Flash Storage Systems

Tyler Harter

December 21, 2010

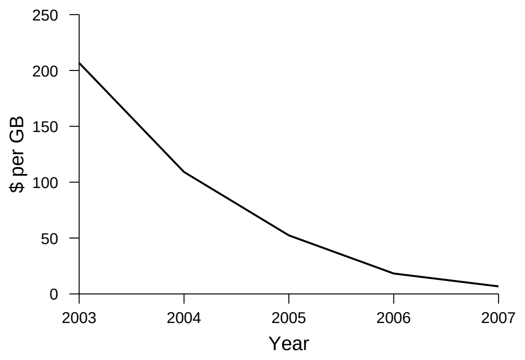

Figure 1: Flash Prices. The cost per GB of flash is shown for the years between 2003 and 2007. Source: plot adapted from figure 2 of [6].

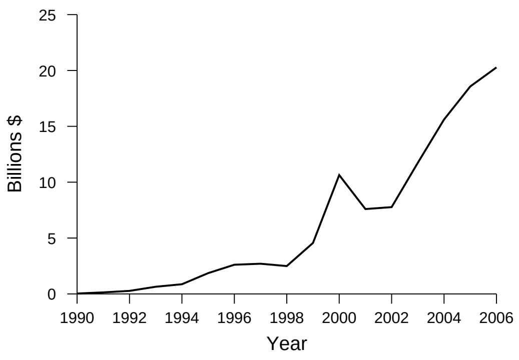

Figure 2: Flash Market. The growth of the flash market is shown between 1990 and 2006. Data source: table 1 of [15].

1 Introduction

In recent years, the evolution of flash storage technology has significantly impacted the everyday lives of consumers, the computer hardware industry, and the work of systems developers. Millions of people unknowingly use flash storage on a daily basis as they chat on their cell phones, snap pictures with digital cameras, listen to music on iPods, and back up important files to USB thumb drives.

Flash has many advantages over hard drives, which are the currently dominant storage technology. Flash is greener, more compact, and often faster than hard drives. The primary reason hard drives are used so often while flash use is mostly limited to mobile electronic devices is that a gigabyte of flash storage costs significantly more than a gigabyte of hard drive storage. The cost barrier to greater flash use is, however, quickly diminishing. Figure 1 shows the rapid decline of flash prices in recent years. Even as flash prices are rapidly falling, the flash industry is rapidly growing as illustrated by figure 2. As can be seen, flash is already a $20+ billion market.

The emergence of flash as a viable storage option and the trend towards more frequent flash use has forced systems researchers to rethink the design of file systems used with flash (a file system is the part of an operating system that decides how to organize the raw bits of a hard drive, flash drive, or other storage device in order to store a user's files). Since optimal organization for flash drives differs from optimal organization for hard drives, many traditional file systems designed for use with hard drives are inappropriate for use with flash.

In section 2 we provide basic background information about how various types of storage hardware function and about file systems in general. In section 3 we discuss in further depth the tradeoffs between different storage technologies that must be considered when a storage system is being designed. The importance of using the software that is appropriate for the chosen hardware is also discussed. In section 4 we describe some specific techniques that are used by the software of flash based storage systems. Finally, section 5 considers research into the use of flash in large-scale systems. Section 6 concludes.

2 Background

In sections 2.1 and 2.2 we provide basic background information about storage hardware and software respectively.

2.1 Storage Hardware

Computers are basically useless unless they can remember information. Some information, such as the contents of emails or documents, needs to be remembered long-term. It is acceptable, however, to forget some information when the computer is turned off, such as the locations of different program windows on the screen. Hardware that allows computers to remember information long-term, even after the computer is powered off, is called non-volatile storage. Hardware that forgets information once a computer is powered off is called volatile storage.

All the data that computers store is encoded as 1's and 0's. Storage hardware uses a variety of techniques to store these 1's and 0's. We discuss the techniques used by DRAM, hard drives, and flash in sections 2.1.1, 2.1.2, and 2.1.3 respectively.

2.1.1 Dynamic Random Access Memory

When people refer to the memory in a regular desktop or laptop computer, they are typically referring to DRAM (Dynamic Random Access Memory) [12]. DRAM uses capacitors to store 1's and 0's. A capacitor holding a charge represents a 1, and an uncharged capacitor represents a 0.

One major disadvantage is that the capacitors in DRAM leak charge. This means that DRAM capacitors have to regularly be recharged so that their charge doesn't drop too low, resulting in 1's being unintentionally changed to 0's. In addition to drawing extra power, this disadvantage makes DRAM a form of volatile storage since it forgets everything when power is removed.

DRAM is commonly used because it's a relatively simple and fast technology. Storing the data electronically allows the data to be accessed in any order at high speeds. DRAM is called memory because it stores data electronically using semiconductor technology. DRAM is also classified as a "random access" technology because it can access data in any random order quickly (many storage technologies are only fast when they access data in order, from beginning to end).

