UNIVERSITY OF OSLO
Department of Informatics

Solid State Disks vs Magnetic Disks: Making the right choice

Master thesis

Torkild Retvedt -
torkildr@ifi.uio.no

Solid State Disks vs Magnetic Disks: Making the right choice

Torkild Retvedt - torkildr@ifi.uio.no

Acknowledgments

I would like to thank my advisor Pål Halvorsen, for being a great source of valuable feedback, and for being extremely flexible and easy to work with. I would also like to thank the entire lab at Simula, for providing an incredibly stimulation work environment, and for being an infinite supply of information and entertainment.

In addition, I would like to express my gratitude to both friends and family for being a great inspiration, and for providing me with just the right amount of distraction...

Oslo, August 17, 2009

Torkild Retvedt

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The last few years, SSDs has received much focus as a possible replacement for magnetic disks. SSDs ability to do random operations with a near constant latency has changed the way we look at disk performance, making random access of data less costly. For many workloads, this reason alone, has been enough to consider the technology. Even though SSDs show great improvements over magnetic disks on random read operations, it still faces challenges caused by the physical limitations of Flash memory, making write operations a more complex operation than on magnetic disks. This property of SSDs means that it can, in many cases, be hard to give a clear answer to what disk technology is the *best*.

Though many simple performance studies have looked at the performance of both magnetic disks and SSDs, these have a tendency to look at the performance of very basic, and limited, operations. By simply looking at performance in a few narrow cases, it can be hard to identify where a potential bottleneck lies. To get an insight into the limitations and advantages of the different technologies, we will take a close look at both magnetic disks and SSDs.

1.2 Problem statement

We will, in this thesis, closely investigate the performance of SSDs and magnetic disks. Looking at physical attributes of these two disk technologies, we will try to give an understanding of how they perform in different scenarios based on real workloads. Based on what we learn from investigating the performance in these scenarios, we will try to get an understanding of what performance issues the different storage technologies faces. We will look at how optimizations in Operating System (OS) and at application-level have an impact on SSDs and magnetic disks, as well as what we can do to achieve best possible performance.

1.3 Main contributions

In our benchmarks, we show that early generation SSD suffer from write amplification, and will, because of that, have problems achieving good performance on random write operations. By introducing a log-based file system, we see that we can remove the effect of write amplification received by random write operations, and give us a more stable performance. We discuss, and benchmark, possible alternatives for optimizing existing applications for magnetic disks, by introducing SSDs, as an alternative to magnetic disks as storage device. By going from high-end magnetic disks to SSDs, we see that we, in some cases, can achieve orders of magnitude better performance, without changing application level code.

1.4 Structure

In chapter 2 we will look at different, existing and future, non-volatile technologies. We look into what fundamental changes these technologies introduces, and why we are interested in these changes in storage tech-

nology. Chapter 3 will give an overview over magnetic disks, SSDs and how these two compare. We will also give a short insight into what expectations we have from both magnetic disks and SSDs, with regard to different aspects of system performance. In chapter 4 we will look closely on the performance of magnetic disks and SSD in different scenarios. We will try to identify the main characteristics and try to point out possible weaknesses, as well as solutions to these. Finally, in chapter 5 we will summarize our findings, as well as do some reflection on possible implications.

Chapter 2

Non-Volatile Memory

The last couple of decades, we have seen an increase in availability and interest of non-volatile memory technologies. In this chapter we will look into different types of non-volatile memory. We will give an overview on how the different types of memory compare to each other, and what makes these interesting for bulk storage.

Non-volatile memory is generically speaking all semiconductor based memory devices with persistent storage. We will look at both advantages and challenges introduced by using non-volatile memory as storage. Also we will take a look at possible future technologies, and how these compare to existing non-volatile memory.

2.1 EPROM

Erasable Programmable Read-Only Memory (EPROM) was introduced in 1971 by Frohman-Bentchkowsky as a new way of storing data in semiconductor materials [1]. Historically, the most viable solution for storing data permanently has been magnetic storage on disk platters, e.g., a hard drive. This new technology however, was far from being able to satisfactory replace bulk storage, having major shortcomings, like no fast auto-

mated way for erasing data. EPROM did supported erasure, but mainly as a way for testing chips when in production. To erase an EPROM chip, it has to be exposed to Ultra Violet (UV)-light, making it highly impractical for any scenario where erasures are commonplace.

As a simple permanent storage, however, EPROM has proved valuable, growing more and more popular into the 1980s. It has since been included in a number of application, both in industry and consumer product. This ranges from internal permanent storage in hard drives and network switches, to automotive usage. Being included as a simple storage for small amounts of data in consumer products, EPROM quickly received attention in form of research and refined production.

2.2 E^2PROM

A refined version of EPROM, called Electrically Erasable Programmable Read-Only Memory (E^2PROM) was proposed in 1985 [2, 3]. This technology introduces a more flexible approach to erasing, making it possible to erase data with an electrical current, using a technology called field emission. Not only does this make it possible to simplify test procedures during production, but it also has the side effect of enabling on-device erasure. These memory cells are organized in manner resembling NOR-gates, as will we will look at in detail in section 2.3.2, and are sometimes being referred to as NOR Flash cells [1].

Being a relatively young technology, early E^2PROM implementations were still considered to be a form of permanent storage. During the 1990s, the technology did, however, prove itself powerful. Because of lacking moving parts, having a relatively low power consumption and getting increasingly cheaper to produce, E^2PROM soon proved to be a vital part of many mobile devices.

2.3 Flash Memory

The term *Flash memory* was coined in the early days of E^2PROM to emphasize the ability to erase the entire chip in a fast and effortless way, or in a *flash* [1]. Through the years, the term has been used for numerous variants of non-volatile memory types. In recent years, it has, however, become common to refer to a modified variant of E^2PROM. This variant uses a cell structure resembling NAND-gates, as will be explained in section 2.3.2.

What is common for both EPROM and E^2PROM (or variants of both), is that data is being stored by altering the electrical attributes of a floating-gate cells, changing the threshold voltage. This voltage is later being used to determine what bit the cell represents.

2.3.1 Multi-level cell vs Single-level cell

Regular Flash cells, described in section 2.3, are called Single-level cell (SLC). These cells can distinguish between two discrete values, thus being able to store a single bit of information. To give a higher storage density, another type of cell has been introduced, Multi-level cell (MLC). Unlike SLC, these cells can hold several distinct threshold voltages, something that in turn will make it possible to store more than one bit in a single cell. By having four or eight different values, the density is increased two or four times respectively.

2.3.2 NAND vs NOR

As we mentioned briefly in section 2.3, Flash cells can be organized in different ways. The two ways used is named after what logic-gate structure they resemble, NAND and NOR. In short, NOR Flash is connected in parallel, giving it great performance in random access, whereas the more serial approach in NAND Flash gives it a higher storage density. Today,

NAND Flash is more or less the *de-facto* standard used in Flash storage [4]. figure 2.1 illustrates multiple reasons for this. One of the more convincing reasons for choosing NAND over NOR is probably the cost-pr-bit, making Flash storage a viable option to the well established magnetic storage.

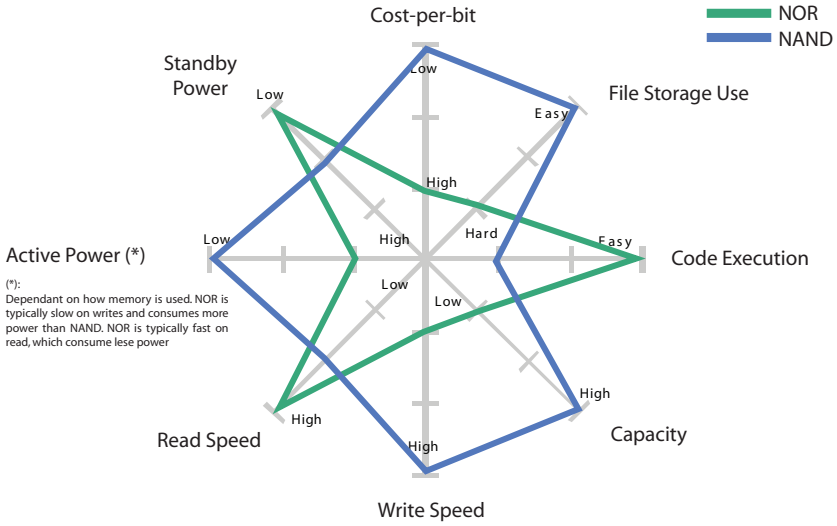

Figure 2.1: *NAND* vs *NOR* memory [4]

2.3.3 Structure

The internal structure of Flash memory is rarely identical from chip to chip. As the technology has matured over the years, many smaller architectural changes are been made. There are, however, a few fundamentals for how Flash memory is constructed. Each chip will have a large number of storage cells. To be able to store data, these will be arranged into rows and columns [1]. This is called the Flash array. The Flash array is connected to a data register and a cache register, as seen in figure 2.2. These registers are used when reading or writing data to or from the Flash array. By having a cache register in addition to a data register, the controller can process a request while the controller serves data. This enables the Flash memory bank, to internally, process the next request, while data is being read or written.

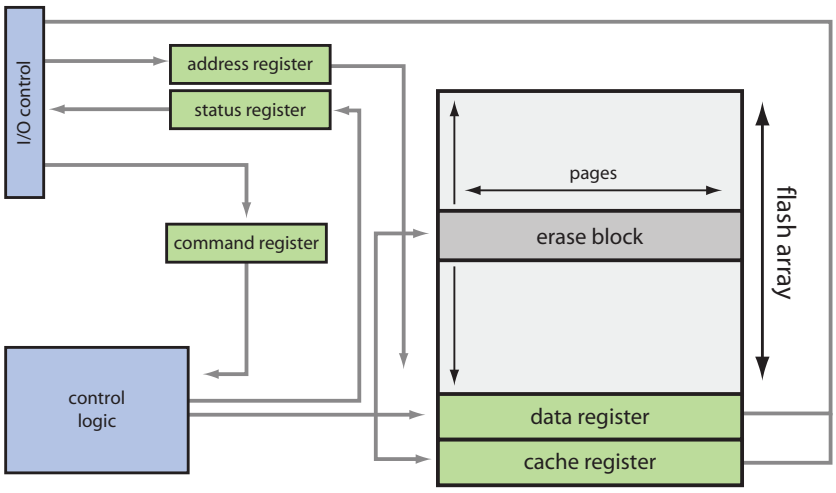

Figure 2.2: A generic overview of a Flash memory bank. Based on [5]

Page

Pages in a Flash array is the smallest unit any higher level of abstraction will be working on. The size of a page may vary, depending on the specifics of physical structure, but are typically in the size of 2kB [6, 5]. In addition data, each cell will also have a allotted space for Error-Correction Code (ECC). During a read operation, all the data from the page will be transferred to the data register.

In a similar way, write operations to a page will write all data in the data register to the cells within a page. What we need to know about Flash cells when writing is that these cells only support two operations. A cell can be in a neutral or a negative state. When writing data to a page, it is only possible to change from neutral (logical *one*) to negative (logical *zero*) state, meaning that to be able to change from *zero* to *one*, we need to reset the entire page.

Erase Block

When reseting cell state with field emission, multiple pages will be affected by the reset. This group of pages is called an erase block. A typical number of pages contained will be 64 [7], but can be different, depending on how the Flash cells are structure. Given a page size of 2kB, an erase block would then be 128kB in size. This tells us that changing content in any of the pages within the erase block, we would need to rewrite all 128kB. For this simple reason, in-place writes are not possible in Flash memory.

2.3.4 Cell degradation

Each time a Flash cell is erased, the stress on the cell from the field emission will contribute to cell degradation [1]. Modern Flash memory banks are usually rated for approximately 100.000 erase cycle, but to be able to handle a small number of faulty cells, each page will be fitted with ECC-data.

2.4 Related technologies

During the last two decades, Flash memory has become an increasingly stable and widespread technology. However, it does suffer from certain constraints, as we have discussed in section 2.3. To counter these constraints, there are several new non-volatile memory technologies on the horizon, some of which might take over for Flash in the future.

2.4.1 MRAM

Magnetoresistive Random Access Memory (MRAM) is a non-volatile memory where bit information is stored by changing magnetic properties in a

cell. Performance-wise, MRAM places somewhere between Flash and Dynamic Random Access Memory (DRAM) [8]. Being considerably faster at writes than Flash and lacking any known deterioration, this makes MRAM a valid competition for Flash. Using magnetic properties to store bit information instead of electrical charge, MRAM can be able to achieve lower power consumption on writes. Though much research is being done on MRAM, there is still no commercial production, and the maturity of the technology is years from being close to that of Flash.

2.4.2 FeRAM

Ferroelectric Random Access Memory (FeRAM) is a form of non-volatile memory, organised in much the same was as DRAM, but with ferroelectric properties instead of dielectric. The main advantages of FeRAM over Flash is overall better speed, less cell deterioration on writes [9], combined with low power usage. FeRAM does however not scale in the same way Flash does, both with regard to capacity and density. Because of these shortcomings, despite being commercially available, FeRAM is not mature enough to be a viable alternative to Flash.

2.4.3 PCM

Though still being far away from commercial production, Phase-Change Memory (PCM) is probably the most promising alternative to Flash to date. PCM addresses the issues mentioned in section 2.3.3, making it possible to flip bits between logical *one* and *zero*, without the need to erase/reset cells [9]. There are, however, still many unresolved issues or questions regarding deterioration and scalability.

2.4.4 Others

In addition to the technologies mentioned here, there is constant research being done in the field of non-volatile memory. With an increasing amount of embedded devices demanding both higher capacity and higher performance storage, without having to sacrifice low power consumption, no single solution stands out as optimal at this time.

2.5 Summary

In this chapter, we have given an introduction to different non-volatile memory technologies. We have also looked closely at the Flash memory, and seen how this technology is suitable for bulk storage, as well as some of the challenges introduced by cell degradation. In section 3.2, we will see how Flash memory can be used in storage devices aimed at competing with magnetic disk drives.

Chapter 3

Disk Storage

As computing capacity increases, so does the need for permanent storage. Disk storage has since the early days of mainframe computing been used in some way or another to store data. In the last decade, we have seen a change in how we think of storage with the introduction of SSD. Our goal in this chapter is to get an overview over what the different disk storage technologies have to offer, what they have in common, what sets them apart and what impacts these differences might impose.

In this chapter, we will look at the current state of disk storage, represented by magnetic disks in section 3.1. We will then take a look SSDs in section 3.2. In section 3.3, we will give a comparison of these two technologies.

3.1 Magnetic disks

Rotating magnetic disks is without doubt the most used disk storage technology today. Being in use for half a century, magnetic disks are today considered very mature, and have seen many major improvements. We will in this section look at some of the advances done in magnetic disks over the years, as well as try to give an overview over known weaknesses.

Magnetic disks, or Hard Disk Drive (HDD), are storage devices containing one or more rotating platters made out of a magnetic material. Small sections of this material is manipulated into different magnetic states, making it possible to store data. Magnetic disks have had a great ability to scale capacity, and continues to do so today. We can see an internal view of a magnetic disk in figure 3.1.

