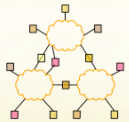
15-446 Distributed Systems Spring 2009



L-18 More DFS

Review of Last Lecture

- Distributed file systems functionality
- Implementation mechanisms example
  - Client side: VFS interception in kernel
  - Communications: RPC
  - Server side: service daemons
- Design choices
  - Topic 1: name space construction
    - Mount (NFS) vs. global name space (AFS)
  - Topic 2: AAA in distributed file systems
    - Kerberos
  - Topic 3: client-side caching
    - NFS and AFS

2

Today's Lecture

- DFS design comparisons continued
  - Topic 4: file access consistency
    - NFS, AFS, Sprite, and DCE DFS
  - Topic 5: Locking
- Other types of DFS
  - Coda – disconnected operation
  - LBFS – weakly connected operation

3

Topic 4: File Access Consistency

- In UNIX local file system, concurrent file reads and writes have "sequential" consistency semantics
  - Each file read/write from user-level app is an atomic operation
    - The kernel locks the file vnode
  - Each file write is immediately visible to all file readers
- Neither NFS nor AFS provides such concurrency control
  - NFS: "sometime within 30 seconds"
  - AFS: session semantics for consistency

4

Semantics of File Sharing

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transactions	All changes occur atomically

- Four ways of dealing with the shared files in a distributed system.

5

Session Semantics in AFS v2

- What it means:
  - A file write is visible to processes on the same box immediately, but not visible to processes on other machines until the file is closed
  - When a file is closed, changes are visible to new opens, but are not visible to "old" opens
  - All other file operations are visible everywhere immediately
- Implementation
  - Dirty data are buffered at the client machine until file close, then flushed back to server, which leads the server to send "break callback" to other clients

6

AFS Write Policy

- Data transfer is by chunks
  - Minimally 64 KB
  - May be whole-file
- Writeback cache
  - Opposite of NFS "every write is sacred"
  - Store chunk back to server
    - When cache overflows
    - On last user close()
  - ...or don't (if client machine crashes)
- Is writeback crazy?
  - Write conflicts "assumed rare"
  - Who wants to see a half-written file?

7

Access Consistency in the "Sprite" File System

- Sprite: a research file system developed in UC Berkeley in late 80's
- Implements "sequential" consistency
  - Caches only file data, not file metadata
  - When server detects a file is open on multiple machines but is written by some client, client caching of the file is disabled; all reads and writes go through the server
  - "Write-back" policy otherwise
    - Why?

8

Implementing Sequential Consistency

- How to identify out-of-date data blocks
  - Use file version number
  - No invalidation
  - No issue with network partition
- How to get the latest data when read-write sharing occurs
  - Server keeps track of last writer

9

Implication of "Sprite" Caching

- Server must keep states!
  - Recovery from power failure
  - Server failure doesn't impact consistency
  - Network failure doesn't impact consistency
- Price of sequential consistency: no client caching of file metadata; all file opens go through server
  - Performance impact
  - Suited for wide-area network?

10

"Tokens" in DCE DFS

- How does one implement sequential consistency in a file system that spans multiple sites over WAN
- Callbacks are evolved into 4 kinds of "Tokens"
  - Open tokens: allow holder to open a file; submodes: read, write, execute, exclusive-write
  - Data tokens: apply to a range of bytes
    - "read" token: cached data are valid
    - "write" token: can write to data and keep dirty data at client
  - Status tokens: provide guarantee of file attributes
    - "read" status token: cached attribute is valid
    - "write" status token: can change the attribute and keep the change at the client
  - Lock tokens: allow holder to lock byte ranges in the file

11

Compatibility Rules for Tokens

- Open tokens:
  - Open for exclusive writes are incompatible with any other open, and "open for execute" are incompatible with "open for write"
  - But "open for write" can be compatible with "open for write" --- why?
- Data tokens: R/W and W/W are incompatible if the byte range overlaps
- Status tokens: R/W and W/W are incompatible
- Data token and status token: compatible or incompatible?

