

Advanced Disk Drive Caching

By Chris Russ and Brent Neal

jcr6@reindeergraphics.com

brent@reindeergraphics.com

Abstract

As computers reach well into Gigahertz performance ranges and memory busses strive to keep up, there is one critical component that fails miserably in the quest for speed: Hard disks. This is a proposal for an advanced disk-drive caching mechanism that trades storage for speed and endeavors to keep the drive light out.

Overview

Personal computers today have one essential component that slows them down more than anything else. The disk drive. Data rates from the disk drives is continually increasing; from SCSI to SCSI-3 and from ATA-33 to ATA-133. However, the time that it takes for the read/write heads on the disk drives to travel from one area to another is still approximately the same as it was 10 years ago. Rotation speeds are improving incrementally, not exponentially.

When reading or writing small pieces of data (a very common event inside a computer), the bulk of the time is spent waiting for the drive heads to get to the appropriate place. Very little is spent actually sending the data. This is especially obvious when the disk drive light turns on and continually stays on.

This is known as a "disk-bound" condition. The computer's speed is fundamentally limited by the seek speed of the disk. (Simple tests using RAM disks can demonstrate how much time is spent accessing the disk.) These seek

times are measured in *milliseconds*, not *microseconds* or *nanoseconds*. Thus hundreds of thousands of processor cycles can be lost while waiting for data to be located.

Note: Streaming applications are different -- music and video access are not dominated by disk seeking times, but rather by the amount of data that can be streamed in or out at a sustained rate. However, attempting to multitask on a device that is doing streaming will illustrate the performance loss -- while the drive is thrashing between different locations.

Drive performance

So how do we measure disk drives? There are three metrics:

- **Size:** Megabytes, Gigabytes, Terabytes, PetaBytes, ExaBytes
- **Latency:** head seek time and rotational latency (measured in milliseconds)
- **Maximum data rate:** MB/sec.

Disk size is affected by the density of the recording medium, the size of the drive head, and the number of platters (and sides) that have data on them. A common method for increasing disk drive size is to add platters. This increases the power consumption, the mass of the drive heads, and can negatively impact latency.

There are two components of latency, **head seek time**, which is the amount of time that it takes to accelerate the drive head from one position, move it, and decelerate it into the desired position. This is affected by the distance that the head has to move -- small moves, especially ones as small as one track, can be nearly instantaneous, while large moves can take dozens of milliseconds.

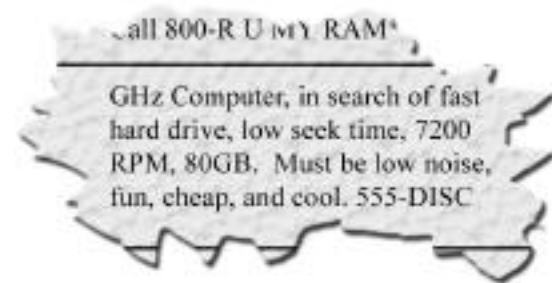
The second component is **rotational latency**. This is the amount of time wasted while the desired spot on the media rotates into a position under the drive head. For a platter rotating at 4200 RPM (revolutions per minute), this means that once every 14.2 milliseconds the platter rotates completely. Thus, the average rotational latency would be 7.1 milliseconds (or one half revolution). Some platters rotate at 7200 RPM which would correspond to an average latency of 4.2 milliseconds.

The assumption is that the seek delay and the rotational delay will occur concurrently, and that if the data is close enough, the rotational delay will be the dominant factor. This is not a good assumption. When the head travels a significant distance, this matters.

Finally, the **maximum data rate** is a function of the position on the platter that the data is stored. Since the platter spins at a constant rate, the velocity of the platter under the heads is higher around the outside of the disk. This means that either the data is closer together on the inside (resulting in a constant data rate everywhere) or the data is accessed at a faster rate on the outside rim than on the inside edge. This suggests that streaming applications will perform better if the data is placed on the outside edge of the platter.

The drive light is on, but nobody is home

It would be an interesting experiment to create a "ThrashMeter" utility that measured the amount of time for a given thread that the computer is in a synchronous read or write. This, expressed as a percentage, would better evaluate the performance of a computer.



