Interposing Flash between Disk and DRAM to Save Energy for Streaming Workloads

M.G. Khatib, B.J. van der Zwaag, P.H. Hartel, and G.J.M. Smit
Department of Computer Science
University of Twente, Enschede, the Netherlands
{m.g.khatib, b.j.vanderzwaag, pieter.hartel, g.j.m.smit}@utwente.nl

Abstract— **In computer systems, the storage hierarchy, composed of a disk drive and a DRAM, is responsible for a large portion of the total energy consumed. This work studies the energy merit of interposing flash memory as a streaming buffer between the disk drive and the DRAM. Doing so, we extend the spin-off period of the disk drive and cut down on the DRAM capacity at the cost of (extra) flash.**

We study two different streaming applications: mobile multimedia players and media servers. Our simulated results show that for light workloads, a system with a flash as a buffer between the disk and the DRAM consumes up to 40% less energy than the same system without a flash buffer. For heavy workloads savings of at least 30% are possible. We also address the wear-out of flash and present a simple solution to extend its lifetime.

I. Introduction

Due to the increasing electricity prices [5], the demand for energy-efficient computer systems is increasing. The conventional storage hierarchy, comprising the disk drive and the DRAM, accounts for a large portion of the energy consumed by the computer system. If the workload is predictable, which is typically the case for streaming workloads, the disk can be spun off most of the time, thus saving orders of magnitude on energy [12]. Most systems therefore pre-fetch data into DRAM to maximize the spin-off period. However, retaining data in DRAM is not free, because each DRAM refresh cycle dissipates a few milliwatts per MB. The maximum energy saving is attained when the energy consumption of the disk and the refresh energy of the DRAM are in balance.

Our contribution is to avoid the saving ceiling by putting a flash buffer in between the disk and the DRAM as a traffic reshaper. Since flash does not require refresh, the size of the flash buffer is limited only by the amount of money that the user is willing to spend (once) to save energy (during use). The latency of flash is lower than the latency of the disk, but higher than that of DRAM, so that a small DRAM buffer is sufficient to sustain the required streaming rates. The combination disk-flash-DRAM makes optimal use of the disk (by extending its spin-off period) and it makes optimal use of the DRAM (by using little of it). The cost of the system increases through the addition of the flash but decreases through the reduction of the amount of DRAM.

To evaluate the idea, we present energy simulations of two streaming systems: a mobile multimedia player and a media server. The simulations are based on models of DRAM, NAND flash, and two disk drives: a typical laptop drive and a server drive. Using energy measurements on an existing setup in our laboratory, we are able to validate our model partially.

The outline of our paper is as follows. In the following section, we review related work on energy conservation for storage systems. In Section III, we introduce our architecture model. Buffer capacities are quantified in Section IV. We present our simulation methodology and discuss the results in Section V. An experimental validation of the simulation results follows in Section VI. We discuss the wear-out limitation of flash in Section VII. Section VIII concludes.

II. Related Work

Since the disk drive is the main energy-consuming component, most related work extend the spin-off period of the disk to increase energy saving. Two approaches are used: pre-fetching expected data and/or redirecting I/O traffic.

For predictable workloads of high temporal locality of reference, such as audio and video streaming, pre-fetching is applied. Mesut el al. [12] pre-fetch as much streaming data as possible into DRAM, so that the disk spins off for long time. Here, the disk energy and the DRAM energy are balanced to minimize the total energy consumption.

For unpredictable workloads, however, pre-fetching is challenging. All future demands should be predicted and pre-fetched to guarantee long-enough spin-off periods for the disk. Providing hints about the future demands of applications [13] and the usage patterns [10] is proposed to improve predictability and thus maximize energy saving. Bisson et al. [6] redirect write traffic to flash to extend the disk spin-off period. Doing so, read requests hit mostly the flash, further extending the spin-off period.

In environments like web servers where a huge DRAM capacity is demanded, the DRAM also consumes a considerable amount of energy. Kgil et al. [8] use flash as a cache for DRAM to offload infrequently accessed files (approximately 80% of the total accessed files). As a result, the capacity of the DRAM decreases, saving orders of magnitude on its retention energy.

