CSCI 5105

Instructor: Abhishek Chandra

Today

- Distributed File Systems

Distributed File Systems

- Why do we need these?
  - Data sharing in distributed systems
  - Provide easy interface to users
  - Hide distributed nature of data
- Distributed File Systems:
  - Form the basis for several distributed systems
  - Also illustrate various principles of distributed systems

Distributed File Systems: Issues

- File Service
  - What interface/operation is provided to users?
  - How is data stored and fetched?
- Naming and location
  - Where is a file located?
  - How do different users refer to the same file?
- Sharing, caching and replication
  - What happens for concurrent writes?
  - How to tradeoff consistency-performance?
- Fault Tolerance and Security
  - What if file server fails?
  - How to authenticate remote users?

Distributed File Systems: Design

- Depends on the usage environment
  - Scale of system
  - Location of data
  - Fault/security model of system
  - Read/write patterns of users

5

Distributed File Systems: Examples

- NFS: Transparent user access to distributed data
- Coda: High availability in mobile environments
- GFS: Handle large files and data volumes
- PAST: Scalable archival system

6

Coda

- Main Goals:
  - Availability: Work in the presence of disconnection
  - Scalability: Support large number of users
- Successor of Andrew File System
  - Developed at CMU
  - Uses similar basic architecture
- Client-server architecture

7

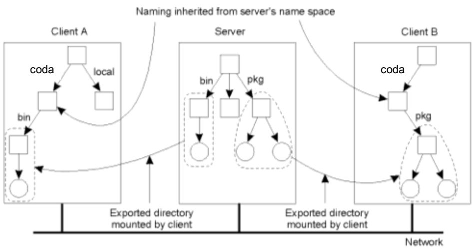
Naming: Volumes

- Volume is a subtree in the naming space
  - Like a disk partition, but finer granularity
- Mounting and replication takes place at volume granularity
  - Whole volume must be mounted from its root
  - Crossing mount points allowed

8

Naming: Shared Name Space

- All clients see same shared name space
  - All coda volumes are mounted under /coda
  - Each mounted volume inherits name from server namespace



9

RPC2

- Coda runs on top of RPC2
  - Enhanced version of RPC
- RPC2 supports
  - Reliable RPC over UDP
  - Side-effects: Application-specific modules called by client and server stubs
  - MultiRPC: Using parallel one-to-one RPCs or IP multicast

10

Caching and Replication

- Client Caching
  - Each file is completely transferred to the client on access
- Combination of session and transactional semantics
  - Each session is considered to be a transaction
  - Effect of each session applied sequentially

11

Consistency

- Server makes a callback promise
  - Will notify when file is updated
- Callback break
  - On file modification, server sends invalidation
- Session-transaction semantics
  - Current session unaffected
  - Client needs to download updated file before the next session

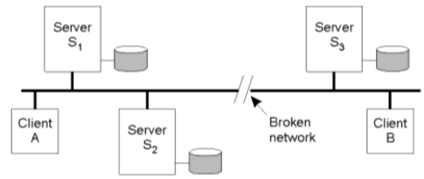
12

Server Replication

- Volume Storage Group (VSG): Group of servers holding a volume replica
- Available Volume Storage (AVSG): Members of VSG connected to a client
- Update policy: Read-One, Write-All
  - Performed on members of AVSG

13

Network Partitions



- Optimistic strategy: Commit changes to local AVSGs
- Conflict detection: Use version vectors
- Conflict resolution: Application-dependent or manual

14

Disconnected Operations

- Disconnected clients can operate on their local copies
  - Update propagation and conflict resolution done on reconnection
- How does a user get a file if it is disconnected?

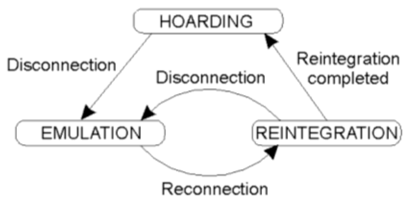
15

Hoarding

- Files are pre-cached on the client
- Files to be hoarded determined using priority
  - Hoard database: Important files specified by user
  - File recently referenced
- Files are fetched to ensure that:
  - Higher priority files always cached first
  - Get non-zero priority files if cache has space
- Hoard walk: Reorganizing the cache periodically to meet above requirements

16

Hoarding: Transitions



Google File System (GFS)

- Main Goals:
  - Scalability: Large amount of data
  - Performance: High rate of distributed processing
  - Fault-tolerance: Server failures common

Google File Characteristics

- File sizes very large
  - Generally a collection of crawled documents
  - Several MB-GBs
- Processing:
  - Dumping crawled content
  - Parsing files, building indices

Google File System Requirements

- Built from commodity hardware
- Modest no. of very large files
- File reads:
  - Large sequential reads
  - Small random reads
- File writes:
  - Large, sequential appends
  - Many concurrent appends
- High bandwidth rather than low latency

Google Server Cluster

- Each cluster maintains some files
  - Each file divided into chunks
  - Chunk has a global handle
- One master and multiple chunk servers

Google Server Cluster

- Multiple chunk servers
  - Each maintains one chunk of the file
  - Chunk maintained on local FS
- Master
  - Responsible for naming
  - Maintains metadata
  - Location of <filename, chunk index>

Client File Access

- Client maps file offset into chunk index
- Master sends chunk handle and chunk server locations to client
- Client reads chunk directly from chunk server

Reliability and Consistency: Chunk Data

- Each chunk is replicated
- Updates performed using a primary-backup scheme
  - Server grants lease to a chunkserver to become primary replica
  - Data disseminated separately from update control flow
- Caching: No caching on client or chunk server

Replication and Consistency: MetaData

- Master keeps information in main memory
- Naming index of chunks
  - Updated by polling chunk servers from time-to-time
  - Could have stale information
- Critical metadata changes
  - Logged to an operational log
  - Reconstructed by log replay
  - Checkpointing from time-to-time

25