Fast Processors, Slow Hard Drives... What's wrong with this picture?

The anatomy of an "ideal" disk optimizer.

By Chris Russ

Reindeer Games, Inc.

This paper describes methods of achieving optimal performance from mechanical memory where latencies measured in milliseconds are the primary culprits in the loss of computer performance. This includes file arrangement, data structures, and the kinds of prediction of future events that are really possible.

Overview - Why is disk performance the issue?

Generally, the computer's organizational structure is a pyramid. The really fast accesses are in a small memory space with slower accesses in large memory spaces. The following illustration shows this:

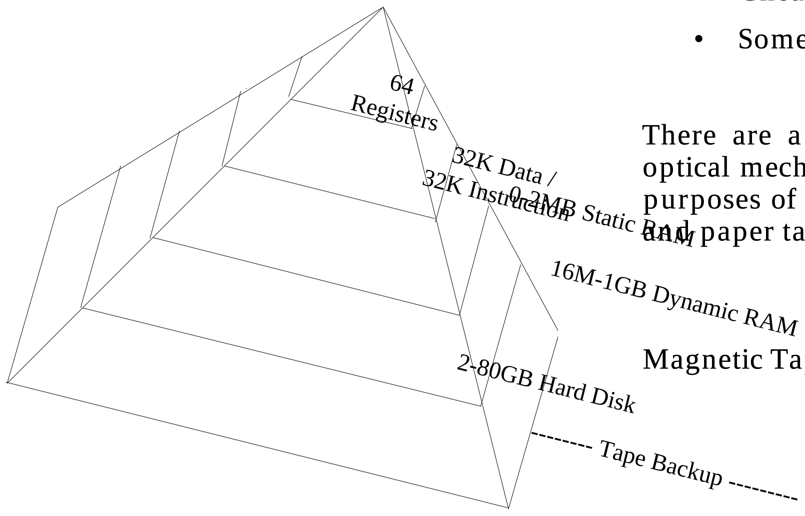


Figure 1 - Memory Hierarchy Pyramid

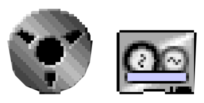
Mechanical Storage

The bottom of the pyramid is filled with very large and slow devices. Although the sizes have changed over the years (and some devices like paper tape and punch cards are omitted here) generally they have the same properties when compared to non-mechanical memory:

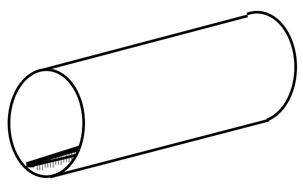
- Big
- Slow
- Cheap
- Sometimes removable

There are a few basic types of magnetic and optical mechanical devices for storage. (For the purposes of this discussion, ignore punch cards and paper tape.)

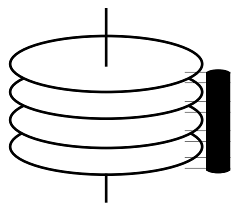
Magnetic Tape



Drum



Disk



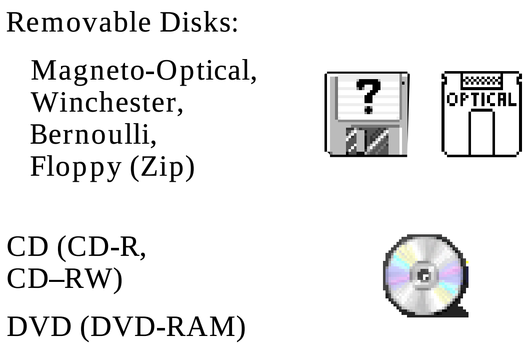


Figure 2: Mechanical R/O and R/W Mass-Storage Device

Tape has a *really* long latency, depending upon where the data is on the reel. (Even longer if the reel is not mounted.) Because of this property, it is generally better to get more than one thing at a time while searching. However, streaming tape can be a very good way to record large volumes of data at high speed. (To this day, NASA still has magnetic tape from the Apollo program that has never been read or transferred to another format. It may no longer be readable.)

Drum has only a rotational latency that is relatively short, but requires many heads, and the amount of storage is relatively small. It is listed here as a blast from the past, but it does illustrate that there are other kinds of mechanical storage than disks and tape.

Normal hard disks have a short rotational latency, but potentially high seek latency. Thus, while a disk may have a rotational velocity of 7200 RPM it may have an average seek time of 13 msec. What is average seek time? It depends upon the data provided by the manufacturer. It is not well defined. The easiest way to increase the size of a hard disk is to increase the number of platters, which has little effect on the seek time.

There are many kinds of removable media such as optical disks, magneto-optical disks, Winchester and Bernoulli drives, Zip and floppy drives.

Optical disks and magneto-optical disks tend to be very non-volatile, but require relatively large heads. Because of their high mass, the seek times for these heads are also high. This includes CD-R, CD-RW, and DVD-RAM. Note: the description 12x or 20x merely refers to a faster rotation speed and a higher data transfer rate. The head seek times are still 3-5x longer than equivalent hard disks. Some of the drives like CD-R's are very slow when writing data and seeking.

