File Systems (III)

(Chapters 39-43,45)

CS 4410

Operating Systems

Cornell CIS

COMPUTING AND INFORMATION SCIENCE

[R. Agarwal, L. Alvisi, A. Bracy, M. George, F.B. Schneider, E. Sirer, R. Van Renesse]

File Storage Layout Options

- ✓ Contiguous allocation

  All bytes together, in order

- ✓ Linked-list

  Each block points to the next block

- ✓ Indexed structure (FFS)

  Index block points to many other blocks

- Log structure

  Sequence of segments, each containing updated blocks

- File systems for distributed systems

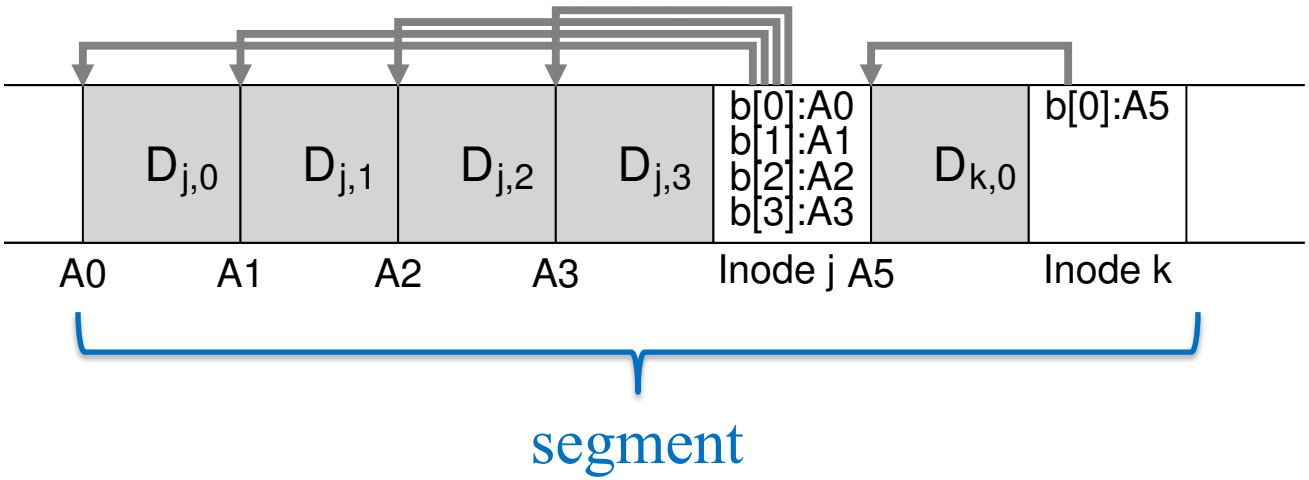
Log-Structured File Systems

Technological drivers:

- System memories are getting larger
  - Larger disk cache
  - Reads mostly serviced by cache
  - Traffic to disk mostly writes.
- Sequential disk access performs better.
  - Avoid seeks for even better performance.

Idea: Buffer sets of writes and store as single log entry (“segment”) on disk. File system implemented as a log!

Storing Data on Disk



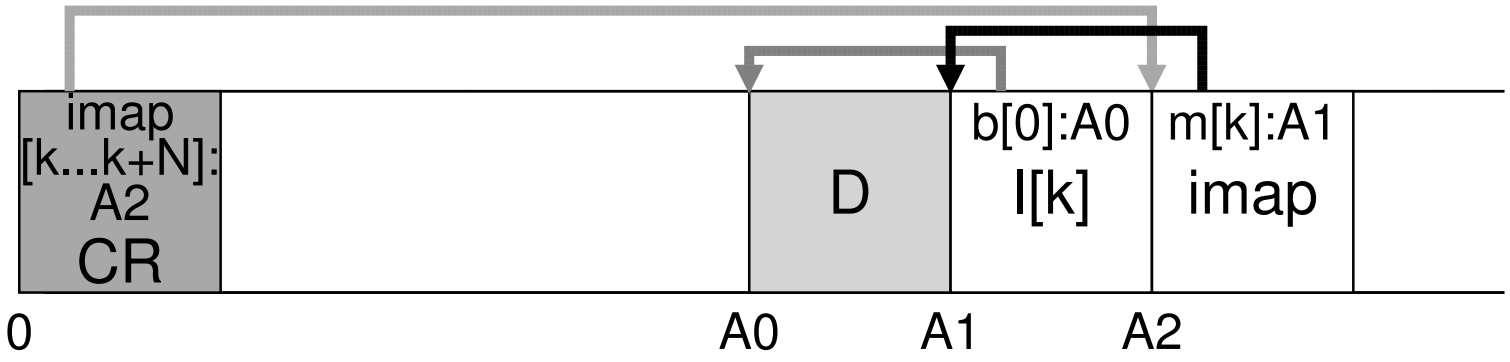
- Updates to file j and k are buffered.
- Inode for a file points to log entry for data
- An entire segment is written at once.

