Provides Different Models of Consistency

• Causal ordering

• Forced ordering (linearizable, think Raft)
  – both causal and total order

• Immediate ordering
  – forced ordering with minimal delay
Gossip

- A special form of multicast
  - Each node periodically sends updates to a subset of nodes
  - The subset is chosen at random each time

- How do we get forced updates? Or immediate updates?
  - Normal Gossip only provides casual ordering
  - Recall we want total ordering
Forced Updates (1)

• Need a causal order that’s also total
  – all clients go through same RM
Forced Updates (2)

- What if primary crashes?
  - elect new primary
Immediate Update

- Forced update + time constraints
Immediate Updates (1)

- Primary requests logs and replica timestamps
  - backups respond and stop processing queries
  - updates are accepted but not executed
Immediate Updates (2)

- Backups respond with logs and timestamps
  - primary stops processing queries and updates
  - processes logs and timestamps
Immediate Updates (3)

- Primary assigns timestamp to update
- Primary sends update to backups
Immediate Updates (4)

- Backups acknowledge updates
Immediate Updates (5)

- After half the backups respond, primary commits (updates val) and responds to client
  - half the backups + primary = majority
Immediate Updates (6)

- Primary sends log to each backup (gossips)
Problem?

• What if client sends update request to multiple RMs?
  – multiple copies of the request are propagated
  – all are executed
  – probably aren’t idempotent

• How can you make this idempotent?
Solution

• Client assigns unique ID (CID) to each request

• RMs keep track of CIDs of completed requests
  – completed requests go to invalid CIDs list
  – check list before doing a request
    - don’t perform requests that have already been performed
Updated Update Model

Client

Replica Timestamp

Distributed Service

Replica Manager

Replica Manager

Invalid CID

Log

Value

TS

TS

TS
Another Problem?

• Won’t logs and invalid CIDs lists grow without bound?
  – yes …
Bounding Logs (1)

• Each log entry \( r \) must be kept on RM \( i \) until it is present on all RMs
  – so that gossip from \( i \) will inform other RMs
• \( r.\text{node} \) is the node that created the log entry \( r \)
• \( r.\text{ts} \) is the vector timestamp assigned to the log entry by \( r.\text{node} \)
• \( r.\text{ts}[r.\text{node}] \) is the logical time on \( r.\text{node} \) when the entry was created
• \( r \) may be removed from \( i \)'s log when:
  – \( \forall k: r.\text{ts}[r.\text{node}] \leq \text{rm}_k.\text{replica.ts}[r.\text{node}] \)
Bounding Logs (2)

• How does RM $i$ know $\text{rm}_k.\text{replica.ts}[r.\text{node}]$?
  – gossip messages contain replica timestamps
    - timestamps on logs
  – each RM keeps a table of the most recent timestamps obtained from all other RMs
    - $\text{rm}_i.\text{ts\_table}$
    - $\forall k \ \text{rm}_i.\text{ts\_table}[k] \leq \text{rm}_k.\text{replica.ts}$
  – RM $i$ may remove log entry $r$ when:
    - All other RM nodes have a higher timestamp for node $r$ that the log message being deleted
    - $\forall k: \ r.\text{ts}[r.\text{node}] \leq \text{rm}_i.\text{ts\_table}[k][r.\text{node}]$
Trimming the Invalid CIDs List

• When can an entry be removed?
  – when it will never be received again

• Assuming perfect communication, how can you tell?
  – you can’t: client’s front-end might send same update to multiple RMs
  – what’s more, communication might not be perfect

• More machinery needed …
Replication Strategies

• Lazy, Passive, Active

• Provide different consistency guarantees
  – Passive, Active: Linearizable
  – Lazy: both linearizable and causal

• Gossip provides Lazy replication
  – Uses vector clocks to maintain causal consistency
  – Switches to something similar to passive replication for linearizable
Distributed File Systems
Outline

• Failure

• Basic concepts

• NFS version 2

• CIFS (Common Internet File System)

• DCE DFS (distributed computing environment distributed file system)

• NFS version 4
Synchronous vs. Asynchronous

• Execution speed
  – synchronous: bounded
  – asynchronous: unbounded

• Message transmission delays
  – synchronous: bounded
  – asynchronous: unbounded

• Local clock drift rate:
  – synchronous: bounded
  – asynchronous: unbounded
Failures

- **Omission failures**
  - something doesn’t happen
    - process crashes
    - data lost in transmission
    - etc.

- **Byzantine (arbitrary) failures**
  - something bad happens
    - message is modified
    - message received twice
    - etc.

- **Timing failures**
  - something takes too long
Detecting Crashes

• Synchronous systems
  – timeouts
• Asynchronous systems
  – ?
• Fail-stop
  – an oracle lets us know
DFS Scenario
DFS Components

• Data state
  – file contents

• Attribute state
  – size, access-control info, modification time, etc.