Figure 3.1: Internal view of a magnetic disk [10]

3.1.1 Physical layout

The rotating platters in magnetic disks may use both sides for storage. Each of these are are divided into sectors and tracks. The intersection of a single block and a single track makes up a block. As seen in figure 3.2, tracks on the outer part of the disk platter are made up of more sectors. This is due to the fact that the surface will pass faster under the disk head, also more surface mean we will be able to store more data in these tracks. We call these different sections zones.

To get a higher data capacity in a disk, several platters are put together in

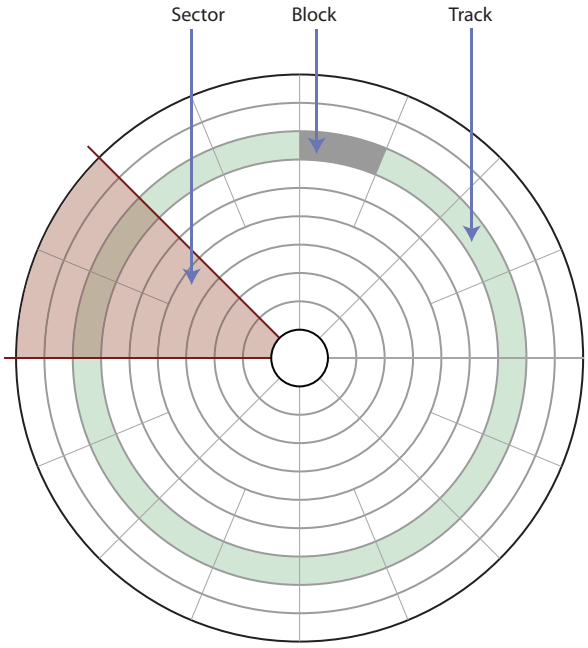

Figure 3.2: Example on layout of a magnetic disk surface

a spindle. The disk arm will have a separate head for each surface, and is able to write to more sectors with out seeking to a different track. The same tracks across all surfaces are called a cylinder. Having cylinders will make it possible to increase read and write operations, as the disk arm can perform operations on multiple surfaces without needing to move to different position.

3.1.2 Disk access time

Disk access time in magnetic disks are made up of three different operations. The time the different operations take will vary on position of disk head, where in the rotation the disk surface is and physical abilities of the disk.

Seek time The time needed for the arm to move into position, in other words changing track. When idle, the disk will place the arm in the middle of the disk to minimize this distance.

Rotational delay Rotational delay is the time it takes for the disk to spin into position. This time will be dependent on the rotational speed of the disk, as well as where in the rotation cycle the head is.

Transfer time Determined by the rotational speed of the disk, as well as the data density of the track being read.

With these characteristics, we see that seek time and rotational delay become a significant part of a random read or write operation. For sequential operations, the disk will be able to work on entire tracks/cylinders at a time, continuing with neighboring tracks/cylinders. Doing sequential read will, because of a short physical distance between the location of data, minimize time used on seeks, resulting in an overall lower access time for the data.

3.1.3 Reliability

In order for the disk head to read/write data, it has to *float* as close as a few nanometers from the surface of the disk. With a platter rotating up to as much as 15.000 Rounds per Minute (RPM), we have a piece of hardware very susceptible to damage. That being said, most disks today have a relatively high Mean Time Between Failures (MTBF) (like 1.200.000 hours in [11]), making reliability a minor issue.

Errors, however, do occur from time to time, making a way to handle these necessary. Magnetic disk drives today, has a small portion of spare sectors reserved for this purpose. When having tried correcting errors discovered by ECC, the disk will re-map the logical block [12].

3.1.4 Future

Magnetic disks has during the last decades followed Moore's Law, doubling in capacity roughly every 12 months. As well as capacity, bandwidth has also followed this trend. Latency does, however, improve with a

smaller factor, making random seeks more and more expensive [13]. Continuing this trend, we will either need to rethink the way magnetic disks are used or move to an alternative storage solution.

3.1.5 Summary

In this section, we have given a quick overview over the state of magnetic disks today. We have seen that magnetic disks have a challenge when it comes to random disk operations, and that moving parts make improving performance further a challenge.

3.2 Solid State Disks

In this section, we will take a close look at SSDs. Our goal is to identity what separates SSDs from magnetic disks, and how these differences will have an impact on different aspects of performance. We will also try to get an overview of the weaknesses in SSDs and how we might be able to counter these.

3.2.1 General

Contrary to its name, Solid State Disks (SSDs) do not use *disks* for storing data. An SSD is a storage medium made up of multiple Flash memory banks, combined to provide a seamless way to store data in ways similar to magnetic disks. SSDs provides the same physical interfaces, as well as command interfaces as conventional magnetic disks. This means that it is possible to switch from a magnetic disk to an SSD without having to upgrade additional hardware components, or doing alterations to the OS. We will take a close what the changes brought forth by SSDs means, and some of the considerations we need to take when replacing magnetic disks.

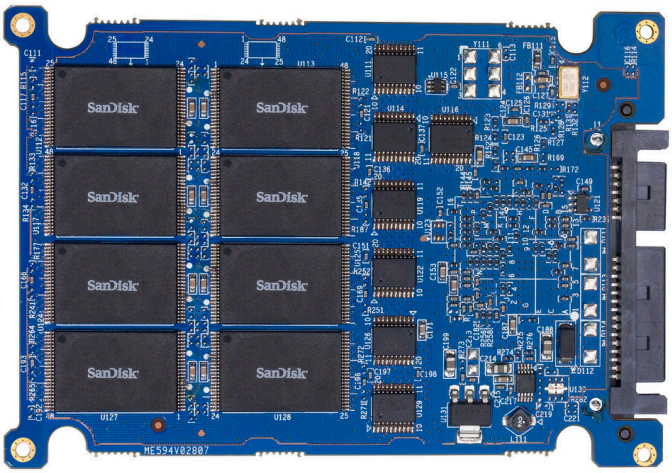

Figure 3.3: Internal view of a 64 GB SSD, exposing the Flash memory banks [14]

What makes Flash interesting for storage is that the performance places it somewhere between RAM and high-end magnetic disks [15]. In figure 3.4, we can see that SSD perform over 50 times better than magnetic disks with regard to access time. At the same time price is currently only increased with a factor of 5. Constant improvements like higher density, cheaper components and faster read/write speeds are making SSDs an increasingly attractive alternative to magnetic disks.

Though we see an increasing interest for use of SSD in applications such as servers and desktop machines, the success can most likely be attributed to the widespread inclusion in many capacity centered embedded applications[1] like digital cameras and digital music players. As the alternative to Flash in these applications has been magnetic disks in a smaller form factor, there is much to be gained by switching to Flash. These applications also make the main selling points of SSDs apparent, as things like small form factor, low power consumption and lack of moving parts are vital.

Today, several new laptop computers that focus on weight and power con-

[1] as opposed to bandwidth centered

sumption use SSD as an alternative to magnetic disks. Though SSD capacity is overall less than that of magnetic disks, it has been doubled every year since 2001 [5]. At the same time, prices has effectively been cut in half. This is probably the reason why SSD in later years have become a more viable option as primary storage in laptop computers, and not only used as an additional storage device for specific applications. Already in the early days of netbooks[2], many alternatives was produced with 16GB SSD as an option. This has most likely attributed to the mainstream inclusion and given the possibility to see how SSD perform in real scenarios.

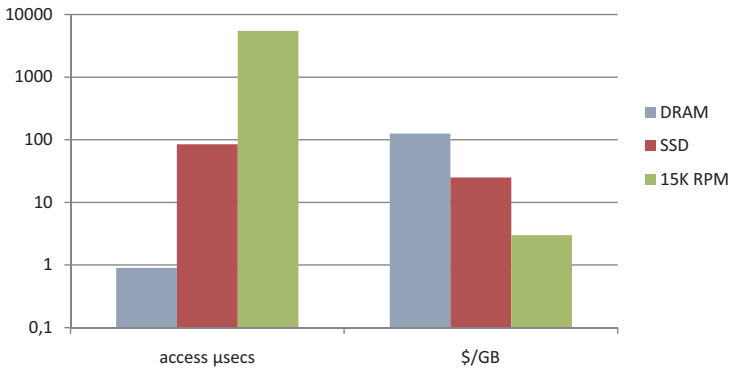

Figure 3.4: Price vs performance on different architectures. Based on [15]

3.2.2 Physical layout

There is little information released by hardware manufacturers about disk layout and how data is organized. To illustrate this, we can take a look at the entirety of what the Intel® X25-E datasheet has to say about its architecture.

> The Intel® X25-E SATA Solid State Drives utilize a cost effective System on Chip (SOC) design to manage a full SATA 3 Gb/s bandwidth with the host while managing multiple flash memory devices on multiple channels internally [16]

[2]Inexpensive laptop computers priced at a low price point, like Asus EEE

Having looked the structure of the Flash memory banks in section 2.3.3, we get a general idea of what to expect, but only a simple read/write operation. As we can see from block diagram in figure 3.5, an SSD connects several Flash memory banks together in a Flash Bus Controller (FBC). In a single SSD there are usually multiple FBCs, which are commonly called channels. As implied by the name, a channel will be able to independently process requests, giving SSDs the ability to internally process a number of operations in parallel. How and if operations are performed in parallel is however very often not known to neither consumer nor OS[3]. We will later see that from benchmarking certain types of operations on an SSD, we can make a few assumptions about the design.

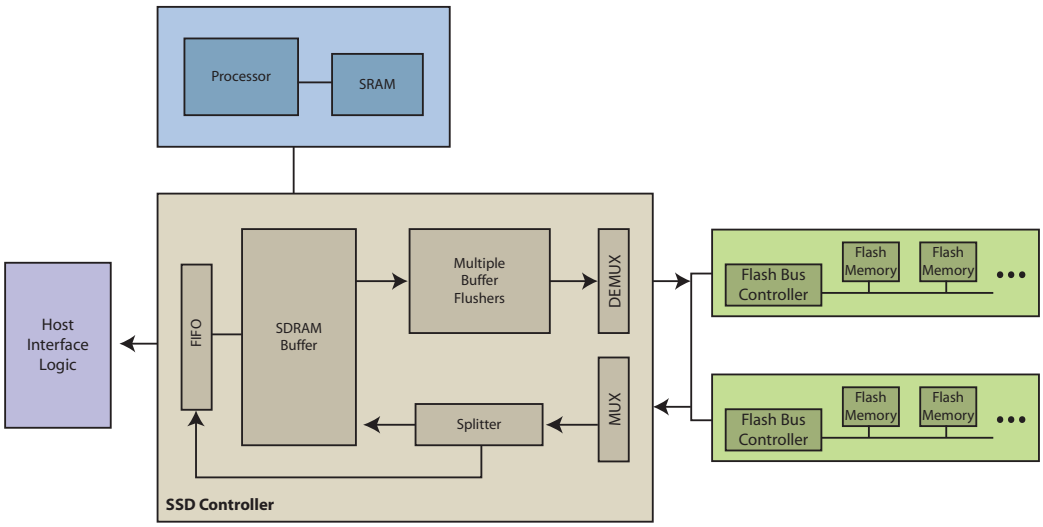

Figure 3.5: Block diagram of an SSD. Based on [18] and [19]

3.2.3 Flash Translation Layer

To be able to provide the same level of consistency on SSDs as on magnetic disks, we need to take into consideration wear and tear. As we have discussed in section 2.3.4, all cells have a limited number of approximately 100.000 erase cycles. This will, according to [18], give an SSD of

[3]This can be seen in a correspondence from the Linux kernel mailing list [17]

32GB capacity the life span of >140 years with a strain of 50GB sequential writes/day. Because of this physical limitation, SSDs is accessed through an abstraction layer known as the Flash Translation Layer (FTL). The main purpose of this layer is to make the SSD appear as any other block device, without the need for the OS to be aware of physical limitations [6]. This means that the OS will be able to use the disk without any additional knowledge. It also means that a potential overhead will be introduced on all I/O.

The FTL performs a mapping between how the data is represented to the OS and how data is stored in the different Flash memory banks. An illustration of this can be seen in 3.6. In magnetic disks, there is also a logical in place, though unarguably, not as extensive as in SSDs. In magnetic disks, blocks are mapped directly to corresponding physical blocks. When the disk observes a bad sector, the data contained in the sector are remapped to a physical block from a pool of spare blocks. As bad sectors in magnetic disks can be viewed as a relatively critical and rare occurrence, the role this layer plays, can mostly be view as a last resort.

Much unlike the mapping layer in magnetic disks, SSDs will remap blocks on a regular basis. When doing in-place write, most wear leveling algorithms will copy the new version of the data to a different physical block, and remap these [20]. This will mean that the OS thinks it is working on the same physical block, even though the data will be moving around constantly. By doing this, the FTL will ensure that no single page will be worn out by constantly changing data, while others are being left unattended.

These abstraction layers gives us the ability to access drives with different architectural design with a common interface, however they also make it hard to optimize for performance. Most of the commonly used file systems are optimized so that the abstractions of magnetic disks will prove less of a bottleneck.

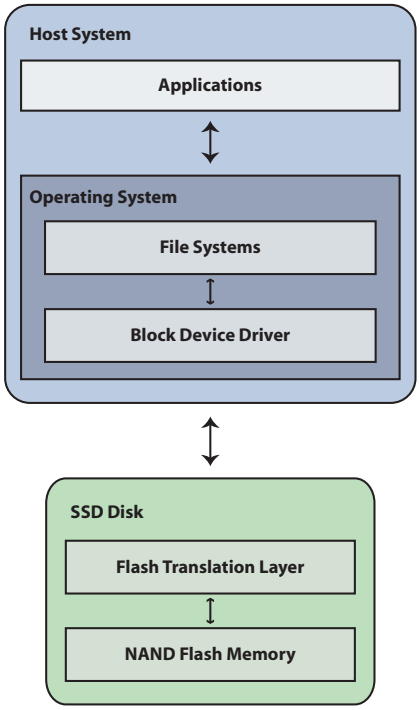

Figure 3.6: Placement of the FTL in an SSD. Based on [6]

Wear leveling

The physical attributes of Flash dictates that each cell will only last for about 100.000 erase cycles [1, 21]. We have seen in section 2.3.3 that we only can erase an entire erase block of typically 128 kB at the time. This means that if we change data in a certain page, data changed in neighboring pages will need to invoke an erase operation on the entire erase block. All unchanged data will then have to be erased and immediately rewritten along with the changed data. As we have learned from section 3.2.3, in reality, we will want to keep data moving around, so that the overhead of writing is keep as small as possible. We can, however, from both these ways of altering data, see that a system with constantly changing data, even in small numbers, will wear out.

SSDs handle these long term reliability constraints by utilizing some sort of wear leveling amongst pages. The goal of a successful wear leveling

mechanism is to, optimally, keep the erase cycle count in all pages as low as possible. At the same time, it should try to keep a level wear state across the entire disk. What this means is that the wear leveling algorithm, regardless of implementation, will monitor the wear of all blocks, and from that data, rearrange the logical representation of physical blocks as data changes.

Implementations of different wear leveling algorithms will differ greatly. As is being discussed in Chang et al. [20], the different ways to do wear leveling put focus on different performance aspects of the SSDs. Some implementations will favor fast availability of requested pages, whereas others will focus on keeping a balanced erase count. This is shown in a comparison of both academic and industry wear leveling algorithms. From the evaluation and benchmarking of these different implementations, we can see that most of these, both academic and industry, suffer from a skewed erase block count. We also see that using a less optimally implemented wear leveling algorithm can take quite a hit on performance, in addition to having large skew in cell wear.