Unlike Bisson et al. and Kgil et al., in our work (1) we use flash not as a cache but as a streaming buffer between disk and DRAM, thus saving both disk and DRAM energy. By extending the work by Mesut et al., (2) we take a holistic

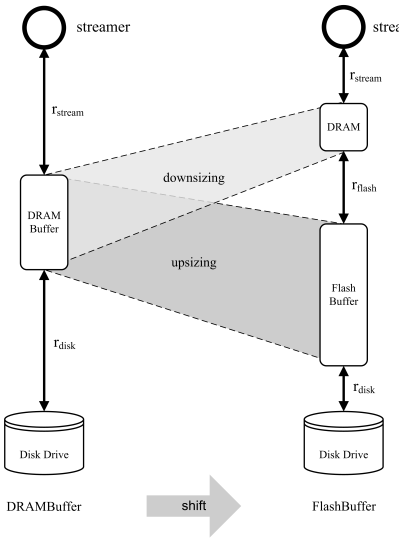


Fig. 1. The DRAMBuffer architecture (*left*) versus the FlashBuffer architecture (*right*)

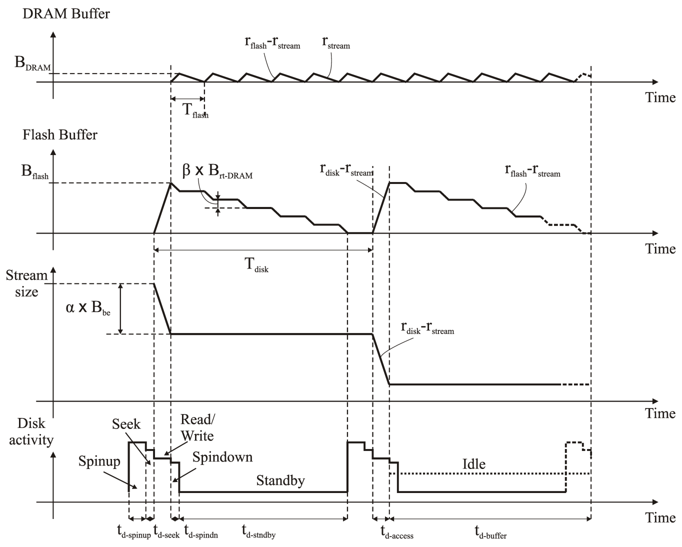


Fig. 2. Activities of the disk drive, the flash memory, and the DRAM in the FlashBuffer architecture during one streaming cycle (T_{disk})

approach and consider the entire storage hierarchy (i.e., disk–flash–DRAM combination) to study the energy merit of flash as a streaming buffer.

Our work complements the work by Marwedel et al. [11], where scratch pad memory (SPM) in the upper level of the memory hierarchy replaces the conventional cache in hosting frequently accessed code segments (and variables) to save energy. We target the lower level of the memory hierarchy, where flash does not entirely replace the main memory but overtakes its long-term retention functionality.

III. The FlashBuffer Architecture

Figure 1 (left) shows the conventional storage hierarchy for a streaming architecture (called the *DRAMBuffer* architecture), where DRAM serves as a primary buffer for the disk drive. The same figure (right) shows our proposed architecture (called *FlashBuffer*), where flash serves as the primary buffer. Since flash is less expensive than DRAM, we use more flash than DRAM in DRAMBuffer. Also, since flash has a shorter latency than disk drives, the DRAM in FlashBuffer is smaller than the DRAM in DRAMBuffer. The disk fills the flash at rate r_{disk} (i.e., the throughput of the disk) and the flash fills the DRAM at rate r_{flash}. The DRAM is emptied at rate r_{stream}, the streamer rate.

A. Traffic Reshaping

Figure 2 shows the activity of the disk, flash and DRAM in FlashBuffer. The disk is started every T_{disk} to fill the flash with a *large* amount of data at a rate $r_{\text{disk}} - r_{\text{stream}}$. As shown, the disk spins up and seeks before every flash refill and spins down immediately after it. It stays in standby to save energy. The flash repeatedly refills the DRAM at rate $r_{\text{flash}} - r_{\text{stream}}$ with a *small* amount of data. When the flash is almost empty, the disk is started ahead, to prepare for a new refill. This process is repeated as long as the stream is running.

