A Docked-Aware Storage Architecture for Mobile Computing

Christopher R. LaRosa
Computer Science Department
Hamilton College
Clinton, NY 13323
crl@chrislarosa.net

Mark W. Bailey
Computer Science Department
Hamilton College
Clinton, NY 13323
mbailey@hamilton.edu

ABSTRACT

We explore how the power-abundant docked state of mobile devices can be exploited to reduce power consumption during mobile operation and expand the capabilities of portable devices. We propose a storage hierarchy, which includes a hard disk, a large low-power cache, and a docked-aware file system that lowers the average power cost of file access from the disk while retaining the storage capacity of the disk. We investigate how hoarding files in low-power memory during a power-abundant docked state can drastically reduce the power consumption of the hard disk during mobile operation. Using traced-based simulation, we determine the effects on battery run-time of adding the storage architecture to a modern palmtop device—effectively adding mass storage capability to the device. We experiment in the palmtop environment because palmtops are frequently and easily docked, and epitomize the battery and power constraints that face compact, portable devices. Our trace-based simulation shows that up to 86% of power used by the storage subsystem can be recovered using simple hoarding algorithms that cache data during the docked state. This power savings translates into simulated run-times 86%–97% as long as the run-time of a diskless palmtop.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability.
D.4.2 [**Operating Systems**]: Storage Management – *Storage hierarchies*.

General Terms

Design, performance. reliability.

Keywords

Hoarding, file system, energy, power, battery life, caching, handheld, palmtop, docked.

1. Introduction

Power conservation is a critical issue in the design of mobile systems. While mobile components have decreased in size considerably and continue to increase in capability, the size-to-performance ratio of batteries has not shown similar improvement. New technologies such as wireless network interface cards and integrated CMOS digital cameras are placing additional demands on the already limited power provided by batteries. If new compact devices are going to provide greater functionality, system designers need to find new ways of conserving power.

An extensive body of research focuses on ways to minimize the power demand placed on batteries during mobile operation. Processor cycling, computational offloading [13, 18], disk spin-down [4, 5, 11, 12], and data caching [14] are some of the approaches that researchers have used to decease power consumption and extend battery life during mobile operation. These software and hardware-based-power-saving methods are often disabled during the power-abundant state when the mobile computer is plugged-in, or "docked." When docked, components run at full-power and full capability without affecting battery life. We propose exploiting the power-abundant docked state in order to minimize the power demands of hardware during mobile use. Concentrating power-costly activities, such as disk access and network access, in higher-intensity shorter intervals has been shown to decrease power consumption in mobile computers by increasing the time peripherals remain in low power states [16, 20]. Our goal is to maximize the durations of these low power states by advancing or delaying power-intensive activity during mobile operation to the plugged-in state. We use the "free power" available during the docked state to minimize power consumption during mobile use. We examine how hardware and software can be redesigned to utilize this means of energy savings.

Palmtop design presents a fertile environment to investigate the benefits of how we can exploit the plugged-in power-abundant state to reduce power use. Palmtops have smaller batteries than laptops and are frequently connected to powered docks, often for short periods. Palmtops are likely to be docked in a user's office and home. If the device contains an integrated cellular phone, it might also be docked in a user's car. The weight and size of laptops prohibit users from having the devices with them continually throughout the day. In contrast, palmtop users are more likely to take their palmtop computer with them in their daily travels.

Current palmtop computers boast processing power that rivals that of laptop computers manufactured only a few years ago. Despite this increase in processing power, palmtops continue to be used primarily as personal information organizers. We believe that improving the storage capabilities of these devices, which are typically limited to less than 100 MB of storage, would give the devices greater functionality and expand their utility. We use the power-abundant docked state, a flash memory cache, and a docked-aware file system to show how disk-based mass storage, a power costly feature absent from palmtop computers, can be provided at minimal power cost.

There are many compelling reasons to add mass-storage to palmtop computers. As palmtops are integrated with digital cameras and other multimedia devices, storage demands are eclipsing the capabilities that solid-state memory can provide at low cost. Mass storage is becoming an increasingly important feature. The rising popularity of digital audio and video means many users want to have their music or videos available as they travel. Users needing constant data availability would benefit from the consistency of having a single palmtop computer if the device were extensible with input, output, and display peripherals via a dock. With a dockable palmtop that provides mass storage capability, a user could take a single handheld computer everywhere. While at work, at home, or at a public terminal the user could dock the device into a full-sized display, keyboard, mouse, and a high-speed network. When away from docking stations, the user would benefit from having access to all of his or her documents and having use of all applications and data at all times. For most users the usefulness of constant data availability surpasses the need to have robust input and display capabilities at all times. This is demonstrated by the popularity of the Palm Handheld/Palm Desktop solution, in which users do the bulk of their data entry and editing on the desktop and regularly access their data in mobile settings. It is also demonstrated by the popularity of dockable laptop computers, which allow users the robust input and display capabilities of desktop computers at their offices and access to data and applications on the go via smaller screens and more awkward input peripherals.

