LD-Server simplifies development at the expense of introducing some overhead. This system allows us to run any real Linux applications and directly measure the energy consumption of the storage system.

The native in-kernel Linux file system performs aggressive asynchronous writes: updates to both metadata and data can be buffered in the buffer cache without being forced to the device.

The UFS module in the LD-Server implements the simplest logical-to-physical address mapping: an identity mapping. Scheduling in the UFS module is also simple: it performs an immediate read or write request upon the receipt of an upcall. To this simple UFS-module, the UFS-Batch module adds a write buffer. It flushes the buffer in a burst only when the buffer is full or dirty data has been present in the buffer for over 30 seconds. Recall that the goal of introducing the write buffer is to increase burstiness to allow the underlying device to benefit from more idle time by triggering its power management mechanism.

The difference between the UFS-Batch and the UFS-Batch-Sched modules is that the latter 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. And sorting writes by their addresses may not be the most effective scheduling algorithm. Despite these limitations, this alternative does provide an opportunity for evaluating the impact on energy consumption as we attempt to increase write locality. (More sophisticated and "correct" scheduling alternatives exist [8, 27], but we would like to gain at least a hint on the energy consumption implications before we attempt these more complex implementations.)

The LFS module implements a *log-structured logical disk* [3]. 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, also *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 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.

### 4.3 Interacting with Devices

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/rawN` 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. (This remote server is typically more powerful than the PDA that originates the I/O and the energy consumption of the storage server is not a concern.) If the LD-Server is configured to run in the UFS mode, 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 the two remaining LD-Server modes, UFS-Batch and UFS-Batch-Sched, is no longer relevant. Although it is possible to flush the UFS-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 UFS-Batch is running on a Microdrive

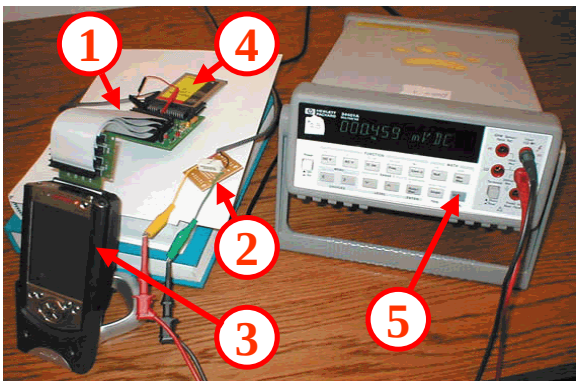


*Figure 2: Measurement apparatus. (1) PC card extender. (2) Shunt resistor. (3) iPaq with a PCMCIA sleeve. (4) Flash PC card. (5) Digital multimeter.*

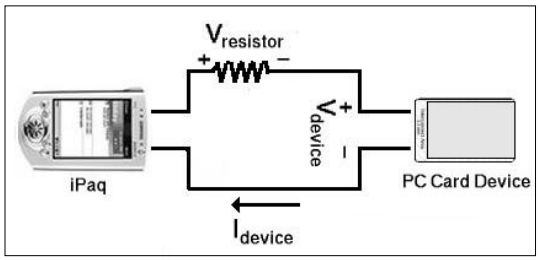


*Figure 3: Schematic diagram of the measurement apparatus.*

or a flash card). We make this choice 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

### 5.1 Apparatus Setup

We need an experimental setup that provides accurate and reproducible measurements without altering or disrupting normal device operation. To achieve these goals, we have fashioned a PC card sleeve to gather energy statistics. The measurement setup is pictured in Figure 2 and its schematic diagram is given in Figure 3. The energy measurement sleeve consists of a PC card extender (1) attached to a shunt resistor (2) in series with the card's power supply. Following the general approach

| Model | iPaq Pocket PC H3635 |
|---|---|
| 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 4: iPaq configuration.*

taken in [7, 14], we measure the voltage across the shunt resistor. The card extender houses a mobile storage device (a flash PC card in the figure) (4). The card extender is inserted into a PCMCIA sleeve of an iPaq (3), whose configuration is given in Table 4. A digital multimeter (5) is then used to measure the resistor's 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 over the course of an experiment. After independently determining the ohmage of the shunt resistor and voltage of the power supply, we deduce the current delivered to the device via Ohm's Law ($V_{resistor} = I_{device}R_{resisor}$). This leads to the average energy consumed by the device using $P_{device} = V_{device}I_{device}$.