The DRAM acronym is a good summary of DRAM's characteristics that we have just discussed:

D	DRAM is "Dynamic" because capacitors must constantly be refreshed.
RA	DRAM is a "Random Access" storage technology because accessing bytes out of order is just as fast as accessing them in order.
M	DRAM is considered "Memory" because it uses semiconductors to store data (in contrast with technologies that use non-electric technologies, such as cassette tapes, to store data)

Although DRAM can't be the sole storage technology used in any computer since it is volatile, DRAM is used in nearly all computing systems in addition to some form of non-volatile storage.

2.1.2 Hard Drive Storage

Unlike DRAM, hard drives are a non-volatile storage technology, so they are useful for long-term storage. Hard drives store 1's and 0's using magnetic material. Inside of a hard drive, there are rotating disks, or platters, that are coated with the magnetic material, as seen in figure 3. The platters are divided into billions or trillions of tiny areas. Each area can be

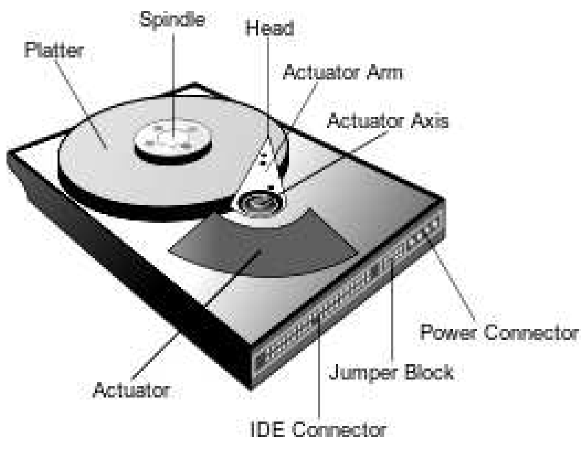

Figure 3: Hard Drive Internals. The actuator arm moves over the spinning platter, allowing the head to read or write bits on the magnetic surface. Source: Wikimedia Commons.

magnetized to store a 1 or demagnetized to store a 0. As the platters rotate at high speeds (7200 RPMs is common for consumer grade drives), a reader/writer head at the end of an actuator arm can move over platters, observing which areas are magnetized or changing whether or not areas are magnetized.

2.1.3 Flash Memory

Like hard drives, flash [11] is also a type of non-volatile storage. Flash can store 1's and 0's in flash cells by trapping electric charges. A cell containing a charge represents a 0 and a cell without a charge represents a 1. Unlike DRAM, which relies on capacitors to hold a charge, flash has a more sophisticated design that prevents charges from leaking.

Charges can be trapped in individual flash cells, setting their values to 0. In order to eliminate a charge, an electric field needs to be applied. The electric field that is used to set a cell's value to 1 affects not one, but many flash cells. This means that it is possible to change an individual cell to a 0, but cells must be changed to 1's in groups.

Flash storage is divided into areas called "blocks" which are further subdivided into areas called "pages." Blocks will often contain 64 or 128 pages, and pages will often contain 2KB or 4KB.

There are several operations that can be performed on flash:

1. Read
2. Write
3. Erase

The read operation retrieves the data stored in a flash page. If one wants to read multiple pages, it is necessary to perform multiple reads. If, however, one wants to read just part of a page, it is necessary to do one complete read. This will result in extra data being retrieved. Because extra data is retrieved when less than a page worth of data is needed, very small reads are less efficient than larger reads.

The write operation can be used to change individual flash cells from 1's to 0's. Like the read operation, the write operation takes the same amount of time when one is trying to change just a couple bits of data as when one is trying to change an entire page.

The erase operation is used to change 0's to 1's (in contrast with the write operation, which changes 1's to 0's). Flash devices execute erase operations by applying the electric field to flash cells as described earlier. This means that it's not possible to erase an individual bit. Setting a bit to a 1 by using an erase operation requires setting all the bits in the containing block to 1's.

2.2 File Systems

Hundreds of file systems have been designed and implemented. Indeed, the name of one popular file system that was optimized for flash, YAFFS (Yet Another Flash File System), humorously illustrates the massive amount of work that has been done in this field. Despite the great variety, nearly all file systems have a few parts in common.

