Section 11: I/O, File Systems

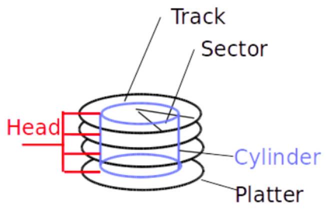
CS 162

July 29, 2020

Contents

1 Vocabulary

- **Hard Disk Drive (HDD)** - A storage device that stores data on magnetic disks. Each disk consists of multiple **platters** of data. Each platter includes multiple concentric **tracks** that are further divided into **sectors**. Data is accessed (for reading or writing) one sector at a time. The **head** of the disk can transfer data from a sector when positioned over it.



- **Seek Time** - The time it takes for an HDD to reposition its disk head over the desired track.
- **Rotational Latency** - The time it takes for the desired sector to rotate under the disk head.
- **Transfer Rate** - The rate at which data is transferred under the disk head.
- **Checksum** - A mathematical function which maps a (typically large) input to a fixed size output. Checksums are meant to detect changes to the underlying data and should change if changes occur to the underlying data. Common checksum algorithms include CRC32, MD5, SHA-1, and SHA-256.
- **Replication** - Replication or duplication is a common technique for preserving data in the face of disk failure or corruption. If a disk fails, data can be read from the replica. If a sector is corrupted, it will be detected in the checksum. The data can then be read from another replica.
- **Simple File System** - The disk is treated as a big array. At the beginning of the disk is the Table of Content (TOC) field, followed by data field. Files are stored in data field contiguously, but there can be unused space between files. In the TOC field, there are limited chunks of file description entries, with each entry describing the name, start location and size of a file.

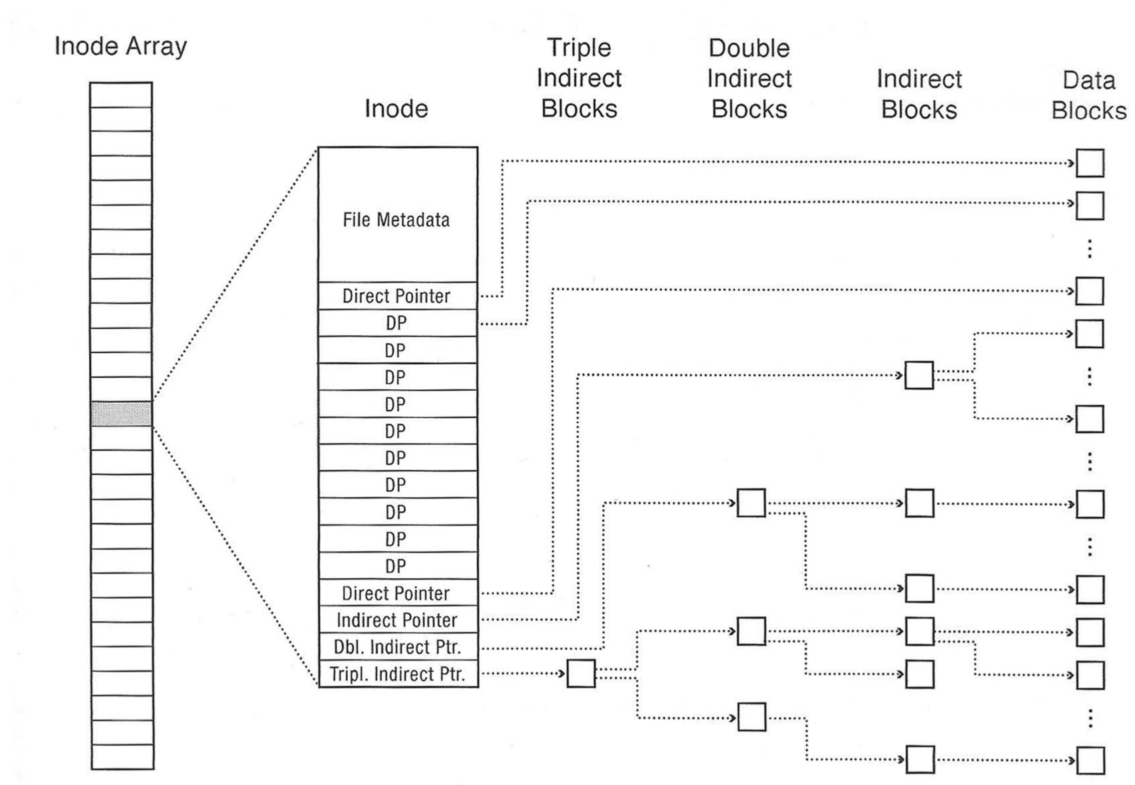
  Pros and Cons

  The main advantage of this implementation is simplicity. Whenever there is a new file created, a continuous space on disk is allocated for that file, which makes I/O (read and write) operations much faster.