One conscious decision that we have 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; and 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 extra devices, the host CPU, and its memory—one may not need the extra devices, may choose to run the storage system under a processor architecture that has comparable speed but different energy characteristics, or may have a memory management system that manages power differently. Indeed, we have reasons to believe that the memory energy consumption on the iPaq is far from optimal. We therefore believe that it is undesirable to burden the analysis with these tangential issues.

### 5.2 Workloads

In choosing workloads for our energy consumption experiments, we made a decision early on that we did not want to confine ourselves to the applications that are available on the iPaq today. This decision was motivated by several considerations. First, the applications, the operating system and hardware support have not matured for the mobile computers; therefore, we do not believe that the limited set of available iPaq applications today constitute in any way a representative set. Second, although we have chosen the iPaq as the computing platform, we do not intend the results of this study to be iPaq-specific—as the capacity and performance of the mobile storage devices continue to improve, one may very well see them being incorporated into higher-end computing devices such as palmtop or even laptop computers. Constraints such as the limited display size on an iPaq should not be allowed to play a major role in narrowing the set of possible applications.

Having refused to confine ourselves to existing iPaq applications, we believe it is in fact not as undesirable as it may first appear to instead use some past workloads collected on workstations, for several reasons. First, while some characteristics such as the I/O rate have seen increases, file system access pattern studies conducted over the years have found remarkable constancy in general access patterns such as sequentiality and data life times [2, 24, 30]. Second, it is our belief that there is very little reason why most of these desktop applications should not be made available to a mobile user at a place and/or time of her choosing. Having said that, we make no claim about the representativeness of the chosen workloads and it is highly likely that there are differences between what an ideal mobile storage system *should* experience and what the chosen traces *do* represent. Nevertheless, we believe that these traces still represent an interesting data point.

We use three workloads. The oldest is a disk trace gathered on a file server at HP Labs in 1992 [26]. 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 persistent storage devices, available on that server then is roughly equivalent to what is available on an iPaq today! We use a five-hour period of user home directory traffic in the middle of a work day to collect measurements, after a ten-day warm-up period. During those five hours, the workload contains 37,416 I/O requests, accessing a total of 150 MB of data. Both the I/O rate and the degree of multiprocessing are low. We will subsequently refer to this workload as "trace 1."

During the five hours of measurement period, we play each of the I/Os contained in the trace synchronously from a user level process running on the iPaq. The logical disk that implements the file system variants performs physical I/Os to the underlying mobile storage devices, allowing real energy consumption statistics to be collected. We also vary the rate at which the trace is played to study the energy effects of different I/O rates. For example, when we use a "speed-up factor" of 2×, we cut all the inter-arrival times by half. During the ten-day warm-up period, we run the logical disk in its "functional simulator mode" (as described in Section4.3), which quickly warms up the file system state (that includes various data structures and the kernel caches) without performing physical device I/Os.

The second workload is an updated version of the first: 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 56,638 I/O requests, accessing a total of 226 MB of data. Experiments involving this trace do not use a warm-up period. Otherwise, the experiments are run similarly as those of trace 1. We will subsequently refer to this workload as "trace 2."

The above workloads are disk traces and they cannot be played to an NFS client, which happens to be one of the options that we would like to evaluate. For this purpose, we use a third benchmark, a synthetic one, called "MMAB". MMAB or "Modified MAB" is an enhanced version of the "Modified Andrew Benchmark" [12, 23]. MMAB has four steps each of which stresses 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 (with the exception of one) has ten subdirectories. The second step is data-write intensive and creates one large file and many small files. The large file created has a size of 256 MB. Each of the small files is 4 KB. 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 in the directory tree; it then computes disk usage of the directory tree by invoking `du`. The fourth and

| | UFS | UFS-Batch | LFS |
|---|---|---|---|
| Microdrive | 136 | 137 | 63.4 |
| WaveLAN | 353 | 366 | 187 |
| SimpleTech | 6.90 | 6.77 | 4.85 |
| Viking | 23.4 | 23.2 | 11.2 |

