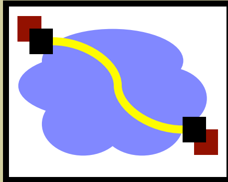
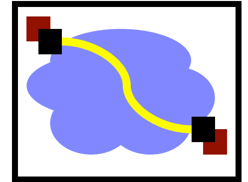


416 Distributed Systems

Distributed File Systems 3

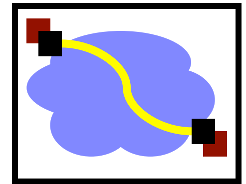
Jan 22, 2015

Today's Lecture



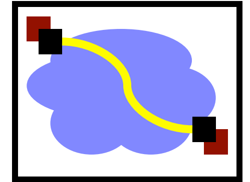
- Last time:
  - Topic 2: file access consistency
    - NFS, AFS
  - Topic 3: name space construction
    - Mount (NFS) vs. global name space (AFS)
  - Topic 4: Security in distributed file systems
    - Kerberos
- This lecture: other types of DFS
  - Coda – disconnected operation

Topic 4: User Authentication and Access Control



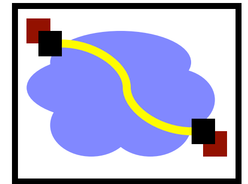
- User X logs onto workstation A, wants to access files on server B
  - How does A tell B who X is?
  - Should B believe A?
- Choices made in NFS V2
  - All servers and all client workstations share the same <uid, gid> name space → B send X's <uid,gid> to A
    - Problem: root access on any client workstation can lead to creation of users of arbitrary <uid, gid>
  - Server believes client workstation unconditionally
    - Problem: if any client workstation is broken into, the protection of data on the server is lost;
    - <uid, gid> sent in clear-text over wire → request packets can be faked easily

User Authentication (cont'd)

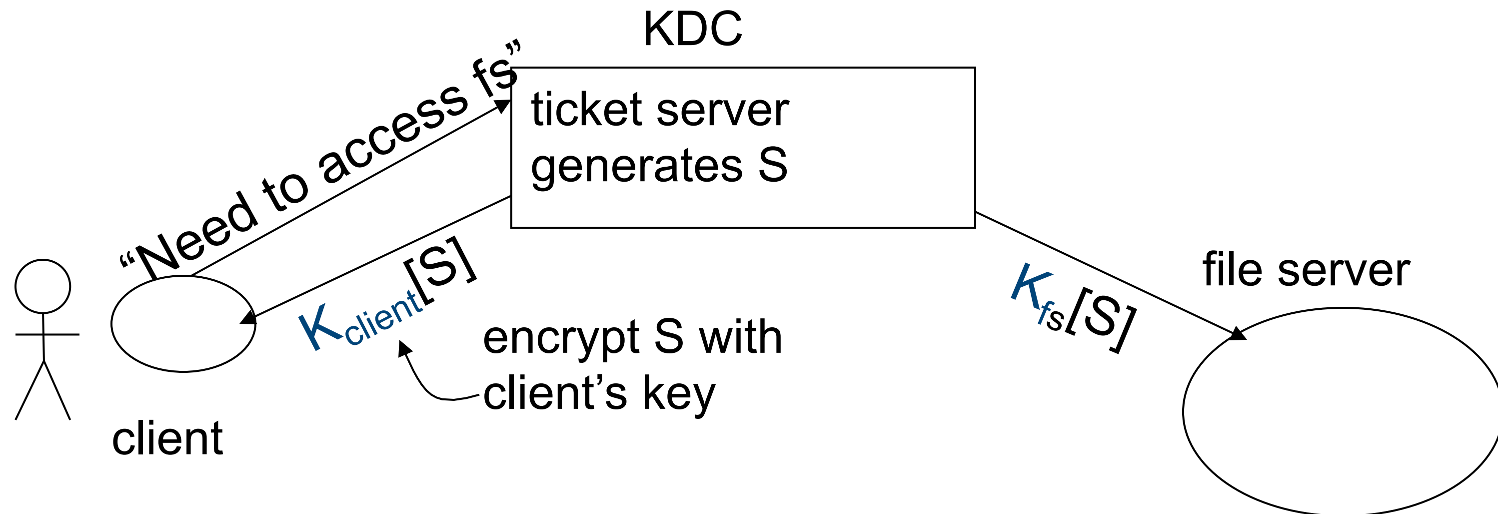


- How do we fix the problems in NFS v2
  - Hack 1: root remapping → strange behavior
  - Hack 2: UID remapping → no user mobility
  - Real Solution: use a centralized Authentication/ Authorization/Access-control (AAA) system