Our goal is to use the plugged-in state with a memory and storage hierarchy and new docked-aware file system to exploit the power available during the docked state by prefetching files off power costly disks, into less power-costly non-volatile memory during the docked state. Flash memory storage has been shown to use less power than disk based storage [3] and caching data into flash memory has been shown to decrease power consumption in mobile systems [14]. Hybrid non-volatile memory/disk-based storage systems have been demonstrated to improve both read and write performance [22, 10]. Prefetching data that is likely to be used has also been demonstrated to improve performance [2]. We adopt the term *hoarding* for our pre-fetching technique from previous research that focuses on providing file availability to mobile users during network disconnect [8, 19, 21].

In Section 2, we present a storage hardware hierarchy for a full-function palmtop and our new file system paradigm, which is conscious of the docked-state. In Section 3, we describe simulation and traces that we believe are similar to the usage patterns of a hypothesized mass storage equipped palmtop. In Section 4, we present models to estimate the power impact the storage architecture has on an existing palmtop device. In Section 5, we present our simulation results. In Section 6, we present our conclusions. In Section 7, we discuss related work and propose future investigation in Section 8.

2. A HIERARCHICAL HOARDING STORAGE SYSTEM PARADIGM

In this section we describe the changes to current hardware and file systems on palmtop computers necessary to exploit the power-abundant docked state and provide low-power-cost mass storage.

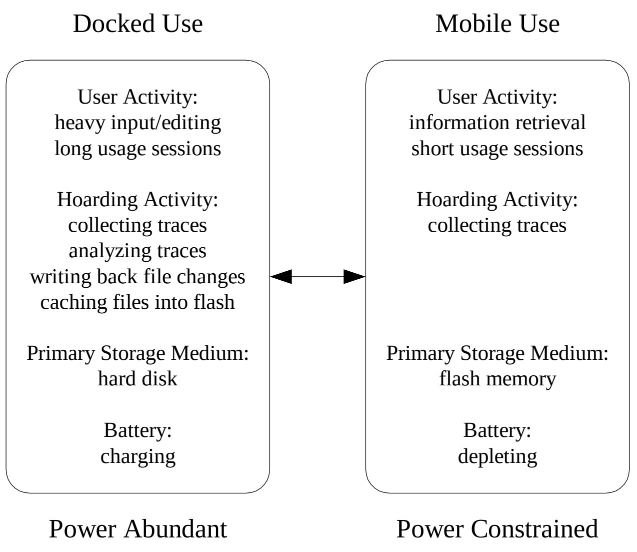

Figure 1 – Docked-Aware Storage Architecture

2.1 Motivation and Guidelines

Previous research shows that it is possible to accurately predict which files users will access during a work session [8, 19, 21]. In this work, file usage predictions are used to copy server files onto laptop hard disks to provide the illusion of seamless network connectivity during network disconnect. Computational off-loading, where processing is transferred from a power-constrained device to a power-abundant server, has been shown to decrease power consumption [13, 18].

Considering these two findings together, we propose a hoarding file system that fills a power-efficient flash memory cache with files from a power-costly hard disk during the handheld's energy-abundant docked state. The system monitors file access patterns and predicts future access to decide which files should be placed in the flash memory cache. When the system is battery powered, files are accessed from the cache. In the event of a cache miss a file is read from the disk. When the device is reconnected to the dock, revised and new files are written back to the disk [Figure 1].

This hoard-use-reintegrate scheme is similar to that used in Coda, a network file system that attempts to provide seamless access to network file systems during network disconnect [19]. In our system, a cache miss has a much less disruptive consequence for the user than Coda — spinning up the disk as opposed to waiting for network reconnect. The benefit of using a non-volatile memory cache to reduce disk use and save energy in mobile computers was demonstrated in [14]. We expand upon this system by concentrating caching activity to the docked state.

We have identified several design guidelines for hardware and software for a docked-aware file system and architecture:

- The system should not alter the physical construction of palmtop computers such that they are no longer highly portable.
- The system should provide mass storage capabilities while minimizing power consumption. The system should provide disk-based capacity at near-flash-based power cost.
- The system should not adversely affect the user experience in either the docked or mobile modes.

2.2 Physical Implications on Architecture

We choose currently available hardware components to show that our proposed single-device paradigm is feasible, both fiscally and physically. A full-function handheld should be capable of running modern productivity and communication software and provide mass storage.