Write Amplification

As mentioned in both section 2.3.3 and section 3.2.3, SSDs will need workarounds to enable in-place writing of data. That is, changing a few bytes of data will either need the entire erase block to move, or the entire erase block to be rewritten [22].

As with magnetic disks, an SSD will do some sort of drive level caching to achieve better performance on write operations. This caching will in some cases make a huge difference. A typical example of this is can be multiple in-place writes on a data contained within a single erase block. If in a queue, the SSD can get away with a single copy-modify-write operations, whereas working without cache, or with a flush between each write, the SSD would do a copy-modify-write for each of the writes.

When writing random patterns, the disk will not be able to rationalize in this way. Seeing as the cache will be full before the same erase block is being written to again, the SSD will have to perform an erase-write or copy-modify-write operation for each of the requests.

This effect is what is commonly referred to as write amplification. Simply put, this can bee seen as a side effect of not being able to perform in-place writes. There are, on the other hand, several ways to try to prevent write amplification. Newer SSDs fights the effect of write amplification in the FTL [23], making an in-place write operation at file system level more symmetric in performance. We have seen that some of these benefits can be achieved without such improvements on the FTL, by moving hardware architecture aware logic to either the file system or application. Though this is the case with magnetic disks as well, workarounds have historically been done in file system.

Error correction

As in magnetic disks, SSD provide a way detecting bit errors at low levels. Knowing that a cell will loose its ability to properly store data after a certain number of writes, the SSD-controller needs to be able to handle erroneous pages in a graceful manner. To detect errors, each page have an alloted space for ECC. This makes it possible to check the consistency of the data on writes. The ECC will be used to handle a given number of damaged cell, but will at some point reach an uncorrectable amount of noise. This page will then be marked as invalid, and no longer be used by the FTL. To be able to provide the same capacity over time, the SSD will have to keep a spare pool of pages, just for this reason. Keeping a spare pool means that the SSD will be able to offer the same capacity to the OS, even if a small number of pages are worn out or otherwise defective.

3.2.4 Future

The availability and maturity of SSD-technology are changed drastically over the last couple of years. Having gone from being a vastly more expensive technology that proved better in only a small subset of scenarios, SSD are now considered to be equal in performance to magnetic disks, if not better in many cases. Recent tests also report that newer enterprise type SSDs will outperform expensive RAID-setups [24].

Scalability

The usage of Flash in SSDs does however impose a few challenges as demand for capacity and bandwidth continue to expand. As we have seen in section 3.2.2, an SSD will be built up of multiple Flash banks. Several of these will be connected to a FBC, giving the internal SSD-controller the possibility of processing some information in parallel. When increasing the size of each Flash memory bank, without increasing the number of FBCs, we might find that bigger capacity drives will have a tougher time achieving the same level of performance. There are however solutions for this. In magnetic disks, we will quickly reach the limit to internal disk scalability, simply because there is a limit to the number of platters it is possible to stack on top of each other. In SSDs, this will not be an issue. As mentioned in [24], increasing the number of channels in SSD, will effectively directly increase the possible level of parallelization.

ATA Trim

In June, 2007, Technical Committee T13, a committee responsible for the Advanced Technology Attachment (ATA) storage interface standardization, started the process of extending the ATA-standard with a new attribute, referred to as the *ATA Trim* attribute [25]. This attribute is the proposed solution to the challenges discussed in section 3.2.3. Today, the

abstraction between storage device and file system removes all knowledge about validity of data from the disk. Because of this, a disk will have to treat all data as currently in use, even if a file can be deleted from the file system. In magnetic disks, this has not been view as a problem, as an entire block will pass the disk head on reads, and entire blocks will be written at a time. There is, in other words nothing to gain from ignoring dead data, as a block containing nothing but dead data will be ignored by the file system.

SSDs, will however, as we have seen in section 2.3.3, work with two different block sizes. One for erase, and one for reads. When writing a block, the SSD will need to consider this data valid, even if it is later deleted in the file system. This also means that once a block is used in an SSD, it will be considered valid for the life time of the SSD, unless told otherwise[4]. By knowing some of the blocks can be discarded, the wear leveling algorithm can also move data more freely around, as not all blocks will be used at all times.

3.2.5 Summary

In this section, we have looked at the technology behind SSDs. We have seen that Flash cells are at a point where production and technology are mature enough to make storage devices capable of competing with magnetic disks. We have also looked at the benefit we get using a storage technology without moving parts. In section 3.2.3, we have discussed some of the challenges SSDs are faced with when using these Flash cells for bulk storage.

[4] A hardware reset of the SSD, invalidating all blocks

3.3 SSDs vs. magnetic disks

As we have seen in section 2.3, the technology in SSDs is built up in a substantially different way than magnetic disks, seen in section 3.1. Magnetic disks consist of a magnetic material that is being manipulated into different states by a strong magnet. These magnetic states are later detected to read out data. SSDs, as we have seen in section 2.3, are on the other hand built up of a large number of Flash cells. Despite these differences, both technologies are able to store data in approximately the same order of magnitude and are in a similar price range.

As all other technologies, both magnetic disks and SSDs have weaknesses. For both SSDs and magnetic disks, certain data access pattern or disk operation might be severely limited by disk attributes.

3.3.1 Magnetic disks

Because of the moving parts in magnetic disks, random seeks will cost time. As we have seen in section 3.1.1, data scattered across the disk will result in the disk performing time demanding repositioning of the disk head in relation to the surface of the disk. The time needed for this is directly, but not exclusively, influenced by the rotational speed of the disk. Higher-end disks usually try to minimize the seek latency, as well as increase bandwidth, by having a higher rotational speed. It can also be challenging to scale performance in disks as size increase. Having multiple disk platters stacked on top of each other will increase density within a single disk. As disks will only have *one* disk arm, the only parallelization it can achieve is to read cylinders at a time.

Disk pricing						
Model	Type	Capacity	Latency	MB/s read/write	Price (NOK)	NOK per GB
WD Caviar®	HDD	1 TB	-	111 / 111	750	0,73
WD VelociRaptor	HDD	150 GB	4.2 ms	126 / 126	1.325	8,83
Seagate Cheetah 15K.5	HDD	300 GB	3.5 ms	125 / 125	3.795	12,65
Kingston V Series	SSD	128 GB	-	100 / 80	1.899	14,84
OCZ Solid Series-SATAII	SSD	250 GB	350 μs	155 / 90	4.795	19,18
Intel® X25-M SSD	SSD	160 GB	85 μs	250 / 70	3.795	23,72
Kingston E Series	SSD	64 GB	75 μs	250 / 170	6.895	107,73

All prices are gathered from komplett.no

Table 3.1: Pricing and capacity of different storage devices as of July 2009

3.3.2 Solid State Disk

As we have mentioned in section 3.2.1, an SSD does not have the ability to perform in place writes. It can be argued that no magnetic disk can do this either, as it will always need to write an entire sector at the time. The big difference between the two is that SSDs are forced to erase an entire erase block to change data in a page. As we mentioned in section 2.3, an erase block will be around 128 kB. However, it is possible to write only single pages, or typically 4 kB. A sector in a magnetic disk will typically be 512 B, making small writes more flexible, though file systems usually works at 4 kB.

3.3.3 Cost

As we have seen in section 3.2.1, there *is* a difference between the cost of SSDs and magnetic disks. In table 3.1, we have compiled a list comparing different consumer-available storage devices. We can see from this list that there is still a huge gap between SSDs and magnetic disks when it comes to available capacity and price per capacity. This means that in any situation where both SSDs and magnetic disks meet our requirements for

performance, magnetic disk will be the preferred choice.

To take implication further, we could say that to consider an SSD over a magnetic disk, it will either have to perform in some way magnetic disk cannot, or to perform better for a given price. As luck will have it, modern SSDs can in some cases do both. We know that SSDs have better performance when it comes to random access reads, and on this point easily beat magnetic disks.

3.3.4 Capacity

The capacity of magnetic disks has proven to more or less follow Moore's law, increasing with a factor 2 every 12 months [13]. This *law* also seems to be valid for SSDs, with gate density being constantly improved. With SSDs having a considerably higher cost per capacity, as seen in table 3.1, the main reason for SSDs being sold with a lower capacity can be said to be cost. We can see this confirmed with recently announced SSDs, matching the 1 TB storage capacity found in magnetic disks [26].

3.3.5 Access time

The access time in disks, regardless of technology used, is determined by the physical limitations of how fast data can be retrieved from the medium in which it is stored. In magnetic disks, as we have discussed in section 3.1.1, this is the limit of how fast the disk arm is able to move to the correct location combined with the rotational speed of the disk platter. In SSDs, as discussed in section 3.2.2, the limitation, for read operations, is how fast the Flash memory chip is able to send data back after a request has been sent and how fast the FTL will handle the level of abstraction in the drive. For write operations in SSDs, the FTL will have much more to say. The main limit is how fast an SSD can perform an erase operation, but by having a smart FTL, the need to erase an erase block can be minimized.

3.3.6 Zoning

In section 3.1.1 we have seen that the different tracks in a magnetic disk has different storage density. This is due to the fact that the disk is divided into what we call zones, giving the outer tracks of the surface more sectors. By having a constant rotational speed, this means that a block at the outer rim of the platter will pass faster under the head on the disk arm than one located near the center. This effect is called zoning. Because of zoning, the location of the block being read or written will determine what speed we will be able to achieve.

SSDs do not, however, suffer from this effect. As we have seen in section 3.2.2, the way the Flash memory banks are connected together, where data is placed will not have an effect on the transfer speed or the latency. Even in the case where this would matter, SSDs would most likely not behave in the same matter as magnetic disks, due to the fact that the FTL will constantly change the mapping of logical blocks.

In figure 3.7, we can see an experimental comparison on the effect of zoning between two different SSDs and a magnetic disk. The results are gathered from running a large sequential read directly from the block device at different points of the disk. We see that the SSDs keep a steady bandwidth across the entire disk, while the bandwidth of the magnetic disk go down almost 30% at the end of the disk, compared to the best bandwidth measured at the beginning.

3.4 File systems

To accommodate the level of performance and or reliability needed for specific applications, different file systems will have different areas of focus. In addition to being created with a specific workload or ability in mind, file systems will also be equipped with options to tweak the performance or choose features to provide. All these design choices and consid-

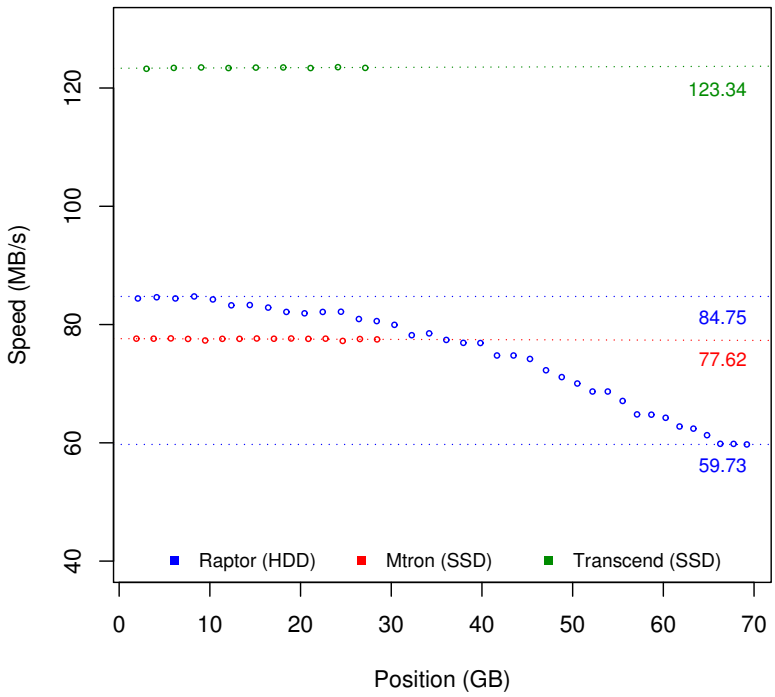

Figure 3.7: Signs of zoning in magnetic disk vs no zoning in SSD

erations will have an effect on the performance the file system will be able to give in certain scenarios.

To make it easier to change between file systems, the Linux kernel provides a common interface for file system operations, called the Virtual File System (VFS). This layer of abstraction will provide a set of interfaces, making it possible to change a file system without changing application level, or even kernel level code. We will later in section 4.2 take a look at performance on different file systems, and will therefore give a short introduction to a few file systems available in Linux.

`ext2` Was until early 2000s considered the de-facto file system for Linux distributions [27, p. 739]. This file system does not use a journal, and will therefore have less overhead on writes, as well as being more susceptible to damaged file system during system crashes.

ext3 Made as an extension to `ext2`, `ext3` tries to address some of the shortcoming `ext2` proved to have. `ext3` is built up in much the same way as `ext2`, but with the difference of introducing journaling. It is also made to be backwards compatible with `ext2`. The journal in `ext3` is used to *back up* blocks of data as they are changed, making recover from a system crash possible.

ext4 The `ext4` file system, is like `ext3` an extension to the previous extended file systems. Unlike the previous versions, `ext4` introduces extents, a features that will break backwards compatibility with earlier `ext` file systems. This feature replaces regular block mapping, and will give better performance on larger files.

reiserfs Is like `ext3` and `ext4`, a journaled file system, introducing optimizations giving better performance working on smaller files.

nilfs2 A log-based file system. `nilfs2`, or New Implementation of a Log-structured File System, will instead of changing data directly in blocks, perform a copy-on-write, writing a new block with the valid data, invalidating the old one. This gives the file system the ability to provide continuous *snapshots* of the state as the file system ages. To clean up invalidated blocks, a garbage collector will clean unused snapshot as the drive becomes full.

On different disk storage architecture, these file systems will handle quite differently. If we have an SSDs where we easily will experience write amplification, journal writes on every write will be costly. This is one of the reasons why many consider `ext2` to be the most suitable file system for early SSDs. `nilfs2`, and other log-based file systems like `logfs`[5], will fight write amplification, simply because they will evade the problem by writing all changed data in different blocks. As the file system has full knowledge of what data is valid, the file system can in this way reorganize how data is placed when writing changes.

[5] A file system made with Flash memory devices in mind, but at a very early point in development

3.5 Disk scheduler

To handle different types of access patterns under heavy load, the Linux kernel will support different types of scheduling algorithms [27]. These algorithms, will schedule the I/O requests to a block device to suit different workloads. The different algorithms will be optimized for different workloads, but common for all is that they will be optimized for magnetic disks.

NOOP A simple queue, doing nothing to reorder requests. This scheduler will merge a request if it is adjacent to a request already in the queue.

Deadline Requests are put in four queues. Two of these are for reads and two are for writes. Both read and write requests are put in both a elevator queue and a deadline queue. This will ensure that requests are handled within a given deadline, but optimally sorted by position on disk. This scheduler will also queue read requests before write requests, because read requests are considered likely to block.

Anticipatory Uses queueing much in the same way as the *deadline* scheduler, but will in addition try to *anticipate* requests from processes. This scheduler will alternate more between read and write requests, but will also favor read. To anticipate requests, the scheduler will gather statistics about a process, making it possible for a process to get a request served immediately after it is issued.

