Karlsruher Institut für Technologie

DM-Relay - Safe Laptop Mode via Linux Device Mapper

Study Thesis
by

cand. inform. Fabian Franz

at the Faculty of Informatics

Supervisor:	Prof. Dr. Frank Bellosa
Supervising Research Assistant:	Dipl.-Inform. Konrad Miller

Day of completion: 04/05/2010

I hereby declare that this thesis is my own original work which I created without illegitimate help by others, that I have not used any other sources or resources than the ones indicated and that due acknowledgment is given where reference is made to the work of others.

Karlsruhe, April 5th, 2010

Contents

List of Figures

List of Tables

Deutsche Zusammenfassung

Das Thema der vorliegenden Studienarbeit ist Festplatten Power Management. Ein signifikanter Energieverbraucher in einem modernen Laptop ist weiterhin die Festplatte. Diese Arbeit zeigt einen neuen Ansatz auf, wie der Energieverbrauch einer Festplatte gesenkt werden kann um die Akkulaufzeit zu verlängern.

Um Strom zu sparen kann die Festplatte in einen Schlafmodus (standby) versetzt werden, sobald ihre Dienste für den Moment nicht mehr benötigt werden. Die Problematik besteht nun darin, dass nicht bekannt ist, wann dies der Fall ist und ob es sich also lohnt die Festplatte schlafen zu legen. Denn wenn die Festplatte zu früh schlafen gelegt wird, kann dies sogar zu Stromverschwendung führen. Zudem hat jede Festplatte nur eine bestimmte Lebensdauer an Anfahrzyklen (50000-600000).

In dieser Arbeit werden zuerst die bereits vorhandenen Lösungen für diesen Problembereich vorgestellt, analysiert und in zwei Kategorien eingeteilt. Die Ansätze der ersten Kategorie beschäftigen sich vor allem mit der Frage, wann die Festplatte schlafen gelegt werden sollte – dafür wird versucht das Systemverhalten möglichst gut vorherzusagen.

Die zweite Möglichkeit Strom zu sparen, basiert darauf zu verhindern, dass die Festplatte wieder aus dem Schlafmodus aufgeweckt wird. Für Lesezugriffe gibt es hierbei eine Vielzahl an gut funktionierenden Lösungen, die auch in den meisten Betriebssystemen implementiert sind (Caching, Read-Ahead, ...). Für Schreibzugriffe existiert dagegen nur die Lösung die Schreibzugriffe zu verzögern. Durch die Verzögerung können im Falle eines Systemcrashs Daten verloren gehen. Beim Linux Laptop Mode – einer in der Praxis eingesetzten Lösung um Energie auf Laptops zu sparen – sind dies sogar ganze 10 Minuten Arbeit, die verlorengehen können.

Deshalb ist ein neuer Ansatz enstanden, diese Daten nicht vergänglich im Arbeitsspeicher zu verzögern, sondern auf einen nicht-flüchtigen Datenträger zu schreiben, wie z.B. einen USB Stick. Allerdings benötigen die vorhandenen Lösungen

entweder spezielle Hardware oder erfordern so große Eingriffe ins Betriebssystem, so dass sie bisher keine breite Verwendung gefunden haben.

Der Beitrag dieser Arbeit schließt diese Lücke: Es wird ein neues Design vorgestellt, welches sich auf Block-Layer Ebene vor dem Festplattentreiber befindet. Diese Lösung kann als ganz normales ladbares Modul eingebunden werden und benötigt damit keinerlei tiefergehende Eingriffe ins System. Es kann auch im laufenden Betrieb eingebunden werden, da das Linux Device-Mapper Framework benutzt wird.

Im Anschluss werden die Probleme, die bei diesem Ansatz auftreten, diskutiert und Lösungen vorgeschlagen. Auf der Basis dieses Designs wird dann eine konkrete Implementierung vorgestellt.

Um die Ergebnisse zu validieren wird der Stromverbrauch quantitativ gemessen. Es kann gezeigt werden, dass im Vergleich zu Oracle (einer häufig verwendeten Basis) die Lösung in einem schlechten Fall nur 3% schlechter ist und in einem günstigen Fall sogar 12% besser. In jedem Fall aber benötigt Safe Laptop Mode viel weniger Anfahrzyklen als Oracle, was zu einer deutlich verlängerten Festplattenlebenszeit führt.

Safe Laptop Mode verbraucht im schlechtesten Fall 30% mehr Strom als der normale Linux Laptop Mode – und 10% im Besten. Dies liegt daran, dass der USB Stromverbauch auch im Idle Modus noch bei 0,1 Watt liegt und es derzeit nur theoretisch möglich ist den USB Stick in einen Standby-Modus zu versetzen.

Außerdem zeigt die Auswertung der Benchmarks, dass weder Leistung, noch I/O-Durchsatz, noch CPU-Overhead oder Speicherverbrauch gegenüber dem normalen Laptop Mode beinträchtigt werden.

Insgesamt gesehen erreicht die Arbeit ihr Ziel einen transparenten und einfach in bestehende Systeme integrierbaren Mechanismus bereit zu stellen, der es ermöglicht, Energie zu sparen, indem Schreibzugriffe verzögert werden ohne dabei aber Datenverlust zu riskieren.

1. Introduction

In laptops a significant energy consumer is the hard disk. While there are many solutions available to save hard disk energy, the most common solutions like Linux laptop mode draw energy consumption against data safety. This might lead to data loss and is unnecessary, because by utilizing some space on an USB flash drive, the risk of data loss can be minimized.

1.1 Problem Definition

Unfortunately so far all available solutions either need specialized hardware or extensive kernel changes to function and/or are only available on paper. So far the only solution that has been officially integrated into the Linux kernel is the so called laptop-mode and in the default setup it is possible to loose up to 10 minutes of work. Even though there has been lots of research in the field of hard disk power management, no other solution has reached the consumer market yet.

1.2 Objectives

The objective of this work is to change that and provide a transparent energy-saving solution inside of the block-layer, which can be integrated into consumer systems with ease. The goal is to make using the solution so easy that it is as complex as loading a kernel module or configuring a LVM module. And that this solution can be as easy as using the laptop mode now. The idea of the proposed solution is to extend the laptop mode with additional functionality – not to replace it.

1.3 Methodology

To show that it is possible to save energy by using the approach outlined in this study thesis, desktop traces have been recorded, replayed and evaluated. Also the power consumption on a live system utilizing the solution was measured with promising results. The results have been examined quantitatively and besides energy also the influence on performance and memory consumption has been tested.

1.4 Contribution

The contribution of this work is a new approach to saving energy on a modern laptop or desktop system by using a non-volatile flash memory. New about this approach is that the core of the kernel does not need to be changed and that energy saving policies can be implemented in user space. Also new about this approach is that the proposed layer is completely transparent when inactive and does not influence system behavior. With this said the solution is mainly an extension of the already well working Linux laptop mode by minimizing its hugest drawback – the possible loss of data.

1.5 Thesis Outline

This study thesis is first taking an extensive look at the state of the art in the field of hard disk power management. Then the pro and contra of the outlined approaches are discussed. Based on this analysis a new solution is presented. The solution is first introduced as a high level design, then a concrete implementation is discussed. Last, the solution is validated by evaluating quantitatively how much energy can be saved compared to the Linux laptop mode and to normal system behavior. At the end a conclusion is drawn and further work outlined that could be done to allow further energy savings.

2. Background

There are many solutions for the problem of extending the battery life of laptops. This study work concentrates on disk power management.

This chapter first introduces the problems related to disk power management, then the field is divided into categories and after wards the state of the art for each of these categories is presented.

2.1 Problems of Disk Power Management

In laptops a significant energy consumer is the hard disk. In idle mode a typical 2.5" hard drive usually requires between 0.5 W and 1.3 W, and from 2 W to 4 W under load [ScRo08]. However in standby the consumption drops to 0.1-0.2 W. In most power managed scenarios the hard disk uses between 3 and 15% of the total power consumption of a laptop [MaVa05]. This number can increase to up to 30% for desktop systems [Gree94] with an average of 8-15% for modern desktop systems [BiJo07].

Significant *energy-savings* are possible by putting the disk into a low power mode ("spinning down the disk") instead of running it in idle mode. However a number of parameters need to be taken into consideration before power management algorithms can be applied. On of those factors is the break-even time [LCSB+00]. The break-even time is the time a disk needs to remain in a low power state until energy is saved at all. The formula is:

$$t_{BE} = \frac{P_{spindown} - e_{standby}(t_{spindown} + t_{spinup}) + P_{spinup}}{e_{idle} - e_{standby}} \tag{2.1}$$

Additional information about the break-even time is provided in figure 2.1.

Another important factor is the *life-time* of a hard disk. While laptop hard disks have a typical *life-time* of 600'000 spin ups and spin downs, most desktop and

$t_{BE} = \frac{P_{spindown} - e_{standby}(t_{spindown} + t_{spinup}) + P_{spinup}}{e_{idle} - e_{standby}}$

where

t_{BE} = Break-Even Time; time after which energy is saved at all
$P_{spindown}$ = energy needed to spin down the disk in Joule
P_{spinup} = energy needed to spin up the disk in Joule
$t_{spindown}$ = time needed to spin down the drive
t_{spinup} = time needed to spin up the drive
$e_{standby}$ = average power consumption in standby
e_{idle} = average power consumption in idle mode

Figure 2.1: Break-Even Time: time after which energy is saved at all

server hard disks have much lower values (200 spin downs per year; 50'000 cycles completely). So for example the Linux laptop mode warns to not enable the laptop mode on a desktop system as the hard disk life-time is heavily decreased by such aggressive power management. [Gree94, Samw04b]

2.2 State of the Art

In general energy saving policies can be implemented at several layers: In the application layer, file system or block layer. There is a loss of information from application to block layer and vice versa. While the application has all information about the data written and read, but almost none about the disk state, the reverse is true for the block layer. The block layer has all information about disk state, but no information about the requests arriving and who is responsible for them.

Four cases of approaches can be distinguished for saving energy by spinning down the disk:

- Predicting request timing (and such predicting sleep time of the disk) without influencing it
- Influencing request timing on any of the three layers (and such influencing sleep time)
- Giving more information to other layers to enable better prediction and influence
- Or a combination thereof

2.2.1 Predictive Algorithms

Normal applications are not aware of the state of the disk layer, but instead rely on it as just being there. They are also unaware of the disk's energy consumption.

The disk layer needs to analyze behavior of the applications to predict if the time between incoming requests will be larger than the break-even time (see figure 2.1). As a baseline often the so called oracle policy is used, which gives optimal energy savings when request timing can only be predicted but not influenced. Oracle is an imaginary device, which does know the timing of all request that will come in the future. Oracle can be simulated by replaying traces and using the following policy: When the time between the current and the next request is greater or equal to the break-even time, then the disk is immediately spun down.

Another often used algorithm – twice as bad as oracle in the worst case – is called DDT (Device Dependent Timeout). In this policy the disk is spun down after the break-even time has passed, because from this moment on energy can be saved.

2.2.2 Cooperative I/O

However even oracle can be surpassed, when application behavior is changed rather than only predicted. Weissel et al. show in [WeBB02] that with just little modifications to the I/O syscall interface of existing programs further energy-savings are possible despite many unmodified applications being run at the same time.

