

CSE 486/586 Distributed Systems Distributed File Systems

Steve Ko
Computer Sciences and Engineering
University at Buffalo

CSE 486/586, Spring 2014

Recap

- Distributed transactions with replication
 - One copy serializability
 - Primary copy replication
 - Read-one/write-all replication
 - Available copies replication

CSE 486/586, Spring 2014

2

Local File Systems

- File systems provides file management.
 - Name space
 - API for file operations (create, delete, open, close, read, write, append, truncate, etc.)
 - Physical storage management & allocation (e.g., block storage)
 - Security and protection (access control)
- Name space is usually hierarchical.
 - Files and directories
- File systems are mounted.
 - Different file systems can be in the same name space.

CSE 486/586, Spring 2014

3

Traditional Distributed File Systems

- Goal: emulate local file system behaviors
 - Files not replicated
 - No hard performance guarantee
- But,
 - Files located remotely on servers
 - Multiple clients access the servers
- Why?
 - Users with multiple machines
 - Data sharing for multiple users
 - Consolidated data management (e.g., in an enterprise)

CSE 486/586, Spring 2014

4

Requirements

- **Transparency**: a distributed file system should appear as if it's a local file system
 - **Access transparency**: it should support the same set of operations, i.e., a program that works for a local file system should work for a DFS.
 - **(File) Location transparency**: all clients should see the same name space.
 - **Migration transparency**: if files move to another server, it shouldn't be visible to users.
 - **Performance transparency**: it should provide reasonably consistent performance.
 - **Scaling transparency**: it should be able to scale incrementally by adding more servers.

CSE 486/586, Spring 2014

5

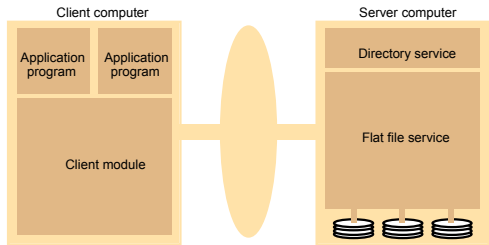
Requirements

- **Concurrent updates** should be supported.
- **Fault tolerance**: servers may crash, msgs can be lost, etc.
- **Consistency** needs to be maintained.
- **Security**: access-control for files & authentication of users

CSE 486/586, Spring 2014

6

File Server Architecture



CSE 486/586, Spring 2014

7

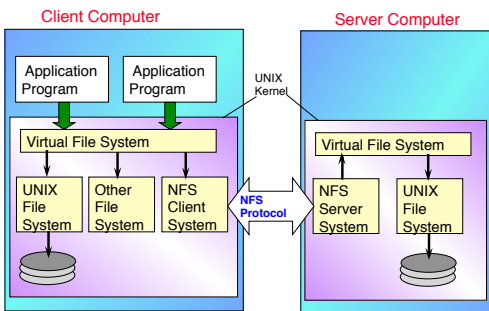
Components

- Directory service
 - Meta data management
 - Creates and updates directories (hierarchical file structures)
 - Provides mappings between user names of files and the unique file ids in the flat file structure.
- Flat file service
 - Actual data management
 - File operations (create, delete, read, write, access control, etc.)
- These can be independently distributed.
 - E.g., centralized directory service & distributed flat file service

CSE 486/586, Spring 2014

8

Sun NFS



CSE 486/586, Spring 2014

9

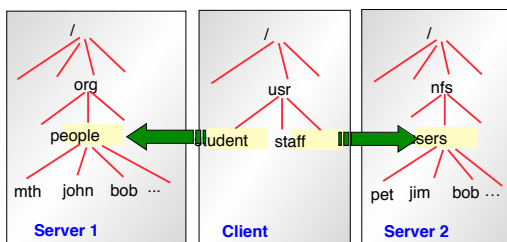
VFS

- A translation layer that makes file systems pluggable & co-exist
 - E.g., NFS, EXT2, EXT3, ZFS, etc.
- Keeps track of file systems that are available locally and remotely.
- Passes requests to appropriate local or remote file systems
- Distinguishes between local and remote files.

CSE 486/586, Spring 2014

10

NFS Mount Service



➡ Each server keeps a record of local files available for remote mounting. Clients use a *mount* command for remote mounting, providing name mappings

Remote Mount

CSE 486/586, Spring 2014

11

NFS Basic Operations

- Client
 - Transfers blocks of files to and from server via RPC
- Server
 - Provides a conventional RPC interface at a well-known port on each host
 - Stores files and directories
- Problems?
 - Performance
 - Failures

CSE 486/586, Spring 2014

12

Improving Performance

- Let's cache!
- Server-side
 - Typically done by OS & disks anyway
 - A disk usually has a cache built-in.
 - OS caches file pages, directories, and file attributes that have been read from the disk in a *main memory buffer cache*.
- Client-side
 - On accessing data, cache it locally.
- What's a typical problem with caching?
 - Consistency: cached data can become stale.

CSE 486/586, Spring 2014

13

(General) Caching Strategies

- Read-ahead (prefetch)
 - Read strategy
 - Anticipates read accesses and fetches the pages following those that have most recently been read.
- Delayed-write
 - Write strategy
 - New writes stored locally.
 - Periodically or when another client accesses, send back the updates to the server
- Write-through
 - Write strategy
 - Writes go all the way to the server's disk
- This is not an exhaustive list!