Winchester drives are much faster, but cannot compete with hard disks since it is very difficult to make a multi-platter version of a removable Winchester. They are also subject to magnetic fields in the same way that a hard disk is (and optical and magneto-optical media are not). There are data density limits since it is difficult to completely seal the platter while at the same time keeping the head and supporting hardware in the drive unit.

Floppy media has come in and out of vogue many times. Everything from single density 8" floppy drives to 100MB zip disks constitutes this family. The primary consideration is price, not speed, for these devices.

Read Only removable media such as CD's and DVD's are not suitable for the main active memory space, but many of the optimization tricks discussed later in the paper will apply.

Because of all of these properties, hard disks (and RAIDs) have become the standard for large capacity active memory (at the bottom of the pyramid in figure 1). At this time, tape is still the leader for raw capacity and is used for archival purposes, although the removable media types do compete.

High Level - Where to start?

The first problem with designing an optimizer for a disk is to examine where the problems lie. Contrary to popular opinion, the problem isn't fragmentation. The primary problem is really how much time is wasted while the disk head seeks from one position to another. The secondary problem is how many seeks are required to get all of the requested data.

Disk Optimization Goals

The goals for optimizing a disk are two-fold:

1) **Minimize Head Seeking**: Fragmentation is only a problem if a lot of time is spent seeking to the fragment required.
2) **Make the most of a seek operation**: If seeking is required, get other data along the way. This also means that a larger block size should be read in hopes that the nearby data will also be useful.

Disk Optimization Solutions

Try not to access the disk if possible. Caching works very well if the same data is requested more than one time, subject to the size of the cache. The major problem with caching is that it does nothing for data that is requested the first time and can cause increased overhead if that data does not get used again.

If it is necessary to access the disk, the following methods minimize the amount of seeking:

A) Put the Most Frequently Accessed data in the middle of the platter. This will reduce the average seek time. (Thus, partitioning is a bad from a speed perspective.)

B) The data that is referenced concurrently should be in close spatial proximity.

C) Data that is accessed sequentially (as opposed to randomly) can be placed toward the inside and outside edges of the disk (movies, audio).

D) Infrequently Accessed data can be placed as far out as possible toward the inside and outside edges. (Help files, reference data, old user data, utility applications.)

There is another method of caching data. If the data is too large to fit in a RAM-based cache, but it is accessed frequently and is static, it can be placed in multiple places on the disk. In this way, a seek must only travel to the nearest copy instead of the only copy. This could be very effective for system related data.

Mid Level - Candidates for Improvement

It is one thing to say "put infrequently used data as far out as possible" and something else to figure out what is frequent and what isn't.

User and OS behavior become an issue when determining frequency of use. It is important to look at what the user does.

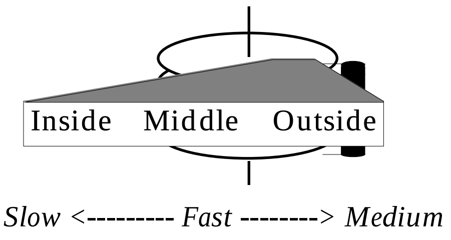
Types of user access:

1) Booting
2) System Files, Libraries, Finder, Fonts
3) Launching an Application
4) Running an Application
5) Running Multiple Applications
6) File Sharing
7) Accessing documents - read only
    a) Large Sequential Access
    b) Random Access
8) Accessing documents, creating new ones
9) Accessing documents, changing portions of them
    a) Large Sequential Access
    b) Random Access

Typically, each of these files has a different disk usage pattern, and are fairly easy to characterize. For instance, the boot sequence files are readily identifiable from their positions in the disk directory and file types.

The disk can be broken into speed zones. Different portions of the disk provide for faster access than others.

One way to visualize this is to picture the disk in zones around the ideal center:



Inside Edge: Medium data rate, slow random access, good sequential access (like a tape)

Middle: Good data rate, fast random access, good sequential access (like a drum)

Outside Edge: High data rate: slow random access, excellent sequential access (like a fast tape)

Booting

While making a Macsbug log of the files opened during the boot process (Mac OS 8.5.1 with IE removed), over 600 file open references were made prior to Finder Launch.

This includes every access to each font, each ‘ndrv’ driver over a dozen times, each ‘cdev’ control panel, each ‘INIT’ extension, QuickTime over 90 times, the System file over 100 times, and over 130 references to preference files.

What the user sees during the boot is a small subset of the files opened. Generally these include only the extensions and control panels from Apple or third parties.