We will discuss the following three common file system structures:

1. Files

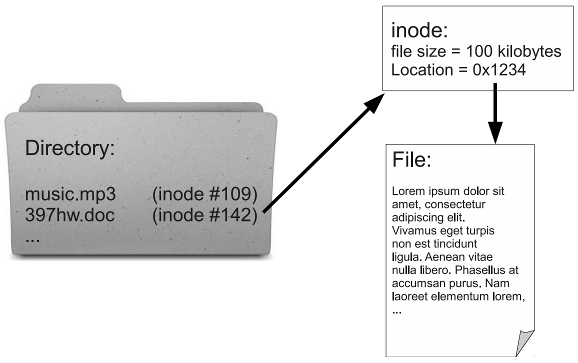
inode:
file size = 100 kilobytes
Location = 0x1234
Directory:
music.mp3 (inode #109)
397hw.doc (inode #142)
...
File:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus eget turpis non est tincidunt ligula. Aenean vitae nulla libero. Phasellus at accumsan purus. Nam laoreet elementum lorem, ...

Figure 4: Common File System Structures. The directory contains a list of file names and corresponding inode numbers. The inode numbers can be used to find the inodes and get more information about the files, such as their locations and sizes.

2. Inodes

3. Directories

A file is mostly what one would expect. A file just contains data that must be stored. What it doesn't contain is organizing information such as the size, location, or name of the file. This organizing information, called metadata (metadata means "data about data"), can be found in inodes and directories.

The size and location of the file is stored in a separate structure called an inode. Nobody knows for sure what the "i" in "inode" stands for. Each inode has an inode number associated with it. An inode number can be used to find the location of the corresponding inode on a storage device.

The names of files can be found in directories. Directories are often informally called folders. A directory can be thought of as special file that acts as a container by maintaining a list of regular files and other directories that it contains. The list will contain a set of file name/inode number pairs.

The relationship between files, inodes, and directories can be best understood by the illustration in figure 4. Suppose a user wants to open their "397hw.doc" homework file, as shown in the illustration. In order to access the file contents for the user, the file system would first locate the containing directory and search for "397hw.doc" within the directory. The file system would observe that the inode number corresponding to "397hw.doc" is 142. The file system knows the location of inode 142 on the storage device, so it will read the inode data, getting the size and location of the file. The file system is then capable of reading the file from the storage device and giving the contents to the user.

3 Storage Systems

A storage system is simply a system that needs to store data. Storage systems have two aspects: hardware and software. A storage system will typically use some mix of storage hardware (memory, hard drives, flash, etc). Good storage systems are designed with the needs of the system in mind. We discuss the needs of storage systems in section 3.1. Then in sections 3.2 and 3.3 we discuss how storage system hardware and software and meet those needs.

3.1 Needs

Storage systems need to satisfy a variety of requirements. Examples include being power-efficient, affordable, and reliable. Performance needs typcially receive the most attention, however, so performance will be our focus for the remainder of this section.

Storage system performance is not a simple scalar metric. Saying a sprinter is faster than a marathon runner or vice versa only makes sense if one has a specific race in mind. For storage systems, the "race" is called a workload (or access pattern).

There are several ways to characterize device workloads. One way is to measure how sequential a workload is. Although the term "sequentiality" is rarely used outside of the computer science community, anyone who has ever watched movies on VCR tapes and DVDs is probably familiar with the concept. Movie watchers usually start watching a movie at the beginning and watch everything in order until the end. This is a sequential access pattern. Since both VCR and DVD players have good sequential performance, the type of media has little impact on people that

	RAM	Flash	Hard Drive
Non-Volatile	✗	✓	✓
Solid State	✓	✓	✗
Gigabytes/Dollar	■□□□	■■□□	■■■■
Compactness	■■□□	■■■■	■□□□
Power Savings	■□□□	■■■■	■□□□
Read Performance	■■■■	■■□□	■■□□
Write Performance	■■■■	■□□□	■■□□
Random Performance	■■■■	■■□□	□□□□
Sequential Performance	■■■■	■■□□	■■□□

Horrible	Poor	OK	Good	Excellent
□□□□	■□□□	■■□□	■■■□	■■■■

Table 1: Hardware Comparison. The table shows what types of storage DRAM, flash, and hard drives are. Many of the common comparisons that are made between these technologies are also shown. The tradeoffs are somewhat subjective and are constantly changing, so the information said again is simply our opinion (albeit an opinion that is probably more or less shared by those who are familiar with the various storage technologies).

watch movies this way. In contrast, sometimes people who are rewatching a movie like to skip around to their favorite scenes. This is a non-sequential access pattern. Non-sequential workloads are usually called random workloads. VCR players are very slow when handling random workloads because it's necessary to mechanically wind a strip of magnetic tape dozens of feet in order to go to a different place in the movie. But it takes relatively little time to start reading at a different location on a DVD that is making hundreds if not thousands of rotations per minute. Thus, people using DVDs can enjoy randomly watching various scenes in a movie, but people using VCR tapes will probably spend most of their time fast forwarding or rewinding.

Another way to characterize workloads is by the proportion of bytes read to bytes written. Sometimes a workload will involve just reading data without writing any data or vice versa.

