Reducing Energy Consumption using a Non-Volatile storage cache

Timothy Bisson Scott A. Brandt

Computer Science Department
University of California, Santa Cruz
{tbisson,sbrandt}@cs.ucsc.edu

Abstract

We can outperform the most powerful disk spin-down algorithm available by using a small non-volatile storage write-only cache to temporarily store hard disk write requests while a disk is spun-down. In our experiments, we use a USB thumb drive device as our non-volatile storage cache. By redirecting writes to a flash memory while a disk is spun-down we avoid costly hard disk cycle start-stop operations, thus increasing hard disk reliability and reducing energy consumption.

We also show that such a storage cache is necessary to provide feasible performance from a disk spin-down algorithm in a general purpose operating environment. The problem arises from typical general purpose operating system applications generating periodic disk activity while a system is not in use. By using small, cheap, and non-volatile flash memory we can improve the energy savings of a highly adaptive machine learning disk spin-down algorithm by 36%, and the 15 and 45 second fixed-timeout algorithm by 57% and 46%, respectively. We can also reduce the relative fraction of hard disk "contact start-stop cycles" by 80% for the DSA algorithm, and with the 15 and 45 second fixed-timeout algorithm 79% and 73%, respectively.

1. Introduction

One of the major problems in mobile computing is power consumption. Simply put, mobile computers do not have an unlimited supple of energy and their users are often forced to temporarily abort computing when their battery supply is exhausted. To reduce the energy consumption in mobile computers, much research has been performed and targeted at different subsystems for mobile computers such as CPU [16], networking [15], and disk [8]. The hard disk is a popular subsystem to attack because there are well-defined power states for a disk which can be harnessed by a disk spin-down algorithm to drastically reduce a system's overall power consumption.

A spin-down algorithm generally falls into one of three different categories: fixed timeout, adaptive, and predictive. Fixed timeout spin-down algorithms maintain a constant timeout value independent of the disk parameters. Adaptive spin-down algorithms have a dynamic timeout value that uses hysteresis and disk state power statistics to calculate the timeout. Predictive spin-down algorithms also use disk request hysteresis and power statistics, however, they use hysteresis to determine when it is beneficial to spin-down and do so following a disk request.

With little hysteresis and hard drive statistics adaptive disk spin-down algorithms can provide significant improvements over fixed time-outs. A new and powerful class of adaptive disk spin-down algorithms use the share algorithm, a machine learning class of algorithm [8]. This algorithm uses multiple experts – each predicting a fixed timeout for its overall dynamic timeout calculation. A dynamic weight is associated with each expert and used to restrict its contribution to the overall timeout calculation. A new timeout is calculated after every idle period. An expert's weight is reduced by considering its energy consumption relative to an Oracle's energy consumption policy.

We verified the correctness of the Multiple Experts Adaptive spin-down algorithm by implementing it in the Linux kernel and running the algorithm on our own personal desktop machine [2]. After implementation, we discovered three drawbacks that were not apparent in simulation: latency, sub-optimal performance, and reliability.

We have developed a novel solution to directly address spin-down algorithm reliability and sub-optimal spin-down algorithm performance, and indirectly address spin-up latency. We redirect the periodic write requests that occur while a disk is spun-down to flash memory, which acts as a write-only cache. Redirection occurs until the flash memory becomes full, or the system becomes active (a read request or many quick write requests). By performing this redirection, a disk can stay spun-down for hours at a time, depending on flash memory size, as opposed to seconds. A beneficial consequence is that we reduce the number of spin up-

and-down operations a system must incur because the disk need not be spun-up on idle system periodic writes. Energy savings are substantially increased; Powering a disk drive in stand-by mode requires significantly less power than keeping it in standby mode for short time intervals, repeatedly spinning it up, and keeping it in active power mode (while waiting for the next timeout to be exceeded).

The rest of the paper is organized as follows: Section 2 motivates our work, Section 3 discusses our design, Section 4 presents our preliminary results, Section 5 discusses related work, and Section 6 concludes.

