Filesystem Optimization and Examples

Module 17

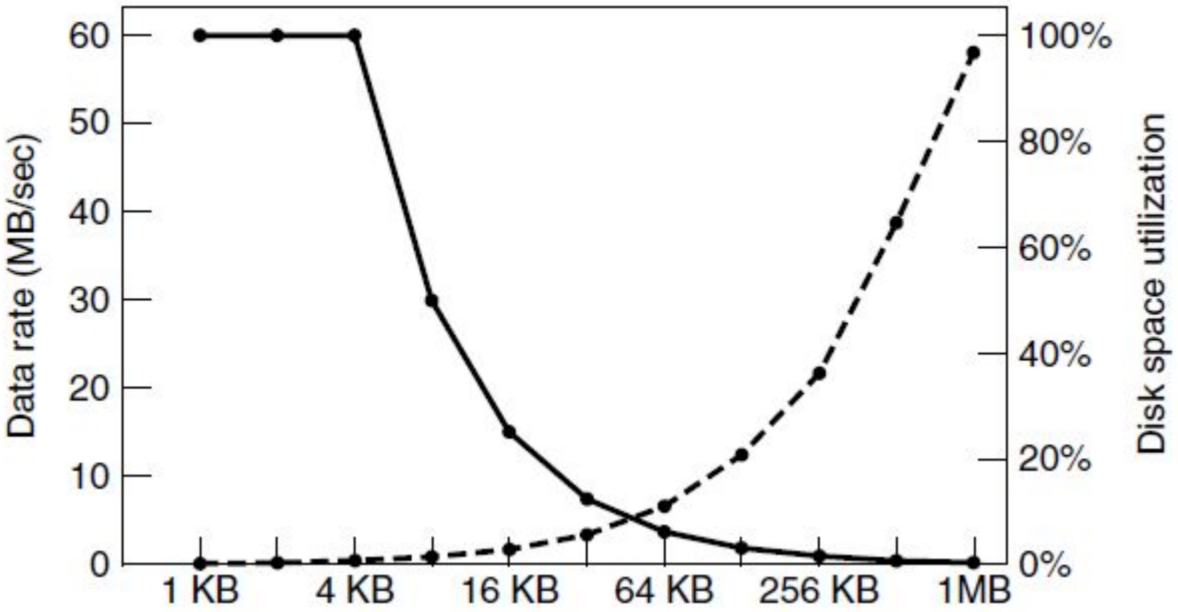
Space Management

- We know files need to be broken into Blocks
- This prevents external fragmentation
  - … but it creates ***internal fragmentation*.**
- How big should blocks be?
  - We could use device values: sector, track, cylinder
  - We could use page size

Block size considerations

- Big block sizes are a problem for small files
  - Lots of small files = lots of wasted space
- Small blocks (scattered) require lots of seek time
  - wasted time

Block size: Space or Transfer Speed?



Solid line = Space utilization

Dashed line = Data Rate

Takeaways

Based on what we know (empirically) about typical file sizes (lots of small files) 1-KB to 4KB has been the standard.

Drives are becoming so big (and cheap) that this might change though.

File System Recovery / Backup

- Hard-drives are supposed to be permanent
  - Nothing in life is permanent
  - When they break, you could lose all your data!
- Actually - there are two reasons for backup:
  - Recover from disaster (physical)
  - Recover from stupidity (far more common)

The problem with backups is they are slow, and expensive...

Backup issues

- Do we want to back up files that can easily be recreated by other means?
  - Binary programs?
  - We should be able to choose
- What if files haven't changed from the last backup?
  - Incremental backups are far superior
  - Of course… then restoration is a bit more complex

Backup Issues

- Even incremental backups can be huge
  - Compression is often employed here
- When to take a backup?
  - Difficult to do this on a system where files are actively changing
- Then of course… don’t lose your backup!
  - Backups need to be stored off-site

A series of small failures....