The specifications of current mobile components show that the addition of a hard disk and flash memory cache would not drastically alter the shape or size of handheld computers like the Hewlett Packard iPaq or Sharp Zaurus. Toshiba manufactures a 2.12-inch wide, 3.09-inch tall, and 0.2-inch thick 10GB hard disk, the HDD1262, which would meet most users' storage demands and meet the compact design requirements of handheld computers. The addition of the Toshiba disk would increase the volume of the iPAQ h5550 by 11% and increase the volume of the Zaurus SL 5600 by 8%. Handheld computer designers might find that many users would trade features such as biometric fingerprint readers and GPS positioners, currently consuming space on handheld computers, for the ability to carry all of their data. Already handheld computers ship with flash memory as the primary storage medium, so increasing the capacity to accommodate a hoarding cache would not alter the memory architecture. In addition, the HDD1262 operates at the same voltage as the iPaq and Zaurus, and can be bundled with a PCMCIA interface, so current handheld power architectures and peripheral busses would not need modification to accommodate the disk.

2.3 An Energy Conscious File System

An energy conscious file system should cache all data that will be needed during mobile operation into a lower-power memory while the unit is connected to AC power. The file system should predict which files will be accessed based on past file access patterns. Those files with the highest probability of access should be placed in the cache.

We cache data at the file level rather than at the block level because tracing block-level access will generate overhead on the processor, memory, and storage subsystems during mobile use. Information that we need to implement a simple hoarding system can be collected during the power-abundant docked state by comparing the access, creations, and modification times of files before and after each mobile session using the low-overhead approach described in [9]. This approach provides the system with a list of files that were read, changed, or created on the disk during mobile operation without requiring any kernel modifications. In our experiments, we use a kernel-level tracing tool to track file access. File system level tracing provides a more complete account of file system activity, including the frequency of file access.

Previous research has shown that about 70% of file accesses are whole-file operations [15]. Of the remaining 30%, some blocks of each file are accessed and others are not. To accurately predict block access, power costly block-level traces would have to be performed during mobile execution. Megabytes of block level trace data would need to be collected, incurring power-costly processor overhead and filling lower power cache memory with trace data. During our experiments, we explicitly traced file access using a modified kernel so that we could determine experimentally when files were accessed. This information allowed us to determine the number of disk spin-ups that would occur using our hoarding system so that the power cost of the system could be measured.

In a docked-aware file system, all file writes during mobile operation should go to the low-power cache. Spinning up a disk for a file write is nearly entirely avoidable with a sufficiently large flash memory cache. Non-volatile memory has already been shown to reduce writes to disk on servers and network file systems [1]. Modifications to existing files should not generally tax the cache's capacity. Long-term file access studies have shown that 80% of new files are usually completely overwritten or deleted within 200 seconds of creation. This is the result of temporary file creation and deletion by applications and the operating system [15]. Since these temporary files do not accumulate over time, they are almost immediately deleted after creation; they do not increase the demand placed on the cache's capacity during mobile operation. Previous disk utilization studies show that 80% of file size increases are less than 32KB and 90-95% are less that 64KB. New file creation is primarily responsible for increases in disk utilization. Also, applications often delete a file and create a new file with the same name when a user is editing a document [9]. Since the file that the user is editing is already in the cache, either it was inserted during the hoard or inserted from the disk at the time of its first access, the space freed by the deletion of the old file makes room for the new version to be written. Given that during mobile operation users will primarily use pen and compact keyboard input devices, we assume only in the case of downloading from a wireless network or capturing multimedia data from an integrated device, will heavy write traffic necessitate a disk spin up and affect power consumption.

We do not address the amount of storage needed to handle write traffic, though for the previously listed reasons we do not anticipate it to be large. We also do not account for the power cost of write traffic in our experimentation, which means our results do not show the energy savings achieved though write caching, which we assume to have nearly full success. The caching file system should also cache metadata, such as file lists and attributes for commonly browsed directories. Although we do not address metadata caching in our research, we expect it will consume a relatively small portion of our cache space.

A hoarding storage system should weigh both the frequency of a file's access and its last access when deciding whether to cache the file. In data caches, the historical frequency of data access has been demonstrated to play a role that rivals that of temporal locality [17]. A flash memory cache size that exceeds the size of the user's file working set should be used so that the system can approach a zero power cost implementation.

3. MEASURING PERFORMANCE

To evaluate the effectiveness of our new storage hierarchy and file system design, we create a model that measures the additional power consumption introduced by the new storage architecture. A power model of the storage architecture accounts for access to two sets of files: *f*, files hoarded in flash memory, and *d*, files stored only on the disk at the onset of the mobile session.