*Table 5: Energy measurements (Joules) of trace 1 at ×1000 speedup.*

| | UFS | UFS-Batch | LFS |
|---|---|---|---|
| Microdrive | 2865 | 1659 | 1616 |
| WaveLAN | 1531 | 1503 | 1444 |
| SimpleTech | 59.9 | 57.6 | 55.4 |
| Viking | 86.5 | 71.7 | 60.8 |

*Table 6: Energy measurements (Joules) of trace 1 at original speed (×1).*

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

## 6.1 Zero Idle Time

We begin with one of the simplest cases: we play trace 1 at ×1000 speedup, effectively eliminating all idle time. (We will present results from trace 2 in later sections.) The power management mechanisms of the devices, if any, are not triggered. In subsequent subsections, we will present results from varying the amount of idleness in the trace and study the interactions between the power management schemes and the file system strategies. We also keep the storage utilization sufficiently low so the devices effectively clean themselves under LFS—the cleaner is not necessary. The results of these experiments are summarized in Table 5. Not surprisingly, this is a simple case where performance difference translates almost directly into energy consumption difference. Due to the relatively high latency on all these devices, the energy consumption of LFS is the lowest.

What is interesting is that UFS-Batch, the UFS variant that attempts to increase burstiness to exploit the power management mechanisms of the devices, is in fact counter-productive—under high I/O load, the smoother traffic presented by UFS results in more efficient I/O than the bursty traffic of UFS-Batch, due to less queueing in the former case, thus saving slightly more energy for the Microdrive and the WaveLAN.

## 6.2 Ample Idle Time

We now play trace 1 at its original speed, restoring the ample amount of idle time present in the trace, and triggering the power management mechanisms of the devices. The results of these experiments are summarized in Table 6. We also play trace 2 at ×8 slowdown, a speed which gives the same average I/O rate as trace 1. The results are shown in Table 7.[1] In both sets of experiments, we see a profound impact of the interaction between the devices' power management schemes and the file system strategies.

Whereas the bursty traffic produced by UFS-Batch has provided no gain in Figure 5, UFS-Batch now delivers various degree of energy saving. The exact amount of gain depends on the power management schemes of the underlying device. This gain is most dramatic on the Microdrive, nudging the performance of UFS-Batch close to that of LFS, both of which are substantially better than UFS. It appears that ABLE-2, the Microdrive power management mechanism, is not very successful at exploiting idle time without a "helping hand" lent by the file system.

The performance differences among the different file systems are least dramatic on the SimpleTech, thanks to this card's almost perfect power management scheme in terms of effectively exploiting idle time. The power management scheme on the WaveLAN card also appears to be successful under these workloads. The effectiveness of the power management on the Viking card lies in between those on the Microdrive and the SimpleTech.

We have learned a number of insights from Tables 5, 6, and 7. First, we see that the impact of idle time on energy consumption is great. Second, the device power management mechanism and the file system can complement each other—file system cooperation becomes more crucial on a device with less effective power management schemes. Third, at least under low storage utilization, LFS provides an effective means of managing idle time—its behavior is more robust than an alternative like UFS-Batch.

## 6.3 Varying Idle Time

So far, we have seen two extreme cases in terms of the amount of idleness in the system. Figure 4 shows that the behavior of the file system variants indeed forms a predictable and smooth continuum as we

[1] During the last days before the submission deadline, the Viking card failed, and we were unable to complete the trace 2 measurements on it.

| | UFS | UFS-Batch | LFS |
|---|---|---|---|
| Microdrive | 6190 | 4824 | 4198 |
| WaveLAN | 3319 | 3200 | 2736 |
| SimpleTech | 105.9 | 105.9 | 100.5 |

