416 Distributed Systems

Distributed File Systems 4

Jan 23, 2017

Today's Lecture

- Wrap up NFS/AFS
- This lecture: other types of DFS
  - Coda – disconnected operation

Key Lessons

- Distributed filesystems almost always involve a tradeoff: consistency, performance, scalability.
  - Notice consistency/performance trade-offs between NFS and AFS (and different assumptions about workload)
- We’ve learned a lot since NFS and AFS (and can implement faster, etc.), but the general lesson holds. Especially in the wide-area.
- Well see a related tradeoff, also involving consistency, in a while: the CAP tradeoff. Consistency, Availability, Partition-resilience.

More Key Lessons

- Client-side caching is a fundamental technique to improve scalability and performance
  - But raises important questions of cache consistency
- Timeouts and callbacks are common methods for providing (some forms of) consistency.
- AFS picked close-to-open (session) consistency as a good balance of usability (the model seems intuitive to users), performance, etc.
  - AFS authors argued that apps with highly concurrent, shared access, like databases, needed a different model

Key to Simple Failure Recovery

- Try not to keep any state on the server
- If you must keep some state on the server
  - Understand why and what state the server is keeping
  - Understand the worst case scenario of no state on the server and see if there are still ways to meet the correctness goals
  - Revert to this worst case in each combination of failure cases (since on failure server loses state)

Today's Lecture

- Wrap up NFS/AFS
- Other types of DFS
  - Coda – disconnected operation

Background

- We are back to 1990s.
- Network is slow and not stable
- Transition from 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!
  - Novel at the time; ubiquitous idea today

Hardware Model

- CODA: Successor of AFS
- 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*** **(our focus)**
  - ***Opposite of ubiquitous connectivity***

Accessibility (aka availability)

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

Design Rationale –Replica Control

- Pessimistic
  - Disable all partitioned writes
  - Require a client to acquire control of a cached object ***prior*** to disconnection

- Optimistic
  - Assumes no one else touching the file
  - conflict detection
  - \+ workload fact: low write-sharing in Unix
  - \+ high availability: access anything in range

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?

Leases mechanism

- A lease grants exclusive/shared control of the cached objects for a ***limited amount of time***
  - A popular way to efficiently implement pessimistic replica control
- Works very well in ***connected mode***
  - Reduces server workload (how?)
  - Server can keep leases in volatile storage as long as their duration is shorter than boot time (why?)
- Would only work for very short disconnection periods

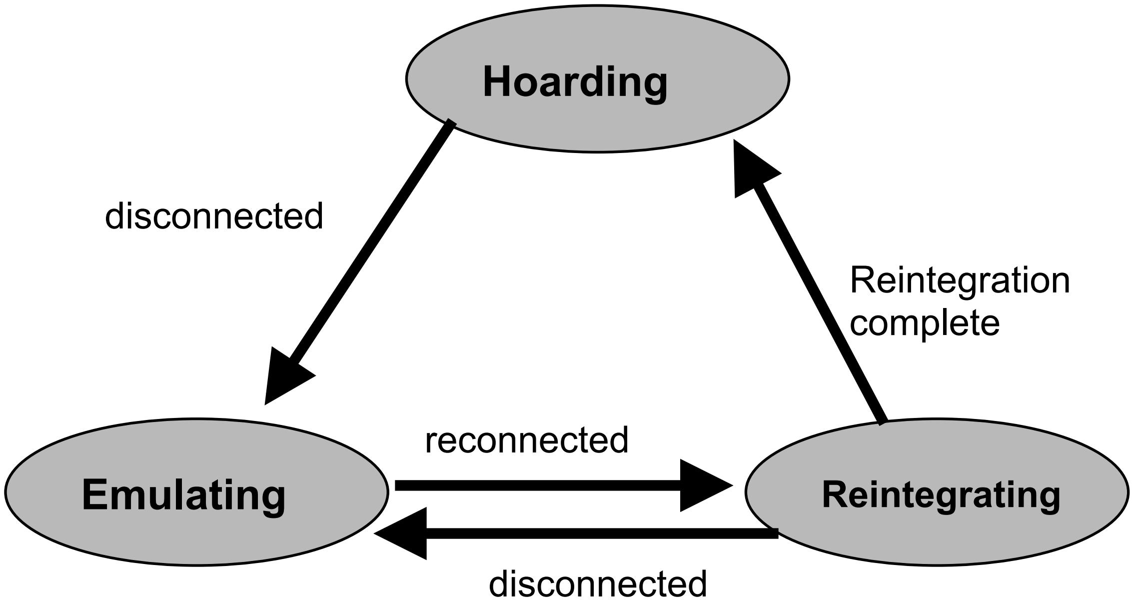
Optimistic Replica Control (I)

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

Optimistic Replica Control (II)

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

Coda node states



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

Hoarding

- Hoard useful data for disconnection
- Balance the needs of connected and disconnected operation.
  - Cache size is restricted
  - Unpredictable disconnections
- Uses user specified preferences + usage patterns to decide on files to keep in hoard

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)
  - Coda removes from log all obsolete entries like those pertaining to files that have been deleted

Reintegration

- When workstation is reconnected, Coda initiates a ***reintegration process***
  - Performed one volume at a time
  - Ships replay log to each volumes
  - Each volume performs a log replay algorithm

- Only care about write/write confliction
  - Conflict resolution succeeds?
    - Yes. Free logs, keep going…
    - No. Save logs to a tar. Ask for help

- In practice:
  - No Conflict at all! Why?
  - Over 99% modification by the same person
  - Two users modify the same obj. within a day: <0.75%

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