B. Implementation Issues

Secondary buffer — Flash in FlashBuffer communicates with the system via the I/O subsystem, which is shared across several components, among which is the flash. Although flash has a short latency, that can be accounted for in streaming applications, we use a secondary pre-fetching buffer. This is because the I/O subsystem should be freed, for some time, for other components.

Dual-Port flash — As Figure 2 shows, the flash memory refills (from the disk) and flushes (into the DRAM) at the same time. To allow these simultaneous activities, dual port memory is needed.

Flash throughput — Interposing flash between disk and DRAM dictates that flash should not become a performance bottleneck. In that regard, several lessons can be learnt from DRAM to improve its throughput, such as deploying multiple banks and/or modules. A good real example is the SanDisk Extreme-IV CompactFlash card, which supports a read/write throughput of up to 40 MB/s [1].

Best-effort I/O — Our work investigates the merit of using flash as a buffer mainly for streaming workloads. Nonetheless, the scheduling shown in Figure 2 can be extended by a fixed slack for servicing best-effort requests in mixed-media environments as proposed by Mesut et al. [12].

IV. Buffer Capacities

In this section, we analytically derive the capacity of the primary buffer of the disk in both architectures, namely the DRAM in DRAMBuffer and the flash in FlashBuffer. Then, we derive the capacity of the secondary buffer, that is the DRAM, in FlashBuffer. The objectives are to maximize the capacity of the primary buffer, thus increasing the idleness of the disk, and to minimize the capacity of the secondary buffer, thus saving on retention energy. To guarantee continuous streaming, the buffers should always contain enough data (i.e., not less than the real-time buffer) to account for the incurred access latencies, thus performance is guaranteed.

A. Primary-Buffer Capacity

Energy can be saved if the disk is spun off (put in standby) for a sufficiently long time. Since streaming workloads are predictable, data can be buffered ahead into the main buffer as Figure 2 shows and then the disk can be spun off. However, putting the disk in standby ($E_{\text{d-stndby}}$) consumes energy to spin up ($E_{\text{d-spinup}}$), seek ($E_{\text{d-seek}}$), and spin down ($E_{\text{d-spindn}}$) every time it is accessed for a new refill. This additional energy should be, at least, compensated for to save energy or the disk should be left in idle ($E_{\text{d-idle}}$). To compensate for, a sufficiently long flush period, and thus standby period, is required ($t_{\text{d-buffer}}$). This can be formulated as follows:

$$P_{\text{d-idle}} \times t_{\text{d-buffer}} \geq (t_{\text{d-buffer}} - t_{\text{d-oh}}) \times P_{\text{d-stndby}} + E_{\text{d-oh}} \quad (1)$$

where

$$t_{\text{d-oh}} = t_{\text{d-spinup}} + t_{\text{d-spindn}} + t_{\text{d-seek}}$$
$$E_{\text{d-oh}} = E_{\text{d-spinup}} + E_{\text{d-spindn}} + E_{\text{d-seek}}; \quad E_{\text{d-x}} = t_{\text{d-x}} \times P_{\text{d-x}} .$$

The flush period that balances the two sides of inequality (1) is called the break-even period, t_{be}. It is calculated as follows:

$$t_{\text{be}} = \frac{E_{\text{d-oh}} - t_{\text{d-oh}} \times P_{\text{d-stndby}}}{P_{\text{d-idle}} - P_{\text{d-stndby}}}$$

and its corresponding break-even buffer capacity (B_{be}) is:

$$B_{\text{be}} = t_{\text{be}} \times r_{\text{stream}} . \quad (2)$$

Different pre-fetching levels (and thus energy savings) can be achieved by deploying a buffer capacity larger than the break-even buffer. We express this by a sizing parameter called $\alpha \in [1, 0)$ based on the break-even buffer. Thus, the capacity of the primary buffer (B_{prm}) becomes:

$$B_{\text{prm}} = \alpha \times B_{\text{be}} \quad (3)$$

To prevent under-run, the primary buffer should contain enough data to fulfill the real-time throughput of the stream:

$$B_{\text{prm}} \geq B_{\text{rt-prm}}$$

Appendix I provides details on the calculation of the capacity of the real-time buffer (Equation (5)).