A Better AAA System: Kerberos

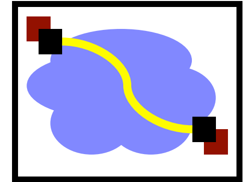


- Basic idea: shared secrets
  - User proves to KDC (Kerberos key distribution center) who he is; KDC generates shared secret between client and file server



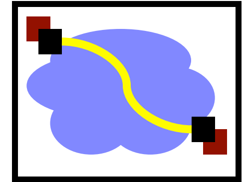
S: specific to {client,fs} pair;
"short-term session-key"; expiration time (e.g. 8 hours)

Key Lessons



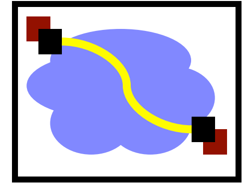
- Distributed filesystems almost always involve a tradeoff: consistency, performance, scalability.
- We'll see a related tradeoff, also involving consistency, in a while: the CAP tradeoff. Consistency, Availability, Partition-resilience.

More Key Lessons



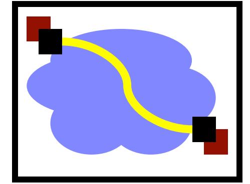
- Client-side caching is a fundamental technique to improve scalability and performance
  - But raises important questions of cache consistency
- Timeouts and callbacks are common methods for providing (some forms of) consistency.
- AFS picked close-to-open consistency as a good balance of usability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

Today's Lecture



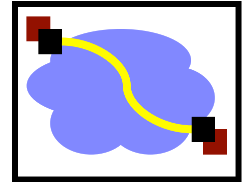
- DFS design comparisons continued
  - Topic 2: file access consistency
    - NFS, AFS
  - Topic 3: name space construction
    - Mount (NFS) vs. global name space (AFS)
  - Topic 4: AAA in distributed file systems
    - Kerberos
- Other types of DFS
  - Coda – disconnected operation

Background



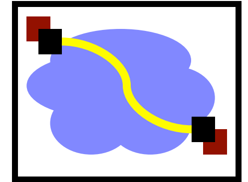
- We are back to 1990s.
- Network is slow and not stable
- Terminal → "powerful" client
  - 33MHz CPU, 16MB RAM, 100MB hard drive
- Mobile Users appeared
  - 1st IBM Thinkpad in 1992
- We can do work at client without network

CODA



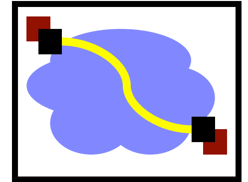
- Successor of the very successful Andrew File System (AFS)
- AFS
  - First DFS aimed at a campus-sized user community
  - Key ideas include
    - open-to-close consistency (session semantics)
    - callbacks

Hardware Model



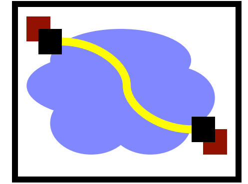
- CODA and AFS assume that client workstations are personal computers controlled by their user/ owner
  - ***Fully autonomous***
  - ***Cannot be trusted***
- CODA allows owners of laptops to operate them in ***disconnected mode***
  - ***Opposite of ubiquitous connectivity***

Accessibility (aka availability)



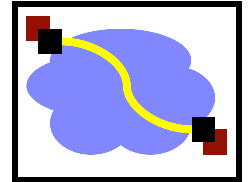
- Must handle two types of failures
  - ***Server failures:***
    - Data servers are ***replicated***
  - ***Communication failures*** and ***voluntary disconnections***
    - Coda uses ***optimistic replication*** and ***file hoarding***

Design Rationale –Replica Control



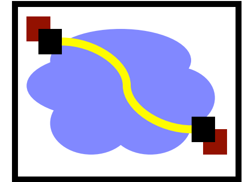
- Pessimistic
  - Disable all partitioned writes
  - Require a client to acquire control of a cached object ***prior*** to disconnection
- Optimistic
  - Assuming no others touching the file
  - conflict detection
  - + fact: low write-sharing in Unix
  - + high availability: access anything in range

Pessimistic Replica Control



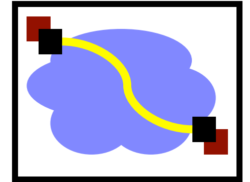
- Would require client to acquire ***exclusive*** (RW) or ***shared*** (R) control of cached objects before accessing them in disconnected mode:
  - Acceptable solution for voluntary disconnections
  - Does not work for involuntary disconnections
- What if the laptop remains disconnected for a long time?