Obviously, it is possible for another thread to make use of the processor while the first is stalled waiting for data. (One of the compelling arguments for pre-emptive multitasking in OSX...) However, this does not improve the performance of the first thread. If we are actively searching for methods to improve perceived performance, it is necessary to reduce the time spent stalled on these threads.

RAIDs

One method of improving the performance of a hard drive is to use several at the same time. A RAID is a **Redundant Array of Independent Drives** that use statistics to help performance. For instance, if the data you are searching for is on two drives, the likelihood is that while one of them may have to wait 7 milliseconds for the data to rotate into position, the other may only have to wait 3 milliseconds. This is an excellent way to combat the problem of rotational latency.

Head seek latency while reading can also be addressed by putting mirrored data in different positions on different drives, but this has to be done in an efficient manner because typical

mirroring uses the same layout on multiple drives. As a result, RAID-0s are not as effective in speeding up access as they could be.

In addition to the non-redundant array (RAID-0), there are at least nine types of RAID strategies:

- **RAID-0:** This technique has striping but no redundancy of data. It offers the best performance but no fault-tolerance.
- **RAID-1:** This type is also known as disk mirroring and typically consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same or slightly worse than for single disk storage. RAID-1 provides the best performance and the best fault-tolerance in a multi-user system.
- **RAID-2:** This type uses striping across disks with some disks storing error checking and correcting (ECC) information. It has no advantage over RAID-3.
- **RAID-3:** This type uses striping and dedicates one drive to storing parity information. The embedded error checking (ECC) information is used to detect errors. Data recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all drives at the same time, RAID-3 cannot overlap I/O. For this reason, RAID-3 is best for single-user systems with long record applications.
- **RAID-4:** This type uses large stripes, which means you can read records from any single drive. This allows you to take advantage of overlapped I/O for read operations. Since all write operations have to update the parity drive, no I/O overlapping is possible. RAID-4 offers no advantage over RAID-5.
- **RAID-5:** This type includes a rotating parity array, thus addressing the write limitation in RAID-4. Thus, all read and write operations can be overlapped. RAID-5 stores parity information but not redundant data (but parity information can be used to reconstruct data). RAID-5 requires at least three and usually five disks for the array. It's best for multi-user systems in which performance is not critical or which do few write operations.
- **RAID-6:** This type is similar to RAID-5 but includes a second parity scheme that is distributed across different drives and thus offers extremely high fault- and drive-failure tolerance. There are few or no commercial examples currently.
- **RAID-7:** This type includes a real-time embedded operating system as a controller, caching via a high-speed bus, and other characteristics of a stand-alone computer.
- **RAID-10 (also called 0+1):** This type offers an array of stripes in which each stripe is a RAID-1 array of drives. This offers higher performance than RAID-1 but at much higher cost.
- **RAID-53:** This type offers an array of stripes in which each stripe is a RAID-3 array of disks. This offers higher performance than RAID-3 but at much higher cost.

The most common RAID geometries are 0, 1, 3, 5, and 10 (or 0+1).

This proposal is for a new kind of RAID that exhibits many of the characteristics of traditional RAID-0s, but is intended to reduce latency as a key to improving performance.

Moore's Law on Steroids

The following figure shows how the cost of storage has been dropping since the days of ENIAC.

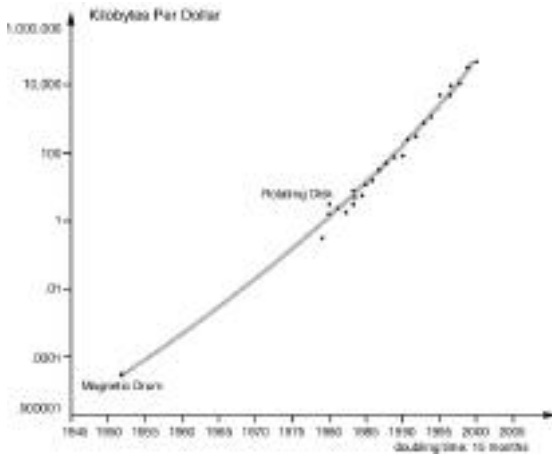


Figure 1a. Moore's Law: Mech. Storage, Kilobytes per Dollar

The price of storage of RAM is also dropping, albeit not as fast. While mechanical storage doubles every 15 months, RAM storage doubles every 18-24 months. This illustrates the need for storage with characteristics inbetween.