How to Find Inode on Disk

In FFS: F: inode nbr → location on disk
In LFS: location of inode on disk changes…

LFS: Maintain **inode Map** (imap) in pieces and store updated pieces on disk. imap: inode number → disk addr

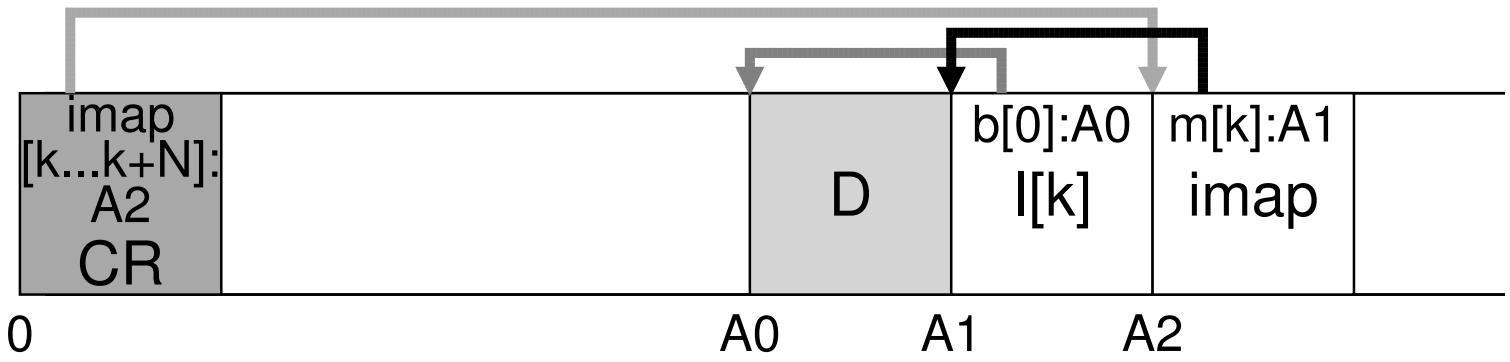
- For write performance: Put piece(s) at end of segment
- **Checkpoint Region (CR)**: Points to <u>all</u> inode map pieces and is updated every 30 secs. *Located at fixed disk address*. Also buffered in memory.



To Read a File in LFS

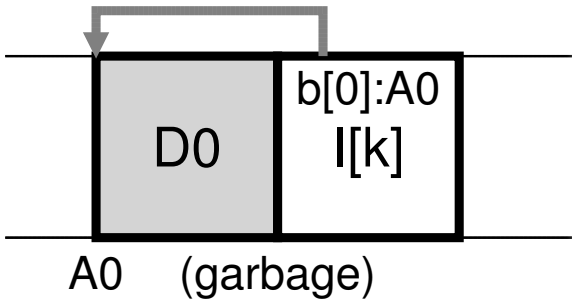
- [Load checkpoint region CR into memory]
- [Copy inode map into memory]
- Read appropriate inode from disk if needed
- Read appropriate file (dir or data) block

[…] = step not needed if information already cached.