CFQ The Completely Fair Queuing (CFQ) scheduler focuses on dividing I/O bandwidth equally amongst processes. It will have a large number of queues, and will always insert requests from a single process into the same queue.

As we see, these schedulers focus on solving different issues with I/O request scheduling. The NOOP scheduler will refrain from ordering and only merge requests. This makes it possible for the devices, which will have more knowledge of physical layout, to schedule requests internally to better suit the characteristics of the drive, and will make the firmware

in the devices more complex. We know from section 3.2.3 that the FTL in SSDs will hide knowledge of the physical layout from the OS. This means that optimizations done in scheduler for magnetic disks will most likely be less optimal for SSDs.

The deadline and anticipatory schedulers will both order requests on sector number, to fit a SCAN-pattern[6] on the disk. This will have a positive effect on magnetic disks, but will most likely have less to say on SSDs, as pages are moved around, changing logical address. It can be argued that providing a good scheduling algorithm for a device with little information about physical layout can be hard, if not impossible. As SSD already have an abstraction layer in place that will do queueing and buffering, it can be argued that this layer should be intelligent enough to do disk scheduling as well. Therefore, with an optimal FTL, SSDs should see the best performance with the NOOP scheduler.

3.6 Summary

We have in this chapter looked at different disk storage technologies; magnetic disks and SSDs. Comparing these two, we see that both are able to meet the requirements we have for disk storage today, both regarding capacity and bandwidth. We do, however, see a difference in how the two technologies perform in certain scenarios. In section 3.2.2 and section 3.1.1 we have discussed how SSDs and magnetic disks, respectively, handle random requests, identifying the challenges present in keeping latency low for magnetic disks. We have, in section 3.2.3, looked at the potential issues with SSDs, regarding small random writes. Last, in section 3.3.6, we have seen that transfer speed in magnetic disks depend on placement on disk, whereas the transfer speed in SSDs is constant.

[6] Processing requests as the sectors passes, from low to high sector number

Chapter 4

Benchmark

The performance of both SSDs and magnetic disks can be difficult to summarize with just a few numbers. As we have discussed in chapter 3, certain aspects of a disk might give different performance results, and one might get different performance depending the workload. In addition to these uncertainties, different file systems will store data in a fundamentally different way. All this put together, we have an overall hard time getting a clear answer for what level of performance a given application can expect to achieve, only looking at numbers from datasheets.

We know from section 3.2 that SSDs have the advantage of not having moving parts, giving it an overall low latency. Magnetic disks, on the other hand, have a harder time keeping latency low, due to seek and rotational delay. In this chapter, we will look at how these general performance characteristics add up when faced with specific application scenarios. Our goal is to get a clear profile of both SSDs and magnetic disk, making the choice, when faced with one, a simpler task.

Disk specifications				
Make	MTRON [18]	Transcend [28]	WD Raptor [29]	Seagate [11]
Type	SSD	SSD	HDD	HDD
Size	32GB	32 GB	74 GB	80 GB
Form factor	2.5"	2,5"	3.5"	3.5"
Interface	SATA	SATA	SATA	SATA
Rotation speed	-	-	10.000 RPM	7.200 RPM
Memory	SLC NAND	SLC NAND	-	-
Access read	0.1 ms	-	4.8 ms	8.5 ms
Access write	-	-	5.2 ms	9.5 ms
Max read	100 MB/s	150 MB/s	-	85.4 MB/s
Max write	80 MB/s	120 MB/s	-	-
MTBF (hours)	1.000.000	1.000.000	1.200.000	600.000

All numbers are from datasheet provided by manufacturer

Table 4.1: An overview of disks in benchmark environment

4.1 Benchmark environment

In our benchmark environment, we have two different SSDs and two magnetic disks. One of these magnetic disks are being used as system disks, and will only be used in some test scenarios, for comparison purposes. Information about the disks, as provided from datasheets is available in table 4.1. We will, for simplicity, refer to make of disk when talking about these in our benchmarks. Our test PC is suited with a Core™2 Duo 2.66GHz CPU and 2GB of Random Access Memory (RAM), running Ubuntu 9.04 with Linux kernel 2.6.28-14.

The Seagate disk in table 4.1 is included to better understand the impact of attributes in magnetic disks, as latency in this disk is quite much more than in the Raptor. We can see a comparison of the seek times of these two disks in table 4.1. It should be noted that the Seagate disk is also being used as a system disk. This will be taken into consideration in the tests performed on this disk.

When choosing disks for benchmark, we have focused on mid-range alternatives, both for the magnetic disks and SSDs. There's a few reasons for this, money being one, but it is also interesting to see what we can be able to achieve with relatively cheap hardware.

In section 3.2 we have talked about FTL and wear leveling. According to the Transcend datasheet [28], this SSD uses wear leveling techniques. Inside this disk, we can see 16 Flash chips of 16 Gbit capacity, giving a total of 32 GB storage. As the entirety of this storage is reported to the OS as available storage, we can make some assumptions about, not only the physical layout, but the wear leveling algorithm.

As we have discussed in section 3.2.4, future SSDs will be able to more effectively perform wear leveling by having *some* level of idea about the validity of data. As this drive does not have ATA-Trim capability, we know that it will have to regard all data as valid at all times. In other words, this means that the drive, or more specifically, the FTL, will have to have all physical blocks mapped to logical blocks at all times, seeing as no spare space is available.

4.2 File system impact

The last decade, magnetic disks have ruled the realm of data storage. This has not surprisingly had an effect on the design choices of file systems. It is common to store inode information on places of the disk that give good performance, but in an SSD, as we have seen in section 3.2.3, this will produce an increased amount of erases of single blocks, unless handled by the FTL. How good this is handled, is much up to the FTL.

In this section, we will try to understand what impact different file systems have on magnetic disks and SSDs. Our goal is to get an overview of what file systems is best suited for the different technologies, as well as try to explain the reason behind this. By testing different file systems on both

magnetic disks and SSDs, we are hoping to gain insight to how the FTL handle some of the assumptions done in these file systems.

4.2.1 Test scenario

To discover the level of impact the file system has on performance, we have tested a set of different file systems across three different disks. These disks are listed in table 4.1, and are a Western Digital Raptor, Transcend and Mtron, one magnetic disk and two SSDs respectively. By having two kinds of SSD, we might see a potential impact of different FTL-implementation.

We will in this test scenario try to focus mainly on the effect we receive by using different file systems. To do this, we will test four different operations; *sequential read*, *sequential write*, *random read* and *random write*. Each of these operations are run, using the iozone benchmark utility [30], on a large file located in the file system. It's important to note here that we are not testing how efficient the file system is able to do operations across a number of files, but how the most basic operations perform.

In addition to testing the commonly used Linux file systems, `ext2`, `ext3` and the newly included `ext4`, we have tested on `reiserfs` and `nilfs2` as well. `reiserfs` is, like `ext3` and `ext4` a journaled file system. This file system is know to have better performance on smaller files, and can therefore be interesting to test on SSDs. `nilfs2`, or New Implementation of a Log-structured File System, is a log-based, snapshotting file system, recently included into the development branch of the Linux kernel. We have tested `nilfs` with the default block size of 4096 bytes, meaning that, if we are lucky enough to guess the right erase page size, the file system will only write an entire erase block worth of data at a time.

Log-based file systems is regarded as well suited for SSDs with less effective FTL, as it will generally avoid in-place writes. Instead the file system will copy the content to memory, modify, and write the changes to a fresh block. Because this file system also provides snapshotting, the old block

will then be invalidated, and later recycled by a garbage collector. Because of this way of writing modified data, the integrity of the file system will be preserved, even if a write operation gets corrupted. It also means that we will spend a lot of capacity on invalidated blocks when modifying data.

4.2.2 Results

When comparing the results from the two SSDs, seen in figure 4.1 and figure 4.2, we observe many of the same characteristics. Both perform relatively close to that of advertised speed when doing random reads, seen in figure 4.6, with Transcend averaging at 118 MB/s, and Mtron averaging at 108 MB/s. The Raptor disk, on the other hand has an average 16 MB/s on random reads. This due to the symmetrical latency properties of SSDs, discussed in section 3.2.2. It is however interesting to see that big impact this latency has, when it comes to doing random writes.

On sequential reads, we see that both SSDs, across all file systems, achieve a lower throughput. This can most probably be attributed to the fact that Flash memory banks are channeled. When getting a series of requests for data located on different channels, the SSDs will be able to handle these requests in parallel.

Comparing the write performance of the two SSDs in figure 4.5, we see that the Mtron SSD achieve a much greater speed than that of the Transcend SSD in all filesystems but `ext4`. An example of this is in `ext2`, where Mtron write at 83 MB/s, and Transcend write at 49 MB/s. In `ext4`, however, we see Transcend writing at 86 MB/s, while Mtron writes at 78 MB/s. This is evidence the limitation of write speed on the other file systems does not lie in the Flash memory used, but the FTL.

With write speeds, we see an overall trend that the SSDs are able to match that of the magnetic disks in sequential writes, but have trouble keeping up on random writes, seen in figure 4.5 and figure 4.7 respectively. This is not surprising, as magnetic disks will have the same basic latency, caused

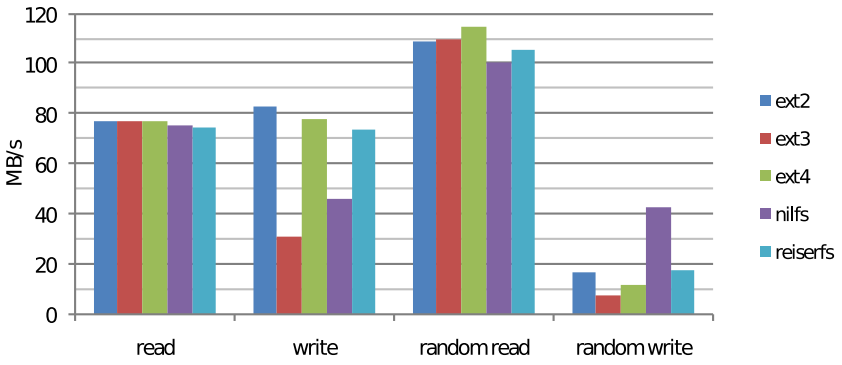

Figure 4.1: Multiple file systems on Mtron SSD

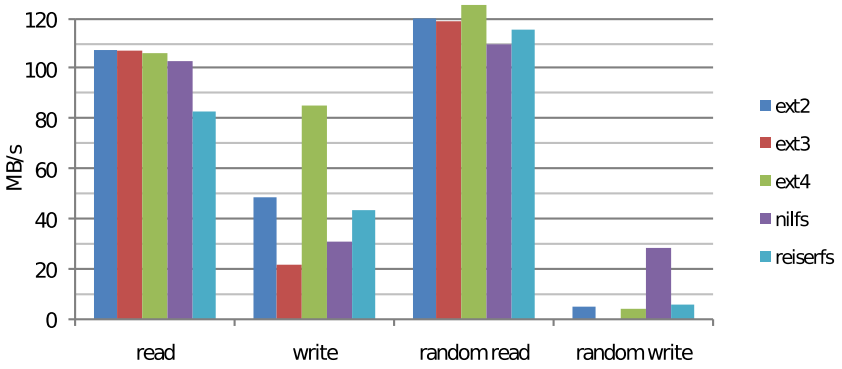

Figure 4.2: Multiple file systems on Transcend SSD

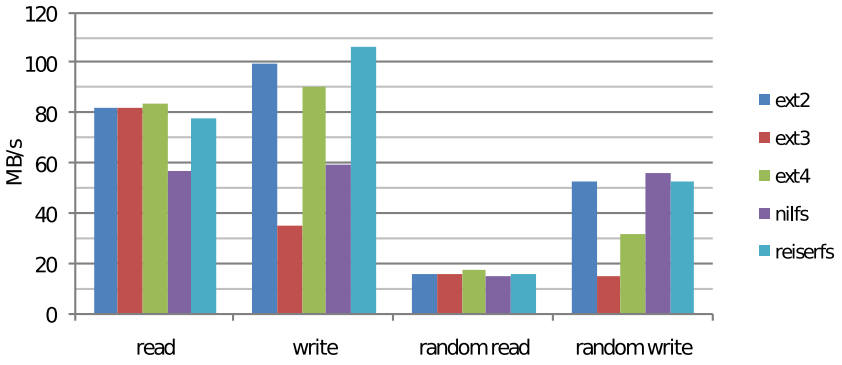

Figure 4.3: Multiple file systems on Western Digital Raptor

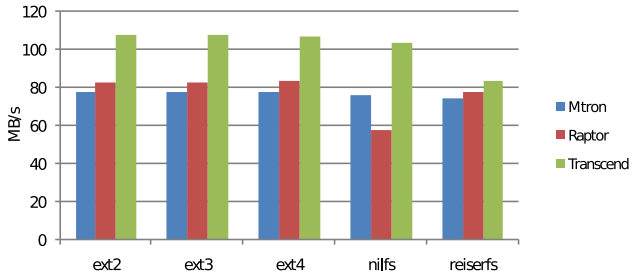

Figure 4.4: Sequential read operations across multiple file systems

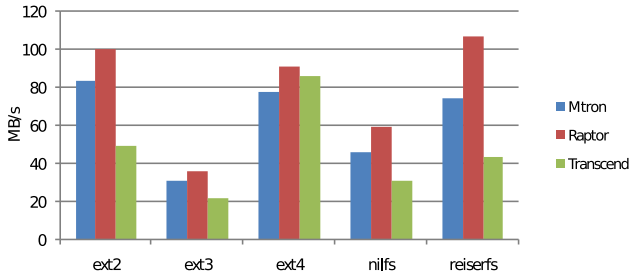

Figure 4.5: Sequential write operations across multiple file systems

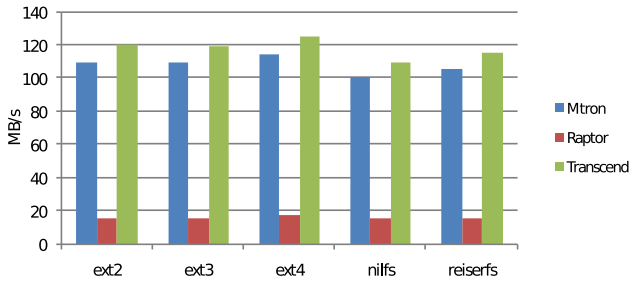

Figure 4.6: Random read operations across multiple file systems

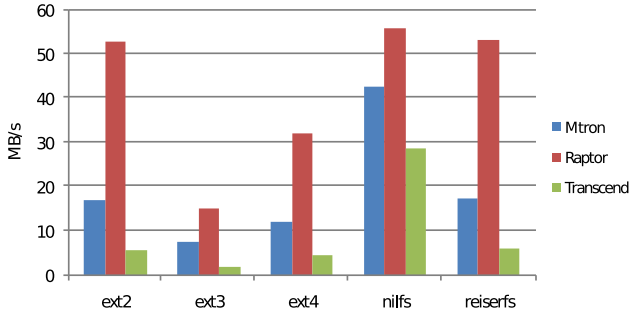

Figure 4.7: Random write operations across multiple file systems

by moving parts, when doing a write operations, as when doing a read operation. SSDs on the other hand, as we have seen in section 2.3, have a much higher write latency than read latency, and will because of this have a highly asymmetrical performance.