B. Secondary-Buffer Capacity

The secondary buffer is smaller than the primary buffer, because the main latency, due to the disk drive, is absorbed by the primary buffer. The minimum capacity of the secondary buffer is equal to the capacity of the real-time buffer ($B_{\text{rt-sec}}$). We derive the real-time buffer capacity in Appendix I.

In practice, however, deploying a capacity, that is equal to $B_{\text{rt-sec}}$, fully reserves the Flash–DRAM link. This is because while the DRAM is flushing its current contents, the flash masters the link and prepares for a new refill, leaving no time for other resources that share the link. This shortcoming can be overcome by increasing the capacity of the secondary buffer.

TABLE I
OUTPUT AND TUNING PARAMETERS OF THE FLASHBUFFER MODEL

Parameter	Description
B_{prm}	capacity of the primary buffer
B_{sec}	capacity of the secondary buffer
B_{be}	capacity of the break-even buffer of the disk
$B_{\text{rt-prm}}$	minimum capacity of the primary buffer
$B_{\text{rt-sec}}$	minimum capacity of the secondary buffer
α	sizing factor of the primary buffer
β	sizing factor of the secondary buffer

We express this in terms of a sizing parameter $\beta \in [1, 0)$ as follows:

$$B_{\text{sec}} = \beta \times B_{\text{rt-sec}} . \quad (4)$$

The refresh power of DRAM scales proportionally with its capacity. Therefore, we tune β such that the DRAM average power dissipation is traded off to the Flash–DRAM link's availability. Table I summarizes the output, internal and tuning parameters introduced in this section. An evaluation of FlashBuffer against DRAMBuffer follows next.

V. SIMULATION

In the previous section, we have quantified the buffer capacities in the DRAMBuffer and FlashBuffer architectures. Equations (3) and (4) give the corresponding capacities. We now use these buffer capacities to calculate the energy consumption of both architectures.

A. Methodology

a) Modeling: Detailed models of the power dissipation of the disk drive and the flash memory are provided in the extended version of this paper [9]. DRAM consumes energy to retain data and to access (i.e., read/write) data. The retention energy of the DRAM scales proportionally with its capacity, whereas the access energy depends on the access pattern. We use Micron's power calculator [7] in our simulation framework to calculate the energy consumption for different DRAM capacities and access patterns.

b) Benchmarks: We simulate both architectures for different workload sizes. The workload size is represented in terms of the total streaming demand (i.e., aggregated stream rate). We experiment with light workloads as representative for applications like streaming in laptops. Further, we experiment with heavy workloads found in media servers. We choose $128, 512, 1024$, and 2048 kbps for single audio and video streaming in light workloads. Whereas $10 \times 1024, 20 \times 1024$, and 30×1024 kbps are chosen as aggregated video demands in heavy workloads.

c) Hardware Setup: In addition to the workload size, different disk drives and number of flash modules have been chosen. We use the 1.8-inch Hitachi Travelstar C4K40 hard disk drive (HDD) [2] and three SanDisk CompactFlash Extreme-III cards [1] as representatives of the disk and the flash memory, respectively, for the light workloads. Media servers deploy large HDDs, therefore we set our model to the specifications of the 3.5-inch Hitachi Deskstar 7K500 HDD [3] and use five flash cards to account for the large

TABLE II

SETTINGS OF THE INPUT PARAMETERS

Parameter		1.8" HDD	3.5" HDD
r_{disk}	[Mbps]	187.2	383.2
$t_{\text{d-spinup}}$	[s]	3	9
$P_{\text{d-spinup}}$	[W]	1.5	29.5
$t_{\text{d-seek}}$	[s]	0.015	0.016
$P_{\text{d-seek}}$	[W]	1.122	8
$P_{\text{d-access}}$	[W]	0.495	11
$t_{\text{d-spindn}}$	[s]	0.5	1.5
$P_{\text{d-spindn}}$	[W]	0.33	10
$P_{\text{d-stndby}}$	[W]	0.099	1
$P_{\text{d-idle}}$	[W]	0.33	5
Parameter		Flash1	Flash2
r_{flash}	[Mbps]	240	400
$P_{\text{f-access}}$	[W]	0.2×3	0.2×5
$P_{\text{f-stndby}}$	[W]	0.005	0.005
$t_{\text{f-oh}}$	[s]	0.002	0.002

streaming demand and the disk's high throughput. Since the write throughput (approximately 10 MB/s) of the CF Extreme-III card is lower than the read throughput (approximately 20 MB/s), we use more than one card to increase the throughput, so that flash is not a bottleneck. Table II lists the settings of the 1.8-inch and 3.5-inch HDDs, and the flash memories. We select Micron's DRR SDRAM [7] as representative of the DRAM in both architectures. We use CompactFlash cards in our study because of their availability. In practice, however, pure flash modules can be used instead that can be controlled by a simple wear-leveling management algorithm (see Section VII) to avoid two levels of management.