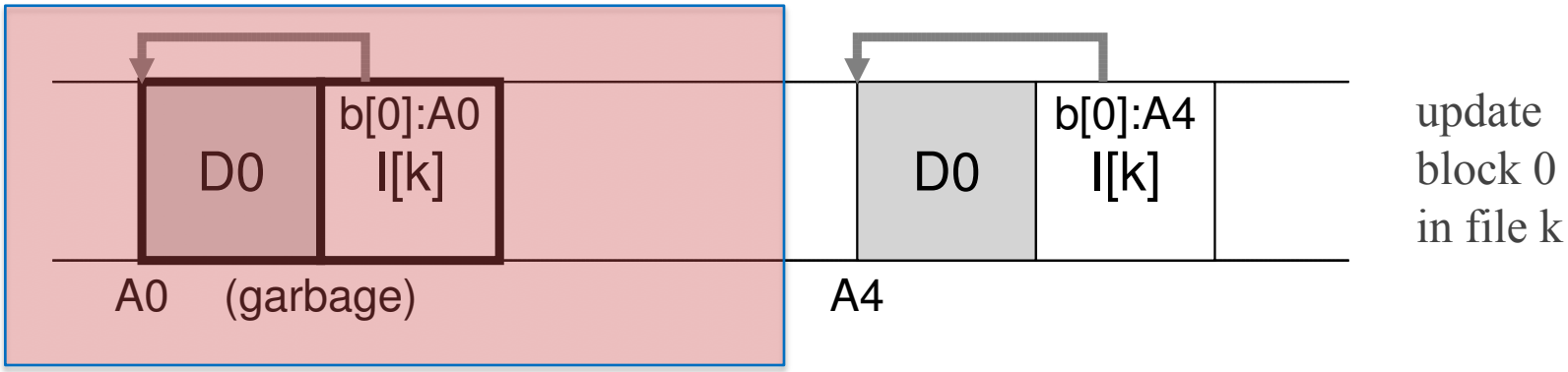
Garbage Collection

Eventually disk will fill. But many blocks (“garbage”) not reachable via CP, because they were overwritten.



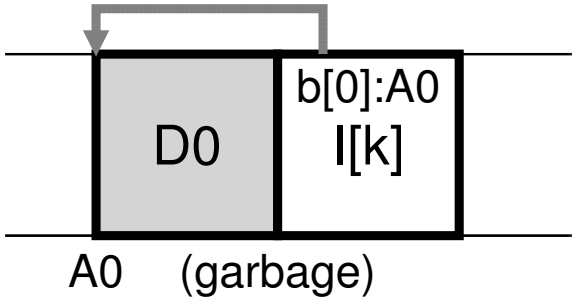
Garbage Collection

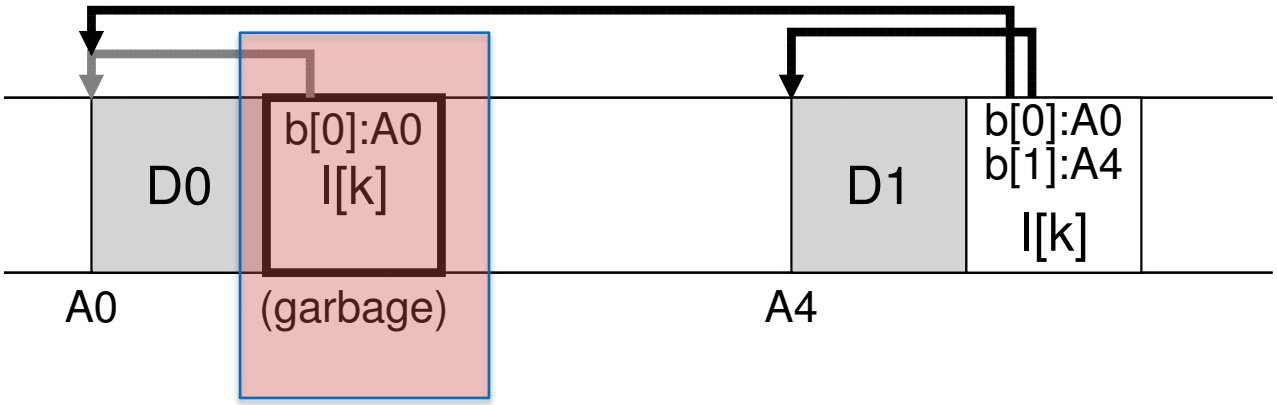
Eventually disk will fill. But many blocks (“garbage”) not reachable via CP, because they were overwritten.



Garbage Collection

Eventually disk will fill. But many blocks (“garbage”) not reachable via CP, because they were overwritten.





append block to file k

LFS Cleaner

Protocol:

1. read entire segment;
2. find live blocks within (see below);
3. copy live blocks to new segment;
4. append new segment to disk log

Finding live blocks: Include at strt of each LFS segment a **segment summary block** that gives for each data block D in that LFS segment:

- inode number *in*
- offset in the file *of*

Read block for < *in*, *of* > from LFS to reveal if D is live (=) or it is garbage (=!).