It is also interesting to see how the SSDs perform on random writes when using `nilfs2`. On the Transcend SSD, `nilfs2` is able to achieve 28.7 MB/s, whereas the next best result is 5.7 MB/s, using `ext2`. The Mtron SSD, getting overall better results on writes, see a similar improvement, going from 17.0 MB/s in `ext2`, to 42.5 MB/s in `nilfs2`.

4.2.3 Summary

We have, by running benchmarks on two SSDs and one magnetic disk, been able to compare the impact different file systems have on the different architectures. In our tests, we have observed an overall similar performance on sequential reads, though slightly favorable to SSDs, especially the Transcend disk. We have seen that SSDs have a clear advantage when it comes to random reads, due to no mechanical parts. Mechanical parts give, as we have discussed in section 3.1.2, large seek and rotational delays

A general observation over the results of both sequential and random reads across all disks is that file system has little to no impact on these operations. This is especially true for random reads, where the standard deviation over all file systems is 5 MB/s in the SSDs, and 1 MB/s on the magnetic disk. This tells us that there's a high probability that the limitations for random read speeds lies directly in the physical attributes of the disks. In the case of magnetic disks, latency is the bottle neck, as we see we get better read speeds when read sequentially, whereas on SSDs, Flash memory bandwidth is the limit. This can be solved, as we have discussed in section 3.2.4 by having multiple channels, effectively multiplying the bandwidth by the number of channels.

By having tested a log-based file system, `nilfs2`, on the SSDs, we ob-

server that the FTL in neither SSD remove penalty of in-place writes. An optimal FTL would give the SSDs the same level on performance on a file system not aware of this constraint as one trying to minimize in-place writes. Instead of writing modified data back to the same block, `nilfs2` will do a copy-on-write operation, writing a new block with the data and then marking the old one as invalid and unused. We also observe that the sequential writes using `nilfs2` are only slightly better on both SSDs, with 46 MB/s to 43 MB/s in Mtron, and 31 MB/s to 29 MB/s for sequential and random writes respectively. This means that `nilfs2` is able, even though sequential write performance drops, to make writes more symmetrical in nature, and therefore more predictable.

We observer that using a `nilfs2` on the Raptor disk gets the highest bandwidth for random writes, at 56 MB/s. This implies that SSDs should be able to achieve close to the best possible random write speeds as well when using `nilfs2`. In magnetic disks, the drop in speed from sequential to random writes can be explained by latency from rotating the platter and performing seeks. We do however have a constant latency in SSDs, so any drop when doing random writes can, most likely, be attributed to the implementation of the FTL, or to be more exact, the effect known as write-amplification, as discussed in section 3.2.3.

4.3 File operations

We have in section 4.2 seen the impact different file systems can have on magnetic disks and SSDs. These tests tell us how the disks perform on isolated use of the basic operations sequential read, random read, sequential write and random write. When working with large sets of files, these operations are used together, depending on the specific operation. This produces a much more complex performance profile, and it can be hard to foresee what will become a bottle neck.

We will in this section take a look at the performance of doing different file

operations on a large set of files. These operations, which we will look at in detail later, will be tested on the disks listed in table 4.1 in section 4.1. Our goal is to get an understanding of what kind of performance we can expect from magnetic disks and SSDs under different scenarios working on a large set of files.

4.3.1 Test setup

We know that SSDs can suffer from write amplification, as seen in section 3.2.3, and it is therefor interesting to observe how changing small amount of data in a large set of files will perform. In addition to looking at write performance, we also want to know how time consuming operations such as directory tree traversals handle on SSDs, as opposed to magnetic disks.

For the basis of our tests, we will use a recent `git clone` of the Linux kernel repository. There are multiple reasons for the choice here. First off, the Linux kernel has large amount of files, almost 29.000, distributed over a reasonable amount of directories, just under 1.800. As a code base, it is quite big, at just over 750 MB. This means that we use both quite a few inodes as well as a bit of disk space. The reason we have chosen to use a `git` repository, is not only that it gives us the possibility to play back activity, but that this activity is actual development history in a project, not only simulated requests.

For the tests, we are using the `ext4` file system, this is mainly because it is a modern file system, made as an improvement for the much used and tested `ext3`. This file system has recently been included into the Linux kernel, making it reasonable to expect future Linux systems to have this file system as a default option. This is a big argument for testing on this file system, as in systems where many applications work together, we would want a modern, general and well tested file system. From our results in section 4.2.2, we can see that `ext4` perform reasonably well in all cases

but random write. This is most probably because of the fact that `ext4` is a journaled file system, making it less prone to error on interrupted writes.

4.3.2 Checkout of Linux kernel repository

To test how the disks performs in file intensive scenario, we will do repository operations on the Linux kernel source. For each release candidate of a Linux kernel version, a git tag is created. Tags do, in other word, provide snapshots in time of the development history. To create file system activity, we will do a `git checkout` of these revisions. A `git checkout` simply changes state in the entire directory tree to match that of the state in the repository when tagged. It is safe to assume that most of the differences are very incremental in nature, and that they will change a large number of files. In this scenario, we have run a `git checkout` of 160 tags, in order, of the Linux kernel source code.

The workload generated by a `git checkout`, will have much in common with many other application scenarios. One checkout will read data from a internal database type structure. Calculate changes in files, read meta data from a number of files and write changes to a number of files. This will mean that we have a very mixed workload, depending on many different aspects of the disk. Random read performance will highly influence the reading of meta data, writing small changes to a wide range of files will depend on the ability of the disk to do random write operations.

The tests have been run on all disks shown in table 4.1. We should, however note that the Seagate disk in this table is used as system disk. This disk is included so we can observe how the performance will be when sharing resources. With the increased latency, seen in table 4.1, and resource sharing, we can expect less performance from this disk. This will give us some idea of a *worst case* scenario of this kind of task.

Results

In table 4.2 we can see the resulting run time of running 160 consecutive checkouts of different tags in the Linux kernel git repository. These results are also listed in Matching our expectations from earlier in this chapter, the Seagate disk gets the lowest performance in the test, with 920.28 seconds run time. Shortest run time, we achieved on the Raptor disk, clocking in at 178.86 seconds. One of the reasons behind the run time in the Seagate disk being much higher is the fact that we're sharing this disk with the system

Disk	Time	Stdev	Mean
Seagate	920.28	8.84	5.75
Raptor	178.86	1.43	1.12
Mtron	184.76	1.50	1.15
Transcend	236.65	1.99	1.48

Table 4.2: Statistics of time used on checkout of 160 git tags

Some variance is to be expected on tasks like these, as some of the tags will be at a point in the revision history where there has been more changes to the files. These checkouts will take more time, as a `diff` will have to calculated, located on disk and merged with the current version. We can see in figure 4.8, how the time accumulates as we checkout tags. Though there are variations in the time it takes to perform a checkout, we can clearly see a trend here.

Block trace

To better understand the performance the different disks have achieved on the `git checkout` tests, we will take a look at a block trace of activity generated by the git command. To trace traffic, we have used `blktrace`, a program for tracing block layer IO. This will give us the ability to trace multiple events in the block layer of Linux.

In figure 4.10, we can see a block trace on all disks during a `git checkout`. This trace is parsed from the data provided by `blkparse`, a part of the

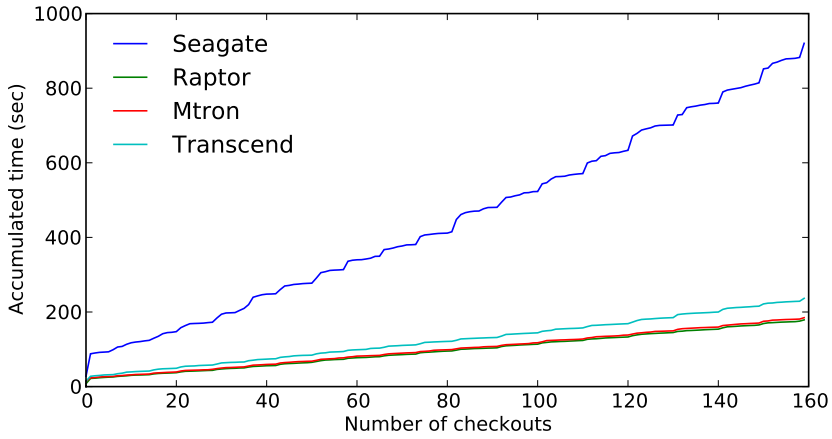

Figure 4.8: Accumulated time of checking out 160 git tags

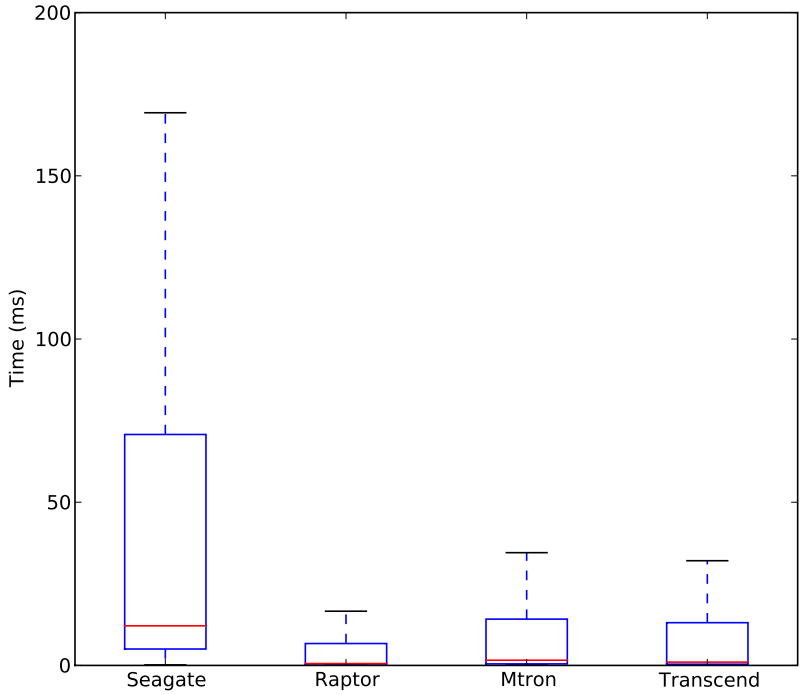

Figure 4.9: Boxplot of time spent on block requests

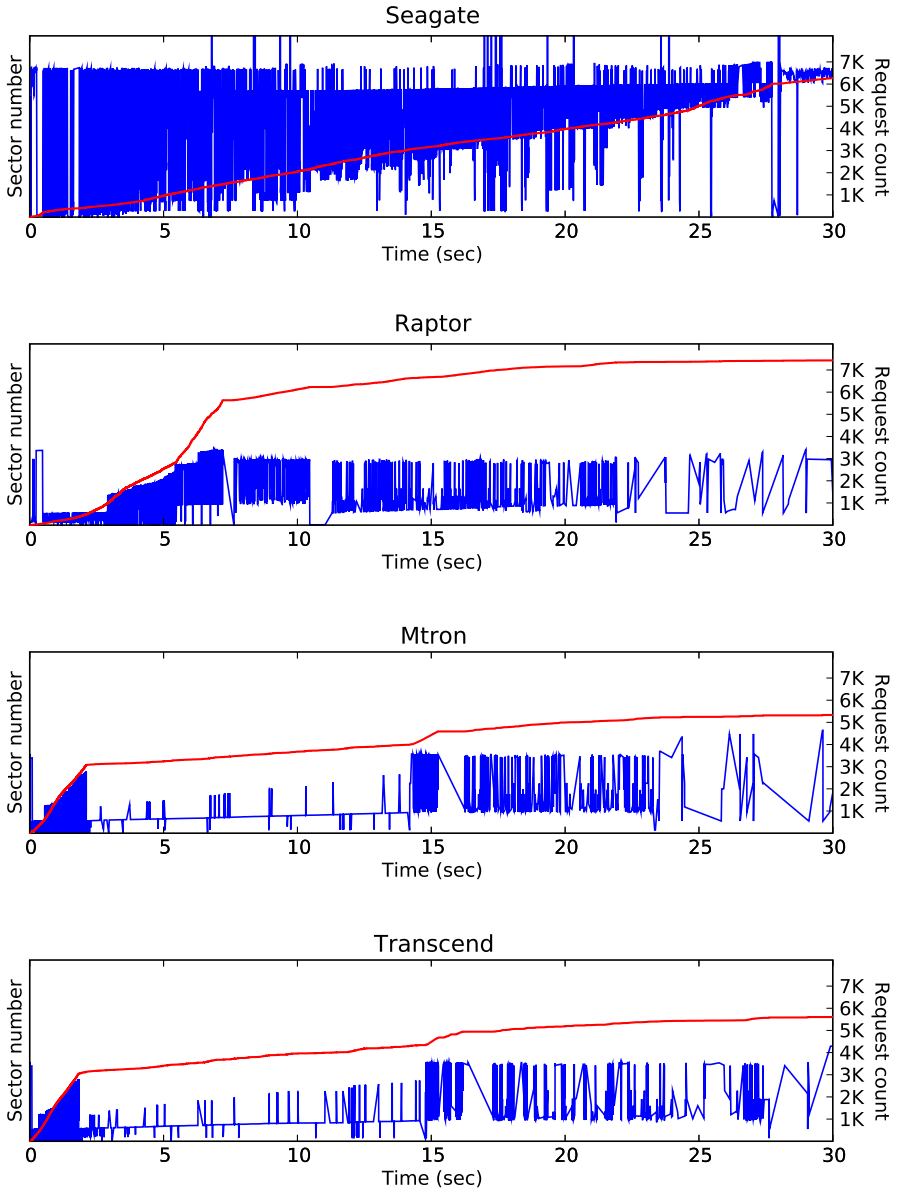

Figure 4.10: A trace of block requests when doing a `git checkout`

`blktrace` software package. An example of this output can be seen in Appendix B, showing block requests as they are queued, merged and completed. What we see traced in this graph is requests of blocks, being reported as they are completed on the disk. This is, in other words, the activity of the disk, not how blocks are put in the queue. The red line seen in the graph is a representation of the accumulated number of blocks processed by the disk. This does not necessarily translate directly to data transfered, but it gives us an idea of how much activity the disk is able to process.

What we see in the graph is the first 30 second of the series of `git checkout`s. During this time, we can clearly see both SSDs serving block requests from the block layer at a much higher rate than both the magnetic disks in the beginning, but changing during the course of the test. The Raptor disk clearly has an advantage when it comes to write operations in general, and will catch up when doing these. With the Seagate disk, which is being used as a system disk in addition to servicing the `git checkout` request, we see the signs of resource sharing, as the time between completed blocks are much higher than the increased latency would indicate. The request count indicates that this disks serves about the same number of block requests as the other disks, the difference being that the other disks will have time to merge more requests together, and thus, serving more data per request.

We can in figure 4.9, see a boxplot of time spent from block requests are inserted into queue to block requests are completed from the disk. The red line indicates the median of the samples, the bottom and top of the box represents the upper and lower quartile of the samples, respectively. Here we see that, despite being less efficient on random reads, the raptor disk see both a lower variance in process time for block requests, as well as having quicker response. These results are across the entire test, and tell us that, in a scenario like this, the ability of magnetic disks to do faster random write operations than that of SSDs weighs up for being less effective at random read operations. This can also be said the other way around,

that SSDs faster random read operations compensate for its slower random write operations.