2. Motivation: Latency, Performance, and Reliability

Adaptive disk spin-down algorithms use a hard disk's power requirements in different power states to help calculate timeout values. Today's hard disks generally consume 5-10 times more energy in active than standby mode [19]. As a result, spin-down algorithms become very aggressive as it most efficient to spin-down after only a few seconds of idle time. Unfortunately, spin-up latency becomes an issue with such timeout values. Spin-up latency is the time it takes to completely spin-up a disk after it has been in standby-mode. Typical latency times for modern hard disks are in the low seconds. With such short timeout values and long spin-up latency, it is easy to imagine system usage patterns in which a user consistently must endure spin-up latency. A quick example is a user editing a document. The hard disk spins down while the user is editing the document, but whenever the user saves their document, they must wait several seconds for the disk to spin-up and service their application's requests. This is clearly unacceptable.

Operating systems incur periodic writes during idle periods when there is no user activity. Applications, such as system loggers, web browsers, and e-mail clients are responsible as well has system daemons. Figure 1 shows the system activity on a graduate student's personal desktop Linux box for a 24-hour period running a typical Linux file system, Reiser V3, no unnecessary daemons, and mounted with the "no_atime" option. In this figure we see that even during periods of non-user activity there are still at least 4 writes per minute. Actually, the writes occur in bursts roughly every 40 seconds. This is due to the parameters specified by pdflush, a kernel daemon in the Linux kernel that controls when dirty buffers are flushed to disk. Thus the problem is that we lose a large percentage of the potential disk spin-down energy saving because our hard disk is periodically spun-up and down every few seconds even when no user-activity is present.

With idle times less than a minute (and no user activity), the number of hard disk "contact start-stop cycles" increase dramatically. A "contact start-stop cycle" is the completion of enabling the spindle motor and moving the disk

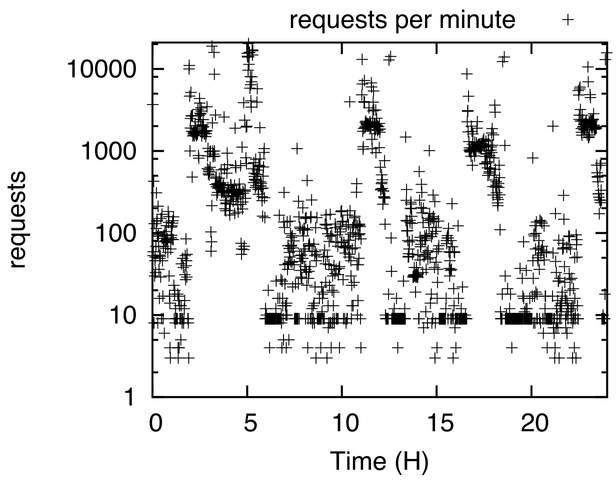


Figure 1: Trace Log (Requests per Minute)

head from its parked position to activating the spindle motor and head, and back to parking the head and disabling the spindle motor. In one hour of non-user activity, roughly 60 contact start-stop cycles occur. Desktop hard disks are rated for 10s of thousands of contact start-stop cycles [13, 17, 6]. Therefore, using an aggressive disk spin-down algorithm will result in exceeding the maximum contact start-stop cycles in a matter of days. Exceeding the maximum contact start-stop cycles may result in disk failure, which at best is inconvenient and at worst a disaster.

3. Design

We detail the design of redirecting writes from a hard disk to a small amount of flash memory while a disk is spun-down. We present our design in the context of the Linux Kernel.

3.1. Indexing flash data blocks sectors

We first describe how redirected data blocks are stored in flash memory. Blocks in flash memory are logically split into data-blocks and meta-blocks. Data-blocks hold actual data while meta-blocks hold the sector index values for data blocks. The sector index values stored in the meta-blocks are for the hard disk location not the flash sector location. Direct indexing is used for associating sector indexes with data blocks. For example, sector index 32 in the 10th logical block of flash memory (a "meta-block") is associated with the data-block stored in flash memory at logical block 5152 and the value from the meta-block sector index is the data block's sector location on disk. Figure 2 shows the flash block layout of meta and data blocks.