- When an OS crashes, it could have been busy reading/writing **blocks**.
  - What if a block is removed from a file, and about to be put in the free pool… but crashed!
  - What if a block was about to be allocated - it was removed from free pool.. but then crashed!
- We can scan i-nodes and the free pool - every block should be accounted for **once**.
  - If not - we have **missing blocks** - or a block that needs to be removed from the free list.

Inconsistent states

(a)

Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(b)

Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(c)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(d)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

This one is tricky - we clone the block.

We still have an error, but at least blocks are consistent...

Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

Bad Blocks

- It’s nearly impossible to manufacture disks without some blocks containing flaws
  - Over time, more flaws occur
  - Flaw = a bit that cannot reliably hold it’s charge (state)
- Two problems:
  - How do figure out it’s happened?
  - What to do we do?

Finding a Bad Block

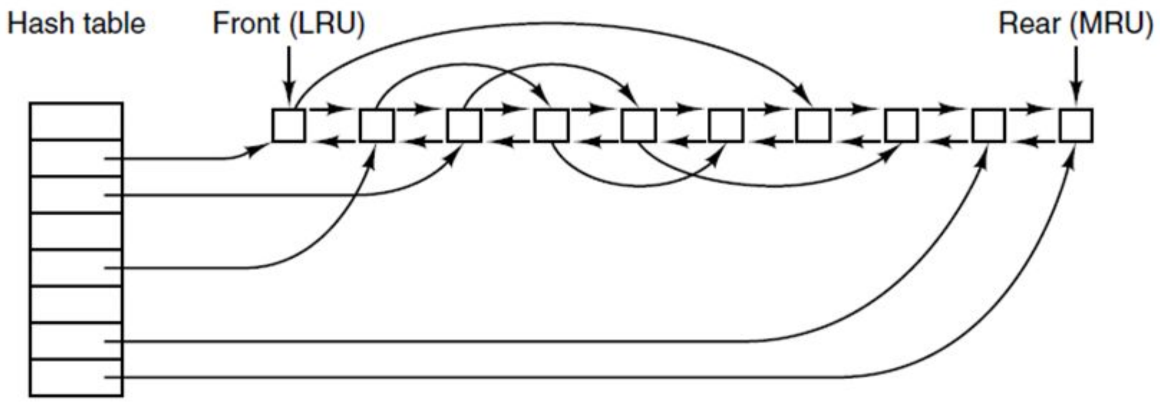
- Each block utilizes a header (overhead)
  - The header contains an ECC
  - On write, generate an ECC of data and write it in the header area of the block.
- On read, look at the data and generate **expected** ECC.
  - Compare with ECC in header
  - If they don’t match, one is bad - you’ve got a bad block.

Bad Block Recovery

- We can’t “fix” the block.
  - We must report the file as corrupted
- In addition, we must remember not to assign this block again
  - Most disk controllers handle much of this - hiding it from the operating system
  - Replacement blocks are held in reserve
  - Block numbers corresponding to bad blocks are reassigned at the controller level

Caching and Performance

- If we have memory to spare, we can hold copies of disk data in it.
- Block cache allows for read/writes to blocks to be done in memory.



Danger of Caches

- What if we cache an i-node… and we crash!
- What if some blocks are used consistently, but not frequently?

We modify LRU:

1. Is the block likely to be used again soon?
2. Is the block essential to consistency?

Blocks aren’t equal

Divide blocks into:

- i-nodes (rarely used again and again)
- directory structures (same thing here)
- full data blocks
- partially full blocks

Blocks aren't equal

Divide blocks into:

- i-nodes (must be written immediately)
- directory structures (same thing here)
- full data blocks
- partially full blocks

Forcing Caches to empty

- System calls like sync (POSIX) can force the cache to be flushed to disk.
  - The OS can do this periodically (every 20 seconds)
  - This is typical for UNIX
- In Windows, "write-through" caches have traditionally been used instead.