  However, this implementation also has many disadvantages. First of all, it has external fragmentation problem. Because only continuous space can be utilized, it may come to the situation that there is enough free space in sum, but none of the continuous space is large enough to hold the whole file. Second, once a file is created, it cannot be easily extended because the space after this file may already be occupied by another file. Third, there is no hierarchy of directories and no notion of file type.
- **External Fragmentation** - External fragmentation is the phenomenon in which free storage becomes divided into many small pieces over time. It occurs when an application allocates and deallocates regions of storage of varying sizes, and the allocation algorithm responds by leaving the allocated and deallocated regions interspersed. The result is that although free storage is available, it is effectively unusable because it is divided into pieces that are too small to satisfy the demands of the application.
- **Internal Fragmentation** - Internal fragmentation is the space wasted inside of allocated memory blocks because of the restriction on the minimum allowed size of allocated blocks.

- **FAT** - In FAT, the disk space is still viewed as an array. The very first field of the disk is the boot sector, which contains essential information to boot the computer. A super block, which is fixed sized and contains the metadata of the file system, sits just after the boot sector. It is immediately followed by a **file allocation table** (FAT). The last section of the disk space is the data section, consisting of small blocks with size of 4 KiB.
- **Unix File System (Fast File System)** - The Unix File System is a file system used by many Unix and Unix-like operating systems. Many modern operating systems use file systems that are based off of the Unix File System.
- **inode** - An inode is the data structure that describes the metadata of a file or directory. Each inode contains several metadata fields, including the owner, file size, modification time, file mode, and reference count. Each inode also contains several data block pointers, which help the file system locate the file's data blocks.

  Each inode typically has 12 direct block pointers, 1 singly indirect block pointer, 1 doubly indirect block pointer, and 1 triply indirect block pointer. Every direct block pointer directly points to a data block. The singly indirect block pointer points to a block of pointers, each of which points to a data block. The doubly indirect block pointer contains another level of indirection, and the triply indirect block pointer contains yet another level of indirection.



2 Storage Devices

What are the major components of disk latency? Explain each one.

```
Queuing time - How long it spends in the OS queue
Controller - How long it takes to send the message to the controller
Seek - How long the disk head has to move
Rotational - How long the disk rotates for
Transfer - The delay of copying the bytes into memory
```

In class we said that the operating system deals with bad or corrupted sectors. Some disk controllers magically hide failing sectors and re-map to “back-up” locations on disk when a sector fails.

If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?

Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

How do you think that the disk controller can check whether a sector has gone bad?

Using a checksum - this can be efficiently checked in hardware during disk access.

Can you think of any drawbacks of hiding errors like this from the operating system?

Excessive sector failures are warning signs that a disk is beginning to fail.

3 I/O Performance

This question will explore the performance consequences of using traditional disks for storage. Assume we have a hard drive with the following specifications:

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

   The time to read the sector can be broken down into three parts: seek delay, rotational delay, and data transfer delay. We are already given the expected seek delay: 8 ms.

   We can assume that, on average, the hard disk must complete 1/2 revolution before the sector we are interested in reading moves under the read/write head.

   Given that the disk makes 7200 revolutions per minute, the time to complete a revolution is 60 sec/7200 Revolution $\approx$ 8.33 ms per revolution.

   The time to complete 1/2 revolution, the expected rotational delay, is $\sim$ 4.17 ms.

   If the controller can transfer 50 MiB per second, it will take:

   $$4 \times 2^{10} \text{ bytes} \times \frac{1 \text{ sec.}}{50 \times 2^{20} \text{ bytes}} \approx 0.078125 \text{ ms}$$

   to transfer 4 KiB of data.

   In total, it takes $8 \text{ ms} + 4.17 \text{ ms} + 0.078125 \text{ ms} \approx 12.18 \text{ ms}$ to read the 4 KiB sector, yielding a throughput of 4 KiB/12.18 ms $\approx$ 328.5 KiB/s

2. What is the expected throughput of the hard drive when reading 4 KiB sectors from the same track on disk (i.e., the read/write head is already positioned over the correct track when the operation starts)?

   Now, we can ignore seek delay and only need to account for rotational delay and data transfer delay.

   We already know that the expected rotational delay is 4.17 ms and we know that the expected data transfer delay is 0.078125 ms.

   Therefore, it takes a total of $4.17 \text{ ms} + 0.078125 \text{ ms} \approx 4.18 \text{ ms}$ to read the 4 KiB sector, yielding a throughput of 4 KiB/4.18 ms $\approx$ 957 KiB/s.

3. What is the expected throughput of the hard drive when reading the very next 4 KiB sector (i.e. the read/write head is immediately over the proper track and sector at the start of the operation)?

   Now, we can ignore both rotational and seek delays. The throughput of the hard disk in this case is limited only by the controller, meaning we can take full advantage of its 50 MiB/s transfer rate.

Note that this is roughly a 156× improvement over the random read scenario!