*Table 7: Energy measurements (Joules) of trace 2 at ×8 slowdown.*

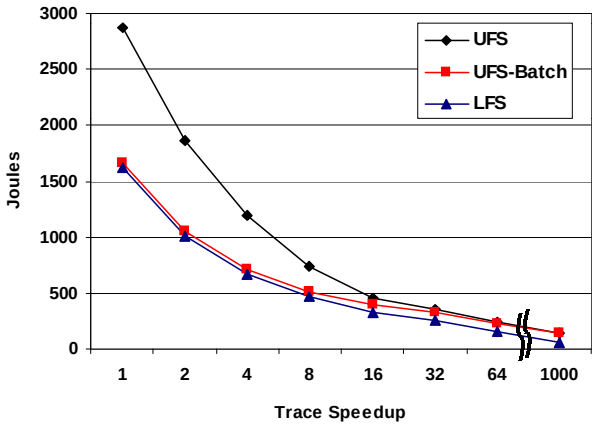


*Figure 4: Energy measurements (Joules) of the Microdrive under trace 1 as we vary the trace playing rate.*

vary the rate at which trace 1 is played on a Microdrive: the performance of UFS-Batch approaches that of LFS as the rate slows, and approaches that of UFS as the rate increases.

The general relationships among the file system variants not only hold for different rates of trace 1, but as Table 8 shows, similar trends hold for trace 2 as well—the I/O rate of this trace lies in between the two extremes that we have examined for trace 1.

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 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, UFS-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

| | UFS | UFS-Batch | LFS |
|---|---|---|---|
| Microdrive | 1939 | 1714 | 1345 |
| WaveLAN | 1372 | 1491 | 985 |
| SimpleTech | 34.1 | 33.5 | 28.8 |

*Table 8: Energy measurements (Joules) for trace 2 run at original speed (×1).*

| | 25% | 50% | 75% | 99% |
|---|---|---|---|---|
| Microdrive | 384 | 383 | 401 | 449 |
| SimpleTech | 11.1 | 11.1 | 11.2 | 15.1 |

*Table 9: Energy measurements (Joules) for the first one-hour of trace 1 played at ×1 for different utilizations.*

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 Impact of Segment Cleaning

We now study the impact of invoking the segment cleaner on overall energy consumption. We make our measurements using trace 1 and vary the utilization from 25% to 99% by limiting the total capacity of the storage device. Table 9 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 cleaner overheads are substantial only at very high utilizations. Even at 99% utilization, the Microdrive incurs less than 20% increase in energy consumption, while SimpleTech incurs a 40% increase. For utilizations of 75% and lower, which is likely to be the normal operating state of storage devices, the effect of the cleaner on energy consumption is barely noticeable, further strengthening the case for LFS being a energy-friendly file system.

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 10. It is therefore interesting to note that the

| | 25% | 50% | 75% | 99% |
|---|---|---|---|---|
| Lazy Cleaner | 384 | 383 | 401 | 449 |
| Auto Cleaner | 692 | 649 | 772 | 675 |

*Table 10: Energy measurements (Joules) for trace 1 run at different utilizations on the Microdrive.*

| | NFS3 | UFS | Ratio |
|---|---|---|---|
| Step1 | 4505 | 994 | 4.53 |
| Step2 | 4647 | 2660 | 1.75 |
| Step3 | 10611 | 2174 | 4.88 |
| Step4 | 1688 | 1663 | 1.02 |

*Table 11: Energy measurements (Joules) for the MMAB workload. Note that although the third column is labeled as "UFS", it is in fact running on top of a logical disk that is backed by the WaveLAN card.*

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.

### 6.5 Comparison with NFS

In this section, we compare the NFS protocol with one of the alternatives that we have considered in previous sections for accessing remote storage. In particular, we compare NFS (version 3) with an LD-Server that is configured to run over WaveLAN in UFS mode. The latter alternative is explained in Section 4.3 in more detail. We report results for the MMAB workload, since traces 1 and 2 are disk traces, and they cannot be played to an NFS client as mentioned in Section 5.2. Tables 11 and 12 present energy and performance results, respectively, for the four steps of MMAB. The fourth column in each table lists the ratio of the two previous columns.

Observe that the fourth columns of the two tables are almost identical. In other words, the energy consumed is proportional to the time taken to finish the workload. This happens because MMAB is a workload with little idle time, and thus, in either case, WaveLAN never enters the doze mode.