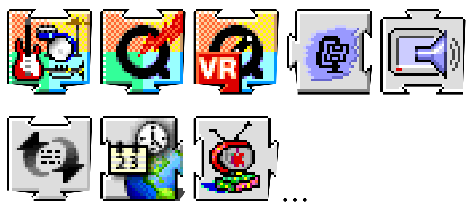


Figure 4 - Parade of icons

If the boot sequence files, including their respective preference files, were all stored together the only significant seeking would be to the directory to find the locations of the files. Ideally, at the start, the directory should be referenced to get all the directory contents of the System Folder, Extensions Folder, Fonts Folder, and Control Panels Folder loaded into the cache. Then, with the disk access time minimized, the remaining boot time is largely execution time. If all the extensions could be loaded as one large file, the disk access time could be virtually removed. *[Author’s Note: in tests this cuts boot times in half.]*

Note: The real problem with this notion is that some members of the boot sequence write to the disk. The directory has to be updated, and writies either to the extension or its respective preference file is much slower. Ideally these writes could be saved up to the end.

Files that remain open

It turns out that under Mac OS 8.5 a large number of files are constantly open. Without considering the effects of IE (which was removed for other reasons) there are 58 files open occupying 64 FCB’s. Two of those FCB’s were the directory files, but the truly interesting part here is which files are open. (See Table 1, below.)

Naturally, the System and Finder are open. There are a few support files for Finder that probably do need to be flushed out to disk as they are changed — the Desktop Database files, the Users & Groups File (assuming it changes much, certainly not a given), and Finder Preferences are really the only ones that should be allowed to change on the disk.

Table 1 - list of files open under Mac OS 8.5.1 (without IE)

4 Background Apps (appe)
- Application Switcher
- Control Strip Extension
- Folder Actions
- Time Synchronizer

3 B*Tree data files (BTFL, DTFL)
- User & Groups Data File
- Desktop DB
- Desktop DF

34 fonts (FFIL)
- Varies, depends upon user

Finder (FNDR)

2 Inits (INIT)
- QuickTime™
- Shared Library Manager PPC

13 Libraries (libr)
- 6 Open Transport (11 FCB's)
- IrLanScannerPPC
- Serial (Build-in)

1 Preference file (pref)
- Finder Preferences

1 Shared Library (shlb)
- File Sharing Library (2 FCB's)

System Files (zsyr, zsys)
- System Resources
- System

Directory (files #3 and #4)
- Catalog Tree
- Extent Tree

Except for the directory files, these open files should either completely exist in memory, or be where they can be accessed quickly. Unfortunately, resource files incur additional overhead. The best bet would be to completely cache in memory as many of these files (except the catalog and extent tree files) as possible. If files aren't used, VM will swap them out. (The VM system will be addressed below.)

The disk directory is really a couple of files. They are the second-most and third-most frequently accessed files in the system behind the VM backing store. Because of this, anything that can be done to reduce seek time will make a significant difference. The obvious place to put this data is right in the middle of the volume. This way, the average seek time is half of what it would be if it was placed on the inside edge (typical position).

Alternatively, the system files in the list rarely get modified and are good candidates for a set of static data caches.

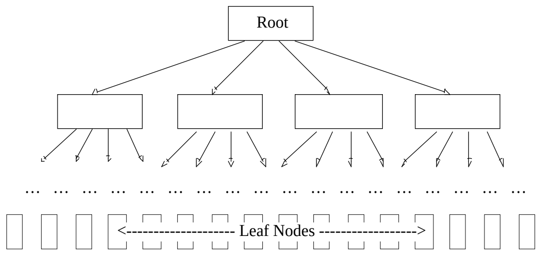


Figure 5 - Disk Directory Tree

It is possible to optimize the contents of the Directory B* Trees as well, with significant performance improvements. To put it simply, placing B*Tree nodes that are used concurrently in adjacent blocks in the catalog or extent files, causes seek latency to be reduced. This is really a low-level issue and will be addressed later.

[Author's Note: The current implementation of PBGetCatInfo() is slow for a catalog indexed miss. Since there is a miss at the end of every directory, Finder and the Standard File Package spend a lot of time building the complete list of files for any path.]

Safe File Updates and Swapping

This is one of the big contributors to making a disk slower. If frequently used files are placed near their respective applications, and they are constantly being replaced by modified copies (that are usually farther away), the performance of the system is degraded. In addition, fragmentation does occur since there are now a lot of holes where the files used to be. Defragmentation programs can address part of this issue, but do not place files in good proximity.

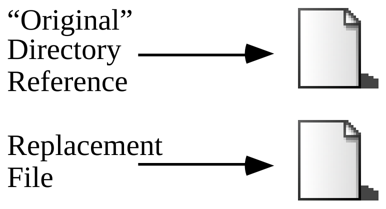


Figure 6a - Write the new file

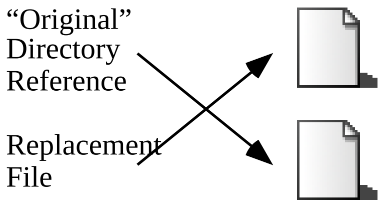


Figure 6b - Swap directory references

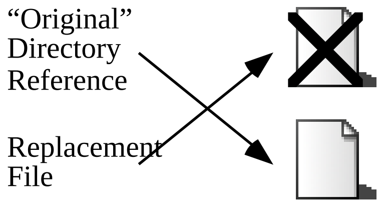


Figure 6c - Delete the old file using new reference

File Categorization

Applications are good candidates for caching. Generally, they are read only and accessed infrequently to load in some new piece of code. VM systems map application files into the memory space at specific addresses and achieve further speed improvements, unless the application files are far away from the VM backing store on the volume.