Since flash memory is nonvolatile, we store sector indexes in the meta blocks to guarantee consistency in the case of system failure during redirection.

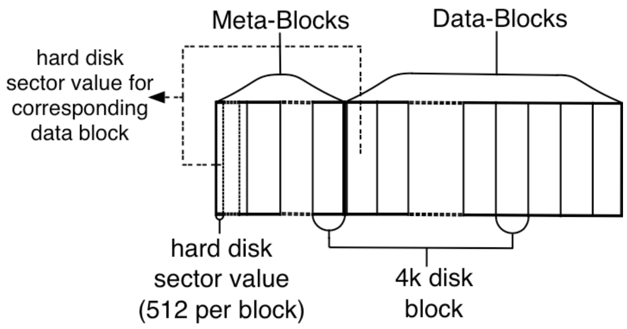

Meta-Blocks
Data-Blocks
hard disk sector value for corresponding data block
hard disk sector value (512 per block)
4k disk block

Figure 2: Flash block layout

Data blocks are 4KB and can hold 512 8-byte sector address per 4KB block. Therefore, each 2 MBs of flash storage requires only 1 meta-block. Figure 5a shows that we reach maximum energy reduction at 128-MB. Using a 128MB of flash memory (32768 blocks), 4 "meta-blocks" (16KB) are needed.

3.2. Redirecting write requests

When a disk is spun-down due to a time-out being exceeded, a global flag (*spun_down*) is set to TRUE. When the disk is spun back up, *spun_down* is reset to FALSE. A kernel daemon, disk_daemon, is used to calculate the current timeout, spin down the disk, and set the *spun_down* flag back to TRUE when the timeout it is exceeded. *Spun_down* is reset to FALSE and requests are no longer redirected to flash memory when the block driver function make_request() is called to send a request to the disk driver. While the *spun_down* flag is set to TRUE, all write requests are redirected to flash.

In order to redirect a write request we record the request size and starting sector. We use these parameters to generate two redirection requests for each hard disk request: one meta-block sector request and one data-block request. The meta-block request appends the new sector indexes associated with the data-block requests to the current meta-block. We keep a memory map of the sector indexes for the current meta-block we are writing and simply append the new sector indexes to this 4KB map and generate a write request for it. This avoids having to read the meta-block into memory, append our sector index entries to it, and write the block back to flash memory. When all 512 sector indexes for the current meta-block are used, we generate a write request for the current meta-block one last time, zero it's memory map and increment the logical block number of the current meta-block our memory buffer maps to.

The data block request is the actual write request generated for hard disk but redirected. Write requests are written sequentially to Flash. The sector location where the write is destined for is calculated with the following function: $sector = MetaBlocks + CurrentMetaBlock * CurrentMetaBlockIndex$

3.3. Flushing data blocks

Figure 3 shows that capacity and user-activity will cause the system to halt redirection and spin-up the disk. Capacity occurs when the flash memory becomes full. We flush the flash memory data blocks to the appropriate sectors on the hard disk. Additionally, when a user begins using the system, we stop redirection. A read request or many write-requests mark user-activity. We use "current write request" (cwr) to denote the write request threshold. While the disk is spun-down, if the number of consecutive write requests exceeds cwr while *idle_time* < *timeout*, the system is deemed active and all data blocks on the USB storage device are flushed to disk.

We denote user-activity as a case to stop redirection because the aim of our functionality is to reduce spin-ups due to periodic application writes while the system is not being used by a real user. In the next section we present the results for different values of write-request thresholds. We show that a real and significant trade-off exists between how much to redirect and the time to flush flash memory data blocks to disk.