d) Assumptions: For the heavy workload, we make two assumptions: (1) the streams are optimally interleaved on the disk, such that no seeks are incurred between a refill of one stream and its successor. Thus, only the overhead of one seek is accounted for. (2) All streams are stored on the same disk drive. Thus, one disk spin-up, seek and spin-down overhead is accounted for. With these two assumptions, we are decreasing the access overhead, and thus decreasing the probability that the FlashBuffer architecture outperforms the DRAMBuffer architecture in pre-fetching. In other words, we give a lower bound for the improvement of the FlashBuffer architecture.

B. Results

In this section we present and discuss the results obtained by simulation for the two streaming environments.

1) Buffer Capacities: Table III summarizes the capacities of the primary and secondary buffers for both workloads. For the light workloads the flash capacity lies in the order of tens of thousands of kilo-bits, whereas the DRAM capacity is in the order of a few kilo-bits, thanks to flash. On the other hand, the DRAM capacity in DRAMBuffer is equal to the flash capacity in FlashBuffer, as both serve as the primary buffer of the disk.

For the heavy workload, the capacities of the buffers in both architectures show the same trends but at larger magnitudes. The flash capacity (as well as the DRAM in DRAMBuffer) is two (to three) orders of magnitude larger, because of the larger overhead of the 3.5-inch HDD compared to the 1.8-inch HDD as well as because of the larger streaming demands. On the other hand, the DRAM capacity in FlashBuffer is one order of magnitude larger, because the streaming demand of this workload is 10 times larger.

TABLE III

CAPACITIES OF THE PRIMARY BUFFER (WHEN $\alpha = 1$, BREAK-EVEN BUFFER) AND THE SECONDARY BUFFER (WHEN $\beta = 1$, REAL-TIME BUFFER) FOR THE LIGHT AND HEAVY WORKLOADS. WE SCALE AS SHOWN IN EQUATIONS (3) AND (4), RESPECTIVELY.

Bit rate [Kbps]	Primary buffer [Kb] ($\alpha = 1$)	Secondary buffer [Kb] ($\beta = 1$)
128	2401.438	0.257
512	9605.752	1.027
1024	19,211.504	2.059
2048	38,423.007	4.138
10×1024	691,495.680	21.006
20×1024	1,382,991.360	43.116
30×1024	2,074,487.040	66.422

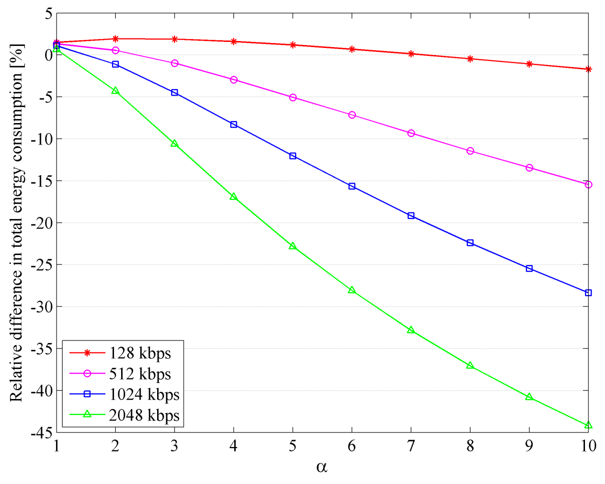
2) Difference in Energy Consumption: Figure 3 (*left*) plots the difference in energy consumption between FlashBuffer and DRAMBuffer for the light workload, calculated as $\frac{E_{\text{FlashBuffer}} - E_{\text{DRAMBuffer}}}{E_{\text{DRAMBuffer}}}$. FlashBuffer consumes more energy for $\alpha \leq 7$ and $\alpha \leq 2$ for the rates 128 and 512 kbps, respectively. This observation remains valid for the other two rates when $\alpha = 1$. This is because the incurred energy to read and write from/to the flash in FlashBuffer outweighs the additional energy consumed by the DRAM in DRAMBuffer. FlashBuffer, however, consumes 13% to 17% less energy than DRAMBuffer for $\alpha = 9, 6,$ and 4 (the optimal saving points of the DRAMBuffer architecture), of the streams 512, 1024, and 2048 kbps, respectively. Figure 3 (*right*) plots the same but for the heavy workload. As α increases, the DRAM capacity in DRAMBuffer increases and consumes more energy, so that the whole architecture consumes more energy than FlashBuffer. At the optimal points of the DRAMBuffer, FlashBuffer consumes 28% to 33% less energy than DRAMBuffer. The reduction percentages are larger for the heavy workload compared to the light workload, because of the higher streaming demand and the larger overhead of the 3.5-inch disk.

