

# Distributed File System

Presented By:  
Amir Khella

## Introduction and Definitions

- Goal: allow users of physically distributed computers to share data and storage resources by using a common file system.
- Service: software entity running on one or more machines and providing a particular type of function to a priori unknown clients
- Server: service software running on a single machine
- Client: a process that can invoke a service using a set of operations that form the client interface.

## Introduction (cont'd)

- Different configurations of DFS:
  - Servers run on dedicated machines
  - A machine can be both a client and a server
- Distinctive features are the multiplicity and autonomy of clients and servers in the system

## Presentation layout

- Terminologies
- Naming and transparency
- Semantics of sharing
- Remote access methods
- Fault tolerance
- Scalability
- Example Systems: Sun NFS and Andrew (AFS)

## Trends and terminologies

- Multiplicity and dispersion of servers should be transparent to clients
  - Network transparency: Client should be able to access remote files using the same set of file operations applicable to local files
  - User mobility: User can log in to any machine in the system
- Performance measure: the amount of time needed to satisfy a service request
- Fault tolerance: Communication faults, storage device crashes, decays of storage medias,...
- Scalability: capability of the system to adapt to increased service load.

## Naming and transparency

- Naming: mapping between logical and physical objects
- In transparent DFS, the system should hide where in the network the file is located
- Naming and transparency involves:
  - Location transparency and independence
  - Naming schemes
  - Implementation techniques

## Location Transparency and Independence

- Location transparency: The name of the file should not reveal any hints as to its physical storage location
- Location independence: The name of the file need not to be changed when the file's physical location changes (also known as file migration or file mobility)
- Aspects that differentiate and contrast the two above concepts:
  - Divorcing data from location is exhibited by location independence
  - Sharing data is provided by location transparency.
  - Location independence separates the naming hierarchy from the storage devices hierarchies and the inter-server structure

## Naming schemes

- Three main approaches:
  - Files are named by combining their host name and local name
    - Unique system-wide names
    - Network transparent (same file name is used for local and remote files)
    - Neither location independent or location transparent
  - Machines attach (mount) remote directories to their local name space
    - Shared name space may not be identical on all the machines

## Naming schemes (cont'd)

- Single global name structure that spans all the files in the system
  - The composed file system structure should be isomorphic to the structure of a conventional file system
  - Many special files can make the ideal goal difficult to attain (e.g. I/O devices in Unix are treated as files)

## Implementation Schemes

- Pathname translation
  - Mapping of textual names to low-level identifiers is typically done by a recursive lookup procedure based on the one used in conventional Unix
- Structure identifiers
  - Aggregating sets of files into component units and providing the mapping on a component unit basis rather than a single file basis
  - Bits of strings that usually have two parts.
    - Component unit to which file belongs
    - File identifier within the unit
  - These Identifiers are location independent and hence can be replicated and cached freely without being invalidated by migration of component units

## Implementation Schemes (cont'd)

- Hints
  - A piece of information that speeds up performance if it is correct and does not cause any semantic negative effect if it is incorrect
- Mount mechanism
  - Joining remote file systems to create a global naming structure
  - A mount operation binds the root of one system to a directory of another system
  - The directory that glues the two file systems together is called a mount point
  - Mount operations are recorded by the operating system kernel in a mount table

## Semantics of Sharing

- File session: a series of file accesses attempted by a client to the same file are always included between an the Open and Close operations
  - Unix semantics
  - Session semantics
  - Immutable shared files semantics
  - Transaction-like semantics

## Unix Semantics

- Every Read of file sees the effects of all previous Writes performed on that file.
- Writes to an open file by a client are visible immediately by other clients who have this file open at the same time
- It is possible for clients to share the pointer of the current location into the file. Advancing the pointer by one client affects all sharing clients

## Session semantics

- Writes to an open file are visible immediately to local clients but are visible to the same file open simultaneously
- Once a file is closed, the changes made to it are visible only in later starting sessions.
- Already open instances of the file do not reflect these changes