When we stop redirecting, we flush all blocks in the flash memory to the hard disk to maintain consistency between the hard disk and flash memory. We choose this design for its simplicity and minimal impact to the Linux IDE device driver. Alternative designs exist, such as lazily flushing the flash buffers or flushing buffers at specific rates. Such designs add a significant amount complexity and it isn't clear whether the added complexity justified.

To flush the data-blocks to disk, we repeat the process of reading a meta-block into memory, reading the corresponding data-blocks into memory, then writing those buffers to disk, and finally invalidating the meta-block descriptors and writing it to flash. This process is repeated until a meta-block does not contain all valid data blocks or the last meta-block was read. To determine which data blocks have valid buffers for a particular meta-block, we do a sequential scan of the data block descriptors after the meta-block is read into memory. If a descriptor contains a non-valid identifier, the scan is stopped and the current index is noted so it is known how many data blocks to read in and write to disk. Since we invalidate the meta-block descriptors after reading them in during the flushing process, we know which data blocks will have valid meta-block descriptors. Note that we can read the blocks back into memory from flash with a few large requests because the data blocks are located sequentially on flash in logical sector order. Assuming the first meta-block, contains 512 valid entries, the 0th - 511th data-block would be read into memory. As a result,

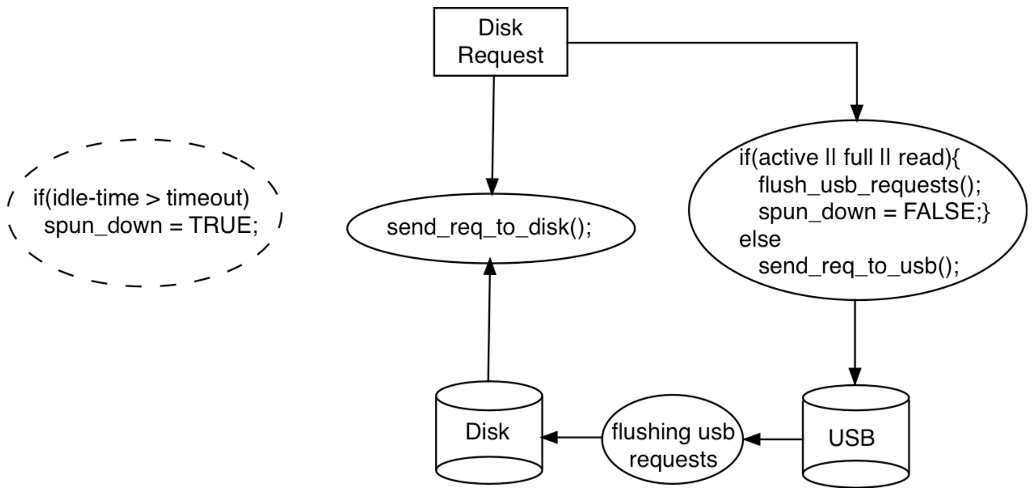


Figure 3: Diagram of redirecting disk requests

we can potentially read into memory at most 2MB of data at a time.

4. Preliminary results

To verify the performance of redirecting writes during periods of spin-down, we traced the disk activity on a desktop machine and executed our algorithm against the generated traces. We examine the energy consumption and number of spin-ups used by adding redirection to a spin-down algorithm, and we benchmark the cost of synchronizing a hard disk to USB by reading X data blocks into memory from USB and then writing them to the hard disk. Note, we use a USB thumb drive to test our hypothesis, but envision a small amount flash memory embedded on the disk or motherboard.

4.1. Trace Generation

We traced the disk access on desktop system for 24 hours. The system runs Linux 2.6.9, uses Reiser V3 as the root file system mounted with no_atime option, and has syslog-ng as the system logger. The system has a 3.2 P4 HT processor with 1GB of memory. The system is used by a graduate student that uses the system for programming (mostly Linux kernel development), listening to music streams, reading e-mail, web-browsing, and text processing. The system also has a nightly cron job to update all packages on the system every morning at 1AM. Package updating is source-based, so depending on the type packages needing to be updated, the update process may take 5 minutes to a few hours.