> With Coop-I/O, applications can declare open, read and write operations as deferrable and even abortable by specifying a time-out and a cancel flag. This information enables the operating system to delay and batch requests so that the number of power mode switches is reduced and the device can be kept longer in a low-power mode. [WeBB02]

2.2.3 Energy Aware File Systems

I/O Requests can also be influenced efficiently in the file system layer. Here two approaches need to be distinguished. File systems, which are aligned to the specific needs of hard disks and file systems optimizing its own behavior by using the knowledge available.

Sherl shows in [Sche] that for mobile and embedded systems the file system can be optimized using a log structured approach to save energy by minimizing seek and rotational latencies, which is especially important if no caches are available.

The Linux laptop mode [Samw04b] on the other hand configures the file system in a way so that non-crucial write requests are omitted: For example by removing writing of access times.

A file system could also prioritize unimportant writes lower than important writes. For example this imaginary file system could hold back access time data as well as volatile data to /tmp or log files on a desktop and not spin up the hard disk for this type of requests, while it would write "user data" directly to the disk. This file system would weight possible data loss of unimportant data against energy consumption. Background processes would not spin up the disk unnecessarily.

2.2.4 Request-Reordering, Read-Ahead

Another technique used in most operating systems is request reordering on the block layer, to minimize rotational and seek delays. As the operating system nowadays does not know the exact layout of the disk, most modern disks support a mode called NCQ (Native Command Queuing), which does exactly that. Request-Reordering does improve performance and can also save energy. However this is only possible for read requests as reordering write requests could lead to a inconsistent system state after a crash.

With the hypothesis that applications are reading data mostly sequentially, but only one chunk at a time, Read-Ahead was developed. By reading more data than requested (pre-caching), further read requests from applications can be satisfied from the read-ahead buffer without actually reading the block-device again. This prolongs idle periods and can conserve energy.

2.2.5 Read-Caching

Caching of requests in general is also a good method to save energy and increase performance.

Caching depends on the idea that things read once are often read a second time and studies show that read-caches are efficient and that many read requests can be satisfied directly from the cache. Saving disk accesses naturally prolongs idle periods, energy is saved.

2.2.6 Write-Delaying

While read requests in most desktop scenarios are not a problem as caching is very efficient – write requests are.

On a Linux system using ext3 as a file system for example data is committed every five seconds. The disk and most of the other file systems have a similar flushing behavior. After data is older than 30 seconds in Linux, it is flushed to disk. This process is called kflushd and runs every 5 seconds. This results in an enduring disk activity of periodic writes, which is completely counter-productive to power management.

The reason for such aggressive flushing is the prevention of data loss. In case of a system crash or power outage all delayed write requests would be lost. On the other hand it would make no sense to directly write all data through as otherwise read requests could be delayed unnecessarily and writing data in a burst gives smooth I/O performance.

For Linux there are several implementations of the write delay method. The most popular are Linux laptop mode and noflushd. Both do configure certain system parameters to be able to spin down the disk for a longer period of time: For example to prolong the already mentioned commit intervals. However in both cases the possibility of data loss is traded for the energy savings.

To mitigate these effects, all data is written back, when the disk becomes active again. Also the 'sync' or 'fsync' syscalls are recognized and requests are flushed, when those methods are called. [Samw04b, Samw04a]

2.2.7 Write-Caching

Another possibility is to not only delay the writes, but to (also) cache them on non-volatile storage and read them back from this storage on disk spin up.

Most write-caching solutions fall in one of two categories: Either they are based on special hardware or are only software based.

The first cache using flash memory was already described in 1994, when a 256 MB (!) flash space did still cost 12’000 $ [MaDK94]. And already then was a flash cache deemed a suitable way to save energy and prolong idle periods.

> We find that a FlashCache can reduce the power consumption of the storage subsystem by 20-40% and improve overall response time by 30-70% when combined with an aggressive disk management policy. When combined with a more conservative policy, power is reduced from 40-70% while overall re-sponse time is improved 20-60%. We also find that durability is not a problem; a 4 MB FlashCache will last 33 years. [MaDK94]

Here the flash cache is used as a normal cache layer between hard disk and main memory.

Eleven years later Bisson and Brandt show in [BiBr05] that this approach is not only still feasible, but can "outperform the most powerful disk spin-down algorithms":

> 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 [BiBr05].

They also show that this approach is necessary on unchanged operating systems due to the previously discussed periodic disk activity. They do use the flash memory just as a write-only cache.

With Smartsaver discussed in [ChJZ06] Chen et al. use the flash-cache also for caching of selected read-requests and pre-caching and also provide a comprehensive flash memory management. Their results look promising as well:

> Trace-driven simulations show that up to 41% of disk energy can be saved with a relatively small amount of data written to the flash drive [ChJZ06].

In 2008 Matthews et al. presented in [MTHC+08] a first practical application of a flash cache approach: The Intel® Turbo Memory technology.

> Intel Turbo Memory [...] by adding a new layer to the storage hierarchy: a platform-based and nonvolatile, disk cache [MTHC+08].

This solution is a very hardware centric solution based around a NAND-Cache and cache controllers by Intel.

2.2.8 Write-Offloading

A variant of this technique was tackled by Microsoft Research in [NaDR08] for enterprise servers. In this technique the requests to inactive disks are not cached on some flash media, but rather distributed to other disks that are in active state at the time of the request. On a read cache miss the data are read back from the media where it was stored to and the data are also synchronized once a disk goes into active mode again.

2.2.9 Hybrid Hard Disks

A small flash cache (NVCache) is now included on most modern hard disks. Those are called HHDs (hybrid hard disks).

But this flash cache is mainly used for pinning data permanently into the hard drive so that system applications can be spawned immediately. Besides it can also be used to cache writes and such avoid spinning up the disk. Hereby the logic for spinning down the disk and pinning writes is configured to be drive specific, though it could be handled also by the operating system.

Bisson et al. show in [BiBL07] four methods of improving I/O behavior to save more energy for HHDs.

2.2.10 Changing Hard Drive Speeds

Another possibility to save energy is to build hard disks with varying rotating media speeds. DRPM [GSKF03] is the most cited approach for this idea, however most modern hard drives still do not offer these multi-speed operating modes, which limits the application of this approach. [GSKF03, HSRJ08]

2.3 Summary of this chapter

In this chapter the problems surrounding disk management were discussed. Then the approaches for solving this problem were classified into four categories: Either the timing of the requests is estimated or the behavior of applications, file system or block layer is changed to get a better usage of resources or information (knowledge) is provided from one layer to another to get better prediction or influence possibilities. Or a combination thereof.

In the last part the state of the art regarding power management for disks was presented. While read requests are nowadays handled quite well with caches, most work now concentrates on avoid spinning up the disk due to write requests. The write requests are either delayed (with the possibility of data loss) or written so some temporary storage internal or external to the hard disk and computer system. With the upcoming of solid state disks a flash cache was included into the hard disk, but still much more memory can be saved if other factors are taken into account, then those which are known by the drive.

3. Analysis

In this chapter different properties of approaches introduced in the previous chapter are presented. It is shown that there is still room for improvement leading to the proposition of a new system design. One important factor analyzed is if the approaches are used in practice today and if yes how widely adopted they are.

3.1 Pro and Contra

Table 3.1 shows an overview of the different approaches and its characteristics. The advantages and disadvantages of these approaches are discussed below in more detail, but the table can be used as a quick reference.

3.1.1 Oracle/DDT

The advantages of oracle and other prediction based algorithms like DDT (device dependent timeout) are that they need no additional hardware and are already in widespread use and such do save energy today. However oracle is only optimal in cases where the timing of requests cannot be influenced. Another disadvantage is that oracle and DDT are spinning the disk down far too often for the average life-time of a disk. In the worst case DDT has a timeout of 20 seconds and Linux is writing (meta)data every 30 seconds, which leads to 120 spin-downs per hour, which is wearing the disk out very fast. The effect of this behavior (using oracle policy) can be seen clearly in figure 6.2 in chapter 6.

DDT is also by no means as optimal as oracle, which only works offline. In fact in the worst case spinning down the disk after the break-even time has passed is twice as bad as oracle and leads to even worse spin-down/spin-up behavior. (A too aggressive spin-down policy in the Ubuntu operating system has lead to complaints of users of failing hard disks and "strange" system behavior in 2007.)

Approach	Layer	without add. HW	data safe	implemented and tested	without add. NV-Storage
Oracle	disk/block	yes	yes	no	yes
DDT	disk/block	yes	yes	yes	yes
COOP-IO	fs/app/block	yes	yes	no	yes
EAFS	fs	yes	partial	yes	yes
Req.-Reord.	block	yes	yes	yes	yes
Read-Ahead	block	yes	yes	yes	yes
Read-Caching	block	yes	yes	yes	yes
Write-Delay	block	yes	no	yes	yes
Laptop-Mode	block/fs	yes	no	yes	yes
NV-Cache	block	yes	yes	?	no
Smart-Saver	block	yes	yes	no	no
ITM	disk/block	no	yes	yes	no
WR-Off-Load.	block	yes	yes	no	no
HHDs	disk/block	no	yes	yes	no

Table 3.1: Comparison of different approaches to save power consumption

3.1.2 COOP-IO

COOP-IO has the advantage that it is directly suited to an applications block-level needs. So it can save energy by optimizing the application behavior, which leads to more knowledge on the file system and block-layers, which then can schedule the I/O more efficiently. The disadvantage is that COOP-IO needs applications and APIs to change and cooperate. This has not happened so far and so the savings unfortunately are still of a theoretical nature.

3.1.3 Energy Aware File Systems

There are two types of energy aware file systems as stated in 2.2.3: file systems optimized for embedded systems with low memory and no caching technology available and file systems that make use of their knowledge. As this work assumes a standard laptop or desktop system, which typically have between 512 MB and 4 GB RAM today, only the second type is analyzed.

The advantage of energy aware file systems is that they would have the necessary data to do some classification and only send the needed requests to the block layer and omitting things like meta-data or data written to /tmp or any other volatile data. This would save energy by prolonging idle periods.

The disadvantages are that this model is theoretical for now and only a certain amount of knowledge is available: Neither the state of the block layer nor the state of the application layer is known. But the biggest disadvantage is that creating a main-stream file system for the sole purpose of saving energy is completely unrealistic as file systems are a very critical part of the operating system and do need lots of testing and verifying before they are considered stable. File systems normally need years to mature and also need to take many factors into account to be a general purpose file system as there are lots of different interests that a file system has to satisfy.

Despite this lack of a direct energy saving method, file systems can be configured to not write meta data and also to commit data less often. And a pure volatile /tmp can be emulated by mounting a small ramdisk to /tmp. However here is still optimization potential by for example flagging data with a priority bit on a VFS layer, so that the block layer can optimize based on that priority.

While optimizing file systems itself is unpractical as stated above, the layers above and below could be optimized to be energy-aware.

3.1.4 Request-Reordering, Read-Ahead

The advantages of request-reordering and read-ahead is that they do improve I/O performance by minimizing rotational and seek delays and also predicting data that most likely will be requested later on. While these techniques have been mainly developed to make I/O faster and more efficient. This leads to higher throughout and lower latency and as a side-effect also saves energy.

The disadvantages are that Read-Ahead can also populate the cache with data never read as it can only predict an applications behavior. And request reordering can only be used in case of reads as reordering writes could lead to an inconsistent state on a crash.

3.1.5 Read-Caching

Read-caching has the advantage that data read once or twice is often read a second time or third time as well. This is especially true for libraries and program files. Linux has a read-twice, cache more policy, which is giving quite good results in practice.