The bigger problem comes from the document and preference files for the (hopefully nearby) applications. The goal is to reduce seek time, so it is a bad idea (and the worst case!) to place documents and their respective applications on opposite ends of the disk.

Q: How should an optimizer tell if an application gets used a lot? (Most Frequently Used)

A: It has a lot of documents with a variety of modification dates.

Q: Which applications were Most Recently Used?

A: If there is a preference file that has a recent modification date, or a document of the application's creator and type with a recent modification date, it is a candidate.

Q: For a document instead of an application, how Frequently Modified is it?

A: The best estimator is the difference between the creation date and the modification date. There will be a lot of documents with little difference between the creation date and the modification date. However, there will be a few that get modified often. (This is especially true of preference files.)

Q: What about a static document? There are no indicators for how often or recently it was accessed.

A: True. The only solution is to watch the OS and see what files fit into this category. Or, on the basis of file type or a list of typical files, this may be possible. For specific application (at least the popular ones) it is possible to make such a list (i.e., CodeWarrior, Photoshop, etc.).

This gives us a few categories to put files into:

- Boot Files (Mostly read only, accessed together and then not again until next time)
- Disk Directories (R/W in place, very frequently)
- System Files (not VM) (Read Only, accessed frequently)
- VM Backing Store (R/W in place, most frequently)
- VM Photoshop (same as VM Backing Store, temporary file, grows, may exist on multiple volumes)
- Frequently Used Applications (Read only, multiple accesses within the file, lots of concurrent read accesses to document files, part of VM memory depending upon implementation)

- Frequently Used Applications that create or modify documents a lot (lots of potential head seeking to the new or modified documents)
- Infrequently Used Applications (generally can place related documents nearby, but can be safely placed in slow zones)
- Static files that are frequently accessed (help, desktop graphics, reference materials, libraries, include files)
- Documents that are frequently modified or created (includes temp files) (Best placed right next to their respective applications)
- Documents that are rarely used
- Sequential access multimedia files

This is a lot more than the three basic categories of Document, Application, and System.

Virtual Memory

Everything revolves around the virtual memory file, called the VM Backing Store. It is the most frequently accessed file in the system.

The best choice is to purchase a RAID device (level 0 is sufficient, no redundancy is needed unless the cost of the system going down is *really* high) and use it for the VM backing store. It only has to be as large as the VM backing store it is supporting. Access time is a big issue. The lower the effective seek time, the better. No other data should be stored on this device, since accessing that data would affect the VM performance. A cheap 4 GB RAID system with its own Ultra SCSI-3 (or firewire) controller card would be perfect.

The second best choice is to place the VM file at the center of the volume. Conversely, if the VM is really large, it should be split into two fragments with the disk directory files at the center of the volume.

This is complicated by mapping applications into the Virtual Memory space. These files effectively become part of the VM backing store, so if they are far away from the center of the disk, performance will again be degraded. The most frequently used applications should be near the VM file, while at the same time as close to their currently used documents. Thus, a swapping space on the volume to push an application and its currently used files next to the VM file (and the center) would make a performance improvement.

Photoshop’s Virtual Memory

A significant fraction of the Macintosh’s user base uses Adobe Photoshop™. Since Photoshop has its own virtual memory system for storing large images (and undo information), it has the same set of problems that the operating system has, except that the PS VM file is temporary. It only exists during the execution of the application, and grows as it needs to. Ideally it should also have its own RAID device, with its own controller card.

Failing a separate hard disk for the PS VM file, the file would benefit from being near the Photoshop application (which in turn needs to be near the OS VM file if VM is on). Fragmentation is also an issue with this large file being created and grown while other files are written to disk. One reason is that the directory extent file needs to be accessed to locate the fragments. Another is that the PS VM file is deleted when Photoshop quits and the holes are partially filled in by other applications’ documents. Fragmentation happens quickly. As a direct result, pieces of files are everywhere and seek times go up. *[Author’s note: this is one of the cases where a traditional Defragger makes some difference -- by creating a single large hole for the PS VM file.]*

File Placement

> **Q:** Now that the file characteristics have been determined, where should they be placed?

The next three figures (7a, 7b, and 7c) show some disk layouts. The first two are the most common optimization patterns, and the third is the proposed “Ideal” layout that incorporates the changes that were discussed above.