CSE 486/586, Spring 2014

14

NFS Client-Side Caching

- Write-through, but only at close()
 - Not every single write
 - Helps performance
- Other clients periodically check if there's any new write (next slide).
- Multiple writers
 - No guarantee
 - Could be any combination of writes
- Leads to inconsistency

CSE 486/586, Spring 2014

15

Validation

- A client checks with the server about cached blocks.
- Each block has a timestamp.
 - If the remote block is new, then the client invalidates the local cached block.
- Always invalidate after some period of time
 - 3 seconds for files
 - 30 seconds for directories
- Written blocks are marked as "dirty."

CSE 486/586, Spring 2014

16

Failures

- Two design choices: stateful & stateless
- Stateful
 - The server maintains all client information (which file, which block of the file, the offset within the block, file lock, etc.)
 - Good for the client-side process (just send requests!)
 - Becomes almost like a local file system (e.g., locking is easy to implement)
- Problem?
 - Server crash → lose the client state
 - Becomes complicated to deal with failures

CSE 486/586, Spring 2014

17

Failures

- Stateless
 - Clients maintain their own information (which file, which block of the file, the offset within the block, etc.)
 - The server does not know anything about what a client does.
 - Each request contains complete information (file name, offset, etc.)
 - Easier to deal with server crashes (nothing to lose!)
- NFS's choice
- Problem?
 - Locking becomes difficult.

CSE 486/586, Spring 2014

18

NFS

- Client-side caching for improved performance
- Write-through at close()
 - Consistency issue
- Stateless server
 - Easier to deal with failures
 - Locking is not supported (later versions of NFS support locking though)
- Simple design
 - Led to simple implementation, acceptable performance, easier maintenance, etc.
 - Ultimately led to its popularity

CSE 486/586, Spring 2014

19

CSE 486/586 Administrivia

- Survey!

CSE 486/586, Spring 2014

20

New Trends in Distributed Storage

- Geo-replication: replication with multiple data centers
 - Latency: serving nearby clients
 - Fault-tolerance: disaster recovery
- Power efficiency: power-efficient storage
 - Going green!
 - Data centers consume lots of power

CSE 486/586, Spring 2014

21

Power Consumption

- eBay: 16K servers, $\sim 0.6 * 10^5$ MWh, $\sim \$3.7$ M
- Akamai: 40K servers, $\sim 1.7 * 10^5$ MWh, $\sim \$10$ M
- Rackspace: 50K servers, $\sim 2 * 10^5$ MWh, $\sim \$12$ M
- Microsoft: > 200 K servers, $> 6 * 10^5$ MWh, $> \$36$ M
- Google: > 500 K servers, $> 6.3 * 10^5$ MWh, $> \$38$ M
- USA (2006): 10.9M servers, $610 * 10^5$ MWh, $\$4.5$ B
- Year-to-year: 1.7%~2.2% of total electricity use in US
- <http://ccr.sigcomm.org/online/files/p123.pdf>
- Question: can we reduce the energy footprint of a distributed storage while preserving performance?

CSE 486/586, Spring 2014

22

Flash (Solid State Disk)

- Unlike magnetic disks, there's no mechanical part
 - Disks have motors that rotate disks & arms that move and read.
- Efficient I/O
 - Less than 1 Watt consumption
 - Magnetic disks over 10 Watt
- Fast random reads
 - $\ll 1$ ms
 - Up to 175 times faster than random reads on magnetic disks

CSE 486/586, Spring 2014

23

Flash (Solid State Disk)

- The smallest unit of operation (read/write) is a **page**
 - Typically 4KB
 - Initially all 1
 - A write involves setting some bits to 0
 - **A write is fundamentally constrained.**
- **Individual bits cannot be reset to 1.**
 - Requires an **erasure operation** that resets all bits to 1.
 - This erasure is done over a large block (e.g., 128KB), i.e., over multiple pages together.
 - Typical latency: 1.5 ms
- **Blocks wear out for each erasure.**
 - 100K cycles or 10K cycles depending on the technology.

CSE 486/586, Spring 2014

24

Flash (Solid State Disk)

- Early design limitations
 - Slow write: a write to a random 4 KB page → the entire 128 KB erase block to be erased and rewritten → write performance suffers
 - Uneven wear: imbalanced writes result in uneven wear across the device
- Any idea to solve this?

CSE 486/586, Spring 2014

25

Flash (Solid State Disk)

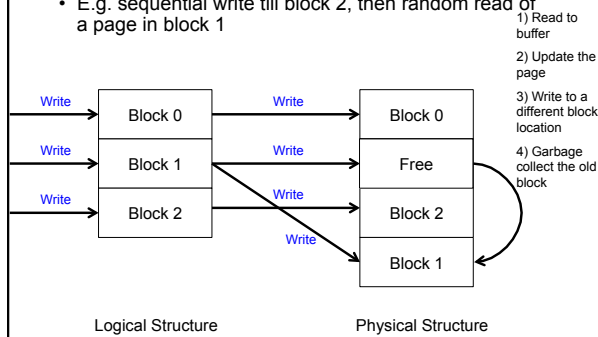
- Recent designs: log-based
- The disk exposes a logical structure of pages & blocks (called *Flash Translation Layer*).
 - Internally maintains remapping of blocks.
- For rewrite of a random 4KB page:
 - Read the surrounding entire 128KB erasure block into the disk's internal buffer
 - Update the 4KB page in the disk's internal buffer
 - Write the entire block to a **new or previously erased physical block**
 - Additionally, carefully choose this new physical block to minimize uneven wear

CSE 486/586, Spring 2014

26

Flash (Solid State Disk)

- E.g. sequential write till block 2, then random read of a page in block 1



CSE 486/586, Spring 2014

27

Summary

- NSF
 - Caching with write-through policy at close()
 - Stateless server
- One power efficient design: Flash storage

CSE 486/586, Spring 2014

28

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).

CSE 486/586, Spring 2014

29