Further reduction in energy consumption in FlashBuffer for both workload sizes is possible for large values of α as opposed to DRAMBuffer, which has a maximum saving point. This confirms our hypothesis in Section I: using flash, we can avoid the maximum saving of DRAMBuffer.

VI. EXPERIMENTAL VALIDATION

In this section, we validate our simulated results by comparing both the DRAMBuffer and FlashBuffer architectures using an existing setup in our laboratory.

Our experimental platform is an HP iPAQ H2215 PDA that runs Linux. A Hitachi 4GB Microdrive (approximately 7MB/s) and a SanDisk 2GB CompactFlash Extreme-III card (approximately 20/10 MB/s read/write throughput) are chosen as representatives of the disk and the flash. The PDA has a single CompactFlash interface, into which we can plug either the Microdrive or the flash card. We measure the energy consumption of the storage device (across the CF interface) and the rest of the system separately.



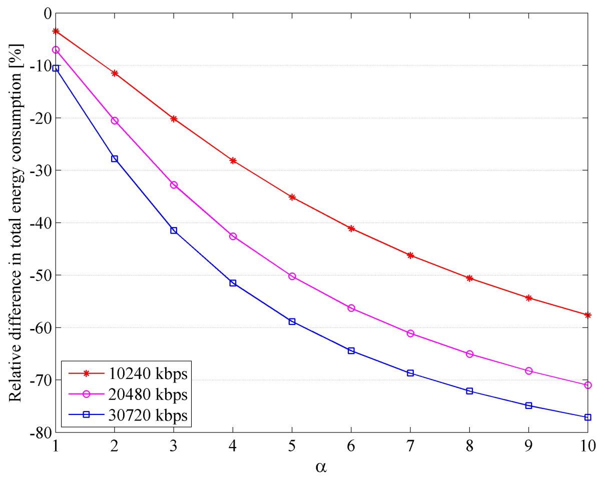


Fig. 3. Relative difference in energy consumption between FlashBuffer and DRAMBuffer, calculated as $\frac{E_{\text{FlashBuffer}} - E_{\text{DRAMBuffer}}}{E_{\text{DRAMBuffer}}}$, for the light (*left*) and heavy (*right*) workloads.

A. *Methodology*

For DRAMBuffer, we measure with different DRAM capacities (i.e., different pre-fetching levels). Given that we use a standard PDA, the amount of physical DRAM cannot be changed. Changing the DRAM capacity is achieved in software by allocating a portion of the total DRAM. As a consequence, we are able to measure the influence of the DRAM capacity only on the Microdrive energy, whereas the DRAM energy is bounded by the energy consumption of the whole physical 64MB.

We also adopt this method to evaluate the Flash–DRAM part of FlashBuffer for different DRAM capacities. Here, the Microdrive is replaced by the CF card. The Disk–Flash part, however, cannot be directly measured in our setup. Therefore, we measure for one point where the disk spins up just once to fill the flash with the whole stream and then spins off (i.e., maximizing pre-fetching into the flash). Thus, the Disk–Flash energy measurement comes down to measuring (1) the energy of one read from the disk and (2) one write to the flash of the stream size. We can actually obtain both measurements from the previous Disk–DRAM and Flash–DRAM measurements, respectively. Adding up the Flash–DRAM energy, disk read energy, and flash write energy, we can obtain an upper bound of the total energy consumed by FlashBuffer.