3.2 Hardware

"Simply put, I wanted to make a chip that would one day replace all other memory technologies on the market."

- Fujio Masuoka (Inventor of Flash) [15]

As Masuoka's quote indicates, flash was designed to be a universal storage technology. Hard drives, DRAM, and other storage technologies are, however, still widely used, and this situation is unlikely to change in the near future. Engineering is all about tradeoffs, and the storage industry is no exception. Flash is not simply a superior storage technology that does everything better than hard drives and DRAM. Rather, flash has distinct advantages and disadvantages that allow it to fill an important and growing niche in the storage world.

Thoroughly and objectively comparing flash to DRAM and hard drives is not feasible and probably not even worthwhile. The storage industry is very dynamic, with prices and technology rapidly changing. Even if a comparison were done at a particular point in time, doing a fair comparison would be difficult because there's no clear best way to choose representatives for each type of storage technology. For instance, a flash device can be made more compact at the expense of the device's longevity.

Although a precise comparison isn't possible, table 1 shows my general opinions about the tradeoffs between DRAM, flash, and hard drives. My opinions were influenced by [1] and [5] and by comparing consumer hardware specifications on shopping sites such as newegg.com. Most storage professionals would likely have a similar view of the tradeoffs.

The first two rows of table 1 show how flash is a different type of storage than DRAM or hard drives. Flash is both non-volatile (it remembers data even without power) and solid state (it has no moving parts). Hard drives are non-volatile but not solid state, and DRAM is solid state but not non-volatile, so flash is very useful in cases where both characteristics are necessary. MP3 players are such a case since people don't want to lose their music when batteries run out and many people like to take MP3 players jogging (using a non-solid state technology would be bad design since moving parts are prone to failure when shaken).

The next three characteristics in table 1 are cost, compactness, and power-efficiency. Hard drives are still much cheaper than flash devices and will probably continue to be cheaper for some time. Cost is probably the main reason hard drives, rather than flash devices, are nearly always used to store data in personal computers. However, flash devices are very compact and power-efficient. These characteristics also make flash very desirable in MP3 players and other consumer electronics).

The last four rows of table 1 compare different types of performance which are described in section 3.1. DRAM has excellent performance when doing anything, so it is typically used as much as possible when it's not necessary to use a non-volatile technology, such as a flash or hard drives, to store data for an extended time.

In many cases, flash and hard drives have roughly comparable performance. The major exception is random access performance. Although hard drive platters spin very quickly, the time it takes for them to rotate is huge compared to the time it takes to perform the vast majority of other things computers do. This means that accessing data that is laid out in order can be done quickly, but accessing data that is scattered over various parts of a platter will take a great amount of time since many rotations will be necessary. Flash doesn't have any moving parts, so it can access data out of order just as quickly as it can access data in order. There are many cases where it is necessary to perform random accesses, so flash is sometimes used instead of hard drives when performance is very important.

3.3 Software

The software component of a storage system is the file system. Although it might seem that performance should depend solely on the type of hardware being used, file systems also play a huge role. An analogy will help clarify this role. Imagine a situation where the CEO of a company has lots of papers that need to be saved. To help her manage all the papers, the CEO hires a secretary that keeps everything organized in file cabinets.

The file cabinets are like storage hardware, such as a hard drive or flash drive. The CEO is like a user/computer program that needs to save and access

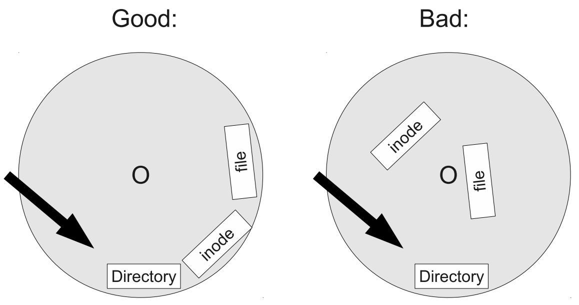

Figure 5: Disk Layouts. On the left hard drive platter, data is arranged in such a way that the directory, inode, and file can all be accessed (in that order) with a single platter rotation. On the right platter, it could taken multiple rotations to access all the data.

files. The secretary is the file system. To be efficient, the secretary needs to be aware of the needs of the CEO. If the CEO often needs certain papers at the same time, the secretary might be faster at retrieving the papers if she uses some sort of categorical organization scheme as opposed to an alphabetical scheme. Similarly, the secretary should consider the layout of the filing cabinets when storing the papers. For instance, it would be wise to keep frequently accessed papers near the front of drawers where it is quickest to retrieve things.

In this analogy, the requests made by the CEO are the file system workloads, and what needs to be done to retrieve the files is the device workload. The secretary's job is to know what device workloads can be performed quickly, know what file system workloads are likely to occur, and then choose an organizational approach that will result in the most common file systems workloads being translated to the most efficient device workloads.

