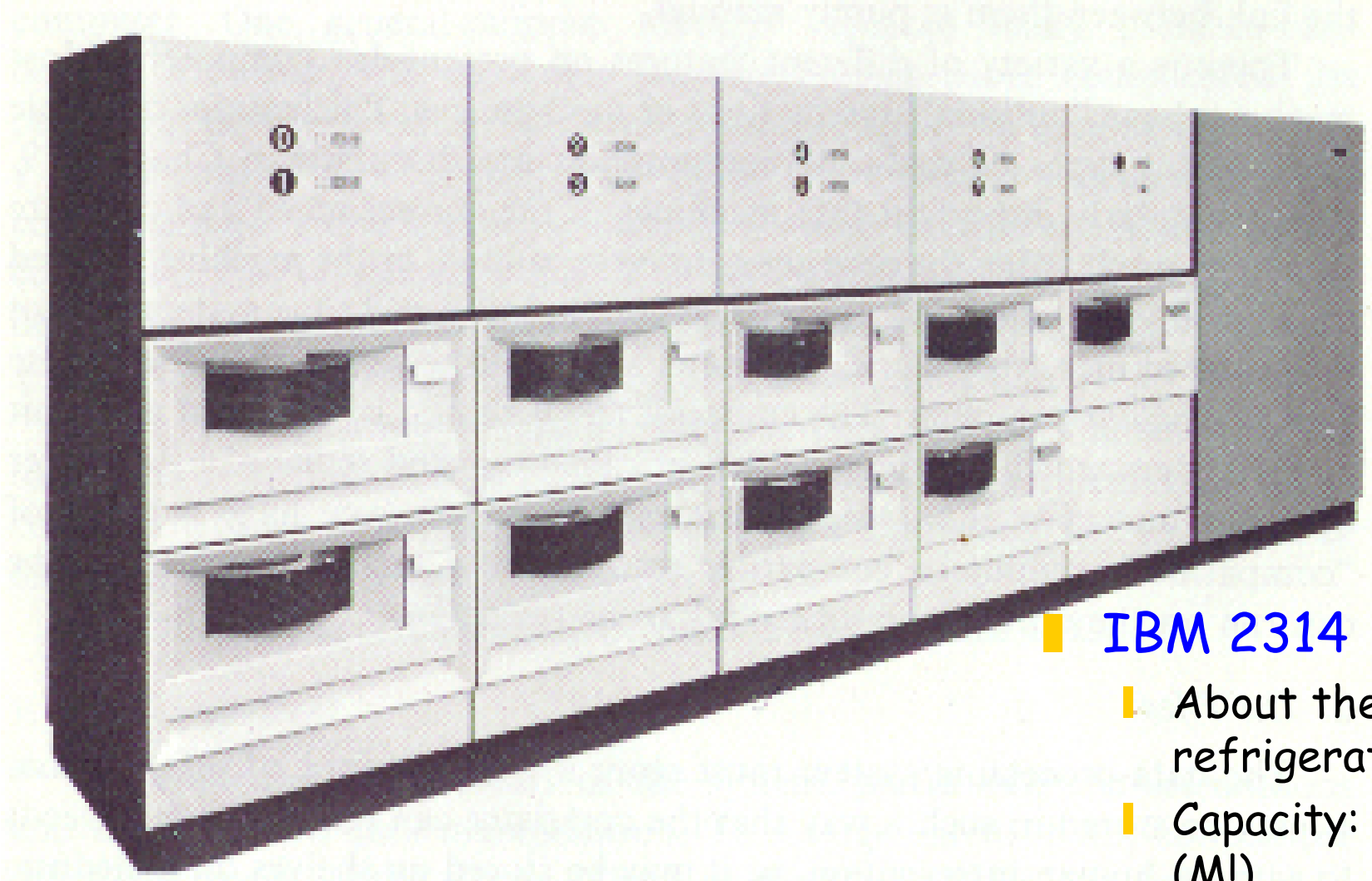
The Google File System

CSE 490h, Autumn 2008

Today …

- Disks
- File systems
- GFS

A trip down memory lane …



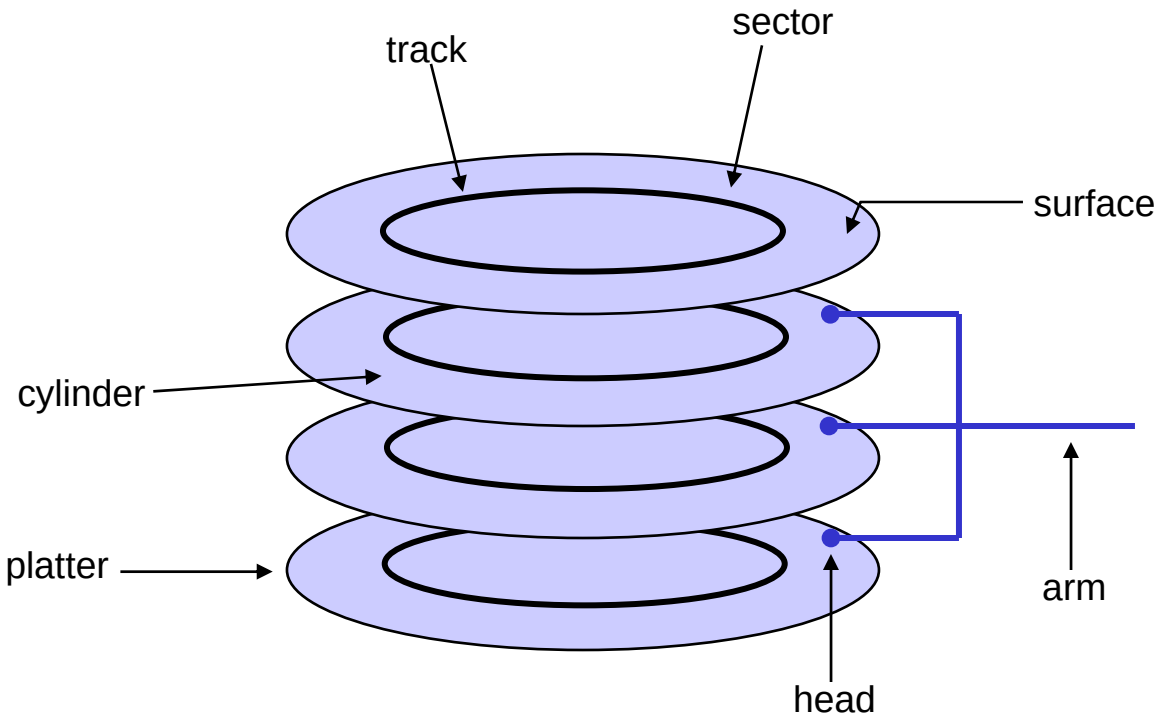
- IBM 2314
  - About the size of 6 refrigerators
  - Capacity: 8 x 29MB (M!)

Today …

- Seagate Barracuda 7200.11
  - Form factor: 3.5″
  - Capacity: 1500 GB
    - 6500 times the capacity of those six refrigerators!!!!!

Physical disk structure

- Disk components
  - platters
  - surfaces
  - tracks
  - sectors
  - cylinders
  - arm
  - heads



Disk performance

- Performance depends on a number of steps
  - **seek**: moving the disk arm to the correct cylinder
    - depends on how fast disk arm can move
      - seek times aren't diminishing very quickly (why?)
  - **rotation (latency)**: waiting for the sector to rotate under head
    - depends on rotation rate of disk
      - rates are increasing, but slowly (why?)
  - **transfer**: transferring data from surface into disk controller, and from there sending it back to host
    - depends on density of bytes on disk
      - increasing, relatively quickly
- When the OS uses the disk, it tries to minimize the cost of all of these steps
  - How?