4. What are some ways the Unix Fast File System (FFS) was designed to deal with the discrepancy in performance we just saw?

- Attempt to keep contents of a file contiguous on disk (first-fit block allocation)
- Break disk into a set of *block groups* — sets of adjacent tracks, each with its own free space bitmap, inodes, and data blocks
- Keep a file's header information (inode) in same block group as its data blocks
- Keep files in the same directory in the same block group

4 File Allocation Table

What does it mean to format a FAT file system? Approximately how many bytes of data need to be written in order to format a 2GiB flash drive (with 4KiB blocks and a FAT entry size of 4 bytes) using the FAT file system?

Formatting a FAT file system means resetting the file allocation table (mark all blocks as free). The actual data can be zero-ed out for additional security, but it is not required. Formatting a 2GiB FAT volume will require resetting 2^{19} FAT entries, which will involve approximately 2^{21} bytes (2 MiB).

Your friend (who has never taken an Operating Systems class) wants to format their external hard drive with the FAT32 file system. The external hard drive will be used to share home videos with your friend's family. Give one reason why FAT32 might be the right choice. Then, give one reason why your friend should consider other options.

FAT32 is supported by many different operating systems, which will make it a good choice for compatibility if it needs to be used by many users. However, FAT32 has a 4GiB file size limit, which may prevent your friend from sharing large video files with it.

Explain how an operating system reads a file like "`D:\My Files\Video.mp4`" from a FAT volume (from a software point of view).

First, the operating system must know that the FAT volume is mounted as "`D:\`". It looks at the first data block on the FAT volume, which contains the root directory, and searches for a subdirectory named "`My Files`". If necessary, the root directory listing might occupy many blocks, and the operating system will follow the pointers in the file allocation table to scan through the entire root directory. Once the subdirectory entry for "`My Files`" is found, the operating system searches the subdirectory's listing for a file named "`Video.mp4`". Once it knows the block number for the file, it can begin reading the file sequentially by following the pointers in the file allocation table.

Compare bitmap-based allocation of blocks on disk with a free block list.

Bitmap based block allocation is a fixed size proportional to the size of the disk. This means wasted space when the disk is full. A free block list shrinks as space is used up, so when the disk is full, the size of the free block list is tiny. However, contiguous allocation is easier to perform with a bitmap. Most modern file systems use a free block bitmap, not a free block list.

5 Inode-Based File System

1. What are the advantages of an inode-based file system design compared to FAT?

   Fast random access to files. Support for hard links.

2. What is the difference between a hard link and a soft link?

   Hard links point to the same inode, while soft links simply list a directory entry. Hard links use reference counting. Soft links do not and may have problems with dangling references if the referenced file is moved or deleted. Soft links can span file systems, while hard links are limited to the same file system.

3. Why do we have direct blocks? Why not just have indirect blocks?

   Faster for small files.

4. Consider a file system with 2048 byte blocks and 32-bit disk and file block pointers. Each file has 12 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer.

   (a) How large of a disk can this file system support?

   2^{32} blocks x 2^{11} bytes/block $= 2^{43} = 8$ Terabytes.

   (b) What is the maximum file size?

   There are 512 pointers per block (i.e. 512 4-byte pointers in 2048 byte block), so: blockSize $\times$ (numDirect + numIndirect + numDoublyIndirect + numTriplyIndirect)

$$\begin{aligned} 2048 \times (12 + 512 + 512^2 + 512^3) &= 2^{11} \times (2^2 \times 3 + 2^9 + 2^{9\times2} + 2^{9\times3}) \\ &= 2^{13} \times 3 + 2^{20} + 2^{29} + 2^{38} \\ &= 24K + 513M + 256G \end{aligned}$$

5. Rather than writing updated files to disk immediately, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every x seconds. List two advantages and one disadvantage of such a scheme.

   Advantage 1: The disk scheduling algorithm (i.e. SCAN) has more dirty blocks to work with at any one time and can thus do a better job of scheduling the disk arm.
   Advantage 2: Temporary files may be written and deleted before data is written to disk.
   Disadvantage: File data may be lost if the computer crashes before data is written to disk.

6. List the set of disk blocks that must be read into memory in order to read the file `/home/cs162/test.txt` in its entirety from a UNIX BSD 4.2 file system (10 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer). Assume the file is 15,234 bytes long and that disk blocks are 1024 bytes long. Assume that the directories in question all fit into a single disk block each. Note that this is not always true in reality.

1. Read in file header for root (always at fixed spot on disk).
2. Read in first data block for root (/).
3. Read in file header for home.
4. Read in data block for home.
5. Read in file header for cs162.
6. Read in data block for cs162.
7. Read in file header for test.txt.
8. Read in data block for test.txt.

9. - 17. Read in second through 10th data blocks for test.txt.

18. Read in indirect block pointed to by 11th entry in test.txt's file header.

19. - 23. Read in 11th – 15th test.txt data blocks. The 15th data block is partially full.