Trace entries were recorded to an in-kernel trace structure that held 8KB of trace entries. We add an entry to the trace memory buffer when the IDE disk driver function ide_do_rw() is called. When all entries are filled up, the memory structure is passed to a user-space process, which then sends the trace data (via sockets) to another desktop system. Doing so, avoids logging trace buffers on the same disk we are tracing, thus skewing the results. The format of a trace entry is:

```
struct trace_entry{
    unsigned long long time;
    int rw;
    unsigned long long sector;
    int size;
};
```

4.2. Preliminary Results

We benchmark the cost of reading different sizes of USB into memory, then copying flushing those memory buffers to disk to get a general idea of the timeline for flushing data blocks from USB to hard disk. Our experiment uses a 64MB USB thumb drive from Dell. Figure 4 shows the results of our experiment. In this figure, Read USB is the time it takes to read the data blocks from the USB storage device into memory. Write Disk is the time it takes to write the memory buffers to disk. For USB disk sizes 8K to 512K, it takes less than a second to synchronize the hard disk with the USB data blocks. Above 512K, the time to synchronize the two devices increases linearly, and reading the USB data blocks into memory contributes most to the overhead.

The intent of this work is to show that we can drastically reduce energy consumption as well as increase reliability by redirecting writes for a spun-down disk that suffers from periodic idle-time writes. In Figure 5 we show the relative benefit of adding redirection to both the DSA algorithm and several fixed-timeout values. The DSA algorithm is the adaptive disk spin-down algorithm developed by Helmbold *et al.* [8]. Figures 5(a) and 5(c) show the results for our experiments using the DSA algorithm. In both figures cwr represents the activity threshold tunable we described in section 3.3. Figure 5(a) shows the percentage of energy used when adding a temporary USB cache to the DSA algorithm. As the smallest request size is 4KB, the smallest USB cache size we use is 8KB. For cache sizes 8 and 16k, we actually increase the amount of energy we use by about .1%. This happens because we can only redirect for 1-2 4KB writes, respectively, and although USB energy consumption is substantially less than disk, in these two cases, keeping a disk spun-down for an extra 1-2 redirect writes does not offset powering the USB device to store those data blocks.

The maximum energy savings occur roughly between 1 and 10MB, with cwr activity threshold values of 25 and 150, respectively. With cwr at 25, we can save up to 36% energy, and with cwr at 150, we can save up to 46% energy. The cwr value shouldn't just be set at 150 because as we mentioned before, the more data we redirect the longer it takes to synchronize the disk with the USB data blocks as we saw in figure 4.

We also measure the fraction of contact start-stop cycles that occur, as a result of adding redirection to the DSA algorithm in figure 5(c). With roughly 10MB of USB cache, we can reduce the number of spin-ups by 60-80%, depending on the cwr parameter. The cwr parameter has a bigger impact on the number of spin-ups than energy savings. As cwr increases, more write requests are served to the USB thumb drives. This results larger flushes when redirection stops, but less frequently (less contact start-stop cycles). The downside is that the longer synchronize times are incurred.

In both graphs, between 8k and 256K, the plots for different cwr values are almost identical, and then the points drift to different limits with different cwr values for larger USB storage device sizes. This occurs because below 512K, the limiting factor for the number spin-ups or energy consumption is the USB device size, not the cwr parameter. With smaller cwr values, the plots values would be different for small USB device sizes.

In 5(b) and 5(d) we show the relative benefit of using redirection with a fixed timeout disk spin-down algorithm using timeout values of 15s, 30s, and 45s. We don't include 60s because to idle periods exist for that long and thus the result would be the same as never spinning down. In this set of graphs we used cwr set at 100 so that we can focus on the different timeout values.