OS / file system accommodations to disk performance

- Seek reduction
  - Increase block size to reduce seeking
  - Co-locate "related" items in order to reduce seeking
    - blocks of the same file
    - data and metadata for a file
  - Scheduling of requests
    - FCFS, SSTF, SCAN, C-SCAN
  - Log-structured file systems

- Caching and pre-fetching
  - Keep data or metadata in memory to reduce physical disk access
    - But what if a crash occurs??
  - If file access is sequential, fetch blocks into memory before requested
- Faster re-boot after a crash
  - Journaling file systems

Example disk characteristics

- Seagate Barracuda 7200.11
  - form factor: 3.5"
  - capacity: 1500 GB
  - rotation rate: 7,200 RPM (120 RPS)
  - platters: 4
  - heads: 8
  - average sector size: 512 bytes
  - cylinders: 16,383
  - cache: 32 MB
  - sustained transfer rate: 135 MB/s
  - average seek: 10 ms (how many bytes worth??)
  - adjacent track seek: 1 ms
  - average latency: 4 ms

File systems: Basic operations

Unix

- create(name)
- open(name, mode)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)
- rename(old, new)

NT

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- CopyFile(name)
- MoveFile(name)

The original Unix file system

- Dennis Ritchie and Ken Thompson, Bell Labs, 1969
- "UNIX rose from the ashes of a multi-organizational effort in the early 1960s to develop a dependable timesharing operating system" – Multics
- Designed for a "workgroup" sharing a single system
- Did its job exceedingly well
  - Although it has been stretched in many directions and made ugly in the process
- A wonderful study in engineering tradeoffs

All disks are divided into five parts …

- Boot block
  - can boot the system by loading from this block
- Superblock
  - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
  - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of freelist is in the superblock
- File contents area
  - fixed-size blocks; head of freelist is in the superblock
- Swap area
  - holds processes that have been swapped out of memory

So …

- You can attach a disk to a dead system …
- Boot it up …
- Find, create, and modify files …
  - because the superblock is at a fixed place, and it tells you where the i-node area and file contents area are
  - by convention, the second i-node is the root directory of the volume

i-node format

- User number, Group number, Protection bits
- Times (file last read, file last written, inode last written)
- File code: specifies if the i-node represents a directory, an ordinary user file, or a "special file" (typically an I/O device)
- Size: length of file in bytes
- Block list: locates contents of file (in the file contents area)
  - more on this soon!
- Link count: number of directories referencing this i-node

The flat (i-node) file system

- Each file is known by a number, which is the number of the i-node
  - seriously – 1, 2, 3, etc.!
  - why is it called "flat"?
- Files are created empty, and grow when extended through writes

The tree (directory, hierarchical) file system

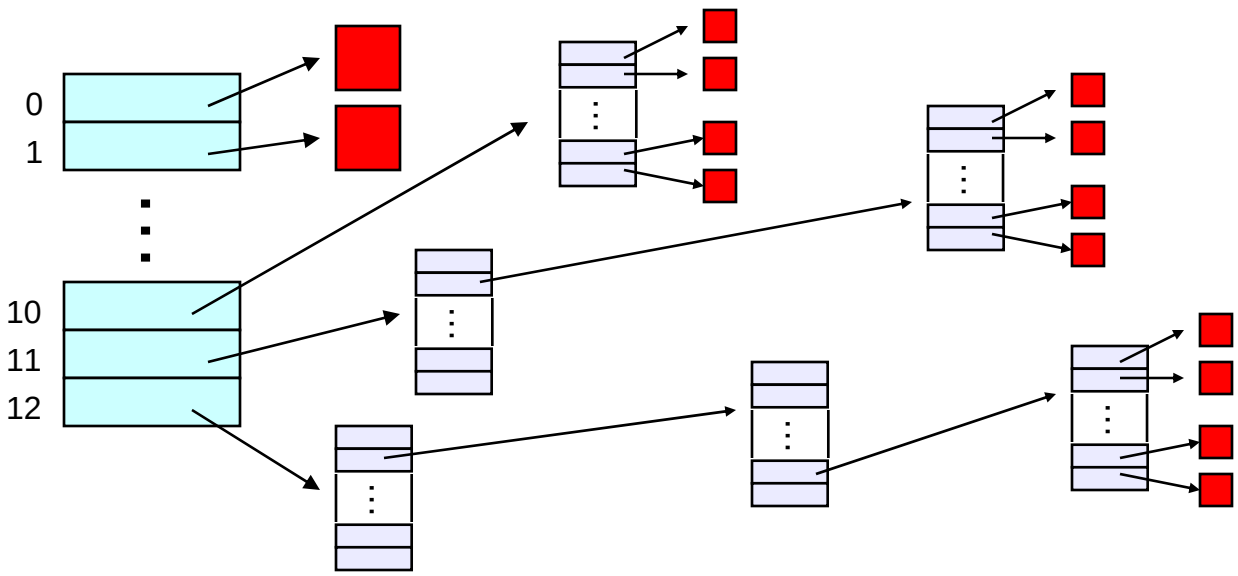
- A directory is a flat file of fixed-size entries
- Each entry consists of an i-node number and a file name

i-node number	File name
152	.
18	..
216	my_file
4	another_file
93	oh_my_god
144	a_directory