In table 4.3, we can see the mean time of how long a block requests takes from insertion to completion, divided into read and write operations. From this table we can see that both SSDs are close to the Raptor disk in respond time for read requests, but fall behind on response time for write requests. Again, we need to note that this is response time for a block request, and tells us little of actual bandwidth. We see that the Mtron disk falls behind the Transcend in request time, but still is able to end up with a shorter total run time. As we know, from section4.2.2, the Mtron disk performs better than the Transcend disk on random writes, with the Raptor disk dwarfing both these. All this points to that that it is very likely that random writes is the critical factor in this test scenario.

Disk	Read	Write	Combined
Seagate	22.13 ms	1133.59 ms	255.84 ms
Raptor	4.70 ms	41.00 ms	10.01 ms
Mtron	4.41 ms	145.22 ms	50.65 ms
Transcend	8.22 ms	106.78 ms	39.28 ms

Table 4.3: Time in queue for read and write operations

4.3.3 Scheduler impact

As we have discussed, in section 3.5, the disk scheduler will have an effect on the efficiency of the disk in different scenarios. To give us an idea of what role the scheduler will play on the different disks seen in section 4.3.2, we have run the same test on the four different schedulers in Linux. As we know that the different schedulers will be optimized for different workloads, we are not necessarily interested in which scheduler will give us the best performance for this exact scenario. What we *do* want to observe is how the different disks react to the different disk schedulers, and how big role the underlying architecture will play.

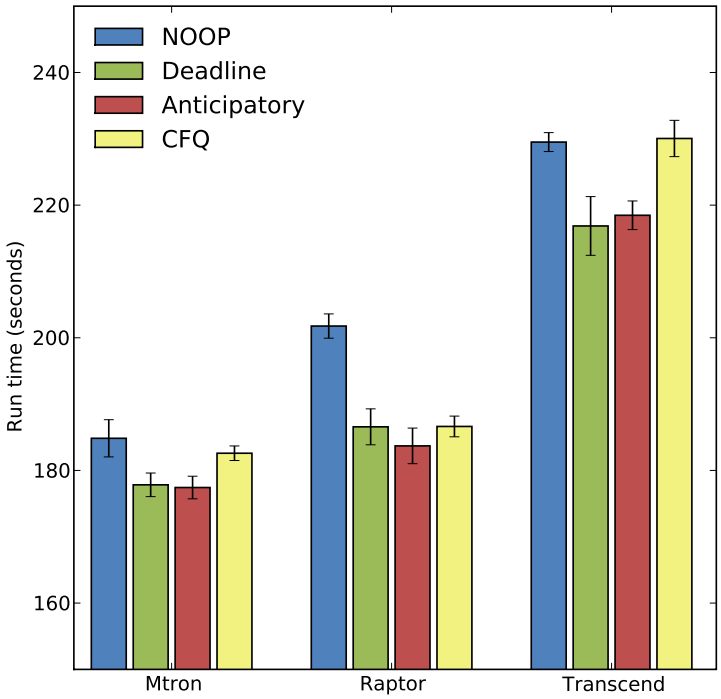

Figure 4.11: Disk scheduler performance impact on `git checkout`

In figure 4.11, we can see the total run time of the same test as in section 4.3.2, using the four different disk schedulers found in Linux. The error bars in the graph show the standard deviation across a series of tests. The Raptor disk, we see have about the same results within the margin of error for all the schedulers, except for NOOP. This is, as we have discussed in section 3.5, expected, as all three schedulers will optimize for certain workloads. In this test have a mixed scenario, using many different aspects of the disks, and some properties of the schedulers will, because of this, at times limit the throughput we are able to achieve. Both SSDs, however, see tendency towards the default Ubuntu Linux scheduler, CFQ, being on par with NOOP.

As we know CFQ will mostly focus on giving different processes a fair

share of I/O time. In our test, we only have one process using the resources from the disk, but will still have some system processes with disk activity. The CFQ scheduler will most likely give better results than NOOP on magnetic disks, as requests from the same process tend to be adjacent to each other, and therefore create less disk seeks. This is also reflected in the results. For SSDs, we see that both the deadline scheduler, and the anticipatory scheduler get better results. A reason for this can be that these two will have a separate queue for write requests, making it more likely that the FTL can consolidate writes.

4.3.4 Inode sorting

Another common operation on large sets of files, at least like our scenario where these files are part of a relatively large code base, is packing and compressing for distribution. The *packing* part here usually means some sort of archiver tool, like `tar`. `tar` can take a series of command line options for filtering content put in the archive, but we will assume that tar is given default options, packing all files within a directory. When processing files, `tar` will recurse through all subdirectories, adding content as it is discovered. For magnetic disks, this can be less than optimal, as content will be added as it is found in the directory tree, not as it is stored on the disk. As the file system ages, chances are files will get more and more spread around on the disk, making this weakness more apparent.

In Lunde [31], we see a proposed improvement of `tar`, optimizing the read order of blocks. In this improved version, the directories will first be scanned to check where the data blocks of files are located, then this list will be sorted by location on disk. This first traverasal will add an extra Central Processing Unit (CPU) overhead, but the theory here is that run time will be reduced in total because of less latency from scattered disk position requests. In Lunde [31], we observe over 80% reduced run time as an effect of the improved `tar`.

With the introduction of SSDs, as discussed in section 3.2.2, we have a reduced and constant latency for read operations. It is therefore interesting to see how an SSD will perform both with and without inode sorting in the same scenario. We have modified this implementation to simply read data to `/dev/null` (due to some portability issues), and might because of this see that run time of sort will be a bigger part of the total run time.

In figure 4.12 we can see the Raptor disk performing in a similar way is in the reference tests, improving time with 67%, from 37.7 seconds to 12.3 seconds. The SSDs, however, see a much lower degree of difference between sorted and unsorted traversal of files. The Mtron disk see no improvement from sorting, using 8.9 seconds in both implementations. The Transcend disk, on the other hand see a slight improvement of 3%, from 9.8 seconds to 9.5 seconds. With both SSDs performing better on random reads in `ext4`, as seen in the results in section 4.2.2, the slight difference in time for the Transcend disk might be attributed to a difference in the implementation of the FTL.

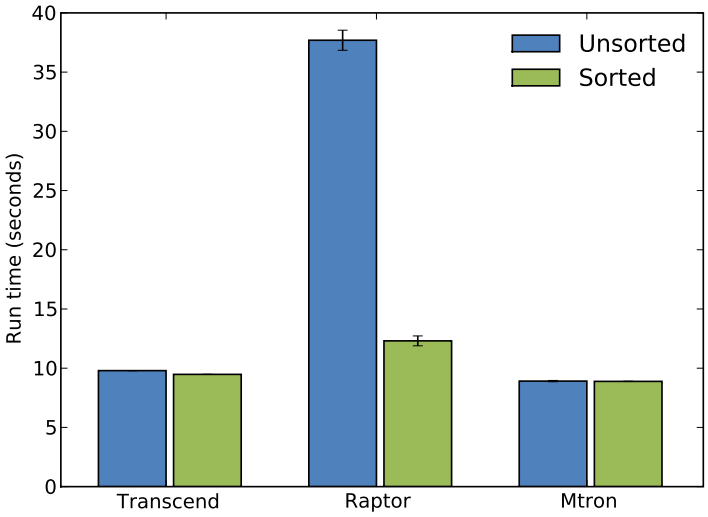

Figure 4.12: Inode ordering impact

4.3.5 Summary

In this section we have looked for possible differences in file intensive operations. We have seen that in a scenario like running `git checkout`, magnetic disks are able to outperform SSDs, due to better performance in mixed (random read, random write) scenarios. By studying trace data collected during benchmarking of in section 4.3.2, we have seen that SSDs ability to do random reads fast, in some scenarios, is only able to compensate for slow writes at a minimal degree. We have also seen that the disk scheduler can have an impact on what performance we are able to achieve on different disk architectures.

In section 4.3.4 we have seen that on magnetic disks, access pattern in applications can be a limiting factor. By optimizing the order data is accessed in applications, performance can be severely increased. We have also seen that in this situation, SSDs can effectively remove this bottle neck, without optimizing application level code.

4.4 Video streaming

As bandwidth on consumer internet connection continue to increase, streaming video become a more and more viable option to video on media. A challenge, however, when it comes to streaming video, is providing a cost-effective infrastructure. We will in this section try to understand how the different disk storage technologies we have looked at in chapter 3 performs when serving concurrent video streams to a large number of clients.

4.4.1 Streaming scenario

A streaming service will typically offer a multitude of videos to a large number of clients. Assuming that the demand for content is Zipf-distributed amongst the clients, a small portion of the content will need to be streamed

to a large group of clients. This means that in a system where we have a true on-demand service, a large groups of clients will be streaming different parts of a relatively small video base.

Our test scenario we will simulate a number of clients, concurrently accessing content stored on a single disk. In a real streaming services, we would in some cases delay a video stream for some users, in order to be able to serve multiple users with the same content, but or goal here is to find out what level of streaming concurrency the different types of disks will be able to serve. Because of this, we will assume that all clients stream different portions of the content, effectively creating a worst-case content distribution. This streaming is done by laying out a number of 200 MB files in a otherwise empty `ext4` file system. There are multiple reasons for choosing `ext4`. As discussed in section 4.3.1, it is an extension to the widely used `ext3`, and is amongst the best performers on read operations, seen in section 4.2.2. We will test with a different number of clients, each streaming at 1 MB per second, a bandwidth capable of delivering video in DVD quality or higher.

We have tested the scenario described in section 4.4.1 with a different number of concurrent clients, 50 through 100 in increments of 10 users. The results we have seen from our file system tests in section 4.2.2 is summarized in table 4.4 for both read and random read operations using `ext4`. We see from these numbers that the Raptor disk is unable to give us at best 83.7 MB/s. Streaming 1 MB/s to 90 and 100 clients is, because of this, an impossible task, but as we want to look at how the SSDs handle this amount of clients, it is also interesting to see the effect it will have on the magnetic disk.

Disk	Read	Random read
Raptor	83.7 MB/s	17.4 MB/s
Mtron	77.5 MB/s	114.9 MB/s
Transcend	106.7 MB/s	125.7 MB/s

Table 4.4: Read performance using `ext4`

4.4.2 Results

Our bar graphs show the mean bandwidth across all the concurrent clients in the benchmark. The error bars show the standard deviation across the results. In our test results, we see that all three disks handle streaming to 50 and 60 concurrent clients, with within 99% of the bandwidth requirements, shown in figure 4.13 and figure 4.14 respectively. This tells us that the magnetic disk will, even though reading streams from different positions of the disk, be able to read enough data in each turn, making the operation perform like a sequential read. In figure 4.15, we see that both SSDs, Transcend and Mtron, still is able to give within 99% of 1 MB/s to all clients, and show little to no standard deviation across the clients. The Raptor disk, does however, show signs of starting to fall behind on some clients, whereas some still get the correct bandwidth, resulting in a higher standard deviation.

In figure 4.16, figure 4.17 and figure 4.18, we see results for 80, 90 and 100 concurrent clients, respectively. Here we see a bit of the same tendency as we have seen in the results for fewer clients, only magnified. For 80 clients, we see both the Raptor disk and the Mtron disk, start falling more behind on bandwidth to clients. This tells us, when looking at the results from our file system tests in section 4.2.2 that both disk perform as with sequential read operations, and both disks are able to achieve close to the maximum bandwidth for this test. What is interesting to note for the results of 90 and 100 concurrent clients, is that though both the Raptor disk and the Mtron disk fall behind on the bandwidth requirement, but quite differently. The Mtron disk will give a lower bandwidth to all clients, giving very little variance in how much resources the different streams get. On the other hand, the Raptor disk will give some clients the full bandwidth of 1 MB/s, whereas others will get less than 400 kB/s. The Transcend disk is able to give 1 MB/s to all clients in all these tests.