History has taught us that designing file systems with the characteristics of the hardware in mind yields much better performance than does designing file systems without paying attention to the hardware. The original UNIX file system ignored the hardware characteristics of hard drives. The file system laid out data in the simplest, most intuitive way imaginable. All the inodes (discussed in section 2.2) were grouped together in a single location on disk and put in no particular order. The files and directories were placed in arbitrary locations elsewhere on disk.

In the early 1980's researchers at Berkeley realized that the layout chosen by the original UNIX file system was non-optimal. In order to access the contents of a file, a file system needs to access the containing directory, the inode, and finally the actual file (in that order). If these structures are scattered over the hard drive platter, it could take several platter rotations before the data can actually be retrieved. Even though the disks are spinning at high speeds, disk rotations take much longer to perform than CPU operations, and hence are usually a performance bottleneck.

The Berkeley researchers realized that keeping related data together, in particular related directories, inodes, and files, would improve performance. They called their new file system the Berkeley Fast File System [7], or FFS for short. Figure 5 compares the way a bad file system might layout data relative to the way a file system such as FFS file system might layout data. FFS provided huge performance gains over the original system (over a factor of 10 for some workloads).

4 Flash Storage Systems

In section 2.1.3 we described how flash works, and in section 3.2 we described flash's strengths and weaknesses that result from the way it is designed. In this section, we describe the techniques flash file systems use to take advantage of flash's strengths and work around it's weaknesses. We consider five common techniques: wear leveling (section 4.1), parallelism (section 4.2), versioning (section 4.3), garbage collecting (section 4.4), and redirecting (section 4.5). These techniques are described in greater detail in [3, 4, 5, 8, 13, 14] and are employed by real file systems such as Red Hat's open source JFFS2 and Microsoft's FFS2 (Flash File System 2, not to be confused with Berkeley's Fast File System, which we mention in section 3.3 and is also abbreviated FFS).

4.1 Wear Leveling

One disadvantage of flash is that blocks wear out after they have been erased a certain number of times (often 100,000 times). Still, flash will often last many years before it wears out. If some blocks are erased much more often than others, however, the device's lifespan can be severely shortened since the heavily used blocks will wear out before the rest, causing the entire device to fail even while some blocks are still relatively fresh.

Good flash file systems have strategies for avoiding writing too often to a single block. Using a random number generator to randomly choose which empty page to write to when a page needs to be written does a fairly decent job of evenly distributing the write load. Sometimes, however, there will be a file on the device that never gets deleted or updated. To evenly exercise the entire device, some file systems, such as JFFS2 (Journaling Flash File System 2) [13] will occasionally move such files to different locations.

4.2 Parallelism

"Many hands make light work"

- John Heywood

Heywood's quote describes the concept of parallelism, the strategy of breaking up tasks into sub tasks and delegating the sub tasks to different components. Since each component only has to do part of the work, it takes much less time to performance a task than it would take if the entire task had to be performed by a single component.

Applying parallelism to storage is not a new idea. Berkeley researchers suggested an approach which has become widely used: RAID (Redundant Array of Inexpensive Disks) [9]. In one type of RAID system (RAID 0), many identical, cheap hard drives are used instead of one large, fast drive. If two hard drives were being used in a RAID 0 system and a user wanted to save a file, half the bits of the file would be stored on one hard drive and half would be stored on the other. This would allow the system to half the time it would take to transfer the file data to the disks.

RAID-like strategies can be used by flash file systems on flash devices. Indeed, parallelization is probably even more useful when applied to flash than hard drives for several reasons:

1. Flash storage can be purchased with much less capacity than can hard drive storage. It is difficult to find new hard drives that store less than 40GB of data. This means that if a storage system needs 80GB of capacity, it would only be possible to split the task of storing data between two 40GB hard drives, but one could conceivably split the task between ten 8GB flash devices.

2. Often individual flash devices are divided into areas called "partitions" that can be accessed somewhat independently. Thus, parallelization can sometimes still be used even when only one device is available.

3. Data transfer is often the bottleneck for flash, and parallelization reduces data transfer time. Seek time (the time it takes for the disk to rotate to the right position) is often the bottleneck for hard drives, and parallelization would typically do nothing to reduce seek time.

For these reasons, a good flash file system will employ parallelization when possible.

4.3 Versioning

Flash is very slow when it has to update a page with arbitrary data because most updates will involve changing some 0's to 1's. As described in section 2.1.3, it is easy to change 1's to 0's in an individual page, but the only way to change a bit from a 0 to a 1 is to erase the entire block containing the bit, setting all the bits in the block to 1. This means that the following process will be necessary if one wants to update an individual page in a block (we'll assume the block contains 64 pages) with 1's:

1. Create a backup (in DRAM) of the 63 pages that don't need to be changed.

2. Erase all 64 pages, setting all the bits they contain to 1's.

3. Write the new data to the flash page that needs to be updated.

4. Write the data we backed up in step 1 to the other 63 pages that didn't change.

This is clearly horribly inefficient. We simply wanted to update a single page, so we should have only had to do one page worth of work. Instead, we had to perform 64 pages of work.