Block Read-ahead

- Another approach used is to “guess” that if block `i` is being read, block `i+1` will be read next.
- This works because it is so often the case that we sequentially read files

File System Examples

MS-DOS

- Supported, but no longer common in modern Windows
- Surprisingly common in embedded systems
  - It’s actually the FS the iPod uses (not iPhone)
- Each partition assigned a “drive” letter
  - The drive letter is essentially the “root”
  - No common “root” (“Computer”)

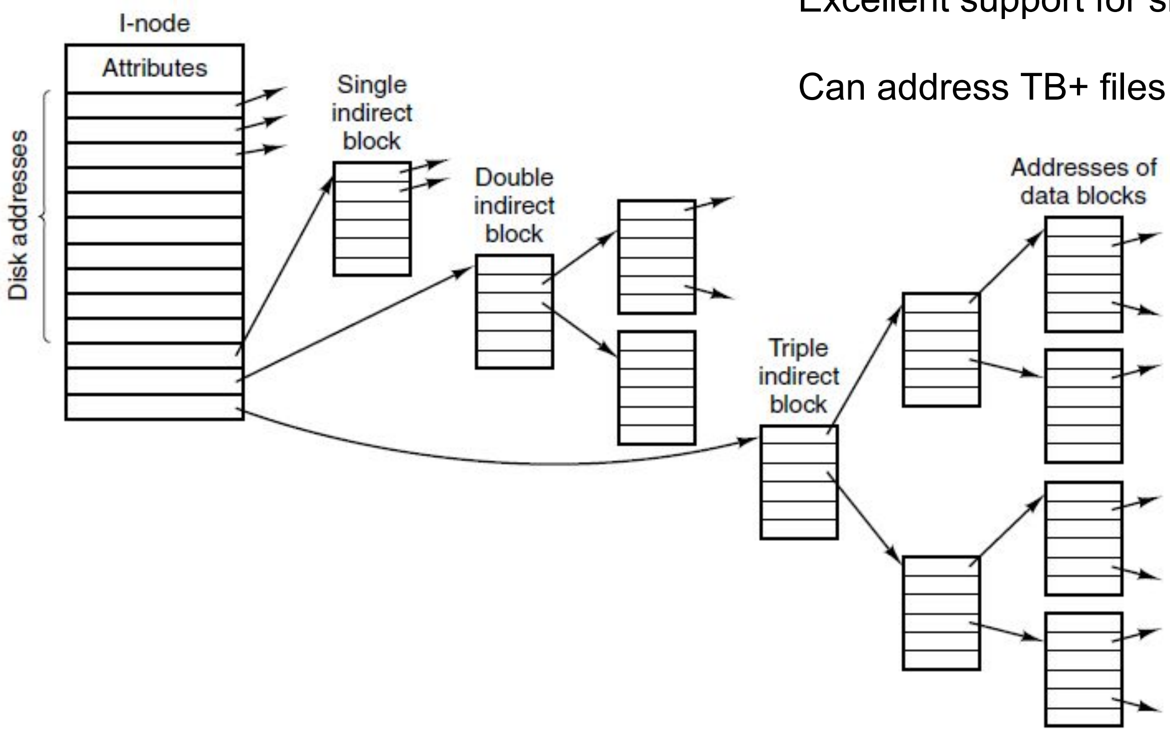
MS-DOS

- Characterized by:
  - Fixed directory entry sizes (fixed file name length)
  - Directory entries contain attributes directly (no linking)
  - File size is a 32-bit number - so no file can be larger than 4GB.
- Uses a FAT *(later MS-DOS renamed to FAT-32)*
  - exFAT supports larger files, licensed by Apple

UNIX

- A single root directory (/) - where all other file-systems can be mounted
- Uses i-node
    - Supports linking (attributes in i-node, not directory entry)

i-node structure in UNIX



Excellent support for small files

Can address TB+ files

Up Next...

Chapter 5 take a look at how the OS interacts with physical devices

There is some overlap with our disk discussion!