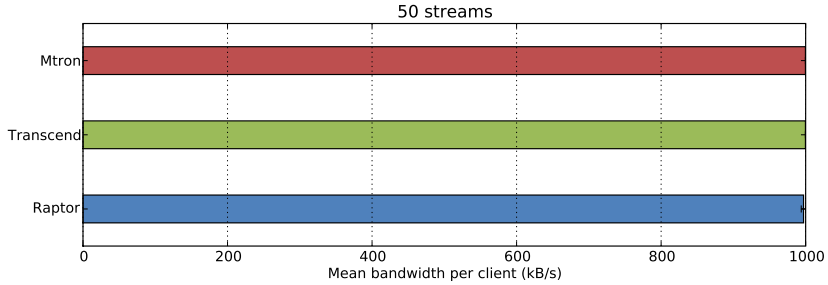

Figure 4.13: 50 concurrent clients, streaming at 1MB/s

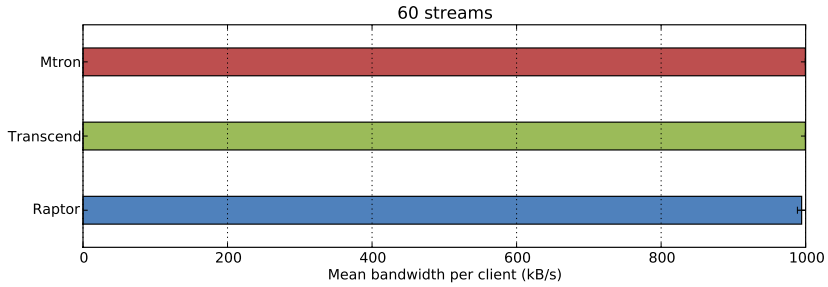

Figure 4.14: 60 concurrent clients, streaming at 1MB/s

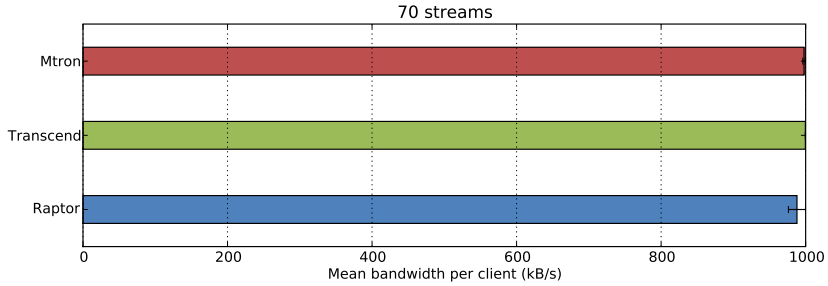

Figure 4.15: 70 concurrent clients, streaming at 1MB/s

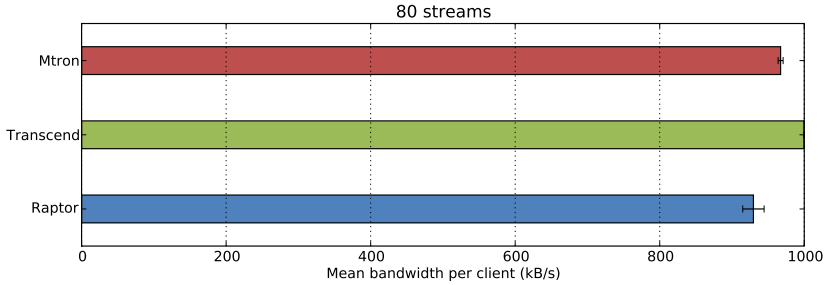

Figure 4.16: 80 concurrent clients, streaming at 1MB/s

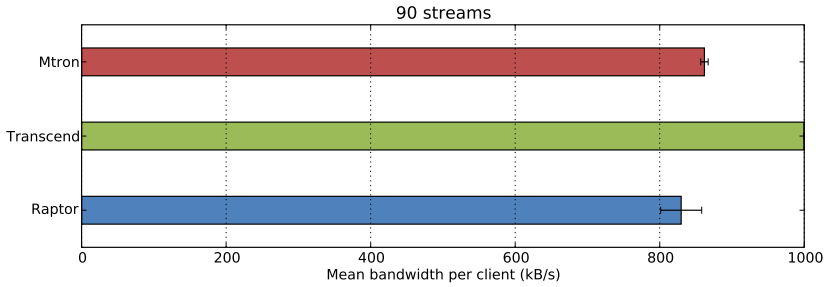

Figure 4.17: 90 concurrent clients, streaming at 1MB/s

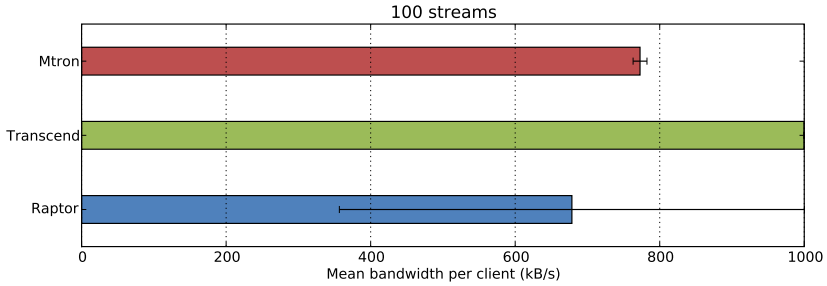

Figure 4.18: 100 concurrent clients, streaming at 1MB/s

4.4.3 Summary

We have in this section seen how the different disk architectures handle video streaming. We see from our tests that video streaming to clients at 1 MB/s will give a sequential-like read performance, even with many concurrent clients, streaming different portions of a stream. From our test results, we see that SSDs handle over-utilization in a more consistent way than magnetic disks. In video streaming, less performance in a consistent way mean we buffer more intelligent, as performance is more predictable.

4.5 Discussion

4.5.1 Benchmark results

From our benchmark, we see that our results fit many of the observations done in section 3.2 and section 3.1. Magnetic disks show an overall low performance on random operations, due to seek time and rotational delay. SSDs show a high performance on random read operations, even showing a higher degree of performance than on sequential reads. This, we most likely can attribute to the fact that an SSD consist of multiple Flash memory chips, connected in parallel, as discussed in section 3.2.2. How much of the effect from these *channels* we see in an SSD, will depend heavily on the FTL, as each channel, can handle requests in parallel.

We see, in section 4.3.4, the effects of these disk properties, when performing file system operations where order of access plays a part. As most applications, in some way or another, will work on a number of files in the file system, there is a high probability that the order these files are read or written in, is a consequence of application design, not disk design. On a magnetic disk, we see we can achieve much better results in the application scenario by simply changing the access order in the application. In SSDs, however, we see that we get the same performance, with both an

original and an optimized application, both performing better than the magnetic disk. We therefore argue that instead of optimizing applications that have a large portion of random read operations, either within files, or simply because we need to do inode lookup on a large number of files, we change disk architecture. Changing an existing application, or even designing an application, for specific access patterns, like sorting inodes by location of data, can be a complex, and sometimes troublesome, task. By changing disk architecture to one better suited for a given workload, we might achieve the same, if not better, level of performance, without the risk of spending time and resources optimizations.

4.5.2 Placement of disk logic

In this chapter we have looked at the performance of SSDs and magnetic disks in different usage scenarios. As we have discussed in section 3.3, SSDs and magnetic disks are built with fundamentally different technologies, even thought they are presented in the same way to the operating system. When changing disk architecture, without doing modifications to application or OS, an SSD will be used as any other magnetic disk would. Many of the big questions today about SSDs is whether or not optimizations should be done at OS-level or device level [32]. There are of course, several sides and arguments to these questions, and giving a text book answer is far from possible. The two biggest arguments can be summed up as follows:

Keep the storage device simple

This solution is more or less the state of magnetic disks as we know them today. Even though newer magnetic disks contain mapping through the Logical-disk Block Address (LBA), this is mostly assumed to be almost a direct mapping to the cylinder, head, sector addresses of the disk. The OS can check capabilities of the drive through common interfaces, mak-

ing it possible for components higher up in the hierarchy to optimize for better performance. This solution gives us almost endless possibilities for customization, and makes it possible to resolve issues by altering the different OS-components. By having all this logic at this level, it also means that the OS will need to be aware of many hardware specific properties to get best possible performance. As hardware architecture changes, so will OS-level code need to change, to best be able to utilize the storage devices.

Put disk-specific logic in the disk

This is generally the solution newer SSDs lean towards. The idea is nothing new, as many Redundant Arrays of Independent Disks (RAID) and Storage Area Network (SAN) systems will want to do optimization of access in ways the operating system is unable to do. This also makes it possible to change between these different types of components, without altering, or even alerting, the OS. Because SSDs connects as any magnetic disk would, changing storage architecture without changing software is, not only possible, but easy. The main reason why this approach has been made for SSDs is likely to be that hardware manufacturers want products quickly into consumer markets when the technology is mature. When the technology quickly becomes available, as we have seen with SSDs, there has been little time do do OS-level performance optimization. In addition, consumers expect to see the potential benefits of these disks in an existing system, without altering the system at software level.

By these two arguments, we see that the latter is chosen for SSDs, simply because it makes it much easier to replace existing disk drives. From our tests, specifically tests with random write operations, seen in figure 4.7 in section 4.2.2, we see that early generation SSDs have trouble hiding the fact that it needs to erase an entire erase block on writes. One can argued that OS-level code should be optimized for this, like file system and disk schedulers. We see from the same tests that `nilfs2` end up with more than twice the bandwidth on random write operations for both SSDs, than

for a commonly used file system like `ext3`.

4.5.3 SSD improvements

As the latency for accessing different parts of the Flash memory is constant, the only different between a sequential write and a random write is the amount of data that is written to a location at a given time. Because of the physical attributes of Flash memory, as we have seen in section 2.3.3, writing data to Flash memory will be done in erase blocks. When the SSD, or more specifically, the FTL know nothing about what data is valid and what is invalid, it will have to keep all data living at all times, making small writes more expensive. We have discussed, in section 3.2.4, the proposed feature of new SSD, making it possible for the file systems to tell a storage device if a block is invalidated.

The SSDs we have used for testing in our benchmark are, early generation SSDs. Later SSDs have better performing Flash memory chips, and not least, a more intelligent FTL. The recently released Intel® X25 SSDs provides a much improved FTL, in addition to better performing Flash memory chips. This may be best seen in the Intel® X25-E [33]. These disks have 10x 4 GB or 10x 8 GB Flash memory chips, giving a raw capacity of 40 and 80 GB. By exposing only 32 and 64 GB to the OS, respectively, the SSD will have 25% of the capacity reserved for use by the FTL. This means that, even without the implementation of ATA-Trim, the SSD have a much needed freedom to keep erase blocks ready, and can do wear leveling, without the need to regard all data as valid.

These improvements all suggest that newer generations SSDs will keep the same level of performance on sequential and random reads. Because of the ability these will have to keep a number of blocks in an erased state, and because the FTL will be able to do garbage collection, without the need to regard all data in the disk as valid, random write operations will be much closer to that of sequential writes, effectively removing the biggest penalty of SSDs.

Chapter 5

Conclusion

5.1 Summary and contributions

In this thesis we have looked at the performance of magnetic disks compared to Solid State Disk. We have, in chapter 4, looked at different benchmark scenarios, to compare performance in magnetic disks SSDs. In section 4.2.2 we have seen that early generation SSDs are able to outperform high-end magnetic disks at random read operations, due to a constant, and low, latency. At the same time, SSDs perform well on sequential read, matching the performance of magnetic disks. This is also reflected in our tests of video streaming in section 4.4.2, where we see that both magnetic disks and SSDs are able achieve their maximum bandwidth, but that SSD handle over-utilization a more predictable manner. By these results, we can see that, when handling scenarios which rely heavily on read performance, either random or sequential, SSD will be able to deliver a much more consistent bandwidth.

As we have seen early on in section 2.3.3, and later discussed in section 4.5.1, random write operations is a performance issue in early generation SSDs, because of write amplification. By using a log-based file system, we see that this write amplification can be, if not entirely removed, strongly re-

duced. This suggests that the FTL implemented in the SSDs used in our benchmarks, is unable to, or does not at all, hide the fact that Flash memory needs to erase an erase block when writing changes to a page.

We have seen, in section 4.3.2, that magnetic disks perform well in workloads with mixed operations. In these same operations, we see that SSDs, even with high write amplification, is able to achieve a total run time, close to that of the magnetic disk. When using different Linux disk schedulers, in section 4.3.3, we see that the SSDs perform better with schedulers that divide operations into write and read queues. This further suggests that the FTL in these SSDs does little or nothing to hide write amplification.

In section 4.3.4, we have seen an example of a file intensive application scenario. This application will traverse a directory tree and process all files contained within all subdirectories. We see that by changing this application to order the requests by block number, a magnetic disk will see a huge performance benefit. This is because ordering block requests will limit the amount the disk needs to seek between each request. For SSDs we see that both ordered and unordered requests give almost identical performance, and that these are better than both cases on magnetic disks. From this, we learn that we in some cases, can get the same benefit as from optimizing application level code, simply by changing disk architecture.

5.2 Future work

As we have discussed in section 4.5.3, much has happened to SSDs in the last few years, and much indicate that newer versions seem to almost completely remove the write amplification effect. For further work, it would be of great interest to see how new generation SSDs, like the Intel® X25 series, handle workloads that we have shown to be a challenge to early generation SSD. By removing the penalty of write amplifications, SSDs quickly become much more attractive to applications with heavy write load, like database systems. Combined with low latency, there are potentially much

to gain from considering performance benefits, both from offloading entire database storage, or just commit logs.

We have not looked at a power consumption comparison between the two architectures, but we know, as seen in section 2.2, that Flash memory uses less power than magnetic disks. It could therefore be interesting to see total cost comparison of a system with magnetic disks, and one with SSD. This can probably give a more realistic image to the total cost of ownership of SSDs, as a number given for price per capacity only tell us part of the story. As newer generation SSDs is closing in on the performance we get from RAID-arrays, it could also be of interest to get a comparison of the cost effectiveness of these versus that of SSDs.

Appendix A

List of Acronyms

ATA	Advanced Technology Attachment
CFQ	Completely Fair Queuing
CPU	Central Processing Unit
DRAM	Dynamic Random Access Memory
ECC	Error-Correction Code
E^{2}PROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
FBC	Flash Bus Controller
FeRAM	Ferroelectric Random Access Memory
FTL	Flash Translation Layer
HDD	Hard Disk Drive
LBA	Logical-disk Block Address
MLC	Multi-level cell
MRAM	Magnetoresistive Random Access Memory
MTBF	Mean Time Between Failures
NILFS	New Implementation of a Log-structured File System
OS	Operating System
PCM	Phase-Change Memory

RAID Redundant Arrays of Independent Disks

RAM Random Access Memory

RPM Rounds per Minute

SAN Storage Area Network

SLC Single-level cell

SSD Solid State Disk

UV Ultra Violet

VFS Virtual File System

Appendix B

Example trace of `git checkout`

Seagate

```
Device   CPU    Seq #   Time (sec)  PID    Action, Mode, Block, Size, Process
  8,1    0      443     3.684757146 23934  G  RM 77736103 + 8 [git]
  8,1    0      444     3.684758448 23934  P   N [git]
  8,1    0      445     3.684759200 23934  I   R 77736103 + 8 [git]
  8,1    0      446     3.684767390 23934  D   R 77736103 + 8 [git]
  8,1    0      447     3.684819135 23934  U   N [git] 1
  8,1    1      463     3.693358108     0  C   R 77736103 + 8 [0]
  8,0    1      464     3.693375442 23934  A  RM 77595439 + 8 <- (8,1) 77595376
  8,1    1      465     3.693375682 23934  Q  RM 77595439 + 8 [git]
  8,1    1      466     3.693376901 23934  G  RM 77595439 + 8 [git]
  8,1    1      467     3.693377550 23934  P   N [git]
  8,1    1      468     3.693377883 23934  I   R 77595439 + 8 [git]
  8,1    1      469     3.693381599 23934  D   R 77595439 + 8 [git]
  8,1    1      470     3.693427170 23934  U   N [git] 1
  8,1    0      448     3.693727687     0  C   R 77595439 + 8 [0]
  8,0    0      449     3.693758990 23934  A  RM 77736111 + 8 <- (8,1) 77736048
  8,1    0      450     3.693759245 23934  Q  RM 77736111 + 8 [git]
  8,1    0      451     3.693760436 23934  G  RM 77736111 + 8 [git]
  8,1    0      452     3.693761233 23934  P   N [git]
  8,1    0      453     3.693761565 23934  I   R 77736111 + 8 [git]
  8,1    0      454     3.693765769 23934  D   R 77736111 + 8 [git]
  8,1    0      455     3.693813272 23934  U   N [git] 1
  8,1    1      471     3.693968435 29272  C   R 77736111 + 8 [0]
  8,0    0      456     3.693977930 23934  A  RM 77595447 + 8 <- (8,1) 77595384
  8,1    0      457     3.693978113 23934  Q  RM 77595447 + 8 [git]
  8,1    0      458     3.693979143 23934  G  RM 77595447 + 8 [git]
  8,1    0      459     3.693979736 23934  P   N [git]
  8,1    0      460     3.693980047 23934  I   R 77595447 + 8 [git]
  8,1    0      461     3.693983662 23934  D   R 77595447 + 8 [git]
  8,1    0      462     3.694030185 23934  U   N [git] 1
  8,1    0      463     3.694531553  4190  C   R 77595447 + 8 [0]
  8,0    0      464     3.694776263 23934  A  RM 77595455 + 8 <- (8,1) 77595392
  8,1    0      465     3.694776515 23934  Q  RM 77595455 + 8 [git]
  8,1    0      466     3.694777829 23934  G  RM 77595455 + 8 [git]
```

```
  8,1    0      467     3.694778584 23934  P   N [git]
  8,1    0      468     3.694778964 23934  I   R 77595455 + 8 [git]
  8,1    0      469     3.694783710 23934  D   R 77595455 + 8 [git]
  8,1    0      470     3.694830240 23934  U   N [git] 1
  8,1    1      472     3.695051220     0  C   R 77595455 + 8 [0]
  8,0    1      473     3.695083367 23934  A   R 77737927 + 8 <- (8,1) 77737864
  8,1    1      474     3.695083628 23934  Q   R 77737927 + 8 [git]
```