3.1 Defining Power Cost

3.1.1 Access to hoarded files

The cost of accessing a hoarded file *i* is the product of the power needed to access the file from flash memory (P^F_i), and the number of accesses to that file during a session (A_i). The total cost of access to all hoarded files, *f*, is:

$$C_h = \sum_{i}^{f} P_i^F \bullet A_i$$

3.1.2 Access to non-hoarded files

We model two scenarios for accessing files off the disk. In the first case, the handheld device is rendered useless to the user while the disk spins up, since either an executing application or the user has requested necessary data and will wait until the data is available. We call this case *blocking*. We also model the case where user and application progress continues during spin up. This type of file access occurs when a daemon or background task requests a file. We call this case *non-blocking*.

When modeling the power cost of accessing a file *i* from the disk, it is necessary to account for the cost incurred by reading the files from the disk, (P^D). It is also necessary to account for the power consumption associated with spinning up the disk if it is not already spinning. The power associated with each spin up includes both the power cost of spinning up the disk (P^F), and, in the case of blocking access, the overhead of powering the rest of the handheld device (P^o). Handheld overhead is the power necessary to maintain the display, operate the processor, and refresh the memory while the handheld is idle waiting for the hard disk to spin up. We assume that when using undocked handhelds, users have a defined task to complete and that waiting for file access in the blocking model prolongs the time needed to complete the task at (and in) hand. The number of spin-ups necessary during a session (*S*) is determined experimentally. *S* is related to the number of accesses to files that are not hoarded and the spin down policy.

We also account for the power needed to spin the disk before the spin down threshold is reached (P^I). The time the disk spins idle (*I*) is determined experimentally as well.

On each initial read of a file from the disk the file is cached since user or application behavior has determined the file to be needed. The file least likely to be used in the cache should be removed to write the newly accessed file if there is not sufficient space to bring the new file in on the cache miss. We consider the file with the lowest historical frequency of access that has not been brought into the cache during the current mobile session the least likely to be used. Subsequent accesses to that file are calculated using the cost equation for hoarded files.

The total cost of file access to non-hoarded files is:

$$C_{nh} = \sum_{i}^{d} P_i^D + S(P^S + P^O) + I \bullet P^I$$

In the non-blocking model system overhead during spin up, P^o, is 0.

3.2 Gauging Battery Impact

To translate the power cost models into real-world usage numbers, we must determine the change in battery life resulting from adding the docked-aware hoarding storage architecture to the palmtop device. We compare the palmtop's runtime with the docked-aware hoarding storage architecture against a theoretical palmtop that has mass storage capacity available at flash memory power cost. We also compare the runtime of the new storage architecture against a palmtop fitted with a hard disk running continuously, to see what percentage of the power consumed by the hard disk our hoarding system is able to recoup.

4. METHODS

To test our hypothesis that adding docked-aware hoarding mass storage to a handheld computer would provide mass storage with minimal impact on battery runtime, we simulate operation of the hoarding software and hardware architecture using a synthetic workload that is intended to represent typical user activity given a mass-storage-equipped handheld with docking capabilities.

4.1 Experimentation Platform

We choose the Linux platform as the basis for our investigation. We favor Linux over the most common handheld operating systems, Windows CE and Palm OS, because Linux provides greater ability to customize file system functions and thereby enables easier prototyping of our system.

Presently two major lines of handheld computer run Linux: the Sharp Zaurus series, which comes with an embedded version of Linux called Embedix installed, and the Hewlett Packard iPaq series, which comes with Microsoft Windows CE installed but is capable of running Linux. We use the Debian distribution of Linux for our experiments because it can be complied to run both on the Zaurus and on a desktop machine where our simulation takes place.

For our experimentation hardware we use a Hewlett Packard Vectra VL with a 166 Megahertz Pentium processor, 52 MB of memory, and a 10 GB 5400 RPM IDE Hard Drive. Current iPaq and Zaurus palmtops have 200-400 megahertz processors and 32-128 MB of SDRAM.

4.2 Tracing file usage

The most significant unknown in our storage paradigm is how having a full-featured handheld computer with mass storage will change users' work habits, affecting file usage patterns from those found on laptops and desktops. This makes the task of defining a representative work session and its associated file trace difficult. Even in traditional laptop and desktop settings, gathering file access traces that are representative of users' real world working habits has been an obstacle in previous hoarding research. Furthermore, much of previous file-hoarding and flash memory storage research has been undertaken on OS/2, DOS, and pre-BSD based Macintosh systems [21, 14], environments that are different from our Linux simulation. More recent file trace data that focuses largely on reducing network traffic and server disk activity in multi-user systems does not account for the fact that applications and their associated data normally reside on local machines, and thus is not ideal for simulating file system activity in a palmtop environment [6].