The disadvantages of read-caching are that it needs system memory to work and that the cache can be invalidated easily by reading one huge file.

3.1.6 Write-Delay

Write-Delaying has several advantages: By not writing all data directly in one synchronous operation to the disk, requests can be scheduled more efficiently and so data written in a burst is not interfering with read requests. Also data that is overwritten can be directly updated in memory before it is written again and again to the disk.

Most operating systems do write the data only periodically to the disk (Linux 5-35 seconds) to flatten the request queue. One disadvantage of write-delaying is the possibility of data loss. To accommodate this most operating systems have a SYNC flag to synchronously write data to disk.

Another disadvantage is that the default write-back delay is too short to allow long sleeping times and as already discussed can lead to the worse-case scenario of the cycle of 20s wait, 10s sleep, wake-up, 20s wait by using the popular DDT disk shutdown algorithm.

3.1.7 Linux Laptop Mode

Linux Laptop Mode with Laptop Mode Tools prolongs the write-delay to up to 10 min and also configures file system parameters not to write meta data like access times. Also – and this is the original laptop mode flag – all data is written back to disk shortly after a read occurs (which does wake-up the disk definitely).

The advantages of this approach are that the disk can sleep longer and as the write-requests are written at the end of a read-request also that data is written as soon as the disk is awake.

The disadvantages are that up to 10 min of data can be lost by utilizing this method, which is unacceptable in some cases. Also if the default configuration is used the disk cannot sleep longer than the configured 10 min even though there is low write traffic with only unimportant data.

3.1.8 Non-Volatile Write-Cache

To solve the problem of occurring data loss a write-cache on a non-volatile storage media can be used.

The advantages of using a non-volatile write-cache are that the data can be saved temporarily or persistent to this storage and the disk can be kept spun down. Also non-volatile RAM is more energy efficient as normal memory as it needs no refresh. However as the non-volatile memory is limited in size and still much more expensive than normal RAM, there need to be policies when to flush and when to write to the cache. In practice the non-volatile memory is also often the only memory, where the data is kept. On write-back all data needs to be first read back into main memory and then written to disk, which has the disadvantage of a small read-back delay.

Intel Turbo Memory (ITM)

The advantages of Intel Turbo Memory are that – as it is integrated into the hardware – there is a high transparency and neither kernel nor applications need to be changed much. Only a small layer in the disk controller is needed to enable the functionality. Also data can be written directly from the cache to the disk and vice-versa.

However as it is a hardware only solution usage is limited to platforms supporting the hardware and additional investment needs to be done to save energy. Also Intel turbo memory also does read caching, which helps as NVRAM is more energy efficient than ram.

Reads are already cached very well by the system itself, so this might not really be needed. However there is one case where the read cache is especially useful. Because at system start up data necessary for the system start up can be read both from the hard disk and from the turbo memory, which increases system start up times. (An example, where this is used in practice is Microsoft's ReadyBoost technology.)

Smart-Saver

The other solution is Smartsaver, which only exists on paper so far. They did an extensive study with a disk simulator to show that by using NVRAM they can save memory. The advantages of Smartsaver are that by using a NV-Cache disk sleep times can be prolonged. The disadvantage of Smartsaver is that according to the paper they need to make extensive changes in the Linux kernel, in the disk drivers and additionally they need to add a complete new policy for writing back blocks.

Write Off-Loading

Another possibility to delay the write requests is to off-load the write requests to other disks that are not sleeping. A challenge hereby is a naturally occurring consistency problem and in case that the data is written to the network, also a small read latency.

An example for that is Write-Off-Loading by Microsoft Research. This technique however is utilized best in enterprise data centers, where lots of data storage is available. Write Off-loading is not that relevant for desktop computer systems as those often just have one or two hard disks and might not have a persistent network connection in case of laptop computers.

3.1.9 Hybrid Hard Disks

Another important field of research is the usage of hybrid hard disks. HHDs do have a small flash cache integrated on the disk. Advantages are that writes can be cached and the hardware can also directly decide to write back cache data. The disadvantages are that there is a custom cache write/invalidate protocol, which means there is always a trade off between disk managed cache and operating system managed cache. This also leads to further overhead.

At the moment the flash cache is also not yet used in the ideal way, as discussed in [BiBL07]. Also HHDs are not yet in widespread use, especially not in laptop computers.

3.2 A new approach

In conclusion there are two main things to do to save power:

First detect patterns and algorithms when to put the disk to sleep. This work only briefly uses techniques from this part to decide if data should be passed through or held back.

Second prevent usage of the block device unless absolutely necessary. This work introduces a mechanism to hold back all writes until the disk is active or the non-volatile request log is full, while preserving the data on a non-volatile storage.

As the analysis shows the existing solutions for using non-volatile storage for writing data are either very hardware centric or need huge changes to the operating systems and such are not yet in widespread usage. However the solutions using only Write-Delay with a volatile cache are working very well in several implementations, but do fail in terms of data loss.

Approach	Layer	without add. HW	data safe	implemented and tested	without add. NV-Storage
Safe LM	block	yes	yes	yes	no

Table 3.2: The new approach safe laptop mode satisfies all requirements

This work closes this gap as it provides a write delay solution, which is robust through using non-volatile storage as backup, but on the other hand needs no extensive hardware or kernel changes (see table 3.2). In fact as can be seen in the implementation chapter (5) only a kernel module is needed to load and the solution could also be hot plugged into a running system using the device mapper framework on Linux.

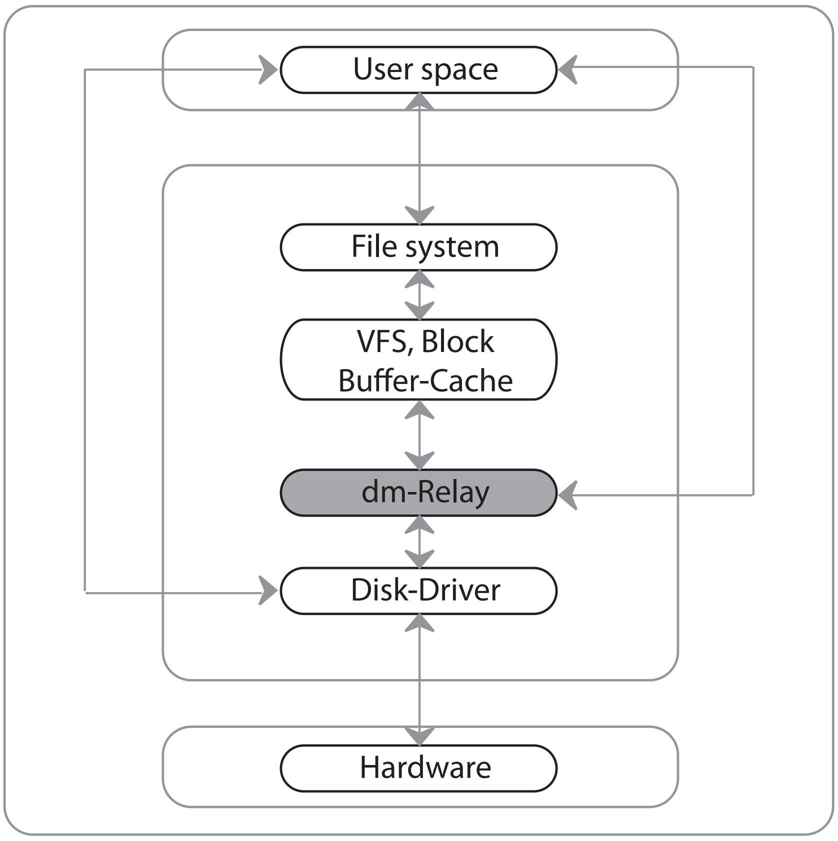

Figure 3.1: The proposed solution is positioned between the buffer cache and the disk driver

So I propose to insert a solution called dm-relay inside of the block layer between the file system and disk layers (see figure 3.1).

This layer has three tasks:

- Pass through all reads (possibly flush log before)
- Delay all writes and keep a persistent log of requests
- Write through when disk is active (possibly flush log before)

This process is outlined in figure 3.2.

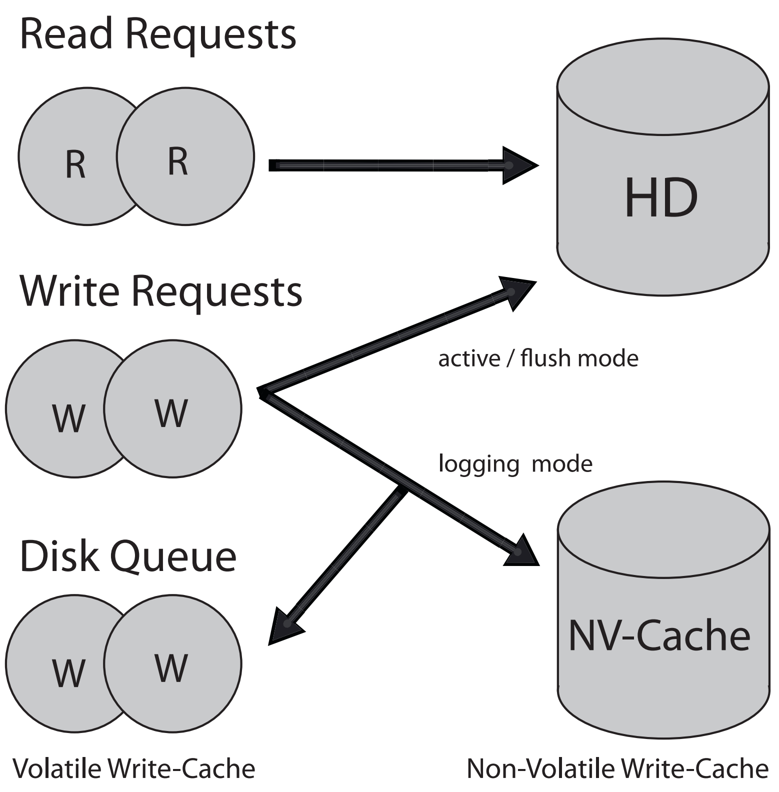

Figure 3.2: Read-Requests are passed through. In active or flush mode Write-Requests are written directly to disk. In logging mode Write-Requests are split and written to a volatile memory queue and a non-volatile cache.

3.3 Analysis of Proposal

The read requests can be passed through directly as most of the reads are cached and adding another cache layer would not significantly improve the cache-hit rate. Read requests might trigger a log flush if the region of the requested data is "dirty". However written data is often also cached by the operating system and such this can be handled like a normal read-miss, which would also trigger a write back of the log after the disk is active. So this is no real disadvantage.

Write requests are delayed until the disk is active again, but a persistent backup of the requests is stored additionally on a non-volatile medium. In case that the log fills up the disk is spun up and the delayed write requests are written to the disk. In this case there will be a small latency before new requests can be processed. However with an assumed log size of 512 MB delayed write requests and a hard disk speed of 70 MB/s, the worst case is only 10 seconds (inclusive a possible disk spin-up time) and this would also only be noticeable by the user in case of a "dirty" region read or a synchronous write operation.

As can be seen in figure 3.3, whenever delayed write requests have been flushed to disk, the log is reset as well.