Note, however, that the relative performance (or energy usage) of the two alternatives is not same across the four steps. UFS performs significantly better than NFS on steps 1–3, and equally well on step 4. Steps 1 (directory tree creation) and 3 (`touch` and `du`) contain a large number of meta-data operations. NFS uses write-through policy for file-attribute operations and periodically refreshes to maintain consistency of cached attributes, which accounts for its poor performance on steps 1 and 3. Step 4 (`grep` and `wc`) is mainly a data-read step, so both alternatives perform equally well. Step 2 creates a single large file and a large number of small files, which results in substantial data transfer, directory tree traversal, and synchronous meta-data operations. Hence, we see an intermediate performance ratio for step 2.

| | NFS3 | UFS | Ratio |
|---|---|---|---|
| Step1 | 5400 | 1183 | 4.56 |
| Step2 | 5196 | 3085 | 1.68 |
| Step3 | 12721 | 2799 | 4.54 |
| Step4 | 1959 | 1975 | 0.99 |

*Table 12: Time measurements (Seconds) for the MMAB workload. Note that although the third column is labeled as "UFS", it is in fact running on top of a logical disk that is backed by the WaveLAN card.*

| | UFS-Batch | UFS-Batch-Sched |
|---|---|---|
| ×1 | 1659 | 1660 |
| ×2 | 1047 | 1047 |
| ×32 | 319 | 307 |

*Table 13: Energy consumption (Joules) of UFS-Batch and UFS-Batch-Sched on trace 1.*

### 6.6 Effect of Scheduling Writes

In this section, we study the effect of sorting blocks by addresses before scheduling writes on the Microdrive. Table 13 presents the energy consumption of UFS-Batch and UFS-Batch-Sched on trace 1 using different speed-up factors. In this experiment, UFS-Batch-Sched does not seem to be significantly better than UFS-Batch. The main reason for this is that the ext2 file system buffers and clusters blocks before handing them over to the LD-Driver. Thus, sorting the blocks inside the LD-Server does not make a significant difference. In general, though, with more information about the disk geometry, it will be possible to schedule a batch of blocks more intelligently.

### 6.7 Total PDA Energy Consumption

We have consciously focused on measuring the energy consumption of mobile storage devices alone instead of including the energy consumption of the whole iPaq. As mentioned earlier, this decision stems from our intent to study the energy behavior of just the storage system without having the analysis of the results be complicated by various artifacts

| | UFS | UFS-Batch | LFS |
|---|---|---|---|
| Microdrive | 339 | 338 | 163 |
| WaveLAN | 1761 | 1601 | 794 |
| SimpleTech | 87.6 | 90.3 | 69.1 |

*Table 14: Energy measurements (Joules) of trace 1 at ×1000 speedup for the whole iPaq.*

of the host device. We do, however, now present results (in Table 14) on the energy consumption of the whole iPaq to give an idea of how the energy savings from different storage alternatives get expressed in the measurements of overall energy consumption. We note that the difference between the storage alternatives is more pronounced (compared to the difference shown in Table 5), although much of the difference is not necessarily intrinsic to the design choices made in the storage system.

## 7 Related Work

In this paper, we examine three technologies for providing persistent storage to mobile computers—IBM Microdrives, flash PC cards, and wireless LAN cards (used for connecting to storage servers.) We analyze several file-system alternatives for these technologies from the perspective of energy-efficiency. To the best of our knowledge, ours is the first systematic attempt to study this large multi-dimensional design space from the perspective of energy-efficiency. Several previous studies have considered different subsets of this space, and most of them have relied solely on trace-driven simulations. Our results, however, are derived from actual energy measurements of real implementations of the alternatives that we consider.

Many papers have studied the effects of aggressively spinning down disks during idle periods. Spin-downs are usually triggered when the time since last I/O request exceeds some threshold [6, 19]. Policies for dynamically varying the spin-down threshold in response to changing user behavior and priorities have also been considered in the literature [5, 11, 17, 9]. Li et al. [19] also examine the effects of caching and delayed writes on energy consumption. None of these papers, however, evaluates the effect of various file-system alternatives on the length or distribution of idle periods, for example, and the resulting energy-savings.