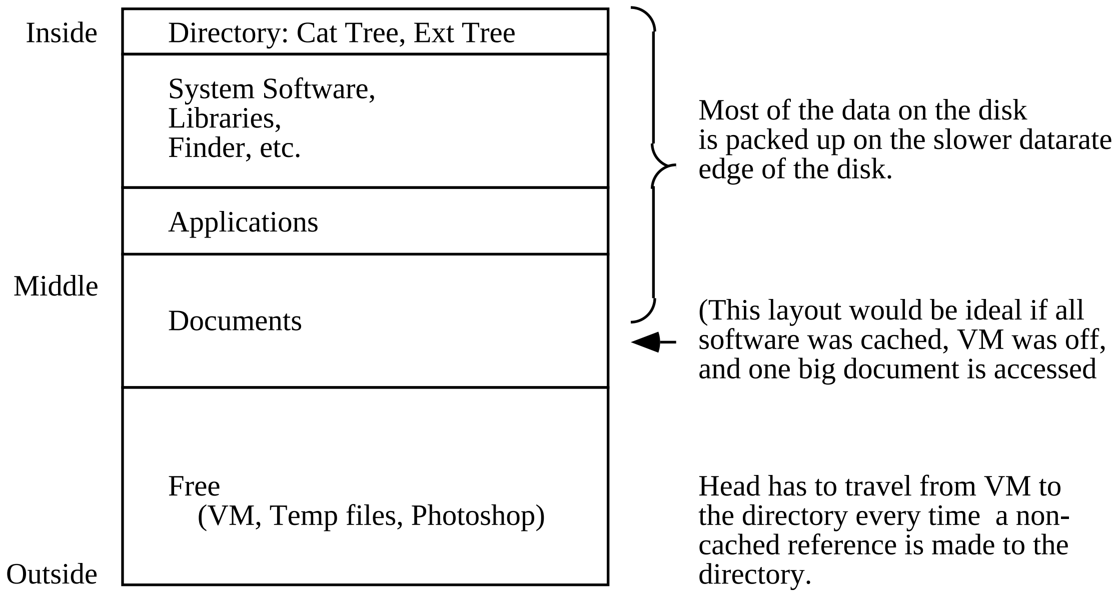

Inside
Directory: Cat Tree, Ext Tree
System Software, Libraries, Finder, etc.
Applications
Middle
Documents
Free (VM, Temp files, Photoshop)
Outside
Most of the data on the disk is packed up on the slower datarate edge of the disk.
(This layout would be ideal if all software was cached, VM was off, and one big document is accessed
Head has to travel from VM to the directory every time a non-cached reference is made to the directory.

Figure 7a - Most common optimization layout

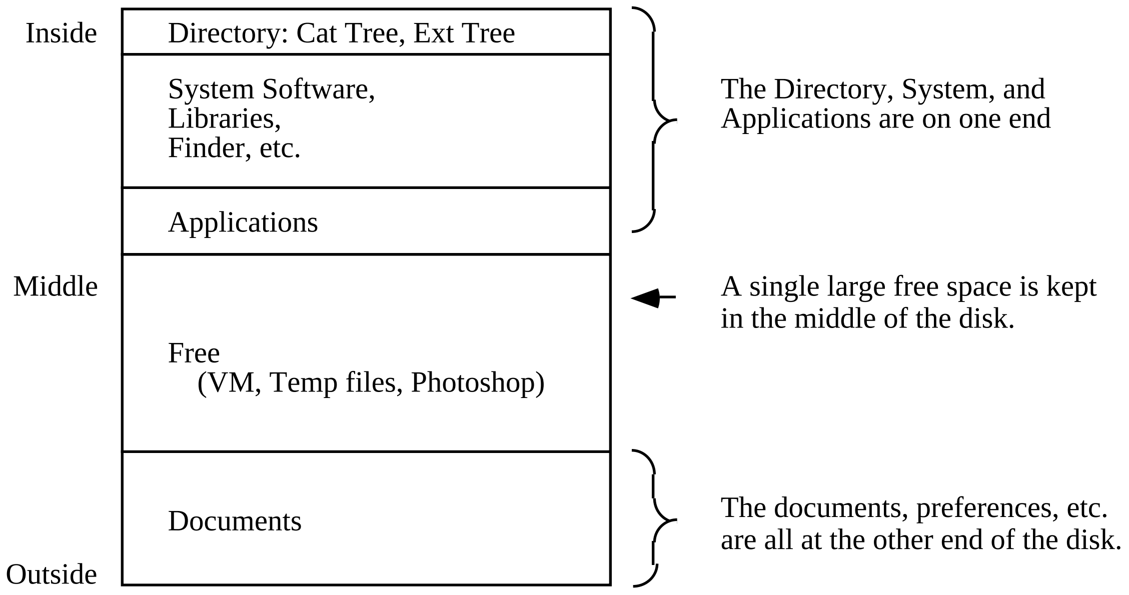

Inside
Directory: Cat Tree, Ext Tree
System Software, Libraries, Finder, etc.
Applications
Middle
Free (VM, Temp files, Photoshop)
Documents
Outside
The Directory, System, and Applications are on one end
A single large free space is kept in the middle of the disk.
The documents, preferences, etc. are all at the other end of the disk.