B. *Results*

We implemented a streaming emulator that reads data from the storage device at a given rate into an allocated buffer of a given capacity. We experimented with a typical rate range for PDA-like devices of 32-512 kbps.

For DRAMbuffer we measure the energy consumed by the disk as well as the energy consumed by the rest of the system for different DRAM capacities, ranging from 32KB to 256KB. As the physical DRAM capacity cannot be changed, the energy consumed by the rest of the system virtually does not change (11.9 to 11.8 joules for 32 to 256 KB). The slight difference is due to the difference in the number of refills and thus the incurred transfer overhead. Since the whole DRAM is always on, the energy measured for the rest of the system is in fact the worst case. However, the energy measured for the disk varies from 11.6 to 6.3 joules for 32 to 256 KB: it increases as the DRAM capacity decreases, because the disk is started and stopped more often.

We measure the energy consumption for FlashBuffer for the same range of DRAM capacities. The energy consumed by the rest of the system is virtually the same (12.2 to 12.1 joules for 32 and 256 KB, respectively) as explained before. The energy consumed by the Disk–Flash part is 7.1 joules for 32 through 256 KB, because, unlike the disk, flash has no spin-up energy. Therefore, varying the DRAM capacity has no influence on the energy. In fact, the main contributor to the energy consumed by the Disk–Flash part is the energy to spin-up, seek, read from, and spin-down the disk.

The previous discussion confirms the following:

1) Deploying large buffers for pre-fetching saves significantly on disk energy. This is in agreement with the related work.
2) FlashBuffer consumes about 17% less total energy than DRAMBuffer when deploying 32KB DRAMs in both architectures.
3) FlashBuffer consumes about 5% less total energy than DRAMBuffer when deploying 32KB and 256KB DRAMs in both architectures, respectively.
4) Large number of refills between the flash and the DRAM due to the small deployed DRAM capacities has a minor influence on the total energy consumption.

VII. Wear-Out of Flash Memory

Flash memory has an endurance problem; a flash cell can be rewritten for a fixed number of cycles (100,000–1,000,000 [4]). After that, its reliability to retain data drops and replacement is needed. To extend its lifetime, wear-leveling algorithms map writes to flash in such a way that all cells are rewritten for the same number of cycles.

When used as a FIFO buffer, wear leveling of flash is straightforward by the inherent circular nature of the buffer refill. Nonetheless, streaming applications can extensively use flash, if the size and number of playbacks are reasonably large. Taking these two factors into account, the designer needs to mount enough flash capacity into the system to guarantee an energy-efficient system for a desired lifetime. Thus, spatial redundancy is needed.

The spatial redundancy comes at a cost. This additional setup cost, however, is earned back during the lifetime of the system in the reduced power bill (or lengthened battery lifetime) due to flash. This is particularly true, since electricity prices increase [5] and flash prices decrease.

For example, assume a 24/7 operating mobile multimedia device is used for playing back one-hour–long videos at 2 Mbps, the amount of data streamed for a single video is $2 \times 60 \times 60 \approx 1$ GB, thus for one day 24 GB. Mounting a 256 MB streaming flash of 100,000 cycles allows the server to operate for approximately 3 years before its flash is worn-out. From the simulation of the light workload in Section V, the power dissipation of FlashBuffer is 0.12 W when $\alpha = \frac{256}{4} = 64$. The minimum power dissipation of the DRAMBuffer is 0.21 W when $\alpha = 4$. Thus, FlashBuffer consumes 43% less energy than DRAMBuffer. Assuming that the memory hierarchy consumes 33% of the total energy, the battery lifetime is extended by $\frac{1}{0.43 \times 0.33} - 1 \approx 17\%$.

VIII. Conclusions and Future Work

We propose to use flash as a streaming buffer between the disk and the DRAM in the storage hierarchy. The disk is spun off for long time and the DRAM capacity is reduced, saving significantly on the total energy consumption. Saving of the disk–flash–DRAM architecture compared to the disk–DRAM architecture mainly depends on the number of and form factor of the disk drive(s) as well as the streaming demand. Our simulations show that, for media-server–like applications, the disk–flash–DRAM architecture consumes at least 30% less energy than the disk–DRAM architecture, whereas in multimedia-player–like applications it consumes up to 40% less energy. For audio-like streaming demands, using flash is not worthwhile, however. We partially validate our simulation on a PDA with a Microdrive and a flash card. Our future work will be to implement multi-level buffering in Linux and test with different DRAM and flash capacities as well as various disk drives.