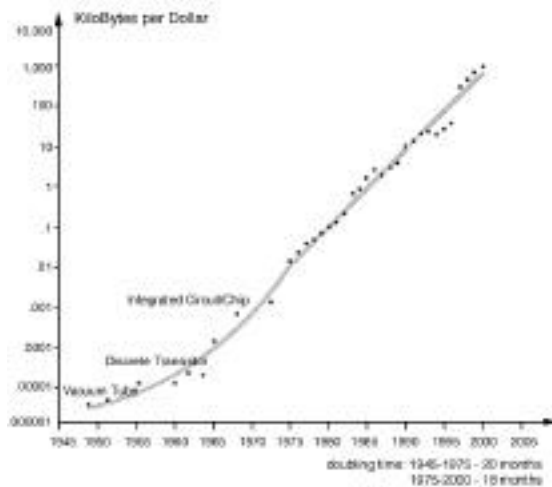


Figure 1b. Moore's Law: RAM, Kilobytes per Dollar

Our performance problem is driven by a large price/performance gap between mechanical storage and RAM storage. Not only is the cost difference more than a factor of 10x, but the Random Access performance difference is more than a factor of 100,000x. Some of this can be helped with disk caching strategies, but only certain types of applications are helped by those strategies.

Caching strategies

One basic principle behind caching is the realization that most of the data that we access, we access repeatedly. For those frequently accessed data, caching is a good thing. The statistics don't lie -- if you use just 5% of your data 95% of the time, then you can take up to 20x longer for the infrequent data before it overwhelms the frequent stuff.

There are two strategies for deciding which data should be cached:

- Most Frequently Used
- Most Recently Used

They are subtly different. Most Frequently Used (MFU) solutions are the desired goal because, statistically speaking, they insure the best possible performance results. However, it is very difficult to obtain those statistics, especially when a process is running for the first time. The Most Recently Used algorithm (MRU) is much easier to determine since the processor just went out and grabbed the data. It tends to work because algorithms generally access the same data over and over. (Except streaming media, of course.)

Just as there are cache filling strategies, there are the corresponding cache replacement strategies, i.e. who to remove from the cache.

- Least Frequently Used
- Least Recently Used

Again, they are subtly different. Least Frequently Used (LFU) solutions are the best possible but very difficult to implement since it requires a lot of historical information. As a result, Least Recently Used (LRU) keep track of how stale the data is, requiring much less effort.

Hits & Misses

If the data is in the cache, this is called a **cache hit**. If the data is not in the cache and needs to be fetched, this is called a **cache miss**. With multiple levels of caching, it is possible to have a variety of types of hits and misses, each taking different amounts of time.

Read caching is where you have data cached for read purposes only, but when data is written, it is generally write-thru so that the data is sent directly to RAM. Finally, there is **write-caching** where the data will eventually get flushed and sent back to RAM. (Write-caching can be vastly faster than write-thru, but there is a serious risk of data loss in the event of a power failure or crash before the data is actually written.)

One strategy for ensuring that data is in the cache when you need it is prefetching. Either by specifically accessing the data a short time before you need it, or by streaming the data into memory in a just-in-time fashion it is possible to take advantage of the data transmission speed and neglect the effects of latency.

However, it takes a lot of application-specific knowledge to determine which data to prefetch. Co-occurrence and repeated use data are critical for making this work.

What is "Random Access?"

For the most part, DRAM is random access memory. That is to say, it takes just as long to access *any* location within the address space -- that *memory can be accessed randomly with no penalty*.

While this was true at one time, it is not any longer. Exploiting the assumption that adjacent memory locations tend to get accessed concurrently, there are now burst modes where several adjacent locations are all read out much faster, after the initial addressing delay. This is done to try to prefill caches on a small scale and improve performance.

So we see that once caches are introduced to the picture, DRAM is not really random access, anymore. Hard drives were also assumed to be random access (to the extent that the directory structures are modeled on linked-lists or tree structures which assume random access characteristics). They most assuredly are not, as we've spent the last several pages discussing disk drive latency in various circumstances.