Leases



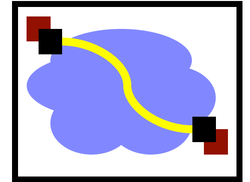
- We could grant exclusive/shared control of the cached objects for a ***limited amount of time***
- Works very well in ***connected mode***
  - Reduces server workload
  - Server can keep leases in volatile storage as long as their duration is shorter than boot time
- Would only work for very short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

Optimistic Replica Control (I)



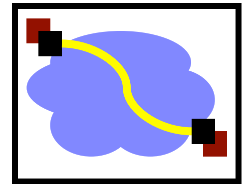
- ***Optimistic replica control*** allows access in ***every*** disconnected mode
  - Tolerates temporary inconsistencies
  - Promises to detect them later
  - Provides ***much higher data availability***

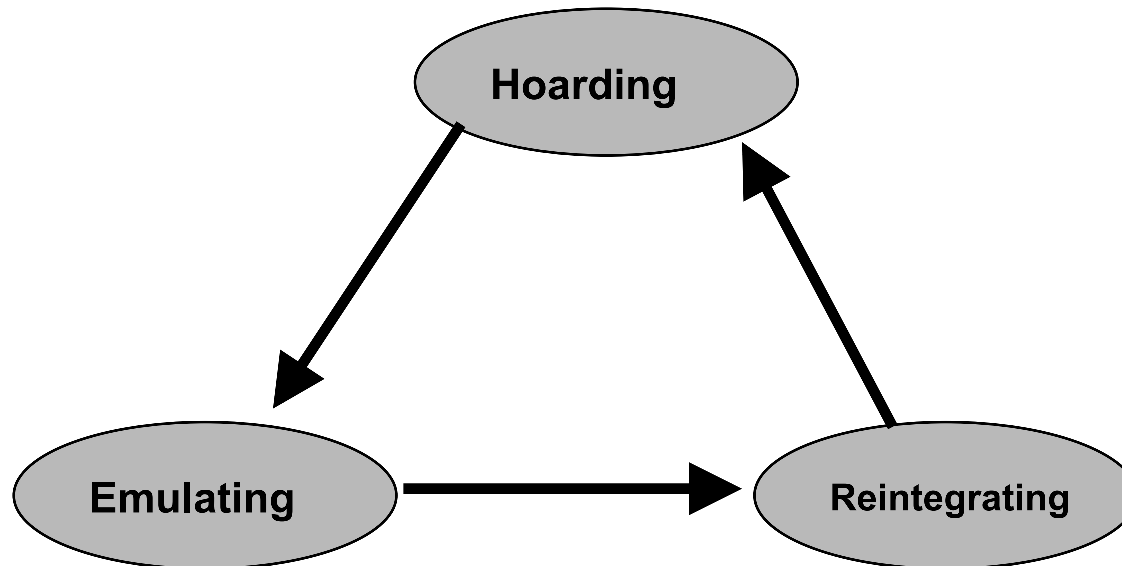
Optimistic Replica Control (II)



- Defines an ***accessible universe:*** set of filesthat 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

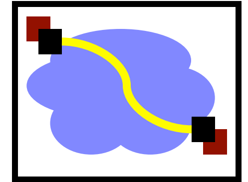
Coda States





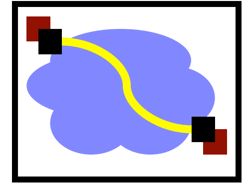
1. ***Hoarding:***
   Normal operation mode
2. ***Emulating:***
   Disconnected operation mode
3. ***Reintegrating:***
   Propagates changes and detects inconsistencies

Hoarding



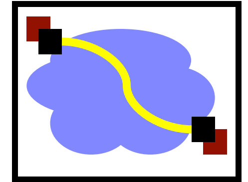
- Hoard useful data for disconnection
- Balance the needs of connected and disconnected operation.
  - Cache size is restricted
  - Unpredictable disconnections
- Uses user specified preferences + usage patterns to decide on files to keep in hoard

Emulation



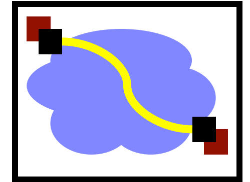
- In emulation mode:
  - Attempts to access files that are not in the client caches appear as failures to application
  - All changes are written in a persistent log, the client modification log (CML)
  - Coda removes from log all obsolete entries like those pertaining to files that have been deleted

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