• Open-file state
  – which files are in use (open)
  – lock state
Possible Locations

Client
- data cache
- attr cache
- open-file state

Server
- data cache
- attr cache
- open-file state
- local file system
In Practice …

- Data state
  - NFS
    - weakly consistent
    - less weak if program uses locks
  - CIFS and DCE DFS
    - strictly consistent

- Lock state
  - must be strictly consistent
Failures in a Local File System
Distributed Failure
In Practice …

• NFS version 2
  – relaxed approach to consistency
  – handles failures well

• CIFS
  – strictly consistent
  – intolerant of failures

• DCE DFS
  – strictly consistent
  – sort of tolerant of failures

• NFS version 4
  – either relaxed or strictly-consistent
  – handles failures very well
NFS Version 2

• Released in mid 1980s
• Three protocols in one
  – file protocol
  – mount protocol
  – network lock manager protocol
Design Goals

• Improve fault tolerance
  – No state at the server
  – Operations are idempotent

• Improve performance
  – Caching at client and server
    - Client: cache in client side to reduce overhead of using the network
    - Server: Cache in memory to reduce overhead of going to disk

• Interesting Design questions
  – What state to maintain at server or clients?
  – How to maintain cache coherence?
Design Trade-off: Consistency vs. Performance

• **Strict consistency is easy …**
  – … if all operations take place on server
  – no client caching

• **Performance is good …**
  – … if all operations take place on client
  – everything is cached on client
Distribution of Components

NFSv2 client

- data cache
- attr cache
- open-file state

NFSv2 server

- data cache
- attr cache

local file system
Consistency in Basic NFSv2

Data cache

file x block 1
file x block 5
file y block 2
file y block 17

Attribute cache

file x attrs
validity period

file y attrs
validity period
More …

• All write RPC requests must be handled synchronously on the server
• Close-to-Open consistency
  – client writes back all changes on close
  – flushes cached file info on open
Client Crash Recovery
Server Crash Recovery

Hard mount:
Client keeps retrying until server returns

Soft mount:
Client retries a few times and gives up
File Locking

• Exception to stateless server rule

• State is required on the server!
  – recovery must take place in the event of client and server crashes

• Locking Protocol is independent of the File Protocol
  – locking is *advisory*
  – one can lock a file and ask if a file is locked
  – not required to honor locks
    - may read/write a file locked by others!
Network Lock Manager Protocol

Lockd
- Lock manager
- Lock/unlock

Statd
- Status monitor
- Cop with crashes

Server failure
- Loses lock info
- On restart ask clients for this info

Client failure
- Server detects failure and revokes locks
NFS Version 3

• In use at Brown and in most of the rest of the world

• Basically the same as NFSv2
  – improved handling of attributes
  – *commit* operation for writes
  – various other things
CIFS

• Common Internet File System
  – Microsoft’s distributed file system
  – Alternate locking approach

• Features
  – strictly consistent

• Not featured …
  – depends on reliability of transport protocol
  – loss of connection == loss of session
Consistency vs. Performance

- Strict consistency is easy …
  - … if all operations take place on server
  - no client caching
- Performance is good …
  - … if all operations take place on client
  - everything is cached on client
- Put the two together …

Ø

- or you can do opportunistic locking
Opportunistic Locks

Client 1

Open A
OK, Op Lock
Revoke Op Lock
OK, changes

Server

Client 2

Open A
OK

Files and Directories
Back to NFS

- File system name space
  - how is distributed file system perceived on clients?

- Cross-computer links
  - how are files on other computers referred to?
NSF servers and File Systems

• files divided up into disjoint collections called file system
• Can migrate file system between servers
  – File system is the smallest granularity for migration
• Servers maintain ACL
  – Mapping clients to allowed file systems
NFS Mount Protocol
Server File Systems
File Handles
(wrapper around file IDs)

- Servers provide opaque *file handles* to clients to refer to files
  - contents mean nothing to clients
  - identify files on server

- Clients contact server via mount protocol to obtain file handles of roots of exported file systems
File Handle Contents

- File-System ID
  - which server file system

- File ID (inode in unix/linux)
  - which file within file system

- Generation #
  - Inode #s get re-used
  - guards against inode reuse
Client vs. Server Mount Points (1)

mount server:/B  /C2
Client vs. Server Mount Points (2)

mount server:/B/C2
mount server:/B/F/K/C2/F/K
Local vs. Global Namespace

• Local namespace
  – each host configures its own file-system namespace
  – NFS clients each mount the appropriate remote file systems

• Global namespace
  – all hosts share the same namespace
  – not done in early NFS
Mount Protocol Problems

• Local namespaces don’t work
• Achieve global name space by having each client mount everything consistently
  • giving each client a table listing all possible mounts is administratively difficult
  • performing all possible mounts is time consuming
    • mounting is a “heavyweight” operation
Rather than this ...

dev  etc  home

max  louisa  twd  carlos  rohil  rodrigo  atty  haris  ishan
… this

dev  etc  home

automount database

Autoofs
Automounting: 2000

- Maintain description of global namespace in global database: NIS (network info. System)

- Do mounts only when needed

- Automount times out after period of unuse
Automounting: 2017

- Global namespace maintained in LDAP database
  - lightweight directory access protocol
    - vendor neutral
  - everything mounted at boottime
    - fewer, but larger, file systems
  - no timeout