In reality, very few things are random access. Instead, we should consider memory and disk storage to be *Pseudo Random Access*. (It just depends how long you're willing to wait for the data to arrive!) Furthermore, our analysis of algorithm efficiency should take into account the cost(s) of accessing memory.

The goal of caching is to find patterns in access and make the cases where those patterns exist exploitable. As a side-effect, some other access patterns could become much worse.

Characteristics of the problem

There are some common applications and computational problems that require frequent disc accesses for large amounts of data. This limits the utility of using RAM as a cache for the disk.

Some of these problematic applications include:

- **Virtual Memory:** Generally, the size of the "Virtual" space is more than 2x the size of available RAM. In many cases this can be as much as 8x larger. With the introduction of 64-bit processors and address spaces this will get drastically worse.
- **Image Processing:** Programs, including Photoshop, that use large images, run through large amounts of data in a short period of time. It is quite common to have a 400MB image, with 10-20 stages of history for that image. Even though there are "tiled" methods for keeping track of the changes from image to image, this can represent several Gigabytes of non-predictable disk access.
- **File and Web Servers:** A network file server can "serve up" hundreds of gigabytes of data to the network that accesses it. With some minor exceptions, most of this data will not fit in RAM. (Google, notably, uses RAM-based storage for all of their searches. It is an expensive but vastly faster solution than disk drive-based storage.)
- **Scientific computing:** Many scientific applications, including molecular dynamics and fluid dynamics problems, require datasets exceeding 150 GB in size. Algorithmic tricks improve performance, but many scientific codes block frequently on reads and writes.
- **Database programs:** These have similar problems to file servers. The non-predictable nature of accesses make caching strategies mostly useless. An intelligent scheme to aggregate statistics on most-frequently used tables at the hardware level would improve database performance incredibly.

Each of these applications access a considerable amount of data, often unpredictably, and generally not the same data repeatedly. When they do repeat sections of data, the sizes can often be larger than the size of available RAM, making the advantages of caching limited.

With Apple shipping the XServe and making a strong move towards the low-end enterprise market, this is no longer a problem reserved for the "big-iron" jockeys and systems administrators at huge network operations centers. The increase in efficiency for programs like Photoshop where virtual memory performance is critical would alone result in phenomenal cost savings for end-users.

Characteristics of a solution

We've learned a couple of things that can be done to make hard drive access faster:

- 1) It is possible to save multiple copies of the data to different positions on the drive to solve the seek time problem. In this way, you only have to travel to the nearest copy of the data. *It is necessary to know the orientation of the platter to make the decision **which copy is closest**.*
- 2) It is possible to reorder the data on the disk so that things that are used together are temporally closer, or can even be read into memory together in anticipation of use. *A virtual remapping of physical data on the drive is reasonable, even while the drive is in use.*
- 3) Many computers are limited in the address space for the memory that can be installed. However, there is no reasonable upper limit on the size of the cache if it is part of the drive controller instead of main memory. It is quite feasible (and relatively inexpensive) to have significant

amounts of external memory to do smart caching, including write-caching, if it can be backed up with battery power.

- 4) The drive is really unaware of files. Often, just parts of files are actually used. It isn't necessary to defragment files. It is really only important that data that are used concurrently be physically adjacent. (Who said fragmentation is a bad thing? As long as I get the fragment that I want in a hurry, who cares?)

Applying what we now know about fast methods for accessing drives, we can infer some other things about an optimal solution.

- 1) The drive is unaware of which operating system that is installed on it. A robust solution should be equally unaware of the OS.
- 2) A robust solution would provide intelligent organization and optimization of cached data.

Proposed Solution "L5 Cache"

With a hard-drive or other permanent storage, it is necessary to use write-thru caching, so that in the event of a power-failure, the data is already written to the disk and no critical results are lost. Just as a RAID-1 is safe in the event of a drive failure, it is necessary for this device to remain safe even if the host has a failure.