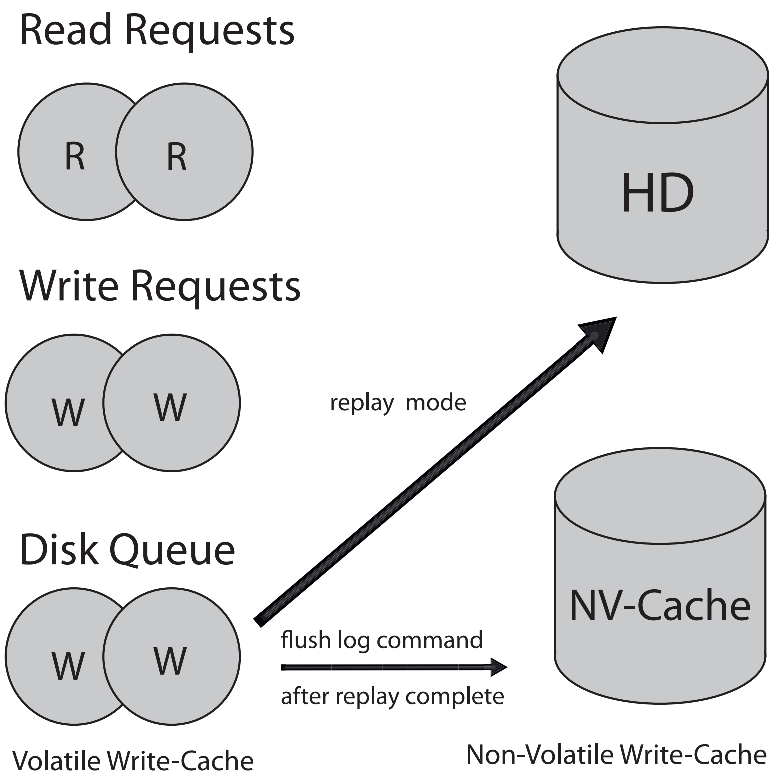

Figure 3.3: While the log is replayed no new read or write requests are processed. Once the replay is complete the log is reset.

In case that the system crashes or a power outage occurs, the initial state is restored by reloading the log into memory. While the log is re-read all other requests are blocked. After log and main memory are synchronized again and the data was written successfully through, the log is reset (see figure 3.4). This can lead to a small recovery "delay" before the system starts up.

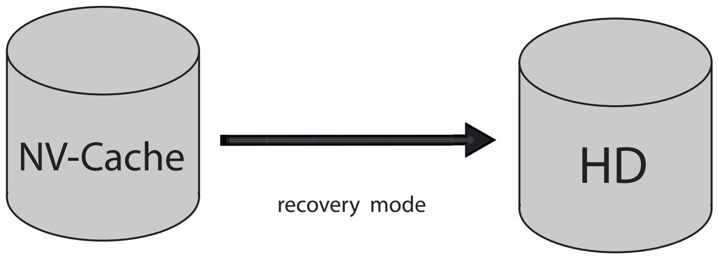

Figure 3.4: In recovery mode the log is replayed from the non-volatile storage to the hard disk.

This whole approach has several advantages:

- No extensive changes to hardware or kernel
- Easy to implement and use
- Does one job and does it well

- Data is at minimum as safe as the log media, in normal cases much safer (as the system does not crash every time)
- Log can be saved to even safer media (raid1, drbd) to increase reliability
- Very fast, there is only a small delay in case of log replay
- Log can be everywhere (USB flash drive, SD Card, SSD Flash, Flash, Network, other active Hard disk, ...)
- Extensible (Log can be exchanged)
- Completely transparently passes through the data in active mode
- Optionally provides user space access to mechanism (flush, active, standby/logging modes)
- Possible to implement different energy-saving policies

The disadvantages are:

- internal caching leads to doubling of data (once in memory and once on non-volatile media)
- uses more memory than what would actually be needed when the data was replayed always from cache
- needs additional non-volatile storage, which also consumes energy
- flash storage can wear out by such usage (however its possible to optimize for this case)
- The storage must not be removed, while the system is running/after an unclean system shutdown, as this could lead to inconsistencies or data loss

However the simplicity of the solution far out weights these disadvantages.

3.4 Summary of this chapter

In this chapter the pro and contra of the currently existing approaches were discussed. Based on this analysis of the existing solution space a new approach was introduced. This approach can significantly reduce power consumption (as shown in chapter 6) by delaying all write requests while keeping a permanent backup on non-volatile storage. The proposal was then analyzed and advantages weighted against disadvantages.

4. Design

This chapter describes a possible system design. Common problems that do occur by choosing this path for the solution are discussed and solutions presented to solve them.

On a high level the design needs to only solve the following tasks (also compare figure 3.2):

Pass through all data

In this mode (active) the solution is completely transparent and just passes through the data from input to output.

Delay write requests, but write a backup to non-volatile storage

In this mode (standby) the solution delays write requests until the disk is active again. Additionally a backup is written in the form of a log structure to the non-volatile storage.

Allow configuration of the mode used

An external (user space) program can set the mode (active/standby) that should be used and so start or stop the logging part of the solution. This could be done for example in anticipation of a disk going to sleep now.

4.1 Common problems

However a number of common problems can occur in these tasks, that are discussed below.

4.1.1 Handling of data consistency

Because data is not written directly to the disk, but to a second medium a number of data inconsistencies can occur. Reads cannot be passed through to the disk when the requested region has been written to since the last flush. A dirty log can be used to detect dirty regions and flush the log and delay reads till the flush is completed. To make sure that data written to the log is not updated on the disk a strict mode setting is used. Data is only passed through to the disk, when the log is clean. Otherwise first all requests from the disk-queue are processed and then the log is reset.

4.1.2 Handling of synchronous write requests

Write-Requests cannot just be delayed, because then synchronous write requests would be delayed until the next log flush. Instead the write requests have to be submitted to the log and just a clone of the write request has to be written into the disk-queue. The synchronous operation does end then, when the data was written successfully to the log.

4.1.3 Handling of log overflow

In case that the log overflows, all operations have to be stopped and the log has to be flushed to disk. All pending data has to be delayed until the flush is complete.

4.1.4 Handling of system crash

In case of a system crash or power outage there are two possible states that the log can be in: clean and dirty. In case that the log is clean the system can start up normally. Otherwise the log needs to be first replayed and just then can new read or write requests be processed.

4.1.5 Handling of removal of log media

As an USB flash drive can be removed easily the solution needs to know when a log is removed or inserted again. Hot-Plug of the log media is currently not handled.

However a possible solution is to generate an UUID to a file (possibly using a user space helper) as soon as the disk gets active in a synchronous operation after the log was flushed to disk. The log is then reset with this UUID.

On a system crash the recovery program would then mount the disk read/only, read the UUID and compare it with the log. If it does not match the log is not replayed and the user alerted.

When the log medium is removed, the solution could either fallback to a delay-only mode or not allow operation in standby mode at all until the log media is inserted again in which case it would be reset.

4.1.6 Data-Rate is faster than log media

When the data rate of incoming write requests is faster than the log media, synchronous writes will be slower than expected by the user and also data safety can no longer be guaranteed as the log lags behind.

To solve this a simple rate limit filter is needed, which is triggering a log flush when the amount of not processed write requests is greater than some threshold.

4.1.7 Un-smooth system behavior

The last problem is that flushing of the log by any of the above methods could always lead to a pattern of flush-logging-flush-logging, which should be avoided. To solve this a timer can be utilized, which is delaying the flush signal by some amount of time. while the flush signal is active and the log state is clean data is directly passed through to the disk without touching the log.

Additionally this timer can be reset as long as a certain amount of data is still arriving. Before the flush signal is set to off and the log is used again, it is also a good idea to write all dirty buffers to disk first.

4.2 System-Design

To solve all the outlined problems the proposed design consists of several sub tasks:

- Incoming request function - Classifies the requests and passes them through a rate-limit filter
- Background and processing task - Chooses a mode based on power state and either passes through the request, writes data to the log, or writes data back to disk
- Processed request function - Passes the processed requests through a rate limit filter
- Timer function - Allows "smoothing" of write periods
- User space Message function - Allows configuration of power state
- User space Helper programs - Recovers the log or sets the power state

4.3 Summary of this chapter

This chapter showed a possible system design. Common problems have been discussed and algorithms proposed to solve them. In the last part the system-design was proposed with several high-level sub tasks, which are explained in finer detail in the next chapter.

5. Implementation of a dm-module for the Linux kernel

This chapter shows how the proposed design was implemented. It starts with a high level system-architecture overview, then gives some insights into the characteristics of flash memory and how a log can utilize those and it finishes with an insight into how the solution can be utilized in practice.

The example implementation was done for the Linux kernel based on the device mapper framework as that allowed the most modular approach. During the development I also wrote block level trace and replay utilities and a log analyzer.

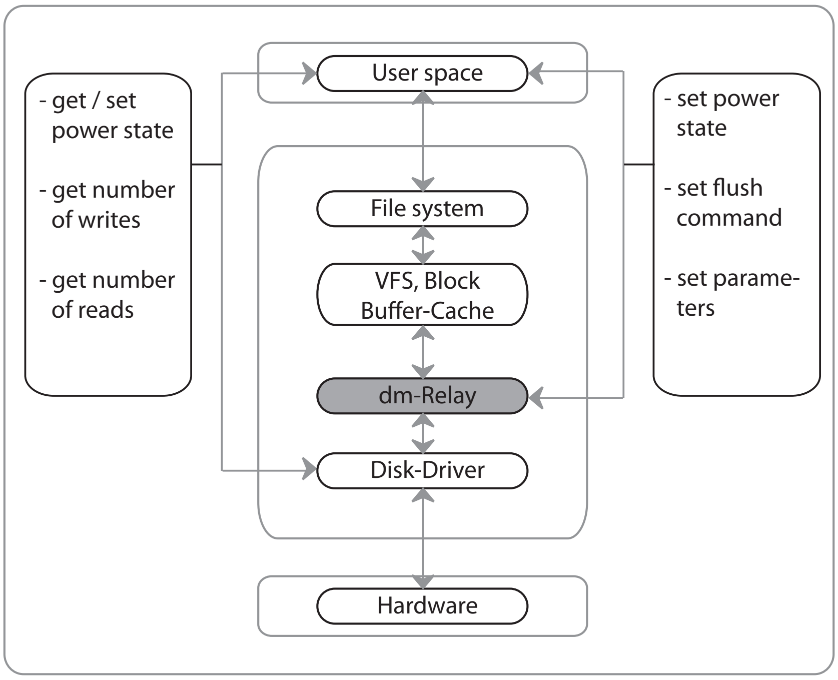

Figure 5.1: The proposed solution is positioned between the buffer cache and the disk driver and an user space interface is provided.

The design was implemented in two stages: A kernel module and a user space utility that is controlling the flush policy and also does recovery if the need arises (see figure 5.1).

The kernel module is using standard kernel work queue and device mapper I/O client functionalities.

5.1 System-Architecture

The architecture of the safe laptop mode module consists of several sub tasks:

- Incoming request function
  - arrival function - pre-processes and classifies the incoming requests
  - rate limit module - flushes the log when it is n MB behind.
- Background and processing task
  - background task - processes the request queues based on mode, sets new modes
  - passthru module - passes through all requests
  - disk module - clones requests and flushes request queue to hard disk
  - log module - logs data to log device, reset the log, can take flash characteristics into account
- Processed request function
  - end_io function - post-processes processed requests
- Timer function
  - timer module - Allows flushing of the request queue for a certain amount of time
  - sync module - Allows syncing of all written data via sys_sync system call
- User space Message function
  - user space interface - allows setting of parameters and modes
- User space Helper programs
  - smart spindown program - Utilizes the user space interface to spin down the disk after the break-even time.
  - replay program - replays the log from log device
- dirty log module - tracks dirty and clean regions on the disk

The flow of data and signals between the sub modules is shown in figure 5.2.

