Distributed Information Processing

8th Lecture

Eom, Hyeonsang (엄현상)
Department of Computer Science & Engineering
Seoul National University

Outline

- Distributed File Systems
  - Introduction
  - Naming
  - Remote Access
  - Fault Tolerance
  - Scalability
  - AFS
- Q&A

Storage Systems

File Systems: Systems to Manage Files w/ Facilities for Creating, Naming, and Deleting Files

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
Distributed shared memory	✓	×	✓	✓	Ivy
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server

Types of consistency:
1: strict one-copy. ✓: slightly weaker guarantees.

Distributed File Systems (DFSes)

- Definition [Levy90]
  - File System, Whose Clients, Servers, and Storage Devices Are Dispersed among the Machines of a Distributed System
- Implementation
  - Part of a Distributed OS
    - E.g., Sprite
  - Software Layer to Manage Communication between Conventional OS and File Systems
    - E.g., Andrew File System (AFS)

Transparency Requirements

- Access Transparency
  - Multiplicity & Dispersion of Servers and Storage Devices
  - Network
- Location Transparency
  - Link between the Name and Location: e.g., NFS
    - C.f., Location Independence (Dynamic Mapping): e.g., AFS
- Mobility Transparency
  - File Mobility
- Performance Transparency
  - Comparability
- Scaling Transparency

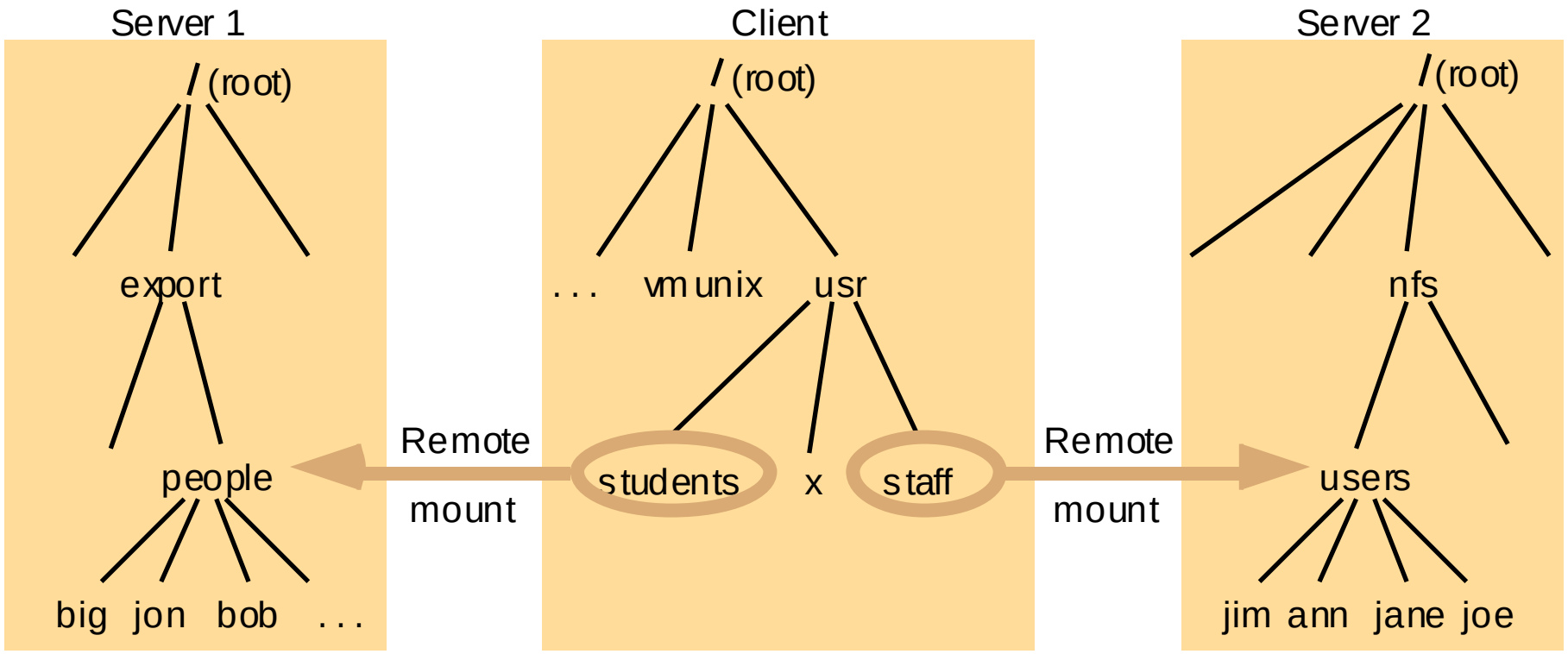
Naming Schemes

Approach	File Names	File Name Examples	Location Transparency/ Independence	Example
Simple	Some Combination of Their Host Name and Local Name	host:local-name	X/X	
Directory-Mount (Attaching Remote Directories to Their Local Name Spaces)	Some Combination of Their Remote and Local Name	combined-name	O/X	NFS
Global	Structured ID	structured-name	O/O	AFS, Sprite