IX. Acknowledgement

We thank the anonymous reviewers for their useful comments. This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs under project number TES.6369.

References

[1] SanDisk CompactFlash cards. `http://www.sandisk.com/Products/Catalog(1020)-SanDisk_CompactFlash_Cards.aspx`. Accessed on August 22, 2007.

[2] Hitachi Global Storage Technologies. OEM Hard Disk specifications for HTC424040F9AT00. Hitachi Travelstar C4K40 1.8-inch drive, November 2003. Revision (7).

[3] Hitachi Global Storage Technologies. OEM Hard Disk specifications for HDS725050KLAT80. Hitachi Deskstar 7K500 3.5-inch drive, September 2006. Version (1.5).

[4] International Technology Roadmap for Semiconductors (ITRS) 2006 update. Technical report, 2006. `http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm`.

[5] G. Basheda, M. W. Chupka, P. Fox-Penner, J. P. Pfeifenberger, and A. Schumacher. Why are electricity prices increasing? Technical report, The Brattle Group, June 2006.

[6] T. Bisson, S. Brandt, and D. Long. NVCache: Increasing the Effectiveness of Disk Spin-Down Algorithms with Caching. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 422–432, September 2006.

[7] J. W. Janzen. *TN-46-03 - Calculating Memory System Power for DDR*. Micron Technology, Inc., October 2003.

[8] T. Kgil and T. Mudge. FlashCache: a NAND flash memory file cache for low power web servers. In *CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 103–112. ACM Press, 2006.

[9] M. G. Khatib, B. J. van der Zwaag, P. H. Hartel, and G. J. M. Smit. Interposing Flash between Disk and DRAM to Save Energy for Streaming Workloads. Technical Report TR-CTIT-07-43, June 2007.

[10] C. R. LaRosa and M. W. Bailey. A docked-aware storage architecture for mobile computing. In *CF '04: Proceedings of the 1st conference on Computing frontiers*, pages 255–262. ACM Press, 2004.

[11] P. Marwedel, L. Wehmeyer, M. Verma, S. Steinke, and U. Helmig. Fast, predictable and low energy memory references through architecture-aware compilation. In *Perspectives Workshop: Design of Systems with Predictable Behaviour*, no. 03471 in Dagstuhl Seminar Proc., pages 4–11, 2003.

[12] Ö. Mesut, B. van den Brink, J. Blijlevens, E. Bos, and G. de Nijs. Hard disk drive power management for multi-stream applications. In *International Workshop on Software Support for Portable Storage (IWSSPS)*, March 2005.

[13] A. Papathanasiou and M. Scott. Energy efficient prefetching and caching. In *USENIX '04 Annual Tech. Conf.*, pages 255–268, June 2004.

Appendix I
The Capacity of the Real-Time Buffer

Let us assume a given storage device with throughput r_{storage}. Let us also assume that every time this device is accessed a latency l is incurred. This latency is due to the storage device itself, the communication buses, and the operating system. To sustain a stream in real time at throughput r_{stream}, a sufficient amount of data should be buffered in advance to account for the access latency. Thus, a minimum buffer of size $B_{\text{rt-storage}}$ should be maintained, and refilled periodically every $T_{\text{rt-storage}}$ time units. The refill period ($T_{\text{rt-storage}}$) is the sum of the incurred latency and the refill time, as follows:

$$T_{\text{rt-storage}} = l + \frac{T_{\text{rt-storage}} \times r_{\text{stream}}}{r_{\text{storage}}}$$
$$\Rightarrow T_{\text{rt-storage}} = \frac{l \times r_{\text{storage}}}{r_{\text{storage}} - r_{\text{stream}}}$$

where $r_{\text{storage}} > r_{\text{stream}}$.

The minimum corresponding real-time buffer capacity ($B_{\text{rt-storage}}$) of that storage device is:

$$B_{\text{rt-storage}} = T_{\text{rt-storage}} \times r_{\text{stream}} \tag{5}$$