Figure 7b - Another common optimization layout (Worst Case?)

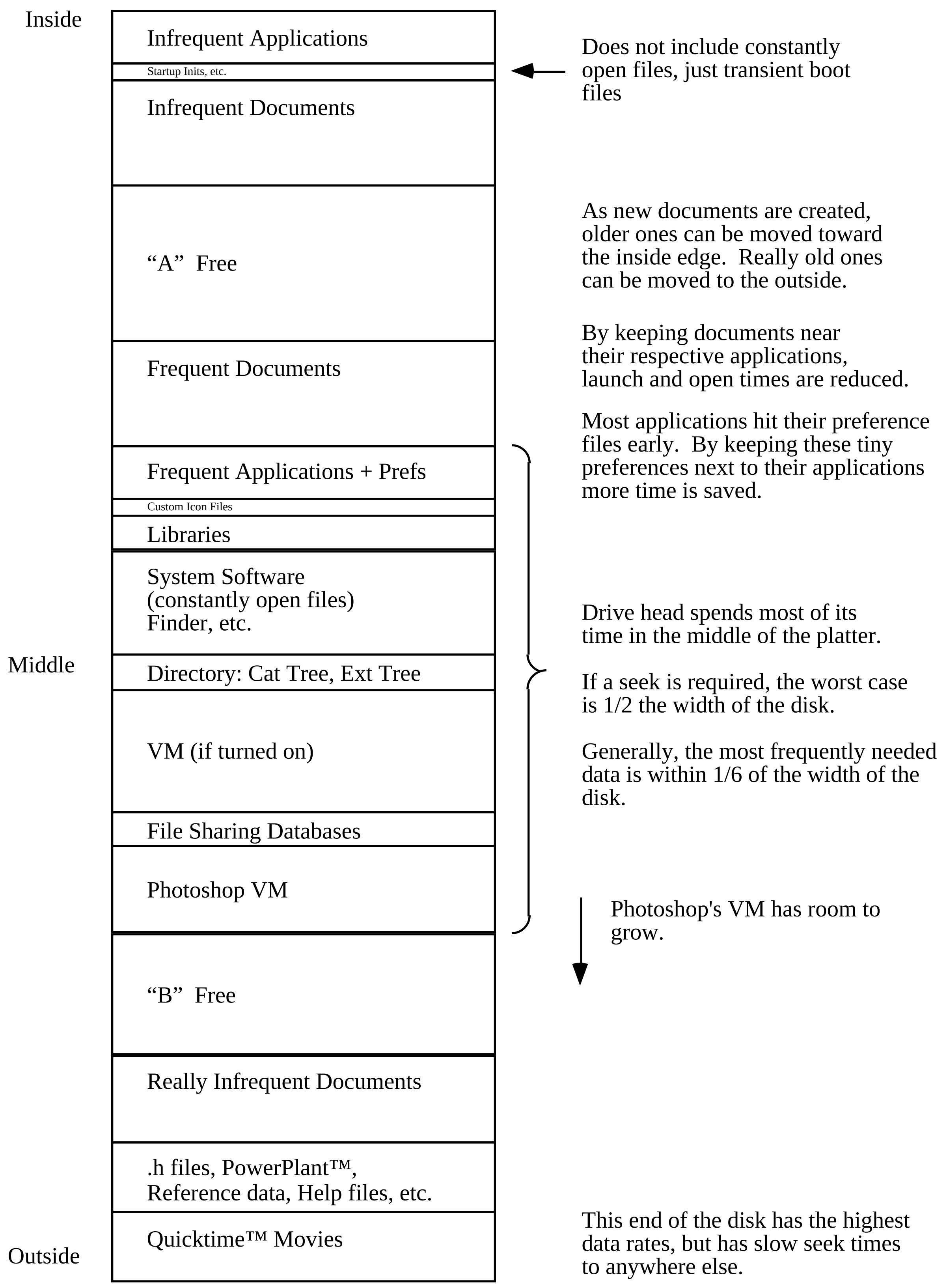

Inside
Infrequent Applications
Startup Inits, etc.
Does not include constantly open files, just transient boot files
Infrequent Documents
"A" Free
As new documents are created, older ones can be moved toward the inside edge. Really old ones can be moved to the outside.
By keeping documents near their respective applications, launch and open times are reduced.
Frequent Documents
Most applications hit their preference files early. By keeping these tiny preferences next to their applications more time is saved.
Frequent Applications + Prefs
Custom Icon Files
Libraries
System Software (constantly open files) Finder, etc.
Drive head spends most of its time in the middle of the platter.
Middle
Directory: Cat Tree, Ext Tree
If a seek is required, the worst case is 1/2 the width of the disk.
VM (if turned on)
Generally, the most frequently needed data is within 1/6 of the width of the disk.
File Sharing Databases
Photoshop VM
Photoshop's VM has room to grow.
"B" Free
Really Infrequent Documents
.h files, PowerPlant™, Reference data, Help files, etc.
Quicktime™ Movies
This end of the disk has the highest data rates, but has slow seek times to anywhere else.
Outside

Figure 7c - "Ideal" disk layout diagram

In the first case (Figure 7a), all the files are packed up against the inside edge of the disk. The primary reasons for this are that the directory was already placed there when the drive was formatted, and the volume bitmap (which contains the list of which blocks are currently in use) is located right there. If writing is going to take place, quick access to the volume bitmap is a good idea. In general, only half the distance of the disk needs to be traversed to get anyplace and the head spends most of its time near the inside edge.