We see that 5(a) and 5(b) have similar plots, however we see that in (b), the curve is significantly steeper between 100k and 1M. The reason for the strong disparity between the DSA with redirect and the fixed timeout with redirect is that DSA is already a fairly optimal algorithm at reducing energy consumption while the fixed timeout algorithm is not. Therefore, by redirecting writes we are able to offset the sub-optimality of the fixed-timeout algorithm through redirection, which is why we see the steep curve in (b).

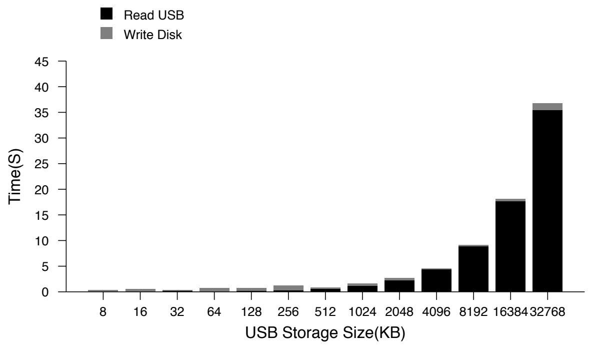


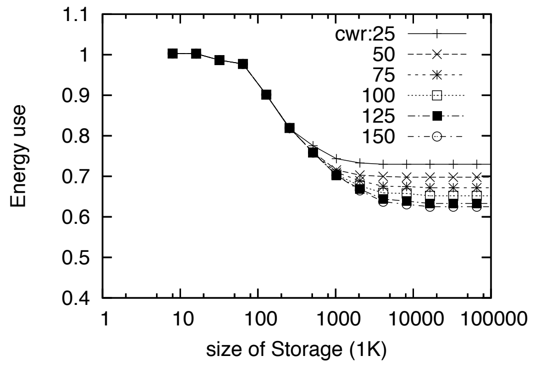
Figure 4: Benchmark: synchronizing hard-disk to USB (USB 100% full)

In figure 5(d), fixed-timeout 30s and fixed-timeout 45s are nearly identical. However, they actually differ by roughly 1% at the larger storage cache sizes. We see that the fixed-timeout 15s plot is more similar to that of the DSA algorithm in (c). We believe this is because the 15s timeout algorithm begins to approach many of the timeout values for the DSA algorithm, thus the curve near the 16-64k are more similar than the fixed timeout values of 30s and 45s. Additionally, we see that the maximum relative reduction in "contact start-stop cycles" is that both (c) and (d) have similar values near the large size of flash storage cache sizes.

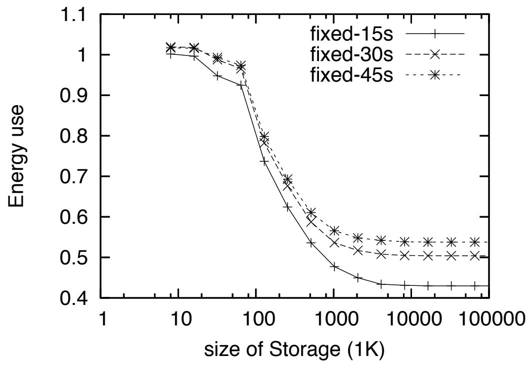
5. Related work

It is well known that using a cache above disk is an effective way of increasing performance. A popular application of this concept is with Distributed file systems. Baker *et al.* looked at using NVRAM as a file cache to reduce write traffic to file server, and NVRAM as write buffers to reduce file disk servers [1, 3, 12].