12

Token Manager

- Resolve conflicts: block the new requester and send notification to other clients' tokens
- Handle operations that request multiple tokens
  - Example: rename
  - How to avoid deadlocks

13

Topic 5: File Locking for Concurrency Control

- Issues
  - Whole file locking or byte-range locking
  - Mandatory or advisory
    - UNIX: advisory
    - Windows: if a lock is granted, it's mandatory on all other accesses
- NFS: network lock manager (NLM)
  - NLM is not part of NFS v2, because NLM is stateful
  - Provides both whole file and byte-range locking
  - Advisory
  - Relies on "network status monitor" for server monitoring

14

Issues in Locking Implementations

- Failure recovery
  - What if server fails?
    - Lock holders are expected to re-establish the locks during the "grace period", during which no other locks are granted
  - What if a client holding the lock fails?
  - What if network partition occurs?

15

Wrap up: Design Issues

- Name space
- Authentication
- Caching
- Consistency
- Locking

16

AFS Retrospective

- Small AFS installations are hard
  - Step 1: Install Kerberos
    - 2-3 servers
    - Inside locked boxes!
  - Step 2: Install ~4 AFS servers (2 data, 2 pt/vldb)
  - Step 3: Explain Kerberos to your users
    - Ticket expiration!
  - Step 4: Explain ACLs to your users

17

AFS Retrospective

- Worldwide file system
- Good security, scaling
- Global namespace
- "Professional" server infrastructure per cell
  - Don't try this at home
  - Only ~190 AFS cells (2002-03)
    - 8 are cmu.edu, 14 are in Pittsburgh
- "No write conflict" model only partial success

18

Today's Lecture

- DFS design comparisons continued
  - Topic 4: file access consistency
    - NFS, AFS, Sprite, and DCE DFS
  - Topic 5: Locking
- Other types of DFS
  - Coda – disconnected operation
  - LBFS – weakly connected operation

19

Background

- We are back to 1990s.
- Network is slow and not stable
- Terminal → "powerful" client
  - 33MHz CPU, 16MB RAM, 100MB hard drive
- Mobile Users appeared
  - 1st IBM Thinkpad in 1992
- We can do work at client without network

20

CODA

- Successor of the very successful Andrew File System (AFS)
- AFS
  - First DFS aimed at a campus-sized user community
  - Key ideas include
    - open-to-close consistency
    - callbacks

21

Hardware Model

- CODA and AFS assume that client workstations are personal computers controlled by their user/owner
  - ***Fully autonomous***
  - ***Cannot be trusted***
- CODA allows owners of laptops to operate them in ***disconnected mode***
  - ***Opposite of ubiquitous connectivity***

22

Accessibility

- Must handle two types of failures
  - ***Server failures:***
    - Data servers are ***replicated***
  - ***Communication failures*** and ***voluntary disconnections***
    - Coda uses ***optimistic replication*** and ***file hoarding***

23

Design Rationale

- Scalability
  - Callback cache coherence (inherit from AFS)
  - Whole file caching
  - Fat clients. (security, integrity)
  - Avoid system-wide rapid change
- Portable workstations
  - User's assistance in cache management

24

Design Rationale –Replica Control

- Pessimistic
  - Disable all partitioned writes
  - Require a client to acquire control of a cached object ***prior*** to disconnection
- Optimistic
  - Assuming no others touching the file
  - sophisticated: conflict detection
  - + fact: low write-sharing in Unix
  - + high availability: access anything in range

25

What about Consistency?

- ***Pessimistic replication control protocols*** guarantee the consistency of replicated in the presence of ***any non-Byzantine failures***
  - Typically require a quorum of replicas to allow access to the replicated data
  - Would **not** support disconnected mode

26

Pessimistic Replica Control

- Would require client to acquire ***exclusive*** (RW) or ***shared*** (R) control of cached objects before accessing them in disconnected mode:
  - Acceptable solution for voluntary disconnections
  - Does not work for involuntary disconnections
- What if the laptop remains disconnected for a long time?