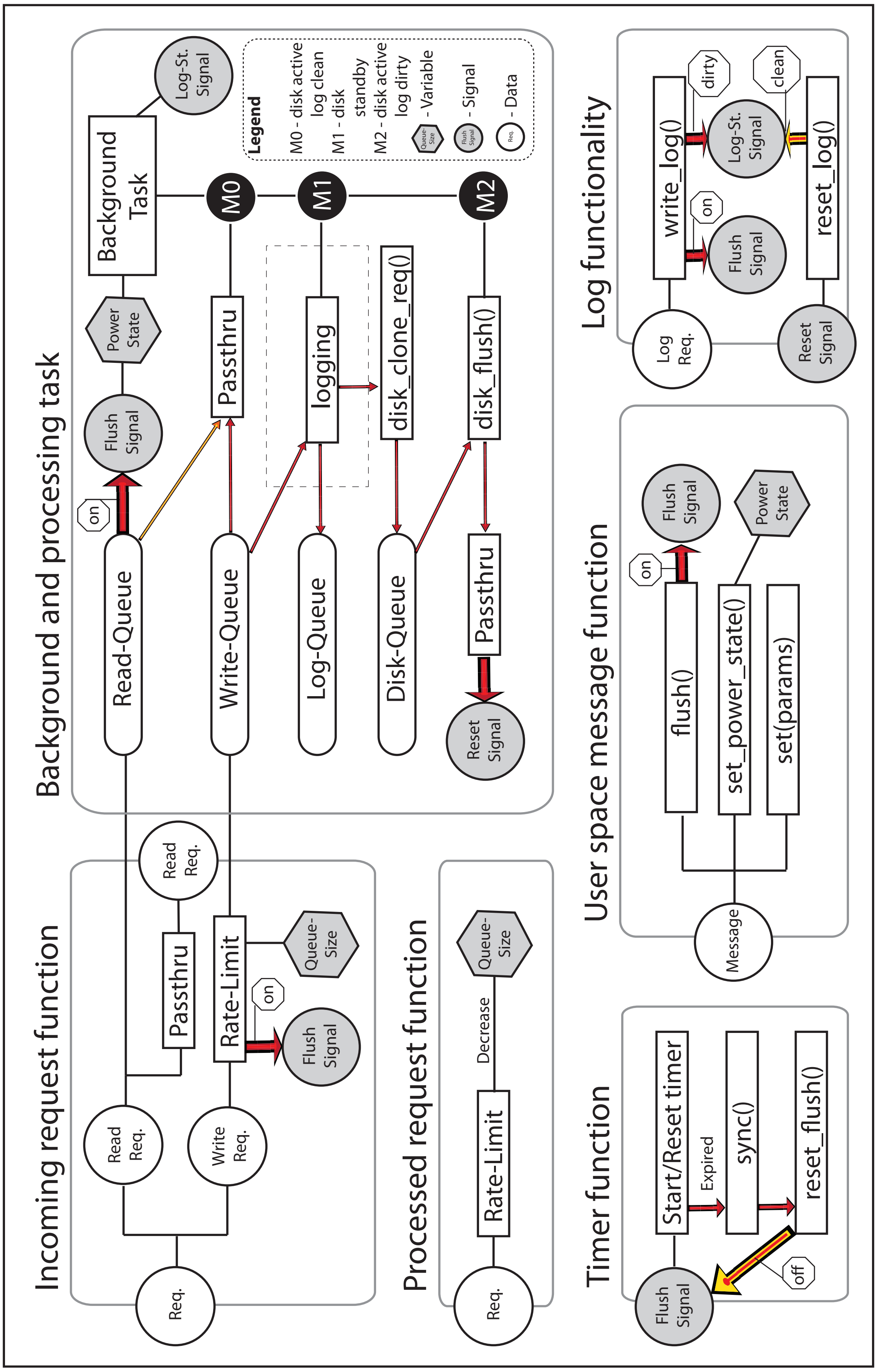

Figure 5.2: The design consists of several sub tasks that communicate through signals. An activity can be started through a signal, a queue write or an incoming request. Tasks can also utilize variables.

Mode	State	Action
M0	disk-active, log-clean	Pass through all data
M1	disk-standby, log-*	Clone write requests and log data to nv-cache
M2	disk-active, log-dirty	Only write-back, do not touch request queues

Table 5.1: The background task has 3 different modes of operation.

5.1.1 Incoming request function

Incoming requests can be classified either as read or write requests. The read requests can depend on written data. In this case the read-requests are pushed into a read-queue and the flush signal is set. The other read requests are just passed through to the disk.

Write requests are passed through a rate-limit filter. This filter is incrementing a queue-size variable by the amount of sectors to be written. Once the queue size is above a certain threshold, the flush signal is set. This threshold is the maximal number of sectors the log can fall behind. This prevents that the user needs to wait for huge amounts of data until it is written to the log, as the log media is often slower than the real media. Also at that point we can no longer guarantee the safety of the data, so it is important to flush the data to disk.

As long as the flush signal is set and the queue size is still above another threshold, the flush signal is set again. This prevents an enduring on-off-on-off cycle which would result in permanent flushing-log writing-flushing while a huge file is copied. The idea is that if there is still much data to process, it is not a good idea to change back to logging mode.

The write requests are then pushed onto a write-queue.

5.1.2 Background and processing task

Both queues are associated with a work queue and a background task is responsible for handling those queues. On push to one of the queues this background task is woken up.

The background task has 4 finite states:

- disk-active/log-clean
- disk-standby/log-clean
- disk-standby/log-dirty
- disk-active/log-dirty

that map to the three modes passthrough (M0), logging (M1) and write-back (M2) (compare table 5.1).

To calculate the disk-state both the power state and the state of the flush signal are taken into account. The flush signal always overrides a standby power mode and sets disk-state to active. On the other hand the flush signal is ignored when the power state was already active. The log-state can only be set by the log module.

M0 - Passthrough mode

In this mode all requests from the write and read-queues are directly passed through to the disk.

M1 - Logging mode

In this mode only write requests are processed (as read requests would have set the flush signal anyway).

The write requests are cloned (inclusive data) and pushed to the disk queue and then the original request is pushed to the log queue. To clone data in a memory-saving way, only the memory map counter should be increased.

The actual logging takes place after the queues are processed. This can be implemented either as a separate logging daemon or inside the background task.

The Log requests are written with a header to the flash-cache (see also section 5.2) and the log-state is set to dirty.

The log also can get a reset signal, which discards all remaining requests, writes a clean log head and sets the log-state to clean.

M2 - Write-Back mode

In this mode only requests from the disk queue are passed through to the disk; neither read nor write-queues are touched.

Once all pending requests have been written to disk, the reset signal is sent to the log (which then sets the log state to clean).

5.1.3 Processed request function

When the requests have been written to either the log media or to the disk media, they are processed by the second part of the rate limit filter and the queue-size variable is decreased by the amount of sectors written.

5.1.4 Timer function

The timer is started or reset once the flush signal is set. The timer is the only function, which can reset the flush signal. After the timer expires, the sync() system call is run to write all pending data to disk. This is done to avoid that the timer expires and with the next write back of data by the kernel large amounts of data are written to the log. Just after all data has been processed and written to disk the flush signal is reset.

A timer is used to avoid flush-logging-flush cases like in the rate-limit filter. The duration of the timer however should be below the write-back threshold of the kernel, because else the flush signal might never be reset.

5.1.5 User space Message function

There are three possible messages that user space can send:

- Flush Signal - Sets the flush signal to flush the log
- Power State Signal - Sets the power state of the disk
- Set Parameters - Sets parameters like max-queue-size or timer duration

The power state is not the actual power state of the disk, but rather telling the module that now data should go to the log as the disk is about to go to sleep. Or that the disk has just woken up and data could be written back to disk or passed through directly. Or whatever policy the user space program wants to implement.

Power-State is directly setting the power-state variable.

5.1.6 User space Helper programs

While the recovery functionality could be also implemented inside of the kernel, user space has the advantage that it can directly communicate with the user if necessary (for example in the case of log corruption).

The replay program reads the log from the flash cache, checks if it is dirty and replays the dirty parts back to the disk.

The task of the Smart Spindown program is to spin down the disk after n secs of inactivity. Before the disk gets the standby command, smart spindown makes sure that the log is flushed to disk, all data is synchronized via sync() command, and that the power state is set to "standby" to avoid waking up the disk due to further write requests.

Smart spindown also sets the mode to "standby" at start up to start the logging functionality.

5.1.7 Utilizing a dirty log

Not shown in the diagram, but also needed is a dirty log kept in memory that is tracking, which region of the disk is dirty and which is clean. This allows classifying read requests as dependent on pending write requests. The dirty log is segmenting the available space and maps it to a bitmap. On a write-request the section counter is incremented. When the write request was processed it is decremented again. On an incoming read-request the bitmap is checked and if the region is dirty the request is marked as dependent.

5.2 Log suitable for Flash-Storage

While any log could be used for the purpose of this work, it or the underlying file system should support the special characteristics of flash memory, where each cell can only be written a limited number of times.

Unfortunately in the case of commercially available USB-sticks a small software layer is hiding the complexity of the flash memory so all work done to optimize the usage of the flash memory is guess work as the wear-leveling-software is a black box. However looking at how the manufacturer has formatted the stick can give points how an ideal utilization could look like.

5.2.1 Flash Memory characteristics

As shown in figure 5.3 flash memory consists of erase blocks and pages. Each page can only be written once. If a page needs to be re-written, the whole erase block needs to be erased first and pages still valid in this block need to be copied to other blocks. If a block was erased too often it fails and needs to be replaced, so that data is always read and written to and from a backup block. Data can also be corrupted on a read, though that is seldom the case.

Reports of failing USB flash drives after using FAT file system with "-o sync" option indicate that most consumer wear-leveling-software is very basic and maps most blocks in a linear way.

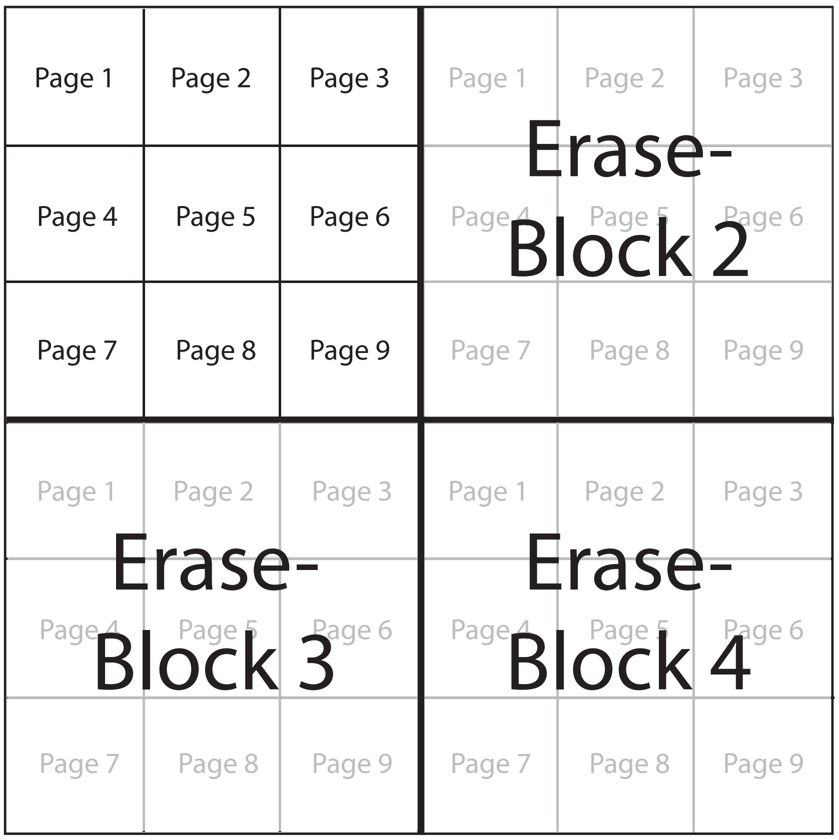

Figure 5.3: Flash memory has erase-blocks and pages. Data can be written only on a per page basis and each page can only be written once. If a page needs to be re-written the whole erase-block needs to be erased first.