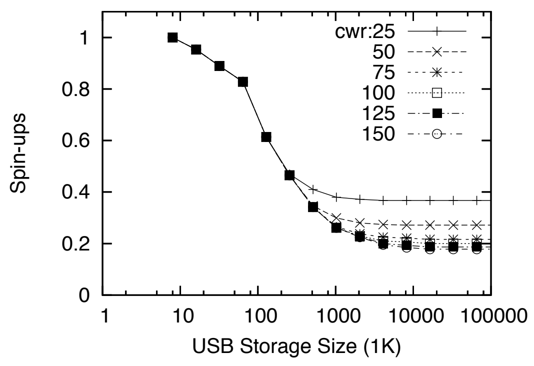
Similar to our own work, Marsh *et al.* developed a "FlashCache" which is a read-write non-volatile buffer cache that exists between disk and memory [11]. They show that by using such a cache, energy savings are significant. In this work, FlashCache buffers writes and services reads even while the disk is active, to reduce latency. Our design focuses on maximizing energy reduction. Therefore, we use a write cache and buffer only writes while the disk is spun-down so that spin-down intervals are maximized. In Marsh's design, FlashCache buffers may be filled while the disk is active, thus reduc-



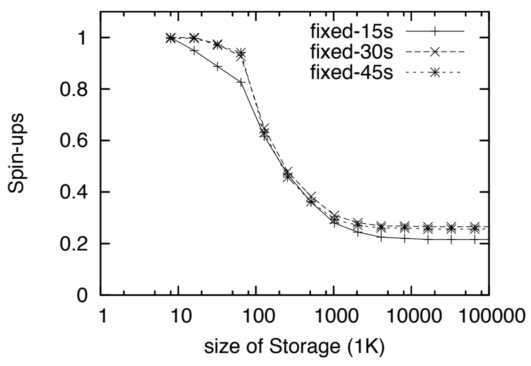
(a) % energy using DSA w/ redirect



(b) % energy using a fixed timeout w/ redirect



(c) % CSS cycles w/ DSA and redirect



(d) % CSS cycles w/ fixed timeout and redirect

Figure 5: Performance: Graphs show benefit of using redirect with DSA algorithm in the context of energy consumption and number start-stop operation cwr is the number of writes we allow when *idle_time* $<$ *timeout* before we deem the system active and execute flush_usb().

ing the number of available buffers when spun-down. Additionally, in FlashCache the metadata needed to map logical blocks between the FlashCache and disk is not discussed, nor how consistency is maintained when serving read requests when the requested data lives on both the FlashCache and disk.

A new application of using NVRAM in a storage hierarchy is to tier multiple disks vertically with NVRAM at the highest caching level [9]. A second disk exists between primary storage and NVRAM, which serves as a cache-disk. When, the cache-disk is idle, which occurs when it has finished copying its buffers to primary storage, buffers NVRAM are copied to the cache-disk.

Douglis *et al.* looked at storage solutions for storage in Mobile computing [4]. They primarily investigate the trade-offs using flash memory and hard disks for primary storage. Additionally, they look at using SRAM and DRAM to function as a buffer cache. They found that flash memory can provide significant energy conservation while providing decent I/O performance. However, flash memory performance is not uniform.

There are several adaptive disk spin-down algorithms that achieve within 10% of the the optimal energy consumption [5, 8, 14, 7]. Our work is complementary to any disk spin-down algorithm including fixed timeout algorithms. Our goal is to enhance the performance of disk spin-down algorithms by increasing the available time a disk can be spun-down by buffering writes during periods of idle time.

LaRosa and Bailey attempt to provide a new approach to reduce energy consumption of mobile devices such as laptops. Their approach is to use nonvolatile memory to reduce energy consumption during mobile use by perfecting likely to be used files to a nonvolatile cache during plugged-in mode. When the laptop enters mobile-mode the hard disk is spun-down and files are accessed from the nonvolatile memory cache [10]. h

Cooperative I/O introduces the concept of processing all pending dirty-memory before a disk should be shut-down to maximize the time spent spun-down [18]. This approach will help reduce energy consumption in our work as well, since this concept will reduce the number of blocks that persist in flash memory.