27

Leases

- We could grant exclusive/shared control of the cached objects for a ***limited amount of time***
- Works very well in ***connected mode***
  - Reduces server workload
  - Server can keep leases in volatile storage as long as their duration is shorter than boot time
- Would only work for very short disconnection periods

28

Optimistic Replica Control (I)

- ***Optimistic replica control*** allows access in ***every*** disconnected mode
  - Tolerates temporary inconsistencies
  - Promises to detect them later
  - Provides ***much higher data availability***

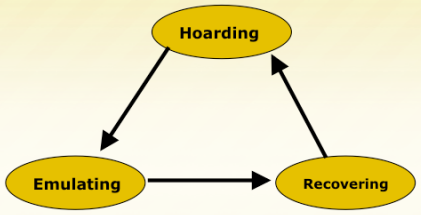
29

Optimistic Replica Control (II)

- Defines an ***accessible universe:*** set of replicas that the user can access
  - Accessible universe varies over time
- At any time, user
  - Will read from the latest replica(s) in his accessible universe
  - Will update all replicas in his accessible universe
  -

30

Coda (Venus) States



1. ***Hoarding:***
   Normal operation mode
2. ***Emulating:***
   Disconnected operation mode
3. ***Reintegrating:***
   Propagates changes and detects inconsistencies

31

Hoarding

- Hoard useful data for disconnection
- Balance the needs of connected and disconnected operation.
  - Cache size is restricted
  - Unpredictable disconnections
- Prioritized algorithm – cache manage
- hoard walking – reevaluate objects

32

Prioritized algorithm

- User defined hoard priority *p*: how interest it is?
- Recent Usage *q*
- Object priority = f(p,q)
- Kick out the one with lowest priority

+ Fully tunable
  Everything can be customized
- Not tunable (?)
  - No idea how to customize

33

Hoard Walking

- Equilibrium – uncached obj < cached obj
  - Why it may be broken? Cache size is limited.
- Walking: restore equilibrium
  - Reloading HDB (changed by others)
  - Reevaluate priorities in HDB and cache
  - Enhanced callback
- Increase scalability, and availability
- Decrease consistency

34

Emulation

- In emulation mode:
  - Attempts to access files that are not in the client caches appear as failures to application
  - All changes are written in a persistent log, the client modification log (CML)
  - Venus removes from log all obsolete entries like those pertaining to files that have been deleted

35

Persistence

- Venus keeps its cache and related data structures in non-volatile storage
- All Venus metadata are updated through ***atomic transactions***
  - Using a lightweight ***recoverable virtual memory*** (RVM) developed for Coda
  - Simplifies Venus design

36

Reintegration

- When workstation gets reconnected, Coda initiates a ***reintegration process***
  - Performed one volume at a time
  - Venus ships replay log to all volumes
  - Each volume performs a log replay algorithm
- Only care write/write confliction
  - Succeed?
    - Yes. Free logs, reset priority
    - No. Save logs to a tar. Ask for help

37

Performance

- Duration of Reintegration
  - A few hours disconnection → 1 min
- Cache size
  - 100MB at client is enough for a "typical" workday
- Conflicts
  - No Conflict at all! Why?
  - Over 99% modification by the same person
  - Two users modify the same obj within a day: <0.75%

38

Coda Summary

- Puts scalability and availability before data consistency
  - Unlike NFS
- Assumes that inconsistent updates are very infrequent
- Introduced disconnected operation mode and file hoarding

39

Remember this slide?

- We are back to 1990s.
- Network is slow and not stable
- Terminal → "powerful" client
  - 33MHz CPU, 16MB RAM, 100MB hard drive
- Mobile Users appear
  - 1st IBM Thinkpad in 1992

40

What's now?