So, in order to solve the write-caching problem, there needs to be a hardware component. This device would have the following features:

- read caching
- write caching
- safe shutdown

- smart sleep and wake operations
- power surge protection
- multiple copies of the frequent to reduce seek time
- pre-fetching
- real-time safe optimization (no danger of power failure)
- pass-through device, works with ATA drives and controllers on *any* platform (Mac, PC, Sun, Linux) because it is unaware of the underlying OS.
- variable DRAM requirements, up to and exceeding main memory size
- Could reprogram device for encryption so that all stored data is completely encrypted as well as distributed.
- Reduced storage space on drive, returns much higher effective performance
- If device was aware of the OS, it could self-heal FAT, NTFS, and HFS+ datastructures.

Faster than a hard drive. Cheaper than DRAM. Not limited to a 32-bit memory space.

So how do we build it? Clearly it has both software and hardware components.

Hardware component

The hardware would need the following:

- sufficient battery or capacitor power storage, multiple voltages
- assurance write-caching is safe
- safe shut-down
- preload data on power-up

- intelligent spin-ups and spin-downs (assuming it ever spins-down)
- platform independence -- to the outside it appears to be just another drive
- Onboard memory (DIMMs) conceivably more than the computer's address space. After all, RAM is *cheap*. Yet, still less memory than size of drive. (Shutdown times will increase with the size of the onboard memory.)
- Onboard embedded processor to do the work

Software component

The software would need to handle the following:

- read and write caching
- statistical analysis of coincident access (for optimization and prefetching)
- real-time optimization categories:
 1. most recently used,
 2. most frequently used,
 3. most commonly used together,
 4. least frequently used,
 5. least recently used,
 6. only used at machine startup

requires a self-correcting database from *virtual* to *physical* locations
- power management of the hardware device, including safe flushes and handle errors in instructions from the host in a robust fashion

Intelligent RAID

Thus, we get some of the characteristics of RAID storage. There can be redundant data storage like a RAID-1 (mirroring). The read and write access times can be significantly improved as you would get in a RAID-4 or -5. It is possible to include error correcting characteristics from a single drive. There is no hot-swap ability unless more than one drive is attached to the system.

Common data is written more than once on the drive giving a faster effective seek time.

If more than one drive is used, really frequent data can be written to the faster of the two drives. However, the best results will come from having the data on both drives.

One drive may be more appropriate for seeking while another may be better for streaming. This is very true with the inner edge and outer edge of drives in general

This L5 strategy could be implemented in conjunction with other RAID strategies to improve latency as well as data transmission speeds.

Conclusion

There are three obvious ways to implement these kinds of strategies with hardware that will fit into a standard PC, each with advantages and disadvantages.

The first (see Figure 2a, below) is a PCI card with space to attach an unused 2.5" hard drive. This is especially interesting since many of us have left-over laptop drives from upgrades. The serious disadvantage is that a driver for each OS must be written.

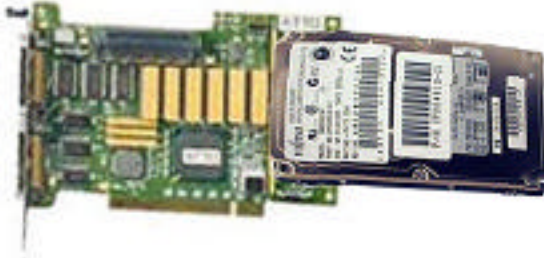


Figure 2a. PCI card with attached spare notebook 2.5" drive

The second (see Figure 2b, below) is a pass-through connector that makes the computer think it is attached to a normal, run-of-the-mill hard drive. In this way, it doesn't matter what kind of target system is used. It should work equally well with anything from Windows to Linux to MacOS to Solaris. The drive will appear smaller because we're writing some data multiple times on it.



Figure 2b. Pass-through ATA connector implementation

Eventually, this type of battery and caching technology will end up *inside* the hard drives themselves. This will offer

less expandability, but could provide more fuel to the hard drive market.



Figure 2c. "Smart" hard drive with built-in battery backup and DIMM memory slots.

Each of these devices could be easily built. Which one will reach the consumer first? It depends on the end consumer. Some applications require the performance that a faster hard drive could provide, including Photoshop users, web servers, database systems, etc.

In addition, most current and older hardware can benefit from this technology. They all increase the efficiency of the host computer drastically reducing the latency of current hard drives.

I eagerly await the day that my "drive light" goes out.