6. Conclusion and future work

In this work we have shown that using a small amount of flash memory as a temporary write cache in conjunction with a disk spin-down algorithm we can reduce the the energy consumed by 36% over the Dynamic Spin-down

Algorithm developed by Helmbold and others. We exploit fixed timeout based algorithm by reducing the consumed energy by 46-57%. We can reduce the relative fraction of contact start-stop cycles by up to 80% for the DSA algorithm and 79%, 73%, and 74% for fixed-timeout values of 15, 30, and 45 seconds, respectively. For mobile computing systems this provides a significant enhancement to disk spin-down algorithms by making them not only feasible, but practical.

In our experiments we have discovered there is a significant tradeoff between the amount of data to be redirected and the time to flush that data from flash memory to hard disk. With flash memory currently available in MBs, it takes seconds to flush the data blocks, with time increasing linearly proportional to redirection size. Fortunately, we've shown only a few megabytes of memory is necessary to reap the benefits of using a write-only cache in flash for redirection.

Flash memory is the appropriate hardware solution for this problem. It requires significantly less power than disk. It is inexpensive in the amounts we require (a few MBs for a few dollars), and it is non-volatile so that data persists across crashes.

We are currently investigating alternative designs including, but not limited to, redirecting writes for more than one disk and alternative flushing techniques.

References

[1] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, pages 10–22, 1992.

[2] Timothy Bisson and Scott A. Brandt. Adaptive disk spin-down algorithms in practice. In *3rd USENIX Conference on File and Storage Technologies, (Work in Progress Proceedings)*, 2004.

[3] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The rio file cache: Surviving operating system crashes. In *Architectural Support for Programming Languages and Operating Systems*, pages 74–83, 1996.

[4] Fred Douglis, Ramon Caceres, M. Frans Kaashoek, Kai Li, Brian Marsh, and Joshua A. Tauber. Storage alternatives for mobile computers. In *Operating Systems Design and Implementation*, pages 25–37, 1994.

[5] Fred Douglis, Padmanabhan Krishnan, and Brian Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, pages 130–142, 1996.

[6] IBM Hard Disk Drives. http://www.pc.ibm.com/ww/hdd/hddredirect.html. IBM hard drive specifications.

[7] Y.Lu G. De Micheli. Adaptive hard disk power management on personal computers. In *IEEE Greate Lakes Symposium on VLSI*, pages 50–53, Mar 1999.

[8] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 130–142, 1996.

[9] Yiming Hu and Qing Yang. Dcd
disk caching disk: a new approach for boosting i/o performance. In *Proceedings of the 23rd annual international symposium on Computer architecture*, pages 169–178. ACM Press, 1996.

[10] Christopher R. LaRosa and Mark W. Bailey. A docked-aware storage architecture for mobile computing. In *Proceedings of the first conference on computing frontiers on Computing frontiers*, pages 255–262, 2004.

[11] B. Marsh, F. Douglis, and P. Krishnan. Flash memory file caching for mobile computers. In *To appare in Proceedings of the 27th Hawaii Conference on Systems Science*, 1994.

[12] Ethan L. Miller, Scott A. Brandt, and Darrell D. E. Long. Hermes: High-performance reliable mram-enabled storage. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 95. IEEE Computer Society, 2001.

[13] Western Digital Desktop Hard Drives Overview. http://www.wdc.com/en/products/index.asp?cat=3. Western Digital hard drive specifications.

[14] J.S. Vitter P. Krishnam, P.M. Long. Adaptive disk spin-down via optimal rent-to-buy in probabilistic environments. In *Proceedings of the 12th annual Internation Conference on Machine Learning*, pages 322–330, Jul 1995.

[15] Christian Poellabauer and Karsten Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proceedings of the 10th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 48–55, May 2004.

[16] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 251–259. ACM Press, 2001.

[17] Seagate Disk Drive Technology. http://www.seagate.com/products/discselect/. Seagate hard drive specifications.

[18] Andreas Weissel, Bjoern Beutel, and Frank Bellosa. Cooperative i/o: a novel i/o semantics for energy-aware applications. *SIGOPS Oper. Syst. Rev.*, pages 117–129, 2002.

[19] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption, March 2003.