During our experimentation we gather information about all file system activity using a kernel-level monitoring package, the Linux Trace Toolkit (LTT), that consists of a kernel patch, a trace module, a daemon, and a standalone log file analysis application. The kernel patch adds mechanisms throughout kernel functions that communicate log events to the trace module. The trace module collects log information into a buffer and the daemon periodically empties the buffer into a log file. The trace module's buffer serves the important purpose of minimizing overhead on the system by reducing the frequency of disk write traffic. The LTT has previously been shown to impose a modest 1-2% processor overhead while logging activity [24].

The breadth of data collected by the LTT is beyond the needs of our hoarding algorithm. Approaching 6 MB of file system log data per minute, much of the data collected is records of the file system extending the timeouts of already open files and the module logging its own activity. The analyzer is slow, taking hours to analyze log files, far more time than we would expect the handheld device to be in the dock. We use a Perl program that makes use of Linux's locate command to gather information about the files that were read based on the log provided by the LTT.

Our system, although cumbersome and processor intensive, provides us with a list of files opened during a given trace session, the number of accesses to each file during a trace session, and a basis to test the effectiveness of a simple frequency-based hoarding algorithm. During experimentation the need to know explicitly when all file accesses occur outweighs concerns over data collection size and processor overhead as our energy consumption models need to have the exact times of all file accesses to determine the number of spin ups. During real-world usage, or during prototyping, the timestamp comparison method described in [9] or an LTT-type package that traces only file reads would be more appropriate.

4.3 Defining the trace environment

We collect five user traces in total. We record file access during two 15-minute periods that we believe accurately represent a session when a user would be using their handheld in docked mode, with a large display and keyboard, to perform significant editing on word processing documents, spreadsheets, and email messages. During this time, we switch between applications, from web-based email to word processing and spreadsheet programs. We also record file access during three two-minute trace sessions we believe accurately represent a session in which the user, operating the handheld in a mobile setting, retrieves information from a few documents, checks their email, and briefly edits documents [Table 1]. During our trace, OpenOffice, a Microsoft Office-like productivity bundled is used for word processing and spreadsheets; Mozilla is used for email and web browsing.

Table 1. Trace Profiles

Trace name	Total files read	Total data size	Average file size	Average access interval
15 minute a	504	111.6 MB	226.7 KB	1.8
15 minute b	502	114.2 MB	232.9 KB	1.8
2 minute a	182	87.7 MB	493.4 KB	.7
2 minute b	43	11.1 MB	264.3 KB	2.8
2 minute c	45	6.9 MB	157.0 KB	2.7

4.4 Measuring Hoard Performance

We conduct a series of frequency-based hoard simulations to measure performance. Using our Hoard List Generator to analyze the file trace log we establish: the average size of file access during the session (*ave*), the total number of files accessed during the session, and a hoard candidate list containing the paths of the files accessed during each session in descending order of frequency.

Although we do not explicitly account for temporal locality in our hoard candidate list, one of our design specifications from Section 2.3, we believe that because our trace sessions are relatively short and consecutive, temporal locality is accounted for. The fact that a file was accessed during either the previous or penultimate usage session is sufficient to account for temporal locality in a real system. The hoarding algorithm would limit input data to recent sessions and only examine traces for *n* previous sessions when building the hoard candidate list to establish the frequency of access to recently used files. Previous data cache research has shown that frequency based caching, when coupled an LRU policy, yields performance benefits in data caches [17].

To account for the widely varying average file sizes observed during the traces, especially those used during the two-minute traces, which varied from 156.0 KB to 493.4 KB, we compute an overall average file size from all the traces, *t_ave*. For a flash memory cache of size *n* bytes, we put the first *n*/*t_ave* files of the candidate list into the simulated hoard cache. We then manually simulate what would happen when another trace session is run on the hoard cache, observing which files are accessible from the simulated hoard cache, how many seconds the disk spins idle, and how many spin-ups would be needed during the session. We put the miss count into our cost difference equation from Section 3, along with sample power specifications for a modern mobile hard disk, the Toshiba MK 1003GAL and a Lexar flash memory card. We use the energy specification of the Sharp Zaurus SL-5600 as parameters in our battery runtime models.

The Toshiba disk has a relatively small data cache, 512 KB, in comparison to other hard disks and a very small data cache in comparison to our flash memory cache. Given the small size of the on-disk cache, we doubt that caching at the disk level would result in sufficient hits to decrease the power consumption resulting from disk spin ups.

5. RESULTS

We ran our simulation on a cache size of 64 MB and 96 MB. The former cache size was sufficiently large to hold the contents of *2 minute b* and *2 minute c*; the latter size was sufficient to hold the contents of *2 minute a*. We chose these sizes because they could be provided inexpensively and are sufficient to hold the contents of the mobile working sets.

In our first test one 15-minute file trace, *15 minute a*, was used as input to the Hoard List Generator to see how the cache would perform during mobile operation with a limited amount of historical data. Our results showed poor performance of the cache [Table 2].