In the second case (Figure 7b), the documents are put at the far end of the disk from the applications and other data that is located at the inside edge. The goal is to put the empty space on the drive in the middle so that VM is faster. The unfortunate side-effect is that every time a new document file is opened, the head has to seek from the directory to the document on the far side of the disk. This is the worst case for seeking.

The third case (Figure 7c) tries to place the most frequently accessed material on the disk in the center. The average seek time can be reduced to approximately 1/6 of the second case and approximately 1/3 of the first case. There is one problem with this proposal; while it is possible to move the Volume Bitmap (VBM) to the center of the platter, the Master Directory Block (MDB) is a problem. The MDB contains global information like the next CNID (Catalog Node ID) and how much free space there is on disk and needs to be updated every time a file or folder is written or changed. That would not matter for a Read-Only medium (such as a CD-ROM), but for a R/W device this is a problem.

One viable (but ugly) hack for 7c would be to partition the disk into three equal partitions: Inside, Middle, and Outside. The middle partition receives the OS and the VM files. Then, allocate some additional space on the middle partition to hold the storage for a Foreign File System that will control the other two partitions as one volume. In this way, the complete directory information will be at the center of the disk, the best performance will be achieved, and a really ugly driver or boot-block hack will not be necessary to move the MDB. It will be moved to another partition. This also allows the foreign file system to keep track of the files that are being written and do a better job of optimization.

Low Level - Directory Performance

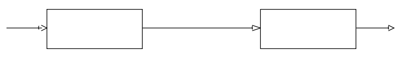
A simple test was constructed to measure boot times. By building a full system (System 7.01) and placing it and a minimal directory tree on a disk, boot times for a MacClassic were 2x faster than an optimized (case 7a - MacTools) disk.

Q: Why would this matter? The system software and the boot files are in the same place. There is no obvious head seeking from one side of the disk to the other.

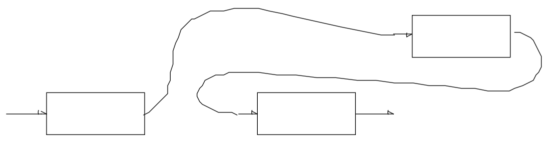
A: The directory tree on the new volume was not fragmented. The old volume had directory nodes and leaves everywhere, so that the process of indexing down a directory with PBGetCatInfo() or locating a file for the first time was very time consuming.

The problem goes like this:

Take a pair of nodes or leaves that are adjacent but need to have new data inserted between them.



A new node is allocated (from the free space at the end of the B*Tree file) and linked in-between.



This kind of insertion in a linked-list (or a doubly linked list) is the whole basis behind pointers and list data structures. Unfortunately, the underlying assumption is that the memory is random access. On a hard drive it isn't exactly random access. Some kinds of accesses are faster than others. Thus, some places in memory are faster than others. Accesses within the same cylinder are the fastest and only wait for the disk latency. Beyond that, accesses that are nearby are much faster than accesses that are far. Since the free space in the B*Tree file is at the end of the file, and the odds are that at least one of the two nodes that the insertion occurs between are not at the end, a lot of seeking will go on.

Q: Doesn't caching fix this problem?

A: Yes and no. Obviously it won't work for the first access, but also there is a limited amount of cache memory available. The additional problem is that the cache would be more efficient with a lot of small pages than a few large ones. However, it takes just about as long to read a full cylinder on a hard disk as it does to read one block. Thus it helps if the necessary data is contained within the same cylinder most of the time. The cache will be much more efficient.

The problem with the B*Tree isn't just theory. There are two practical examples that illustrate the potential gains.

- **MacTools**: When MacTools repaired the directory, it actually made a complete new one. That new directory file was written to disk and the Volume Info Block (Master Directory Block or MDB) was updated. The biggest surprise was the reports from users that the computer seemed faster. In fact, it was.
- **TimesTwo 2.0:** TimesTwo was slow. TimesTwo was very slow. In to make the performance better a number of tricks were used (including not compressing the system files or the disk directory), but the biggest trick was to build an optimized Catalog B*Tree file. On slower media (particularly with low rotation speeds and high seek times) it made a very large difference.

Q: So, how should a Catalog B*Tree be re-arranged to be more efficient (since changing the data structure is not an option)?