Crash Recovery (sketch)

LFS writes to disk: CR and segment.

After a crash:

- Find most recent consistent CR (see below)
- Roll forward by reading next segment for updates.

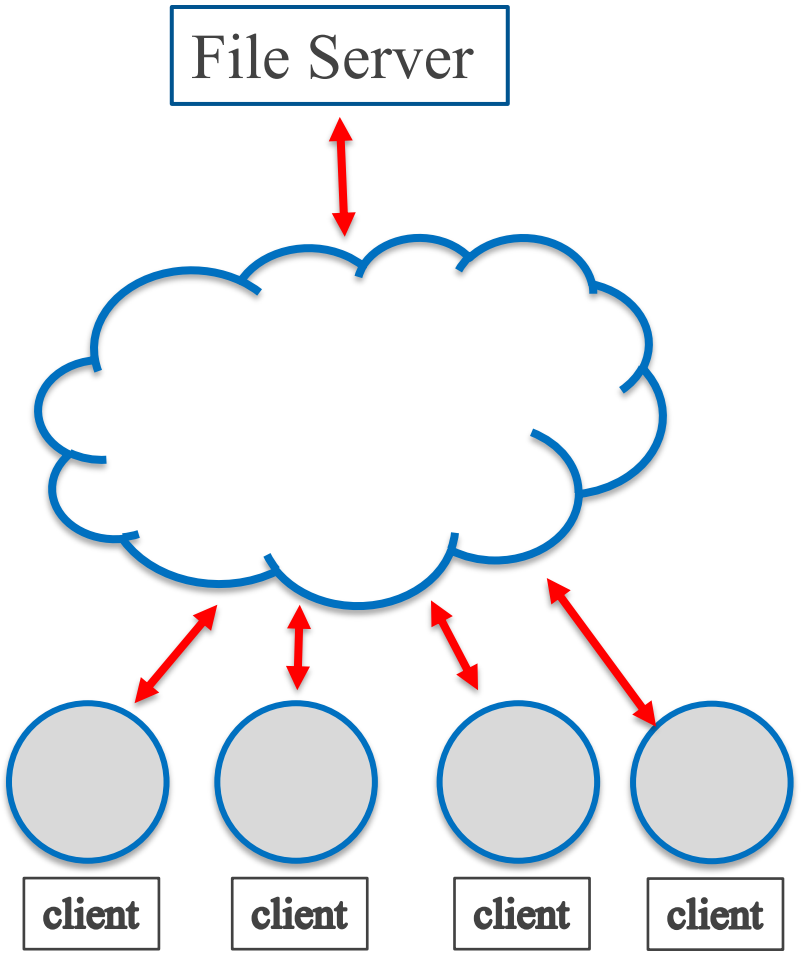
Crash-resistant atomic CR update:

- Two copies of CR: at start and end of disk.
- Updates alternate between them.
- Each CR has timestamp ts(CR,start) at start and ts(CR,end) at end.
  - CR consistent if ts(CR,start)=ts(CR,end)
- Use consistent CR with largest timestamp

Distributed File System

Challenges

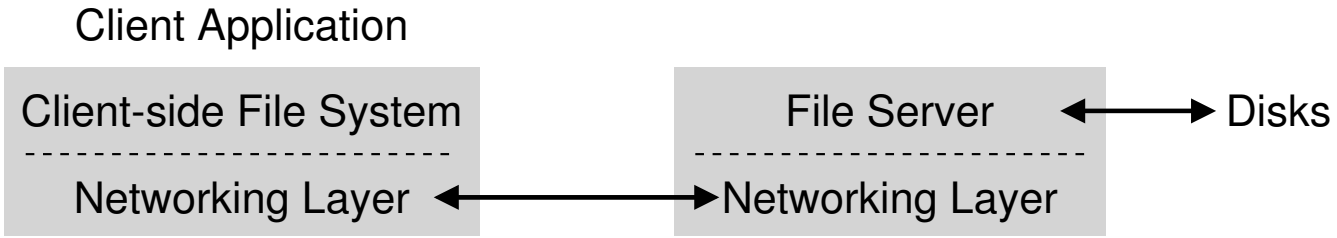
- Client Failure
- Server Failure



NFSv2 (Sun Microsystems)

Goals:

- Clients share files
- Centralized file storage
  - Allows efficient backup
  - Allows uniform management
  - Enables physical security for files
- Client side transparency
  - Same operations file sys operations:
    - open, read, write, close, …



A stateless protocol

- Server does not maintain any state about clients accessing files.
  - Eliminates possible inconsistency between state at server and state at client.
  - Requires client to maintain and send state information to server with each client operation.
- Client uses **file handle** to identify a file to the server. Components of a file handle are:
  - Volume identifier
  - Inode number
  - Generation number (allows inode number reuse)

NFS Server Operations

- Lookup: name of file → file handle
- Read: file handle, offset, count → data
- Write: file handle, offset, count, data
- …

Initially, client obtains file handle for root directory from NFS server.

NFS Client Operations

File system operations at client are translated to message exchange with server.

- fd := open(“/foo”, …) →

  send LOOKUP(roodir FH, “foo”) **to** NFS server
  receive FH_for_foo **from** NFS server
  openFileTable[i] := FH_for_foo {slot i presumed free}
  return i

- read(fd, buffer, start, MAX)

  FH := openFileTable[fd].fileHandle
  send READ(FH, offset=start, count=MAX) **to** NFS server
  receive data **from** NSF server
  buffer := data;

Etc…

Tolerating NFS Server Failures

- Asmpt: Server that fails is eventually rebooted.
- Manifestations of failures:
  - *Failed server*: no reply to client requests.
  - *Lost client request*: no reply to client request.
  - *Lost reply*: no reply to client request.

Solution: Client does retry (after timeout). And all NSF server operations are idempotent.

  - Idempotent = “Repeat of an operation generates same resp.”
    - LOOKUP, READ, WRITE
    - MKDIR (create a directory that’s already present? Return FH anyway.)
    - DELETE <resp> CREATE (failure before <resp>)
      - » Requires having a generation number in object.

Client-Side Caching of Blocks

- read ahead + write buffering improve performance by eliminating message delays.

- Client-side buffering causes problems if multiple clients access the same file concurrently.
  - **Update visibility**: Writes by client C not seen by server, so not seen by other clients C'.
    - *Solution*: flush-on-close semantics for files.
  - **Stale Cache**: Writes by client C are seen by server, but other cache at other clients stale. (Server does not know where the file is cached.)
    - *Solution*: Periodically check last-update time at server to see if cache could be invalid.

AFS: Andrew File System (CMU)

Goal:

- Support large numbers of clients

Design: AFS Version 1

- Whole file caching on local disk
  - » NFS caches blocks, not files
  - – open() copies file to local disk
    - » … unless file is already there from last access
  - – close() copies updates back
  - – read/write access copy on local disk
    - Blocks might be cached in local memory

Problems with AFS Version 1

- Full path names sent to remote file server
  - Remote file server spends too much time traversing the directory tree.
- Too much traffic between client and file server devoted to testing if local file copy is current.

Design: AFS Version 2

- callbacks added:
  - Client registers with server;
  - Server promises to inform client that a cached file has been modified.

- file identifier (FID) replaces pathnames:
  - Client caches various directories in pathname
    - Register for callbacks on each directory
    - Directory maps to FID
  - Client traverses local directories, using FID to fetch actual files if not cached.

AFS Cache Consistency

Consistency between:

- Processes on different machines:
  - Updates for file made visible at server when file closed()
    - » Last writer wins if multiple clients have file open and are updating it. (So file reflects updates by only machine.)
    - » Compare with NFS: updates blocks from different clients.
  - All clients with callbacks for that file are notified and callback cancelled.
  - Subsequent open() re-fetches the file
- Processes on the same machine
  - Updates are visible locally through shared cache.

AFS Crash Recovery

Client crash/reboot/disconnect:

- Client might miss the callback from server
- On client reboot: treat all local files as suspect and recheck with server for each file open

Server failure:

- Server forgets list of callbacks registered.
- On server reboot: Inform all clients; client must treat all local files as suspect.
  - » Impl options: client polling vs server push

File Storage Layout Options

- ✓Contiguous allocation
  - All bytes together, in order
- ✓ Linked-list
  - Each block points to the next block
- ✓ Indexed structure (FFS)
  - Index block points to many other blocks
- ✓Log structure
  - Sequence of segments, each containing updated blocks
- ✓File systems for distributed systems

File Systems: Final Comments

I/O systems are accessed through a series of layered abstractions