Table 2. Cache rates with 15 minutes history.

Trace	250 file cache • 64 MB Cache				400 file cache • 96 MB Cache			
	Hits	Misses	Hit rate	Interval	Hits	Misses	Hit rate	Interval
2 min. a	61	121	.34	.99	110	72	.60	1.66
2 min. b	22	21	.51	5.74	28	15	.65	8.00
2 min. c	21	24	.47	5.00	26	19	.57	6.31

We recorded hit rates as high as 65% but noted the average miss interval was near or below the spin up time. Although file system activity generally comes in bursts [16] the cache performance was sufficiently low that we saw few prospects for significant energy savings and re-ran the experiment providing the Hoard List Generator with more historical data. Later tests on the same data show that these results were in fact too low to yield energy savings using our storage architecture.

We increased the amount of trace data available to the hoard list generator to include the two longer traces, *15 minute a* and *15 minute b*, and the first mobile trace, *2 minute a*. Again we did not reach the level of success we expected, given the consistency of the user behavior during our different traces [Table 3].

Table 3. Cache rates with 32 minutes history.

Trace	250 file cache • 64 MB Cache				400 file cache • 96 MB Cache			
	Hits	Misses	Hit rate	Interval	Hits	Misses	Hit rate	Interval
2 min. b	29	14	.67	8.57	34	9	.79	13.30
2 min. c	30	15	.67	8.00	36	9	.80	13.30

Examining where the cache misses occurred we found two situations where the frequency-based hoard generator resulted in misses. The primary cause of misses while simulating *2 minute c* was OpenOffice accessing a collection of templates and fonts that had not previously been accessed. This failure might be resolved as more trace data becomes available, especially if the user frequently reformats text. Using a hoarding algorithm that traces application execution patterns, like SEER [8], could also address this problem. In the *2 minute b* simulation, misses occurred when

Mozilla attempted to make use of its own file cache on the disk. It is reasonable to assume that for any sufficiently fast network, which is already connected if users are accessing web content, the cost of transferring four files across the network will be less than spinning up the hard disk anywhere from one to four times depending on the density of the cache misses. Mozilla's memory cache could also be increased to avoid both network and disk access costs. Since the cache is a user configurable setting, and does not require rewriting any software, we reran the simulation as if Mozilla's file cache was disabled using *15 minute a*, *15 minute b*, and *2 minute a* as inputs to the Hoard List Generator and obtained the following, more promising, results [Table 4].

Table 4. Cache rates under application adjusted behavior

Trace	250 file cache • 64 MB Cache				400 file cache • 96 MB Cache			
	Hits	Misses	Hit rate	Interval	Hits	Misses	Hit rate	Interval
2 min. b	29	8	.78	15	34	3	.92	40.00
2 min. c	30	15	.67	8.00	36	9	.80	13.30

With the 96 MB cache 80% and 91% of file accesses resulted in file hits during our simulations, levels promising enough that we checked the impact of the system on battery life [Table 5]. We used the hardware specifications described in Section 4.1 to calculate the percentage of battery life for blocking access and non-blocking access as a percentage of the life of an unmodified Zaurus, and as a percentage of runtime for a Zaurus with a non-hoarding hard disk.

Our findings showed that in cases where the hoarding success rate was less than 80%, power was lost due to excessive spin ups in comparison to the baseline measurement of running the disk constantly, when access was blocking. Rates below 80% occurred in all of the 64 MB cache simulations. This means that even accounting for busty access, we would not have been able to save battery life in our earlier traces shown in Table 2. Where access was not blocking, battery runtimes were 89-93% of original runtimes and showed a 5-9% improvement over non-hoarded disk storage, which runs 85% as long as a non-modified Zaurus. For a 96 MB cache, where all hit rates were 80% or better, we saw runtimes of 85-91% of non-modified Zaurus runtimes with our blocking model and 92-97% of original run times with our non-blocking model. Real world usage would fall somewhere between the two values. Compared with non-hoarded disk based storage, our hoarding algorithm with a 96 MB cache showed up to 13 percent battery lifetime improvement and recovered 86 percent of the power used by the storage subsystem and even during its worst performance did not decreased battery life.

We did not account for the power that would be consumed by a wireless network interface card, which has been shown to reduce battery life in palmtops as much as half [20]. The user activity during one of our two-minute trace simulations would necessitate such a peripheral and, as such, would make the percent runtime of the unmodified palmtop somewhat optimistic. At the same time, we did not account for the 100% success rate achieved though the write cache in our experimentation or findings. For this reason we believe our estimated power recovery over the non-modified disk to be somewhat pessimistic.