High Administrative Complexity
- Machine Failure or Detachment Mean That Some Arbitrary Directories on Different Machines Become Unavailable
- File Migration Requires Changes in the Name Spaces of All the Affected Machines

Naming Implementation (Cont'd)

- Illustration: NFS Naming



Note: The file system mounted at *//usr/students* **in the client is actually the sub-tree located at *//export/people*** **in Server 1; the file system mounted at *//usr/staff*** **in the client is actually the sub-tree located at *//nfs/users*** **in Server 2.**

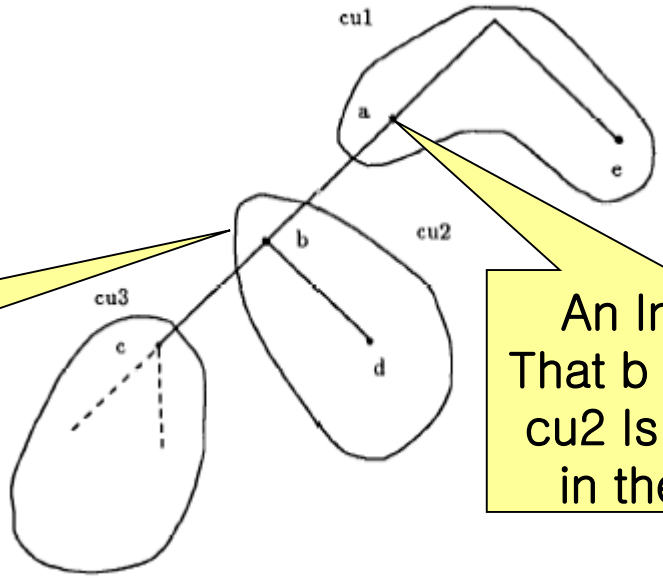
Naming Implementation [Levy90]

Recursive Lookup Procedure for /a/b/c

Smallest Set of Files That Can Be Stored on a Single Machine

component unit	server
cu1	machine1
cu2	machine2
cu3	machine3

Location Table



/b/c Is Passed to Machine2 Machine3 Is Eventually Contacted, and the Low-Level ID of /a/b/c Is Returned

An Indication That b Belongs to cu2 Is Recorded in the b Entry

- Pathname Traversal
  - Server Responding with the Low-Level ID
    - E.g., NFS
  - Client Searching the Dir That Server Replied
    - E.g., AFS

Naming Implementation (Cont'd)

- Structured IDs
  - Aggregating Files into Component Units
    - Bit strings w/ the first part identifying the component unit and the second identifying the particular file within the unit
- Illustration: Structured IDs
  - /a/b/c Translated to the Structured, Low-Level ID <cu3, 11>
  - This Correspondence Not Invalidated Once cu3 Is Migrated to machine2, Requiring Only the Location Table to Be Updated
- Hints: Cached Location Information
- Mount Mechanism

Sharing Semantics

- UNIX (One-Copy Update)
- Session
  - Writes Immediately Visible to Local Clients
  - Writes Invisible to Remote Clients
  - Once Closed, the Changes Visible Only in Later Starting Sessions
- Immutable Shared Files
- Transaction-Like (a Session as a Transaction)
  - Executing File Sessions in Some Serial Order
    - By locking a file during a session

A Series of Accesses by a Client to the Same File, Enclosed between the Open and Close Operations

Remote Access

- Remote Service
  - Requests for Accesses Delivered to the Server
- Caching
  - Accesses Performed on the Cached Copy in the Client Side
    - A Copy of Remotely Present Data Brought
- Caching in a DFS vs Caching in Conventional FS
- Hybrid Method Often Used