Raptor

```
Device   CPU    Seq #   Time (sec)  PID    Action, Mode, Block, Size, Process
  8,16   1        3     0.000007010 22888  G  RM 74103 + 8 [git]
  8,16   1        4     0.000008184 22888  P   N [git]
  8,16   1        5     0.000008867 22888  I   R 74103 + 8 [git]
  8,16   1        6     0.000011235 22888  U   N [git] 1
  8,16   1        7     0.000015823 22888  D   R 74103 + 8 [git]
  8,16   0        1     0.000120473     0  C   R 74103 + 8 [0]
  8,16   0        2     0.000154816 22888  A  RM 74111 + 8 <- (8,17) 74048
  8,16   0        3     0.000155079 22888  Q  RM 74111 + 8 [git]
  8,16   0        4     0.000156418 22888  G  RM 74111 + 8 [git]
  8,16   0        5     0.000157205 22888  P   N [git]
  8,16   0        6     0.000157526 22888  I   R 74111 + 8 [git]
  8,16   0        7     0.000161712 22888  D   R 74111 + 8 [git]
  8,16   0        8     0.000207613 22888  U   N [git] 1
  8,16   1        8     0.000260302     0  C   R 74111 + 8 [0]
  8,16   1        9     0.000298436 22888  A   R 82263 + 8 <- (8,17) 82200
  8,16   1       10     0.000298675 22888  Q   R 82263 + 8 [git]
  8,16   1       11     0.000299903 22888  G   R 82263 + 8 [git]
  8,16   1       12     0.000300669 22888  P   N [git]
  8,16   1       13     0.000301017 22888  I   R 82263 + 8 [git]
  8,16   1       14     0.000305032 22888  D   R 82263 + 8 [git]
  8,16   1       15     0.000351143 22888  U   N [git] 1
  8,16   0        9     0.008269783     0  C   R 82263 + 8 [0]
  8,16   0       10     0.009326639 22888  A   R 81991 + 8 <- (8,17) 81928
  8,16   0       11     0.009326915 22888  Q   R 81991 + 8 [git]
  8,16   0       12     0.009328325 22888  G   R 81991 + 8 [git]
  8,16   0       13     0.009329152 22888  P   N [git]
  8,16   0       14     0.009329496 22888  I   R 81991 + 8 [git]
  8,16   0       15     0.009333709 22888  D   R 81991 + 8 [git]
  8,16   0       16     0.009335997 22888  R   R 81991 + 8 [0]
  8,16   0       17     0.009336653 22888  I   R 81991 + 8 [git]
  8,16   0       18     0.009336988 22888  P   N [git]
  8,16   0       19     0.009338024 22888  U   N [git] 1
  8,16   0       20     0.009338791 22888  D   R 81991 + 8 [git]
  8,16   0       21     0.009339386 22888  R   R 81991 + 8 [0]
  8,16   0       22     0.009339922 22888  I   R 81991 + 8 [git]
  8,16   0       23     0.009340168 22888  P   N [git]
  8,16   0       24     0.013973186  4199 UT   N [gnome-terminal] 1
  8,16   0       25     0.014068467    16  U   N [kblockd/0] 1
  8,16   0       26     0.014071453    16  D   R 81991 + 8 [kblockd/0]
  8,16   0       27     0.014072938    16  R   R 81991 + 8 [0]
```

Mtron

```
Device   CPU    Seq #   Time (sec)  PID    Action, Mode, Block, Size, Process
  8,48   0        3     0.000010310 23264  G  RM 74424 + 8 [git]
  8,48   0        4     0.000012529 23264  P   N [git]
  8,48   0        5     0.000013275 23264  I   R 74424 + 8 [git]
  8,48   0        6     0.000016925 23264  U   N [git] 1
  8,48   0        7     0.000022210 23264  D   R 74424 + 8 [git]
  8,48   1        1     0.000191201     0  C   R 74424 + 8 [0]
  8,48   0        8     0.000224726 23264  A  RM 74432 + 8 <- (252,1) 73920
  8,48   0        9     0.000225238 23264  Q  RM 74432 + 8 [git]
  8,48   0       10     0.000226502 23264  G  RM 74432 + 8 [git]
  8,48   0       11     0.000227187 23264  P   N [git]
  8,48   0       12     0.000227466 23264  I   R 74432 + 8 [git]
  8,48   0       13     0.000231454 23264  D   R 74432 + 8 [git]
  8,48   0       14     0.000276996 23264  U   N [git] 1
  8,48   0       15     0.000369049     0  C   R 74432 + 8 [0]
  8,48   0       16     0.000407985 23264  A   R 78672 + 8 <- (252,1) 78160
  8,48   0       17     0.000408429 23264  Q   R 78672 + 8 [git]
  8,48   0       18     0.000409342 23264  G   R 78672 + 8 [git]
  8,48   0       19     0.000410001 23264  P   N [git]
  8,48   0       20     0.000410267 23264  I   R 78672 + 8 [git]
  8,48   0       21     0.000414058 23264  D   R 78672 + 8 [git]
  8,48   0       22     0.000459812 23264  U   N [git] 1
  8,48   1        2     0.000566355     0  C   R 78672 + 8 [0]
  8,48   1        3     0.001553698 23264  A   R 78360 + 8 <- (252,1) 77848
  8,48   1        4     0.001554258 23264  Q   R 78360 + 8 [git]
  8,48   1        5     0.001555659 23264  G   R 78360 + 8 [git]
  8,48   1        6     0.001556546 23264  P   N [git]
  8,48   1        7     0.001556860 23264  I   R 78360 + 8 [git]
  8,48   1        8     0.001561157 23264  D   R 78360 + 8 [git]
  8,48   1        9     0.001608109 23264  U   N [git] 1
  8,48   0       23     0.001716625     0  C   R 78360 + 8 [0]
  8,48   0       24     0.002763049 23264  A   R 78632 + 32 <- (252,1) 78120
  8,48   0       25     0.002763525 23264  Q   R 78632 + 32 [git]
  8,48   0       26     0.002764959 23264  G   R 78632 + 32 [git]
  8,48   0       27     0.002765867 23264  P   N [git]
  8,48   0       28     0.002766151 23264  I   R 78632 + 32 [git]
  8,48   0       29     0.002770616 23264  D   R 78632 + 32 [git]
  8,48   0       30     0.002817107 23264  U   N [git] 1
  8,48   1       10     0.003065081     0  C   R 78632 + 32 [0]
  8,48   1       11     0.003118737 23264  A   R 78664 + 8 <- (252,1) 78152
  8,48   1       12     0.003119168 23264  Q   R 78664 + 8 [git]
```

Transcend

```
Device   CPU    Seq #   Time (sec)  PID    Action, Mode, Block, Size, Process
  8,32   1      112     0.004719886 23598  G  RM 74424 + 8 [git]
  8,32   1      113     0.004721326 23598  P   N [git]
  8,32   1      114     0.004722132 23598  I   R 74424 + 8 [git]
  8,32   1      115     0.004730705 23598  D   R 74424 + 8 [git]
  8,32   1      116     0.004782480 23598  U   N [git] 1
  8,32   1      117     0.004950250     0  C   R 74424 + 8 [0]
```

```
8,32   1      118     0.004991093 23598  A  RM 74432 + 8 <- (252,0) 73920
8,32   1      119     0.004991504 23598  Q  RM 74432 + 8 [git]
8,32   1      120     0.004992513 23598  G  RM 74432 + 8 [git]
8,32   1      121     0.004993240 23598  P   N [git]
8,32   1      122     0.004993510 23598  I   R 74432 + 8 [git]
8,32   1      123     0.004997534 23598  D   R 74432 + 8 [git]
8,32   1      124     0.005042600 23598  U   N [git] 1
8,32   0       11     0.005216900     0  C   R 74432 + 8 [0]
8,32   0       12     0.005263819 23598  A   R 82744 + 8 <- (252,0) 82232
8,32   0       13     0.005264304 23598  Q   R 82744 + 8 [git]
8,32   0       14     0.005265532 23598  G   R 82744 + 8 [git]
8,32   0       15     0.005266331 23598  P   N [git]
8,32   0       16     0.005266655 23598  I   R 82744 + 8 [git]
8,32   0       17     0.005270823 23598  D   R 82744 + 8 [git]
8,32   0       18     0.005317260 23598  U   N [git] 1
8,32   1      125     0.005477154     0  C   R 82744 + 8 [0]
8,32   0       19     0.006508525 23598  A   R 78360 + 8 <- (252,0) 77848
8,32   0       20     0.006509007 23598  Q   R 78360 + 8 [git]
8,32   0       21     0.006510309 23598  G   R 78360 + 8 [git]
8,32   0       22     0.006511088 23598  P   N [git]
8,32   0       23     0.006511393 23598  I   R 78360 + 8 [git]
8,32   0       24     0.006515711 23598  D   R 78360 + 8 [git]
8,32   0       25     0.006561669 23598  U   N [git] 1
8,32   0       26     0.006741266     0  C   R 78360 + 8 [0]
8,32   0       27     0.007758935 23598  A   R 82704 + 32 <- (252,0) 82192
8,32   0       28     0.007759390 23598  Q   R 82704 + 32 [git]
8,32   0       29     0.007760597 23598  G   R 82704 + 32 [git]
8,32   0       30     0.007761528 23598  P   N [git]
8,32   0       31     0.007761821 23598  I   R 82704 + 32 [git]
8,32   0       32     0.007766241 23598  D   R 82704 + 32 [git]
8,32   0       33     0.007812367 23598  U   N [git] 1
8,32   1      126     0.008109263     0  C   R 82704 + 32 [0]
8,32   0       34     0.008156041 23598  A   R 82736 + 8 <- (252,0) 82224
8,32   0       35     0.008156460 23598  Q   R 82736 + 8 [git]
```

Appendix C

Source code

All source code, benchmark scripts, benchmark results and scripts made for parsing/plotting these are made available at `http://torkildr.at.ifi.uio.no/master/`.

Bibliography

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti. Introduction to flash memory. *Proceedings of the IEEE*, 91(4):489–502, April 2003.

[2] S. Mukherjee, T. Chang, R. Pang, M. Knecht, and D. Hu. A single transistor EEPROM cell and its implementation in a 512K CMOS EEPROM. In *Electron Devices Meeting, 1985 International*, volume 31, 1985.

[3] Fujio Masuoka and Hisakazu Iizuka. Method for manufacturing E^2PROM. US Patent No. US4612212, April 1985.

[4] Toshiba America Electronic Components, Inc. NAND vs. NOR Flash Memory. `http://www.toshiba.com/taec/components/Generic/Memory_Resources/NANDvsNOR.pdf`, Accessed December 2008.

[5] Cagdas Dirik and Bruce Jacob. The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 279–289, New York, NY, USA, 2009. ACM.

[6] Chanik Park, Wonmoon Cheon, Jeonguk Kang, Kangho Roh, Wonhee Cho, and Jin-Soo Kim. A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications. *Trans. on Embedded Computing Sys.*, 7(4):1–23, 2008.

[7] David Roberts, Taeho Kgil, and Trevor Mudge. Integrating nand flash devices onto servers. *Commun. ACM*, 52(4):98–103, 2009.

[8] S. Tehrani, JM Slaughter, E. Chen, M. Durlam, J. Shi, and M. DeHerren. Progress and outlook for MRAM technology. *IEEE Transactions on Magnetics*, 35(5):2814–2819, 1999.

[9] R. Bez and A. Pirovano. Non-volatile memory technologies: emerging concepts and new materials. *Materials Science in Semiconductor Processing*, 7(4-6):349–355, 2004.

[10] Image distributed under the Creative Commons License, Attribution-Noncommercial-Share Alike 2.0 Generic, `http://www.flickr.com/photos/lifeisaprayer/2282011834/`.

[11] Seagate. Seagate Barracuda ST380013AS Datasheet. *`http://www.seagate.com/support/disc/manuals/sata/cuda7200_sata_pm.pdf`*, Accessed June 2009.

[12] L.N. Bairavasundaram, G.R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 289–300. ACM New York, NY, USA, 2007.

[13] D.A. Patterson. Latency lags bandwith. 2004.

[14] Image distributed under the Creative Commons License, Attribution-Noncommercial-Share Alike 2.0 Generic, `http://www.flickr.com/photos/ciaociao/2859153476/`.

[15] Adam Leventhal. Flash storage memory. *Commun. ACM*, 51(7):47–51, 2008.

[16] Intel. Intel® X25-E Extreme SATA Solid-State Drive. *`http://download.intel.com/design/flash/nand/extreme/319984.pdf`*, Accessed July 2009.

[17] Matthew Wilcox. Re: [PATCH 2 of 9] block: Export I/O topology for block devices and partitions. *`http://marc.info/?l=linux-kernel\&m=124058465811408\&w=2`*, Accessed April 2009.

[18] MTRON. MSD-SATA3025 Product Specification. *`http://mtron.net/Upload_Data/Spec/ASiC/MOBI/SATA/MSD-SATA3025_rev0.4.pdf`*, Accessed December 2008.

[19] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on*

Annual Technical Conference, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[20] Li-Pin Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1126–1130, New York, NY, USA, 2007. ACM.

[21] Ji-Yong Shin, Zeng-Lin Xia, Ning-Yi Xu, Rui Gao, Xiong-Fei Cai, Seungryoul Maeng, and Feng-Hsiung Hsu. Ftl design exploration in reconfigurable high-performance ssd for server applications. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, pages 338–349, New York, NY, USA, 2009. ACM.

[22] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives. In *SYSTOR '09: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, pages 1–9, New York, NY, USA, 2009. ACM.

[23] Theodore Ts'o. Aligning filesystems to an SSD's erase block size. `http://thunk.org/tytso/blog/2009/02/20/aligning-filesystems-to-an-ssds-erase-block-size/`, Accessed February 2009.

[24] Sang-Won Lee, Bongki Moon, and Chanik Park. Advances in flash memory ssd technology for enterprise database applications. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 863–870, New York, NY, USA, 2009. ACM.

[25] Data Set Management Commands Proposal for ATA8-ACS2. `http://t13.org/Documents/UploadedDocuments/docs2008/e07154r6-Data_Set_Management_Proposal_for_ATA-ACS2.doc`.

[26] Electronista | OCZ set to ship 1TB Colossus SSD `http://www.electronista.com/articles/09/08/03/ocz.1tb.ssd/`.

[27] Daniel P. Bovet and Marco Cesati. *Understanding the linux kernel*. Number ISBN : 0-596-00213-0. O'Reilly, o' edition, d 2003.

[28] Transcend. Transcend S32GSSD25-S. *http://www.transcendusa.com/support/dlcenter/datasheet/Datasheet%20for%20SSD25.pdf*, Accessed December 2008.

[29] Western Digital. Western Digital Raptor WD740ADFD. *http://www.wdc.com/en/library/sata/2879-001165.pdf*, Accessed March 2009.

[30] IOzone Filesystem Benchmark. *http://www.iozone.org/*.

[31] Carl Henrik Lunde. Improving disk i/o performance on linux. Master's thesis, University of Oslo, 2009.

[32] Theodore Ts'o. Should Filesystems Be Optimized for SSD's? *http://thunk.org/tytso/blog/2009/02/22/should-filesystems-be-optimized-for-ssds/*, Accessed February 2009.

[33] Intel's X25-E SSD Walks All Over The Competition http://www.tomshardware.com/reviews/intel-x25-e-ssd,2158.html.