6. CONCLUSIONS

With our simple frequency-based hoarding algorithm and a 96 MB cache, we were able to simulate runtimes ranging from 85% - 97% of a non-modified handheld. Given the utility of added storage we believe that adding a docked-aware storage architecture with a sufficiently large cache is a promising solution to providing mass storage to hand held devices.

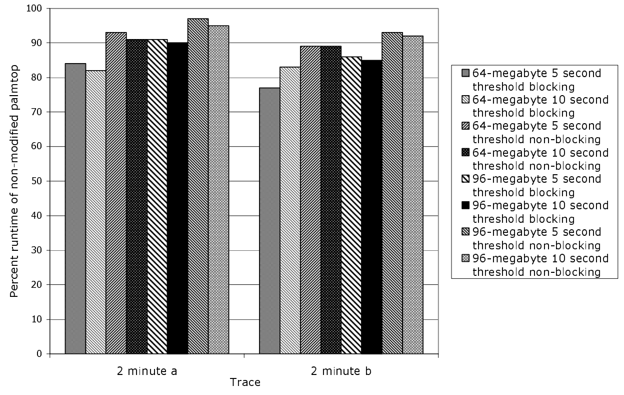

Figure 2 – Docked-Aware File System Performance

While our first experiment, which included only one session, *15 minute a*, as input into the hoard list generator, failed to meet our hopes of caching success, it did indicate that cache size plays a role when dealing with limited historical data. The 64 MB cache the *2 minute a* trace, which has a total file size of 87.7 MB, performed 66-72% as well as the smaller *2 minute b* and *2 minute c* traces, 11.1 MB and 6.9 MB respectively. For the 96 MB cache the larger *2 minute a* test ran at 92 percent of the efficiency of *2 minute b*, and 105% of the efficiency of *2 minute c,* suggesting that when the cache is larger than the working set of the session, increasing the cache size may yield small reward.

The success of our docked-aware simulated file system indicates that designing operating systems to be aware of the power-abundant docked state could save significant energy as more compact devices emerge, which are inherently easier to transport and easier to dock. Our system demonstrates that functionality can be added to compact portable devices with limited demand on battery power by using the free power available during the docked-state.

We also observe that application behavior, such as that of Netscape and OpenOffice, can be very detrimental to the success of hoarding algorithms and may need to be modified, along with the operating system, for the free-power available during the docked state to be fully utilized.

7. RELATED WORK

Based on our observations caching data does not eliminate the busty nature of requests to non-cached data. During our final test, there was little difference between the number of spin ups required when using the five second or ten second threshold. There were an average of 4.13 files requested during each spin up using the 5-second threshold and 5.16 files requested using the 10-second threshold. These thresholds are very small but suggest that previous investigations that use access patterns to delay spin-down during extended bursts of access might benefit our system [11].

Table 5. Battery Runtimes Under Docked-Aware File System

Trace	250 file cache • 64MB Cache, 5/10 second spin down threshold						
	Miss	Idle spin time	# Spin ups	Runtime original blocking	Runtime original non-blocking	Runtime continues disk blocking	Runtime continuous disk – non-blocking
2 minute b	8	22/42	4/4	.84/.82	.93/.91	.99/.96	1.10/1.07
2 minute c	15	43/61	6/3	.77/.83	.89/.89	.91/.97	1.05/1.05

Trace	400 file cache • 96 MB Cache, 5/10 second spin down threshold						
	Miss	Idle spin time	# Spin ups	Runtime original blocking	Runtime original non-blocking	Runtime continues disk blocking	Runtime continuous disk – non-blocking
2 minute b	3	10/20	2/2	.91/.90	.97/.95	1.08/1.06	1.13/1.12
2 minute c	9	27/42	3/3	.86/.85	.93/.92	1.02/1.10	1.10/1.08

We used a simple frequency based hoarding system to demonstrate that through software and hardware design the plugged-in state can be exploited. Using more advanced hoarding algorithms would likely improve performance of our system. We observed that certain types of application behavior can be very detrimental to the effectiveness of the hoard, such as the cache in Netscape and OpenOffice. Instead of hoarding the most commonly used files, hoarding the files associated with the most commonly used applications might allow greater intervals between disk spin ups during mobile use. Application based tracing and hoarding was proposed in [21]. Other hoarding systems that allow users to explicitly specify documents that they will use during network disconnect, or in our case during the undocked state, could also improve reliability and decrease spinups [8, 19].

Another way to circumvent the problem posed by application behavior is to allow applications to adjust their requests for data based on the abundance of energy. This energy aware utilization of data was proposed in [7]. Adding parameters that specify the necessity and urgency of read requests to file system calls could result in fewer spin ups and more bursty disk access [23].

8. FUTURE WORK

We plan to continue to study the behavior of our docked-aware storage hierarchy though simulation and collect further trace data. Based on these findings we plan to build a prototype to measure actual energy savings, which will no doubt be affected by the many layers resting between the file system read calls and the disk media. We also plan to investigate how applications might take advantage of the plugged state by delaying or advancing computation, as with databases or spreadsheets, and concentrating network activity into the docked state.