Caching Schemes

- Cache Unit Size
  - Advantages of a Large Caching Unit
    - High hit ratio
    - Reduced network overhead
  - Disadvantages of a Large Caching Unit
    - Large transfer time
    - Potential consistency problems
- Parameters wrt the Selection of the Size
  - Network Transfer Unit
  - RPC (Remote Procedure Call) Service Unit
  - Block

E.g., 64KB in AFS

In Block-Caching Schemes

Caching Schemes (Cont'd)

- Location
  - Advantages of Disk Caches
    - Reliability
  - Advantages of Main Memory Caches
    - Reduced access time
    - Diskless
    - Using server caches (to speed up disk I/O)
- Modification
  - Flushing Dirty Blocks Back to the Server's Master Copy

The Trend Is Larger and Cheaper Memories

Blocks That Have Been Modified by the Clients

Sending Dirty Blocks to Be Written on the Master Copy

Caching Schemes (Cont'd)

- Modification Policies
  - Write-Through
    - Writing data to the server's disk as soon as it is written to any cache
  - Delayed-Write
    - Delaying updates to the master copy
      - When the block is about to be ejected from the cache
      - At regular intervals
      - When the file is closed (write-on-close)
- Quizzes
  - Which Policy Is Suitable for UNIX Semantics?
  - Which Is Suitable for Session Semantics?

High Reliability, but Poor Performance

Resulting in the Lowest Miss Ratio in a Study

Caching Schemes (Cont'd)

- Cache Validation — Verifying the Validity of Cached Data
  - Client-Initiated Approach — Disadv.: Longer Access & Greater Server Load
    - Validity-check frequency: every access to first
  - Server-Initiated Approach
    - Server's Information on client caches
- Relating Remote Access Methods to Sharing Semantics
  - Caching the Entire File Matches Session Semantics
  - Remote Service Matches UNIX Semantics

Caching Schemes (Cont'd)

- Caching vs Remote Service

Aspect	Caching	Remote Service
Performance Transparency	○	
Low Network Overhead	○	
Optimized Disk Access	○	
Cache Consistency Problem		○
Implementing Any Centralized Sharing Semantics		○
Clients w/o Disks or Large Main Memories		○
Similarity of Intermachine Interface to Local File System Interface		○

For Frequent Writes

Fault Tolerance

- Stateful Service

  Adv: High Performance Due to the Main-Memory Info Kept by the Server on Its Clients

  - Service by a Server Holding on to Info on Its Clients between Servicing Their Requests
    - E.g., connection ID for the client and open file
  - Characterized by the a Virtual Circuit
- Stateless Service

  Adv: Robustness; Disadv: Longer Request Messages & Slower Processing

  - Service by a Server without Any State Info by Making Each Request Self-Contained
    - E.g., file info in full (including the file ID & position)
  - Characterized by No Need for a Session

Scalability

- Decentralization
  - Clustering
    - Associating a server w/ a fixed set of clients and a fixed set of files they access frequently
      - E.g., AFS (w/ read-only replication of key files)
- Serving Many Clients Simultaneously
  - Light Weight Processes (Threads)
    - Assigning a thread for each client w/ threads sharing information extensively making context switches inexpensive
      - E.g., user-level threads in AFS, kernel threads in Mach

Summary: NFS vs AFS [Levy90]

	NFS	Andrew
Other fault tolerance issues	Complete stateless service. Idempotent operations.	Not dealt with fully yet. Stateful service.
Scalability issues	Not intended for very large-scale systems.	Reducing server load and clustering are the main strategy. Replicated location database might be a problem.
Implementation strategy, architecture	Three layers: UNIX system call interface, VFS interface to separate file system implementation from operations, and NFS layer. Independent specifications for mount and NFS protocols. The Current implementation is kernel based.	Augmenting UNIX kernel with user-level processes: Venus at each client, and a single server process on each server using nonpreemptable LWPs. Structured, low-level, location-independent file identifiers are used.
Networking	LAN	Cluster structure, with a router per cluster. All communication is based on high bandwidth LAN technology.
Communication protocol	RPC and XDR on top of UDP/IP (unreliable datagram).	RPC on top of datagram protocol. Whole file transfer as a side effect.
Special features	Stateless service.	Authentication and encryption built into the communication protocol. Access list mechanism for protection. Limited read-only replication.
Main advantage	Fault tolerance, because of stateless protocol. Implementation-independent protocols, ideal for heterogeneous environment.	Ability to scale up gracefully. Clear and simple consistency semantics.
Main disadvantage	Unclear semantics. Performance improvements obscure clean design.	Fault tolerance issues due to maintained state.

Illustration: NFS Architecture