The solution is to use versioning instead of updating. When versioning is used, version numbers are associated with all the data that is saved on the flash device. When there is new data, the old data is left intact and the new data is just written to different pages that are not in use. This allows us to avoid a costly page updates. The file system will know which data is new and which data to ignore because the new data will have a bigger version number than old data.

File systems, such as Red Hat's JFFS2 (Journaling Flash File System), that always write newer versions of data to new locations and then cleanup old data later are often called "Log-structured File Systems." Many of these systems were inspired by the original LFS (Log-structured File System) [10] which was developed by Berkeley researchers. LFS strategies provided performance gains when employed with hard drives under some circumstances, but these techniques have huge performance advantages for flash and flash file systems since updating pages is so slow.

4.4 Garbage Collecting

The problem with versioning as described in section 4.3 is that eventually old data will accumulate and begin to waste too much storage space. At some point, blocks with lots of pages containing outdated data will need to be erased. Since it is common for blocks to have a mix of current and outdated data, the file system will have have to move the current data to other blocks before erasing. This cleanup process is known as garbage collection.

There are a couple of choices that file systems need to make when garbage collection:

1. When to garbage collect (for example, when there is no space left or when the file system

isn't busy doing something else).

2. What blocks to garbage collect (for example, blocks with more outdated data should maybe be chosen over blocks with less outdated data).

Flash file systems apply a variety of different strategies when making these decisions.

4.5 Redirecting

In section 4.3 we described how versioning can be used to avoid updating data. One disadvantage of using versioning as described is that data is scattered over the entire flash device. The file system can use version numbers to determine if data is current, but finding the location of the most current data is more difficult. To learn the locations, it is typically necessary for the file system to scan the entire flash device before the device can be used. This takes significant time.

Microsoft patented the idea, which they employ in their FFS2 (Flash File System), of redirecting the file system from old data, which is at a known location, to new data [3, 4]. The redirection technique can be understood with an analogy: a student is taking a test with a pen (so she cannot erase). After writing all the answers in the test booklet, the student realizes her original answer was wrong. Since she can't erase, she makes a comment in the margin: "ignore what I wrote above and look for the answer on page 13." If she again makes a mistake, she just writes another comment on page 13, directing the grader to her third attempt on page 14. In this way, the student can simply write more instead of erasing or "crossing out" data, and the grader can always find the newest data by following the chain of comments. FFS2 implements this by keeping a "margin" (section of 1's) next to any data. If the data becomes outdated, FFS2 can use the margin to note where the newer data is located. Making the note is quick since it only involves changes 1's to 0's.

5 Distributed Flash Systems

Companies frequently have more data than can be stored by a single computer, and many of the computational tasks they wish to do would take years if they were performed by a single machine. Distributed systems are the solution (a distributed system is simply a collection of computers, often called nodes, that are networked together and cooperate to solve a problem or provide a service). . Flash has established itself as the ideal storage technology for many mobile and compact devices, but researchers are now exploring the usefulness of flash in large-scale distributed systems.

In 2009, researchers at two major computer science research universities published papers describing distributed systems architectures that make use of flash storage. The University of California, San Diego researchers designed a flash-based architecture optimized for data processing called Gordon [5], and Carnegie Mellon researchers designed a flash-based architecture optimized for record lookup called FAWN [1, 2]. Section 5.1 discusses some tasks distributed systems frequently perform. In sections 5.2 and 5.3, we describe how Gordon and FAWN are optimized for some of these tasks. Finally, in section 5.4 we evaluate the usefulness of flash in distributed systems.

5.1 Distributed Tasks

In this section we consider 3 categories of tasks that distributed systems perform:

1. Data processing
2. Data/record lookup
3. Data archiving

5.1.1 Data Processing

Distributed systems are often used to process large quantities of data. The Gordon researchers believe flash can make the processing faster. In particular, a few of the tasks they considered are pattern recognition, sorting, and web indexing.

Pattern recognition simply involves reading over huge bodies of text and finding parts that match a certain pattern. For instance, a "###-###-####" pattern could be used to find phone numbers.

Sorting is the task of putting unordered data in order according to some feature (for example, one might order people alphabetically according to their last names). Sorting tasks typically involves reading a collection of unordered records from the storage medium, putting the records in alphabetic, numeric, or some other order, and then writing the records back to the storage medium. If there are a great number of records, it's not possible to sort all the data using RAM at once, so the sorting algorithms that are typically used involve several phases of writing partially sorted data to the storage medium.