8. REFERENCES

[1] Baker, M., Asami, S., Deprit, E., Ousterhout, and J., Seltzer, M. Non-volatile memory for fast, reliable file systems. In *Proceedings of the fifth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, pages 10-22, 1992.

[2] Cao, P., Felten, E., Karlin, A., and Li, K., A Study of Integrated Prefetching and Caching Strategies. In *Proceedings of 1995 ACM SIGMETRICS*, pages 171-182, June 1995.

[3] Douglis, F., Kaashoek, F., Marsh, B., Caceres, R., Li, K., and Tauber, J. Storage alternatives for mobile computers. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994.

[4] Douglis, F., Krishnan, P., and Bershad, B. Adaptive Disk Spindown Policies for Mobile Computers. In *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing*, 1995.

[5] Douglis, F., Krishnan, P., and Marsh, B. Thwarting the Power-Hungry Disk. In *Proceedings of the 1994 Winter USENIX Conference*, January 1994.

[6] Ellard, D., Ledlie, J., Malkani, P., and Seltzer, M., Passive NFS Tracing of Email and Research Workloads. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST'03)*, pages 203-216, March 2003.

[7] Flinn, J., and Satyanarayanan, M. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles (SOSP)*, pages 48-63, December 1999.

[8] Kuenning, G. and Popek, G., Automated hoarding for mobile computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 264-275, 1997.

[9] Gibson, T. and Miller, E. Long-Term File Activity Patterns in a UNIX Workstation Environment. In *Proceedings of the Fifteenth IEEE Symposium on Mass Storage Systems*, March 1998).

[10] Griffioen, J. and Appleton, R. Reducing File System Latency Using a Predictive Approach. In *Proceedings of the USENIX Summer Conference*, pages 197-207, June 1994.

[11] Helmbold, D., Long, D., and Sherrod, B. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the Second Annual ACM International Conference on Mobile Computing and Networking*, pages 130-142, 1996.

[12] Li, K., Kumpf, R., Horton, P., and Anderson, T. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proceedings of Winter 1994 USENIX Conference*, pages 279-292, 1994.

[13] Li, Z., Wang, C., and Xu, R., Computation Offloading to Save Energy on Handheld Devices: a Partition Scheme. In *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, pages 238-246, 2001.

[14] Marsh, B., Douglis, F., and Krishnan, P. Flash Memory File Caching for Mobile Computers. In *Proceedings of the 27 Hawaii International Conference on System Sciences*, 1994.

[15] Ousterhout, J., Da Costa, H., Harrison, D. Kunze, J., Kupfer, M., and Thompson, J. A Trace-Driven Analysis of the UNIX 4.2 BSD File System, In *Proceedings of the 10th ACM Symposium on Operating Systems and Principles*, pages 15-24, 1985.

[16] Papathanasiou, A. and Scott, M. Energy Efficiency through Burstiness. In Proceedings of the *Fifth IEEE Workshop on Mobile Computing Systems & Applications*, October 2003.

[17] Robinson, J. and Devarakonda. M. Data-cache management using frequency-based replacement. In *Proceedings of the*

1998 ACM Sigmetrics conference on Measurement and Modeling of Computer Systems, 124-142. ACM Press, 1990.

[18] Rudenko, A., Reiher, P., Popek, G., Kuenning, G., *Saving Portable Computer Battery Power Through Remote Process Execution*. In *Mobile Computing and Communications Review*, Vol. 2, No. 1, 19-26, (January 1998).

[19] Satyanarayanan, M.. The Evolution of CODA. In *ACM Transactions on Computer Systems*, 20(2):85-125. 2002.

[20] Stemm, M. and Katz, R. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-held Devices. In *Institute of Electronics, Information, and Communication Engineers Transactions on Communications*, E80B (8):1125-1131. August 1997.

[21] Tait, C., Lei, H., Acharya, V., and Chang, H. Intelligent file hoarding for mobile computers. In *Proceedings of the first annual international conference on Mobile computing and Networking*, pages 119-125, 1995.

[22] Wang, A., Reiher, P., Popek, G. and Kuenning, G. Conquest: Better Performance Through A Disk/Persistent-RAM Hybrid File System. In *Proceedings of the 2002 USENIX Annual Technical Conference*, June 2002.

[23] Weissel, A., Beutel, B., and Bellosa, F. Cooperative I/O – A Novel I/O Semantics for Energy-Aware Applications. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, 2002.

[24] Yaghmour, K. and Dagenais, M. Measuring and Characterizing System Behaviour Using Kernel-Level Event Logging. In *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.