A: The catalog nodes and leaves for the following should be tightly packed at the start of the Catalog File:

1) Links to Desktop DB,DF
2) System Folder and all folders and files contained within, but especially the ones that are hit first during boot: Root of System, Fonts, Preferences, Extensions, and Control Panels
3) Root of the boot volume
4) Folders containing applications that have frequently or recently modified files
5) Folders containing applications with aliases under the Apple Menu, on the desktop, and on the root of the volume
6) Folders containing recently used documents.

Leave strategicly placed gaps (empty leaf records) for future growth.

Note: Metrowerks include files and linking libraries would not fall under any of these since they are neither recently modified nor do they have aliases. However, if their respective nodes are all kept together, there would be significant speed increases in Metrowerks.

Prediction of Future Events and Idle Time

Pre-filling caches is one other method of improving the apparent performance of the machine. If there is some way to determine what the user is likely to do next based upon his previous history, then new data or code can be pre-loaded into the cache in anticipation of the next event.

For instance, if the user is in the process of quitting TeachText and 30% of the time launches ClarisWorks next, but only 2% of the time relaunches TeachText, then it makes sense to remove TeachText from the cache and load ClarisWorks in its place. This would all be done in idle time while the user is still navigating in the Finder and moving the mouse, well before the application launch. Thus, 30% of the time the machine would appear faster.

Photoshop does something like this -- when the screen is updating after a processing operation, the first tile to be rendered on the screen is the one under the cursor. It still takes as long to draw the image on the screen, but the user has a better experience.

Some of the following knowledge can be collected to accomplish this feat:

- What concurrent activities does the user do? Are the same 3 programs always launched together?
- What sequential activities does the user do? It is fair to assume that when an application is terminated that it will NOT be used again? (Is this reasonable when the user isn't trying to benchmark application launches?)
- Are there ways to improve the cache usage? Specifically, what should NOT get cached? Files that are used in a sequential manner and are rarely re-used will completely flush a cache that uses an LRU replacement algorithm. Example: a 50MB QuickTime™ movie and an 8MB cache. The cache will end up holding the last 8MB of the movie after it has played. If the big fireball ending in the movie is about to be replayed (if the application was MoviePlayer, for instance) this might not be a bad thing. However, in most non-editing applications this would not be true.
- Are there certain types of files that only get used with a specific application? If so, can they be pre-loaded when the application is launched (preference files), or can they be purged when the application quits (most documents)?

This information needs to be collected by watching the user's activities. The prefilling is best performed during user idle time or while the disk is idle so that there is no apparent slow-down in the event that the guess is wrong.

Conclusion

There are really two philosophies for optimization: Static Optimization (stand-alone applications) and Dynamic Optimization (extensions). For the most part, the current tools are static. Occasionally the Disk Optimizer is scheduled to run (or the Disk Optimizing Screen Saver) putting all the files in semi-optimal positions.

Static Method - Run the application from time to time to reorganize the contents of the disk and directory based upon statistics from the data present. *BDOW* (Best Disk Optimizer in the West). (Note: this would work very well for improving CD-ROM performance, too.)

Dynamic Method - Get to watch what the user is doing and use that information effectively. Here are some possible utility programs:

- **Least Impact** (tell how much time is being spent waiting for the head to seek) *Thrash Meter.*
- **Low Impact** (pointing the allocation mechanism to *better* places for files to be written, including temp files vs. documents, also when the Safe File Save & Swap occurs, swap positions of the files on disk if they fit and the size < xx) *Hole Master.*
- **Medium Impact** (swap files that are going to be used a lot for a while to the middle of the disk and push files that are not farther out. Do a better job of managing applications next to pref files.) *Disk Meister.*
- **High Impact** (figure out what the user is going to do next, based upon previous activity, and asynchronously preload

data based upon the probabilities.) *Mental Pick-Pocket*.

It is a lot easier to write a static optimizer than a dynamic one. However there are a lot more potential gains from a dynamic one.

A better static optimizer can be constructed that, for a brief period of time, will provide really good results. The challenge is gathering the knowledge that will make an optimal layout. A static optimizer is easier to write than a dynamic one, and it is generally safer since there is no danger of something sneaky going on in the background that could cause data to be lost.

A good dynamic optimizer, on the other hand, can collect the knowledge about the user and the operating system needed to really be optimal and stay that way. The problem to be overcome is safety. It is imperative that any process that changes data structures on the disk not be susceptible to a crash at the wrong moment, and that it be immune to other processes that are running on the system.

The alternative is to make the disk faster. A little money spent on extra drives and controllers or small RAIDs will have a big performance impact. The idea of buying a 800MHz computer with a cheap partitioned 8GB IDE drive and expecting performance is insane.

Remember, adding more memory makes the machine faster because more data can be cached in memory at once. It still doesn't help loading something the first time.

As volume size increases and the number of files on the volume increases, any performance problems with the data structures in the file system will become magnified. An improvement to the B*Tree allocation mechanism, or a simple optimizer for the B*Tree can have profound improvements on the performance of the computer.