5.2.2 Optimizing the log

With that knowledge space can be traded against USB-stick lifetime. A log optimized for the usage with flash memory should:

- Start on a boundary of a erase block (multiple of 256 kB)
- Always align data logged to the page size (4096 Bytes)
- Write pages sequentially, never overwrite a page.

As shown in figure 5.4 the designed log does full fill these characteristics.

The log consists of head and log sections and has two types of records: head records and log records. The log section is used as a ring-buffer. Head records point to the start of the log in the log section and also have an UUID. Log records consist of a meta-data and a data part and are also linked to one UUID. A log record is always padded to the device page size.

On a reset of the log, the head record is written to the next valid position in the head erase block. If the head erase block is full or contains no valid head records, it is zeroed and the head record is written to position zero. The last head record with the correct magic is the currently valid head record and its UUID is chosen.

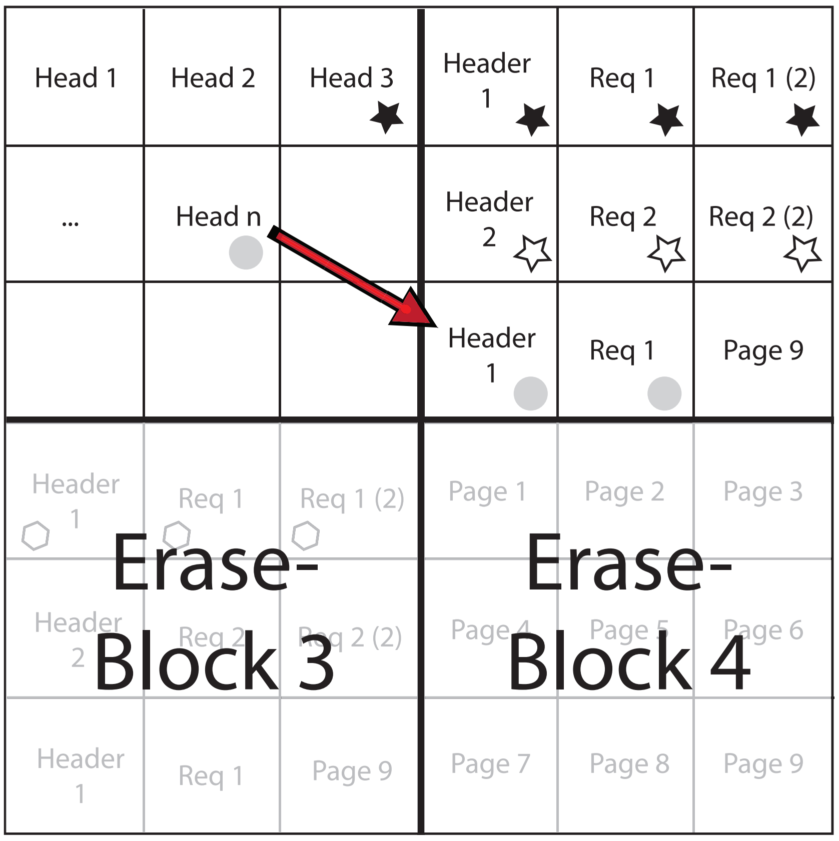

Figure 5.4: The log is optimized for the special characteristics of flash memory. The head-block is zeroed once and then written page per page on a log reset pointing to the start of the log. A special UUID (symbol) allows to differ between valid and old log entries.

This UUID allows to differentiate between valid and old log entries and so the log section does not need to be zeroed each time a log reset occurs. In that case only a new UUID is chosen. In figure 5.4 the different UUIDs are shown as different symbols.

In conclusion this means that the head block is only written on log reset (on a per page basis) and for each request also only one write request is necessary. Also no block is written twice until the log overflows in which case it is reset to position zero. Hopefully the wear-leveling-software will optimize the case of an erase block full of zeroes and never erase the block on subsequent head log write requests to different pages.

5.3 Using dm-relay in practice

dm-relay is a loadable kernel module and to build the solution nothing more is needed than a simple make command. The solution can then be included before the root device is mounted and init is started with the following commands. It is assumed that the root-device is /dev/sdc2 and the USB flash drive partition is available via /dev/sde1.

```
# Load dm subsystem
modprobe dm-zero
# Load kernel module
insmod ./dm-relay.ko
# Setup dm target
echo "0 $(blockdev --getsz /dev/sdc2) relay /dev/sdc2 /dev/
  sde1 $(blockdev --getsz /dev/sde1)" | dmsetup create
  relay-root
# mount root file system
mount /dev/mapper/relay-root /mnt
# Start init process in new root
chroot /mnt /sbin/init </dev/console >/dev/console
```

In the running system dm-relay can be send messages via dmsetup message command like this:

```
dmsetup message relay-root 0 flush
dmsetup message relay-root 0 active
dmsetup message relay-root 0 standby
```

Now only the modified smart-spindown script needs to be run and the solution is working completely transparent.

```
./smart-spindown.sh
```

Smart-spindown takes care of spinning down the disk and setting dm-relay to standby mode.

dm-relay could have also been added to a running system if the system was utilizing the LVM2/device-mapper framework for the root partition, but this was not the case for the KNOPPIX system used for the tests and evaluation.

5.4 Summary of this chapter

This chapter introduced the architecture of the proposed solution. It described the various subsystems that were needed for the implementation. Then the special characteristics of Flash memory and especially USB flash drives were described and a log design proposed that is optimized for this kind of media. Last an insight was given how the solution can be used in practice.

6. Evaluation

This chapter shows how much energy the proposed solution can save compared to the other approaches introduced in this work. It also examines how the proposed system behaves in different scenarios including but not limited to the problems described in chapter 4.

Especially I will show that:

- The solution presented in this work is able to save almost as much energy as the Linux laptop mode, which is only holding back the requests (and flushing at some interval). The solution introduces only a minimal overhead while minimizing the hugest drawback (loss of data) of the Linux laptop mode.
- In terms of memory usage this solution does not use more memory than a solution, where write-requests are just held back and then flushed.
- In terms of CPU the solution does not have more overhead than a comparable RAID1 solution.
- Latency for most read requests and most non-synchronous write requests is the same as without the proposal.

This chapter first introduces the methodology that was used to conduct the tests. Then the benchmarking setup is described. After wards the actual benchmarks are presented, which are then first described and then discussed.

6.1 Methodology

To test the validity of the approach the power consumption of the implemented prototype was actually measured in the power lab. Both, offline traces have been replayed and online (live) measurements have been conducted (see table 6.1). The approach was evaluated quantitatively.

The first test case was a realistic workload trace of someone using a laptop computer for some easy tasks like programming or writing texts. As this is not 100 percent reproducible, for this part of the work traces have been recorded and replayed.

This test case has been conducted to be able to show that energy can be saved by using the approach and how the typical request timing of a system looks like.

The other test cases included copying a big file, compiling a kernel or just leaving the computer idle for a while – without activity.

Test-Case	Practical application	On/Offline
Workstation	Working on a laptop computer	offline
Idle-Pattern	Leaving the system unattended	online
Random-Access Pattern	Compiling a kernel	online
Sequential-Access Pattern	Copying or downloading a large file	online

Table 6.1: Four tests cases of which three are online cases have been done to show that the solution saves energy without degrading performance in common usage scenarios on a modern laptop.

These online measurements have been conducted to measure not only energy consumption, but especially to measure the efficiency of the solution in common use cases; for example Latency, I/O throughput, memory and CPU usage.

6.2 Benchmarking setup

All benchmarks have been done with a KNOPPIX Live CD V6.2-2009-11-18 [Knop03], which has been installed to the notebook disk for the online measurements. The KNOPPIX Live CD environment has also been used to replay the traces. The offline traces have been recorded in virtual machines running two different KNOPPIX versions (compare table 6.2).

Used System	Version	Trace
Knoppix	6.1-2009-02-10-DE	10 minute trace
Knoppix	6.2-2009-11-18-DE	60 minute trace

Table 6.2: Real-time traces have been recorded using two different Linux systems.

The tests have been conducted using a "Western Digital WD3200BEVS 320GB" notebook disk, a 32 GB "Intel X25-E Extreme SATA Solid-State Drive" [Corp09] as an extremely fast backing store for the log and an eight GB USB flash drive "SanDisk Cruzer contour 8GB". The power consumption of each device can be found in table 6.3.

The power consumption was measured by a measurement device with resistors of 100 mOhm using a lab view script to log the measured data every 10th of a second.

Test Hardware	Type	Capacity	active	idle	standby	load
WD3200BEVS	Disk	320GB	1.2	1.0	0.63	3.0-4.0
Intel X25-E	SSD	32GB	0.59	0.59	0.59	1.0
San Disk Cruzer	USB	8GB	0.1	0.1	0.0	0.2-0.36

Table 6.3: The power consumption of a notebook disk, a SSD and an USB flash drive were measured for each of the 4 power states.

Traces were obtained using dm-statlog utility, which is using the same format as the log header written to disk. I have decided against btrace and blkdump, because they do much more than is needed for this work and using the log format for traces was a good test of the log part too. Trace and replay has been verified against /proc/sys/vm/block_dump live output.

Traces are replayed with the dm-replay utility, which is opening the disk device directly and is replaying the data in real time back to the disk (using fsync). dm-replay does not take into account that sleep times can be shorter due to different access times of devices or log media. But as the results show, this is not significant in comparison of the methods and merely a statistical error. In all test cases besides laptop mode fsync was used to get as realistic playback timing as possible.

To create equal circumstances at the beginning of each experiment, scripts have been created that prepare the system always in the same way so that the state is the same.

For the live measurements the system was started from hard disk using the safe-laptop mode/dm-relay kernel module as root device. The log is written to a 1 GB USB flash drive partition starting on an erase-block boundary (see section 5.2.1).

All measurements have been started with the disk being in a low-power-idle mode. After wards the disk was set to standby and after five seconds the measurement was started. At the end of each measurement all data still in buffers was written to disk via the sync command. After a delay of 1 sec the disk was spun down again.

The notebook disk is using 1.2 W in active mode. After 10s with no activity the disk automatically changes into a low power idle mode, where it consumes only 1.0 W. All power consumption data for the used hardware can be found in table 6.3. The behavior of the notebook disk to always change into a low power idle mode after 10s can be seen in the A row of figure 6.2.