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 and analyzed [16, 29, 13]. Active participation of popular applications like email, web browsing and newsgroup in power management of these cards has also been explored, but the use of network to provide file-system services has not been considered from the point of view of energy-efficiency yet.

Flash memories have several technological idiosyncrasies—asymmetric read and write access times, the need to erase before writing, and a limited number of write/erase cycles on any memory cell during its lifetime. Several file systems have been designed to address these idiosyncrasies [15, 18], many of which are log-structured [25]. Energy characteristics of some flash-based storage systems have been studied in [4, 20], where they are observed to be more energy-efficient than disk-based systems.

## 8 Conclusion

In this paper, we study how to provide energy-friendly stable storage for PDAs. We have constructed a logical disk system that can be easily configured to run on different storage technologies and to emulate the behavior of different file systems. We have systematically explored this large design space. We see that the impact of idle time on energy consumption of the storage system is great. Substantial energy saving is possible when the device power management mechanism and the file system complement each other.

## References


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

[2] Baker, M., Hartman, J., Kupfer, M., Shirriff, K., and Ousterhout, J. Measurements of a Distributed File System. *Operating System Review 253*, 5 (Oct. 1991), 198–212.

[3] de Jonge, W., Kaashoek, M. F., and Hsieh, W. C. The Logical Disk: A New Approach to Improving File Systems. In *Proc. of the 14th ACM Symposium on Operating Systems Principles* (December 1993), pp. 15–28.

[4] Douglis, F., Kaashoek, F., Li, K., Cceres, R., Marsh, B., and Tauber, J. A. Storage alternatives for mobile computers. In *First Symposium on Operating Systems Design and Implementation* (Monterey, Californie, US, 1994), pp. 25–37.

[5] Douglis, F., Krishnan, P., and Bershad, B. Adaptive disk spin-down policies for mobile com-

puters. In *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing* (April 1995), pp. 121–137.

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

[7] Farkas, K. I., Flinn, J., Back, G., Grunwald, D., and Anderson, J.-A. M. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proc. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)* (2000), pp. 252–263.

[8] Ganger, G. R., and Patt, Y. N. Metadata Update Performance in File Systems. In *Proc. of the First Symposium on Operating Systems Design and Implementation* (November 1994), pp. 49–60.

[9] Golding, R. A., Bosch, P., Staelin, C., Sullivan, T., and Wilkes, J. Idleness is not sloth. In *USENIX Winter* (1995), pp. 201–212.

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

[11] 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.

[12] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (Feb. 1988), 51–81.

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

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

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

[16] Kravets, R., and Krishnan, P. Power management techniques for mobile communication. In *Proc. of the 4th International Conference on Mobile Computing and Networking (MOBICOM)* (October 1998), pp. 157–168.

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

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

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

[20] Marsh, B., Douglis, F., and Krishnan, P. Flash memory file caching for mobile computers. In *Proceedings of the 27th Annual Hawaii International Conference on System Science* (1994).

[21] Matthews, J. N., Roselli, D. S., Costello, A. M., Wang, R. Y., and Anderson, T. E. Improving the Performance of Log-Structured File Systems with Adaptive Methods. In *Proc. of the 16th ACM Symposium on Operating Systems Principles* (October 1997), pp. 238–251.

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

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

[24] Ousterhout, J., Costa, H. D., Harrison, D., Kunze, J., Kupfer, M., and Thompson, J. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proc. of the 10th Symposium on Operating Systems Principles* (Dec. 1985), pp. 15–24.

[25] Rosenblum, M., and Ousterhout, J. The Design and Implementation of a Log-Structured File System. In *Proc. of the 13th Symposium on Operating Systems Principles* (Oct. 1991), pp. 1–15.

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

[27] Seltzer, M., Chen, P., and Ousterhout, J. Disk Scheduling Revisited. In *Proc. of the 1990 Winter USENIX* (Washington, D.C., Jan. 1990), Usenix Association, pp. 313–323.

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

[29] Stemm, M., and Katz, R. H. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications E80-B*, 8 (August 1997), 1125–31.

[30] Vogels, W. File System Usage in Windows NT 4.0. In *Proc. of the 17th ACM Symposium on Operating Systems Principles* (December 1999), pp. 93–109.