The "block list" portion of the i-node (Unix Version 7)

- Points to blocks in the file contents area
- Must be able to represent very small and very large files

- Each inode contains 13 block pointers
  - first 10 are "direct pointers" (pointers to 512B blocks of file data)
  - then, single, double, and triple indirect pointers



So …

- Only occupies 13 x 4B in the i-node
- Can get to 10 x 512B = a 5120B file directly
  - (10 direct pointers, blocks in the file contents area are 512B)
- Can get to 128 x 512B = an additional 65KB with a single indirect reference
  - (the 11th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to blocks holding file data)
- Can get to 128 x 128 x 512B = an additional 8MB with a double indirect reference
  - (the 12th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks holding file data)

- Can get to 128 x 128 x 128 x 512B = an additional 1GB with a triple indirect reference
  - (the 13th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks in the file contents area that contain 128 4B pointers to 512B blocks holding file data)
- Maximum file size is 1GB + a smidge

- A later version of Bell Labs Unix utilized 12 direct pointers rather than 10
  - Why?
- Berkeley Unix went to 1KB block sizes
  - What's the effect on the maximum file size?
    - 256x256x256x1K = 17 GB + a smidge
  - What's the price?
- Suppose you went to 4KB blocks?
  - 1Kx1Kx1Kx4K = 4TB + a smidge
  - Impact on performance?
  - Impact on disk utilization?

Quick comment on crash recovery

- iCheck and dCheck are hugely expensive
  - Worse as disks get bigger
- Journaling file systems solve this
  - Keep a change log; avoid scanning entire disk
- Will discuss journaling and logs in the "transactions" module

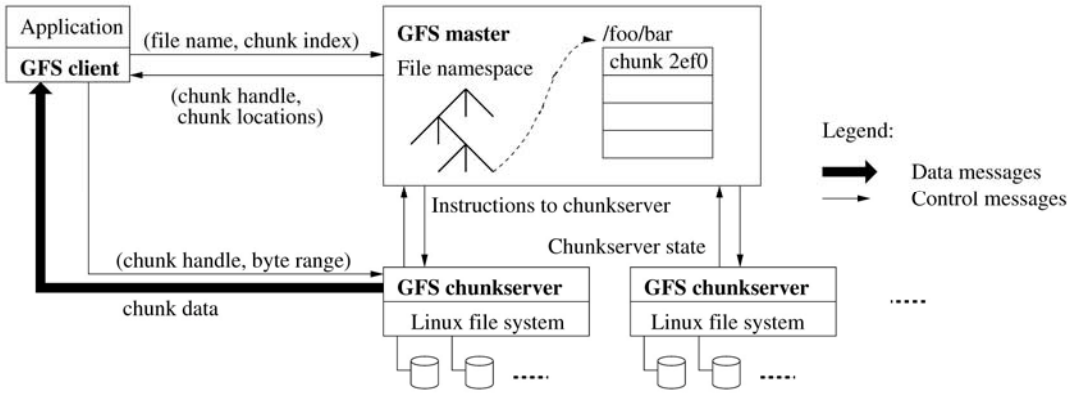
GFS: Environment

- Thousands of computers
- Distributed
  - Computers have their own disks, and the file system spans those disks
- Failures are the norm
  - Disks, networks, processors, power supplies, application software, operating system software, human error
- Files are huge
  - Multi-gigabyte files, each containing many objects
- Read/write characteristics
  - Files are mutated by appending
  - Once written, files are typically only read
  - Large streaming reads and small random reads are typical

- Bandwidth is more important than latency
- Its helpful if the file system provides synchronization for concurrent appends