The disk was set to go into standby mode after 5 minutes, however whatever time was set, the disk changed only its behavior to go after 10s into standby instead of low power mode. To solve this issue and allow valid measurements the smart spindown script [Samw03] was changed to monitor also writes via /sys/block/sd-c/stat (field 5) and suspend the disk always after around 300s.

The reason for using a SSD as a log medium was to find out if a slower or faster non-volatile storage is influencing the performance of the solution.

6.3 Performed Benchmarks

As already described the performed benchmarks were split into an offline and an online (live) part.

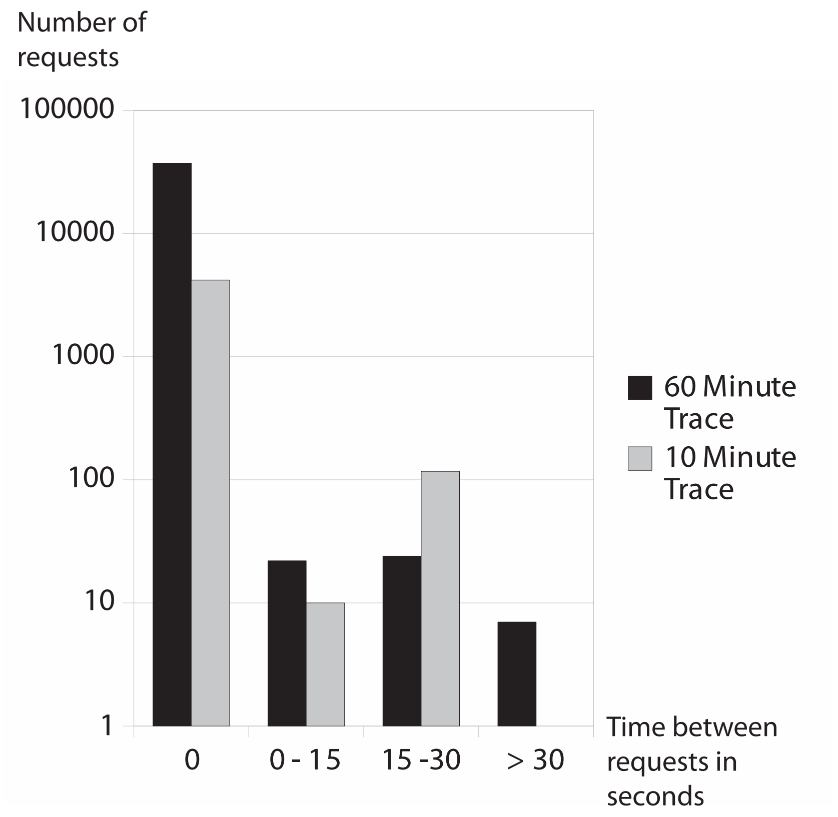

Figure 6.1: The 10 min and 60 min traces show two extremes: While the 10 min trace has no idle times greater than 30s, the 60 min trace has several long idle periods.

Two traces were replayed for the offline part. One from a workstation trace of me working on the implementation while periodically saving my work, which is having no idle time longer than 30s (there are production systems that have this behavior). And another one where I do work on the implementation, then leave my place and come back later to again write some code. This trace has some pretty big idle times, which is the other extreme (also see figure 6.1).

Policy	Description	Log	Spin-Down	Base
Write-All	Write all data, then spin-down	-	0s	min
Oracle	Oracle	-	0s	base
LM	Linux Laptop Mode	RAM	20s	WR-Delay
SLM	Safe Laptop Mode	USB	20s	-
SLM	Safe Laptop Mode	SSD	20s	-
-	Normal system behavior	-	5min	max

Table 6.4: This table shows how the different policies were configured for the offline test cases. The base column is giving a pointer how solutions can be compared. While oracle is a known base in the field of hard disk power management, laptop-mode is used as a base for solutions using delayed writes.

6 Test cases were done for the offline part for both the 1h and 10min trace that are described below (see table 6.4).

6.3.1 Write all

In this test case all data was written as fast as possible. Then the disk was spun down immediately, then 59 min 44 sec/9 min 44 sec standby have been measured. This is the absolute minimum.

6.3.2 Oracle

In this test case the oracle policy was implemented inside of dm-replay. dm-replay was sending a standby command to the disk, when the time between this and the next request was greater then the break-even time (here: 19.34s).

Description	/proc/sys/vm/ variable	def. value	LM value
Flush after n secs	laptop_mode	0	2
Max lost work	dirty_writeback_centisecs	3000	60000
Max lost work	dirty_expire_centisecs	3000	60000
Percentage of dirty pages (self)	dirty_ratio	20	60
Percentage of dirty pages (pdflush)	dirty_background_ratio	10	1

Table 6.5: Laptop mode is configuring certain system parameters. For each variable in /proc/sys/vm/ the default value (def.) and the value configured by laptop mode (LM) is given.

6.3.3 Laptop-Mode

Laptop Mode is configuring the disk to spin down after 20 secs with no activity. All non-synchronous write requests were delayed until the disk is active again. Then all data was flushed. After data has reached the life-time of 10 min it is always flushed to disk. The system parameters that laptop mode is setting can be found in table 6.5.

6.3.4 Safe Laptop Mode with SSD

Safe Laptop Mode dm-relay was configured to log to the SSD. Energy was measured for both devices. The log size was set to 500 MB after which it would have been flushed.

6.3.5 Safe Laptop Mode with USB

Safe Laptop Mode dm-relay was configured to log to the USB device. Energy was measured for both devices. The log size was set to 500 MB after which it would have been flushed.

6.3.6 Normal system behavior

In the normal system behavior, the disk spins down after 5 min of no read or write activity. The trace is replayed in real time with the fsync option.

6.3.7 Normal system behavior, no-fsync

In the normal system behavior, the disk spins down after 5 min of no read or write activity. The trace is replayed in real time, but without the fsync option, which gives the operating system again the chance to flatten the request queue, which leads to a smoother write-out and also saves energy.

For the online part the three measurements were done in the following way each running normal system behavior and safe laptop mode.

6.3.8 Idle-Pattern

Some programs were started in the K Desktop Environment, but the system was not used.

6.3.9 Random-Access-Pattern

The Linux kernel was compiled after having been untared to have a quite random access behavior with mostly writes.

The kernel compilation was done by creating a new directory untaring the kernel source to it and then doing a *make oldconfig prepare* before the measurement. Then starting the measurement with *make bzImage*.

6.3.10 Sequential-Access-Pattern

A single, large file (700 MB) was copied via rsync from the ramdisk. In this case the KNOPPIX CD-ROM Image was used.

The data was copied from a ramdisk to get maximum write performance and test the rate limit filter.

6.4 Results

Figure 6.2 shows the energy consumption of the 10 minute trace. The mean power consumption for this test case is displayed in table 6.6. The mean power consumption for the 60 min trace is displayed in table 6.7.

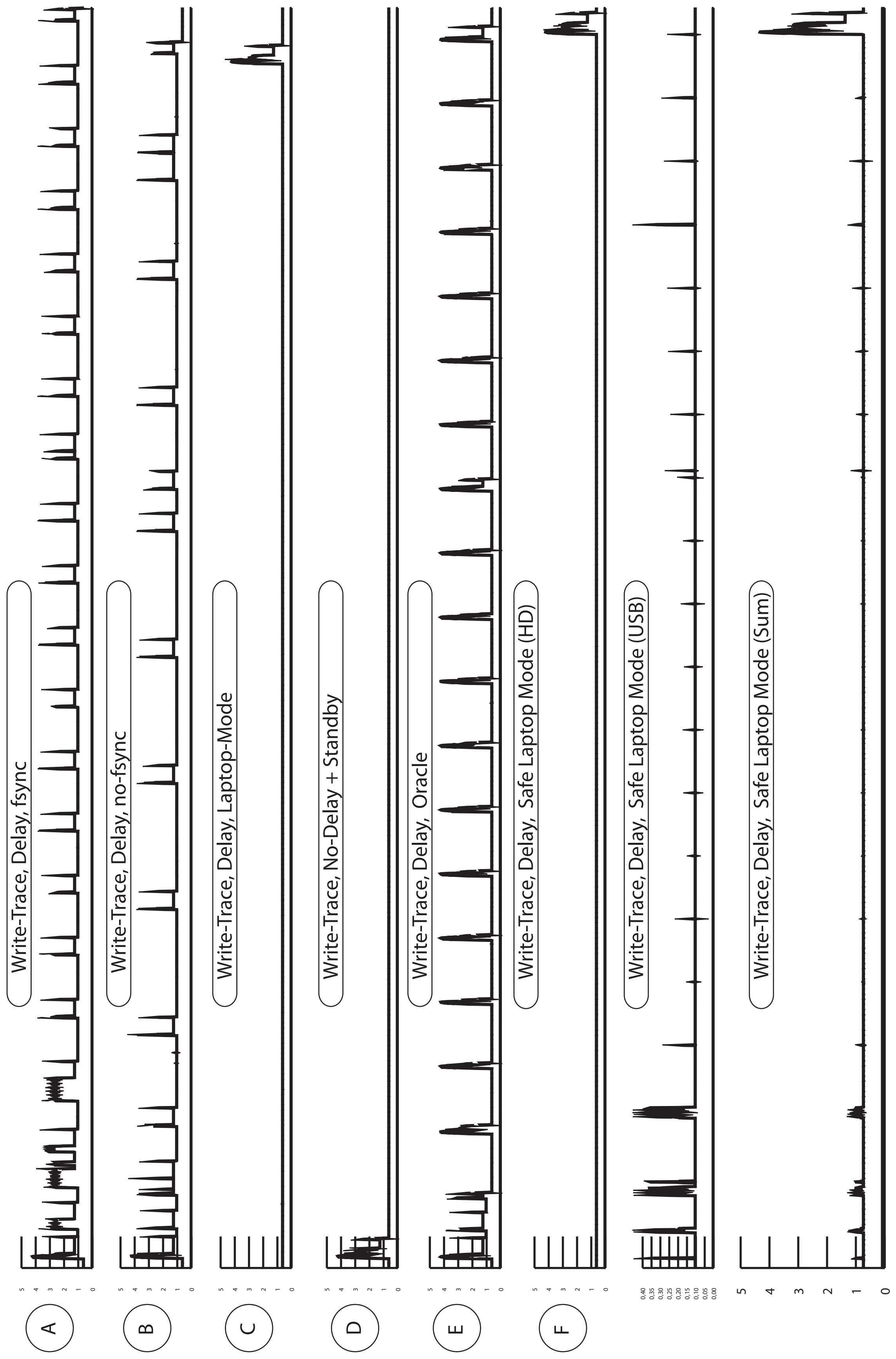

Figure 6.2: This diagrams show the power consumption when the 10-min trace is replayed using different policies and mechanisms.

6.4.1 Offline-Traces

In figure 6.2 the functionality of the laptop mode and also the flushing behavior of the Linux kernel can be seen. For the laptop mode only the last row (F) is interesting. While the disk is idle the write requests do only go to the USB flash drive, which consumes only tiny amounts of energy compared to the hard disk. Besides the overhead that is needed to manage the log, safe laptop mode consumes only as as much power as the write-all/minimal test case.

Sym.	Test-Case	10 min Trace	Extra power	Sum
A	Normal, fsync	1.2169	-	1.2169
B	Normal, no-fsync	1.1073	-	1.1073
C	Laptop-Mode (LM)	0.6559	-	0.6559
D	Write-All	0.6585	-	0.6585
E	Oracle	0.8719	-	0.8719
F	Safe LM, USB	0.6608	0.1066	0.7674

Table 6.6: Different policies were measured in the 10 min trace; all values are the mean power consumption over the measured time span.