- We are in 2000s now.
- Network is fast and reliable in LAN
- "powerful" client → very powerful client
  - 2.4GHz CPU, 1GB RAM, 120GB hard drive
- Mobile Users everywhere
- Do we still need disconnection?
  - How many people are using coda?

41

Do we still need disconnection?

- WAN and wireless is not very reliable, and is slow
- PDA is not very powerful
  - 200MHz strongARM, 128M CF Card
  - Electric power constrained
- LBFS (MIT) on WAN, Coda and Odyssey (CMU) for mobile users
  - Adaptation is also important

42

What is the future?

- High bandwidth, reliable wireless everywhere
- Even PDA is powerful
  - 2GHz, 1G RAM/Flash
- What will be the research topic in FS?
  - P2P?

43

Today's Lecture

- DFS design comparisons continued
  - Topic 4: file access consistency
    - NFS, AFS, Sprite, and DCE DFS
  - Topic 5: Locking
- Other types of DFS
  - Coda – disconnected operation
  - LBFS – weakly connected operation

44

Low Bandwidth File System Key Ideas

- A network file systems for slow or wide-area networks
- Exploits similarities between files or versions of the same file
  - Avoids sending data that can be found in the server's file system or the client's cache
- Also uses conventional compression and caching
- Requires 90% less bandwidth than traditional network file systems

45

Working on slow networks

- Make local copies
  - Must worry about update conflicts
- Use remote login
  - Only for text-based applications
- Use instead a LBFS
  - Better than remote login
  - Must deal with issues like auto-saves blocking the editor for the duration of transfer

46

LBFS design

- Provides close-to-open consistency
- Uses a large, persistent file cache at client
  - Stores clients working set of files
- LBFS server divides file it stores into chunks and indexes the chunks by hash value
- Client similarly indexes its file cache
- Exploits similarities between files
  - LBFS never transfers chunks that the recipient already has

47

Indexing

- Uses the SHA-1 algorithm for hashing
  - It is collision resistant
- Central challenge in indexing file chunks is keeping the index at a reasonable size while dealing with shifting offsets
  - Indexing the hashes of fixed size data blocks
  - Indexing the hashes of all overlapping blocks at all offsets

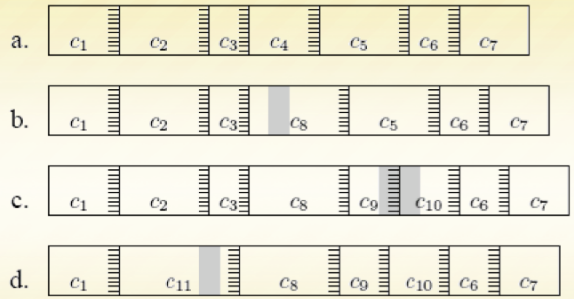
48

LBFS indexing solution

- Considers only non-overlapping chunks
- Sets chunk boundaries based on file contents rather than on position within a file
- Examines every overlapping 48-byte region of file to select the boundary regions called *breakpoints* using Rabin fingerprints
  - When low-order 13 bits of region's fingerprint equals a chosen value, the region constitutes a breakpoint

49

Effects of edits on file chunks



- Chunks of file before/after edits
  - Grey shading show edits
- Stripes show 48byte regions with magic hash values creating chunk boundaries

50

More Indexing Issues

- Pathological cases
  - Very small chunks
    - Sending hashes of chunks would consume as much bandwidth as just sending the file
  - Very large chunks
    - Cannot be sent in a single RPC
- LBFS imposes minimum and maximum chuck sizes

51

The Chunk Database

- Indexes each chunk by the first 64 bits of its SHA-1 hash
- To avoid synchronization problems, LBFS always recomputes the SHA-1 hash of any data chunk before using it
  - Simplifies crash recovery
- Recomputed SHA-1 values are also used to detect hash collisions in the database

52

Conclusion

- Under normal circumstances, LBFS consumes 90% less bandwidth than traditional file systems.
- Makes transparent remote file access a viable and less frustrating alternative to running interactive programs on remote machines.

53