Web indexing is a task commonly performed by search engine companies. If you were to do a Google search for the word "flash," it would not be practical for Google to search through its archived copies of every web page on the Internet. Instead, Google knows that many people will search for the word "flash," so they essentially keep a list on their servers of all the web pages that contain the keyword "flash." Web indexing is the task of reading all the web pages in order to generate the lists for "flash" and other keywords.

5.1.2 Data/Record Lookup

Distributed systems often need to store named data/records that can readily be "looked up." The FAWN researchers believe flash can make these storage/lookup tasks more efficient. In particular, some applications include social networking sites, e-commerce sites, and domain name systems. Social networking websites need to be able to look up a person's profile via that person's name or some other form of identification. Similarly, e-commerce sites have countless books and other products that are accessed via some sort of ID (perhaps an ISBN).

Domain name systems (DNSs) are an extremely important use of a record lookup system that all web surfers use on a daily basis. Computers, including servers that host popular web sites such as www.google.com, are identified by special numbers called IP addresses (for example, 209.85.225.103). When a web surfer visits a website, they must send a request message that specifies a web page to the server that hosts the website. The request must specify the address of the server in order for the request to arrive at the proper destination. Most web surfers can remember an address like www.google.com, but they probably won't remember an IP address such as 209.85.255.103. DNSs solve this problem. When a web surfer types the "www.google.com" URL into their web browser, a lookup request for "www.google.com" is sent to a DNS server. The DNS servers keep track of all the URL/IP address translations, so the DNS server will send 209.85.255.103. The web surfer's browser will receive this IP address and use it to then send the web page request to the appropriate server. Clearly, having efficient DNSs is important. Not only must DNS servers store numerous URL to IP translations, but they must perform the lookups quickly since web surfers have to wait for a response before they can even begin to visit a web page.

5.1.3 Data Archiving

Distributed systems often need to archive data. Archiving involves storing data that will be infrequently, if ever, accessed in the future. Two examples of archiving tasks are logging and backups. Logging involves keeping a list of events that have occurred. For example, a company might keep a log that tracks all the traffic on their network. In most cases, there would be no need to ever look at the logged network traffic, but the data is still worth recording because there's always a chance it might be useful in the future. For instance, if network intrusion was detected, the logs would be invaluable to a security professional.

Flash has not been seriously considered as a storage medium in distributed systems that are primarily used to archive data. When discussing the merits of flash in distributed systems in section 5.4, we offer an explanation for why flash has not been used for these tasks and argue that flash is not a universal solution.

5.2 Gordon

University of California, San Diego researchers designed a flash based distributed system to handle data processing tasks (discussed section 5.1.1 called Gordon. Storage is frequently a performance bottleneck since storage performance has increased at a much slower rate than has CPU performance and network performance. The Gordon researchers address this problem, stating that "the goal of the Gordon design is to reduce the mismatch between network bandwidth, disk bandwidth, and CPU performance" [5].

Flash comes in small packages, but the Gordon researchers wanted a significant amount of storage in each Gordon node since the systems are expected to process large amounts of data, so an array of many flash packages are placed in each Gordon node. This offers huge opportunities for parallelization (discussed in section 4.2), so the researchers designed Gordon to "aggressively pursue dynamic parallelism between accesses to the flash array." [5]

5.3 FAWN

Researchers at Carnegie Mellon University designed a flash based distributed system called FAWN (Fast Array of Wimpy Nodes) that is optimized to handle lookup tasks (discussed in section 5.1.2) quickly. The nodes are "wimpy" because they have relatively slow processors and low power consumption. Like the Gordon researchers, the FAWN researchers argue that it's wasteful to have high performance CPUs that spend most of their time waiting for data to be read from or written to storage.

Unlike the Gordon researchers, the FAWN researchers use cheaper, commodity hardware instead of designing custom hardware, so the cost of a FAWN system is relatively low (under $300 per FAWN node).

FAWN is fast because it's affordable to purchase many FAWN nodes instead of a few high performance servers and flash is ideal for record lookup tasks which result in very random workloads. The FAWN researchers were able to make their design even more efficient by recognizing that some lookup requests are more common than others. In addition to having many cheap FAWN nodes in the system, there are also a few high performance front end servers that use DRAM to remember the answers to some of the more common requests. Any requests that are sent to a FAWN system go to a high performance front end server first. If the server has the answer in DRAM, it will immediately return the answer. Otherwise, the server will pass along the request to the appropriate flash based FAWN node.

5.4 Evaluation

As described in section 3.2, there are various tradeoffs between flash and hard drives, so neither storage technology is a universal solution, either in general, or in the case of distributed systems. Archiving data is one task where hard drives are currently the best solution and will continue to be the best solution for the foreseeable future. Although flash is much faster in many scenarios, capacity, rather than performance, is generally most important for data archival, and hard drives are the clear winner in terms of the capacity-per-dollar metric. Besides speed, power savings are another reason to use flash; however, this is frequently not an important for archival tasks since archived data is rarely read. This infrequent access makes it possible to save power by turning off hard drives when they are not being used.