Test-Case	60 min Trace	Extra power	Sum
Laptop-Mode	0.6605	-	0.6605
Write-All	0.6427	-	0.6427
Oracle	0.7223	-	0.7223
SLM, USB	0.6416	0.1095	0.7511
SLM, SSD	0.6446	0.6154	1.2600
Normal, fsync	0.9542	-	0.9542
Normal, no-fsync	0.9189	-	0.9189

Table 6.7: Different policies were measured in the 60 min trace; all values are the mean power consumption over the measured time span.

6.4.2 Online-Traces

Idle-Pattern

Due to the nature of the Knoppix system almost no background processes were running, so in this trace there was no activity at all even though lots of desktop applications were run. However as the 10 min offline trace shows this is not true for all systems such the consumed energy was the same for all three modes.

Sequential-Access-Pattern

The rate limit filter can determine quickly that a huge file is written. In that case it flushes the log, which leads to the same energy consumption as for the normal case. The same is true for the Linux laptop mode, as it also flushes all data to disk as soon as the sync signal is send – before the end of the measurement.

Random-Access-Pattern

It can be seen in table 6.8 that *safe laptop mode* saves on average 0.46 W power consumption for the whole kernel trace compilation. All data could be read from the cache and so the safe laptop mode could be used in standby mode all the time and did not need to write the data to the disk until the end of the measurement. And because the kernel build is not utilizing synchronous disk accesses there also has been no time difference in building the kernel.

Test-Case	Random-Access	USB power	Sum
SLM, USB	0.6855	0.1494	0.8349
Normal	1.2950	-	1.2950

Table 6.8: Different policies were measured in the live trace compiling the kernel for 15 min; all values are the mean power consumption over the measured time span.

Memory has been traced during the test cases and no change in buffers/cache could have been seen.

Latency is the same as without the solution for the pass-through mode and much more dependent on the latency of the log medium for logging mode. For non-synchronous writes latency is the same as without the solution.

Tests conducted with the SSD as a log medium have shown no performance difference to the USB storage for most system operations. Of course synchronous writes are much faster, when the SSD is utilized, but by using the rate limit filter this case can also be handled well.

The overhead of the solution is except for some queuing of data and some minimal calculations for the rate filter minimal and no influence on system I/O behavior could be noticed.

6.5 Discussion

As the results show, safe laptop mode can save power compared to the normal system operation. However safe laptop mode is in all cases worse than laptop-mode, because it needs to write to the USB flash drive and the USB flash drive needs 0.1 W in idle mode. It is theoretically possible to put an USB subsystem to sleep (and waking up is very fast and consumes only little energy) and such for long idle times the USB subsystem could sleep as well. However: While the standby mode is written in the specification of USB and also already works when the system is suspended to RAM, it is at the moment not possible to enable this functionality for a USB device from inside the Linux kernel. The Linux kernel only can set an auto timeout, which can only be used when no driver is claiming the device.

Table 6.9 shows that Safe Laptop Mode does consume only 3% more energy than the oracle policy in the 60 min trace. In the 10 min trace Safe Laptop Mode even does save 12% more energy. 30% more energy than the normal Laptop Mode is consumed in the worst case, but only 10% in the best case. On average Safe Laptop Mode has an overhead of 16% compared to the normal laptop mode.

Test-Case	10 min Trace (Base Oracle)	60 min Trace (Base Min)	10 min Trace (Base Oracle)	60 min Trace (Base Min)
Normal, fsync	1.3956	1.8479	1.3210	1.4846
Normal, no-fsync	1.2699	1.6815	1.2721	1.4297
Laptop-Mode (LM)	.7522	.9960	.9144	1.0276
Write-All	.7552	1.0000	.8897	1.0000
Oracle	1.0000	1.3240	1.0000	1.1238
Safe LM, USB	.8801	1.1653	1.0398	1.1686
Safe LM, SSD	-	-	1.7444	1.9604

Table 6.9: The results of the 10 min and 60 minutes traces have been normalized to oracle and the minimum base to show that the Safe Laptop mode using USB log solution in a good case is 12% better than oracle and in a more difficult case only 3% worse.

Oracle is beaten, because even though oracle might sometimes have better energy savings, it does this by spinning up the disk 30 times per hour (60 min trace) or in the case of the 10-min trace even every 30 seconds, which means 120 spindowns per hour. This might be good in terms of energy usage (however Safe Laptop Mode is still better in the 10 min trace, and only marginally worse in the 60 min trace), but it is not good at all for the life-time of the disk.

The other measurements show that the solution can be used on a production system without any performance limitations and that it is well suited for example for the average kernel hacker that just needs that extra hour of laptop battery.

6.6 Summary of this Chapter

This chapter described the testing methodology and how the benchmarks were setup. It the described the benchmarks in more detail and the presented the results of this work. In the discussion it is shown that safe laptop mode beats oracle, but will always have the overhead of 0.1 W USB idle consumption compared to the normal Linux laptop mode. Despite this lack the proposed solution still does save enough energy to be usable for every day work.

7. Conclusion

The objectives presented in section 1.2 have been reached. As memory is now available at low-cost, it is no longer a problem to hold most of the written data in main memory and only write a backup to some non-volatile storage. The design of the solution is very simple and to reach the functionality described only a kernel module needs to be loaded. Common problems such as determining when to flush the log and when to spin-down the disk have been solved successfully.

Safe Laptop Mode is mainly suited for the use in mobile computers as that is where energy saving is most important. However the application could also be used on desktop computers as it gets more and more important to save energy here as well.

The evaluation shows that Safe Laptop Mode can beat oracle by 12%. Oracle is used as a common base line to compare disk spin-down algorithms that only predict request timing without changing it. Safe Laptop Mode only needs around 16% more energy than the minimum base line or the normal Linux Laptop Mode. The minimum baseline is the energy needed to write all requests to disk in one synchronous operation and then sleep for the rest of the time. Linux Laptop Mode is at the moment the best base line for disk spin-down algorithms, which delay writes to avoid spinning up the disk. The overhead of the safe laptop mode solely depends on the energy usage of the used non-volatile storage; ranging from 10% to 30% for the USB flash drive. As only up to 1 hour traces have been used for measurements, the results are more in favor of the normal Linux Laptop Mode as the Safe Laptop Mode has its strengths especially in long system usage as it is not bound to flush the data to disk after a fixed interval.

Performance characteristics are excellent and the solution does not need more Memory or CPU overhead than the normal laptop mode. Only synchronous write requests can be as slow as the log medium, but the majority of the requests is handled much faster.

In conclusion this work reaches the goal to provide a transparent and easy implementable mechanism to conserve energy by delaying writes without the possibility of data loss.

7.1 Future Work

Safe Laptop Mode presented in this study work does save energy. However at the moment it is not possible to standby the USB subsystem and such 0.1 W are needed at any moment. An open question is if the USB energy consumption can be decreased significantly and if so which effects this would have.

Another interesting field of research is the log part of the solution. There has been a great deal of work on so called journaling file systems and it remains an open question how this work could be used to optimize the presented log structure and if it would even be worth to for example optimize write-requests to contain only a delta to previous write requests.

While saving energy in mobile systems was in the focus of this thesis, it should also be possibly to apply the presented techniques in desktop or server scenarios. It would be interesting to see, if it is possible to include this one-disk/one-log solution into a system with more disks such as a RAID array. Work also needs to be done in terms of reliability of the flash medium as server systems have other requirements.

While the prototype has been implemented successfully and the easy plug 'n' play structure has been achieved, the work done needs to be cleaned up and published as an open source project. Lastly it remains to be seen if the work is adopted by users that want to save energy on their laptop without risking their data in the process.

Bibliography

[BiBL07] T. Bisson, S.A. Brandt und D.D.E. Long. A hybrid disk-aware spin-down algorithm with I/O subsystem support. In *Proceedings of the 26th IEEE International Performance, Computing and Communications Conference*. Citeseer, 2007.

[BiBr05] T. Bisson und S.A. Brandt. Reducing energy consumption using a non-volatile storage cache. In *Proc. of RTAS*, Band 5. Citeseer, 2005.

[BiJo07] W. Bircher und L. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proc. of*, 2007, S. 158–168.

[ChJZ06] F. Chen, S. Jiang und X. Zhang. SmartSaver: turning flash drive into a disk energy saver for mobile computers. In *Proceedings of the 2006 international symposium on Low power electronics and design*. ACM New York, NY, USA, 2006, S. 412–417.

[Corp09] Intel Corporation. Product Manual - Intel X25-E SATA Solid State Drive, 2009. [Online: http://download.intel.com/design/flash/nand/extreme/319984.pdf].

[Gree94] P. Greenawalt. Modeling power management for hard disks. In *Proceedings of the Symposium on Modeling and Simulation of Computer and Telecommunication Systems*. Citeseer, 1994.

[GSKF03] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir und H. Franke. DRPM: dynamic speed control for power management in server class disks. In *ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE*, Band 30. IEEE Computer Society; 1999, 2003, S. 169–181.

[HSRJ08] A. Hylick, R. Sohan, A. Rice und B. Jones. An analysis of hard drive energy consumption. In *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008*, 2008, S. 1–10.

[Knop03] K. Knopper. Knoppix Linux Live CD. *World Wide Web eletronic publication, accessible at http://www. knoppix. org.(Accessed 18/08/2009)*, 2003.

[LCSB$^+$00] Y.H. Lu, E.Y. Chung, T. Simunic, L. Benini und G. De Micheli. Quantitative comparison of power management algorithms. In *Proceedings of the Design Automation and Test Europe*, Band 160. Springer, 2000.

[MaDK94] B. Marsh, F. Douglis und P. Krishnan. Flash memory file caching for mobile computers. In *PROCEEDINGS OF THE HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, Band 27. Citeseer, 1994, S. 451–451.

[MaVa05] Aqeel Mahesri und Vibhore Vardhan. Power Consumption Breakdown on a Modern Laptop. In *Power-Aware Computer Systems*, Band 3471 der *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, S. 165–180.

[MTHC$^+$08] J. Matthews, S. Trika, D. Hensgen, R. Coulson und K. Grimsrud. Intel® Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. *ACM Transactions on Storage (TOS)*, 4(2), 2008, S. 4.

[NaDR08] D. Narayanan, A. Donnelly und A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 2008)*, 2008.

[Samw03] Bart Samwel. Smart Spindown, 2003. [Online; Stand 3. April 2010; http://www.samwel.tk/smart%5fspindown/index.html].

[Samw04a] B. Samwel. Kernel korner: extending battery life with laptop mode. *Linux Journal*, 2004(125), 2004.

[Samw04b] Bart Samwel. How to conserve battery power using laptop-mode, 2004. [Online; Stand 3. April 2010; linux-source-2.6.31.6/Documentation/laptops/laptop-mode.txt].

[Sche] Holger Scherl. Energy-Aware File System. Studienarbeit im Fach Informatik and der Universitaet Erlangen-Nuernberg.

[ScRo08] Patrick Schmid und Achim Roos. How Can Battery Runtime Be Shorter?, 2008. [Online; Stand 3. April 2010; http://www.tomshardware.com/reviews/ssd-hdd-battery,1955-2.html].

[WeBB02] A. Weissel, B. Beutel und F. Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. *OPERATING SYSTEMS REVIEW*, Band 36, 2002, S. 117–130.