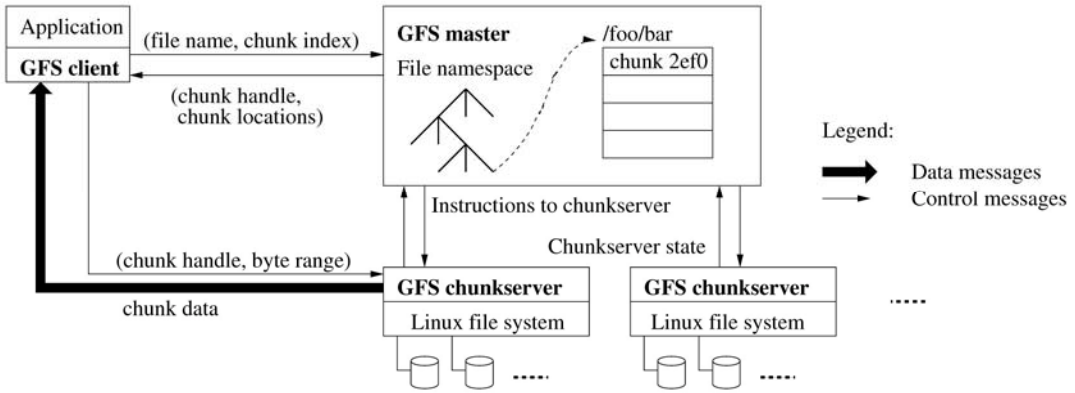
General architecture

- A GFS cluster has one *master* and many *chunkservers*
- Files are divided into 64 MB *chunks*
- Chunks are replicated and stored in the Unix file systems of the chunkservers
- The master holds metadata
- Clients get metadata from the master, and data directly from chunkservers



File read

- From byte offset within the file, client computes chunk index
- Client sends filename and chunk index to master
- Master returns a list of replicas of the chunk
- Client interacts with a replica to access data

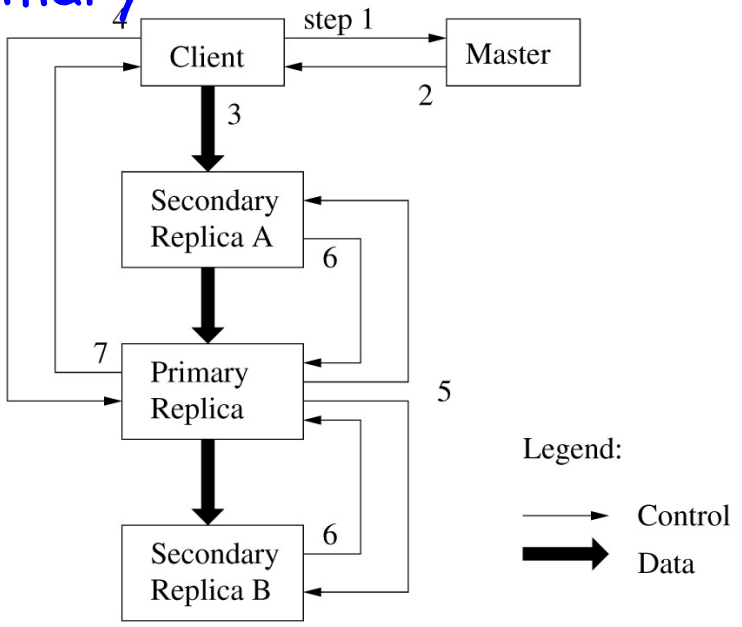


Metadata

- Three types of metadata
  - File and chunk namespaces
  - Mapping from files to chunks (each chunk has a unique ID)
  - Locations of each chunk's replicas
- All metadata kept in memory
- First two types are made persistent via a change log
  - Punt discussion to a later module
- Chunk replica locations learned by polling chunkservers at startup
- Chunkserver is final arbiter of what chunks it holds

File write

- Client asks master for identity of primary and secondary replicas
- Client pushes data to memory at all replicas via a replica-to-replica "chain"
- Client sends write request to primary
- Primary orders concurrent requests, and triggers disk writes at all replicas
- Primary reports success or failure to client



Replica failure

- Master detects a failed "heartbeat" of a chunkserver
- Re-creates contents elsewhere
- Write eventually succeeds