In remains to be seen whether flash will come to be used by systems such as Gordon for data processing. Many of the common data processing tasks identified by the Gordon researchers involve large, sequential data accesses. Although flash may still be faster than hard drives for these workloads, hard drives perform best when doing large sequential accesses, so the performance gains should be less than would be expected for more random workloads. Furthermore, many of the tasks considered by the Gordon researchers involved a significant amount of data being written to storage as well as read. Flash is much faster when reading than writing, whereas hard drives perform both tasks at comparable speeds (ignoring caching), so the significant writes also mean that flash's greatest strengths are not being used. It's not clear whether the tasks the Gordon researchers considered are best handled by flash or hard drive systems. The researchers did report moderate performance gains (50% increase) over traditional hard drive based systems as well as significantly reduced power consumption. It's not clear, however, if these advantages will be worth the extra cost of the flash hardware.

The record/data lookup tasks that FAWN systems are designed to handle seem like a case where flash is a clear winner. Lookup workloads consist of many random reads. Flash is known for its excellent random access performance, and it is also significantly faster at reading than writing. The fast read performance of flash means that bus transfer rates are a bottleneck for flash under heavy load, whereas bus transfer rates don't affect flash write performance. Flash bus transfer rates have not significantly improved for the last 15 years, but just recently there have been technological advances for flash buses, so we can expect heavy-load read performance to increase in the near future, further improving the performance of FAWN systems.

6 Conclusion

We have described flash and compared it with other storage technologies. We have also considered some of the main techniques employed by flash file systems to best leverage flash hardware. Finally, we have described cutting edge research in the use of flash in distributed systems.

There are three main points that can be made regarding flash storage:

1. Use the right storage hardware for the task. Flash isn't a universal solution, but it meets needs that are unmet by other hardware. Knowing flash's strengths (such as random performance and power-efficiency) and weaknesses (such as high cost per GB), as well as the nature of the task at hand, will help guide decisions between flash and hard drives.

2. Create file systems with hardware in mind. File systems have a huge impact on performance, and which file systems techniques should be used depends entirely on the hardware. File systems that are unaware of flash's idiosyncrasies and don't avoid updating data in small batches, for instance, will be extremely slow.

3. Invest in storage if improving performance is the goal. Super fast CPUs are of little value if they spend all their time waiting for the bottleneck: the storage device. Flash has performance advantages in many cases. Where performance is critical, research effort should be invested in investigating ways to use flash and money should potentially be invested in purchasing either higher end hard drives or flash.

References

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: a fast array of wimpy nodes," in *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. New York, NY, USA: ACM, 2009, pp. 1–14.

[2] D. G. Andersen, J. Franklin, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A fast array of wimpy nodes," Carnegie Mellon University, Parallel Data Laboratory, Pittsburgh, PA, Tech. Rep., 2008.

[3] P. L. Barrett, S. D. Quinn, and R. A. Lipe, "Method and system for traversing linked list record based upon write-once predetermined bit value of secondary pointers," US Patent 5,247,658, Sep 21, 1993.

[4] ——, "System for updating data stored on a flash-erasable, programmable, read-only memory (FEPROM) based upon predetermined bit value of indicating pointers," US Patent 5,392,427, Feb 21, 1995, (continuation of 5,247,658).

[5] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications," in *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems.* New York, NY, USA: ACM, 2009, pp. 217–228.

[6] A. Leventhal, "Flash storage today," *Queue*, vol. 6, no. 4, pp. 24–30, 2008.

[7] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for UNIX," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 181–197, 1984.

[8] Numonyx, "Flash file systems overview." [Online]. Available: www.numonyx.com/ Documents/ WhitePapers/ Flash_file_ systems_WP.pdf

[9] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data.* New York, NY, USA: ACM, 1988, pp. 109–116.

[10] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 26–52, 1992.

[11] J. Tyson, "How flash memory works." [Online]. Available: http://electronics. howstuffworks.com/ flash-memory.htm

[12] J. Tyson and D. Coustan, "How ram works." [Online]. Available: http://computer. howstuffworks.com/ ram.htm

[13] D. Woodhouse, "JFFS: The journalling flash file system." Ottawa Linux Symposium, 2001.

[14] ——, "The journalling flash file system," 2001, slideshow used with presentation at Ottawa Linux Symposium: http://sourceware.org/ jffs2/ jffs2-slides-transformed.pdf.

[15] F. Yinug, "The rise of the flash memory market: Its impact on firm behavior and global semiconductor trade patterns," 2007.