## Immutable shared files semantics

- Once a file is declared as shared by its creator, it cannot be modified any more.
- An immutable file has two important properties:
  - Its name may not be reused
  - Its contents may not be altered

## Transaction like semantics

- The effect of file sessions on a file and their output are equivalent to the effect and output of executing the same sessions in some serial order.
- Locking a file for the duration of a session implements these semantics

## Remote access methods

- Two complementary methods to handle data transfer in a request from a client to a remote file on a server:
- Remote service:
  - Requests for accesses are delivered to the server which performs these accesses and returns their results back to the client.
- Caching:
  - Requests for remote access brings a local copy of data blocks to the client side.
  - Usually the amount of data brought is much more than the data needed.
  - Caching works best when the stream of files exhibits locality of reference
  - One problem arises: Cache consistency

## Designing a caching scheme

- Should address the following decisions:
  - Granularity of cached data
  - The location of the client's cache
    - Main memory vs. local disk
  - How to propagate modifications
  - How to determine if the client's version of the cache is consistent

## Cache Unit Size (Granularity)

- Increasing the cache unit increases the likelihood that data for the next access will be found locally
- But the time to transfer data and consistency become a problem
  - Network transfer unit is usually small
  - Parameters such as network transfer unit and RPC protocol service unit should be considered
- Look-Ahead is used in block-caching techniques

## Cache location

- Disk cache:
  - Reliable (in case of crash)
  - Autonomy of single machines
- Main memory cache:
  - Workstations without disks
  - Fast
  - Server caches will be in main memory regardless of where clients caches are located

## Modification policy

- Simplest policy: Write-through
  - Write data through the server's disk as soon as it is written to any cache
    - Reliable
    - Poor write performance
- Delayed updates
  - Write performance is better
  - Data may be deleted before they are written back
  - One policy is to write a block when it is about to be ejected from the cache
- Write on close

## Cache validation

- The client's decision of whether or not its locally cached copy is consistent with the master copy
- Two approaches:
  - Client initiated approach
    - How frequent validity checks should be sent?
  - Server initiated approach
    - If session semantics are implemented, validation on close
    - Violates the traditional client server model
      - Irregular and complex code

## Cache consistency

- Session semantics are perfect match for caching entire file
- Distributed conflict resolution schemes must be implemented
- With immutable files, cache consistency problem does not exist
- Transaction-like semantics can be implemented using locking

## Caching and remote service

- Performance vs. simplicity
- When caching is used, a substantial amount of the remote accesses can be handled locally
  - Server load and traffic are reduced
  - Potential for scalability is enhanced
- Total overhead of transmitting big chunks of data is lower than series of smaller size requests

## Caching and remote service (cont'd)

- Cache consistency problem is the major drawback
- Hard to emulate the sharing semantics of a centralized system in a system using cache as its remote access
- In caching, clients have to have either local disks or large main memory

## Fault tolerance

- Stateful and stateless service
- Improving availability
- File replication

## Stateful vs. stateless service

- Stateful service: Server holds on to the information on its clients between servicing their requests
  - Characterized by a virtual circuit between client and server during a session
  - High performance (file information are cached)
  - Loses all its volatile state in a crash
  - Server should become aware of clients crashing
- Stateless service: Server does not maintain any information on a client once it finished servicing its request
  - Each request is self contained, nothing kept in the server's main memory
  - Effects of server crashes and recovery is not noticeable
  - Longer request messages
  - Slower processing of requests

## Improving availability

- A file is **recoverable** if it is possible to revert it to an earlier, consistent state when an operation on the file fails or is aborted by the client
  - Realized by atomic updates techniques
- A file is **robust** if it is guaranteed to survive crashes of the storage devices and decays in the storage medium
  - May not be available until the faulty component has recovered.
- A file is **available** if it can be accessed whenever it is needed despite machine and storage devices crashes and communication failures

## Improving availability (cont'd)

- File replication improves their availability
  - Some principles have to be taken in consideration in order to insure increased availability
    - The number of machines involved should be minimal
      - Locating is usually done by pathname traversal which in a DFS may cross machine boundaries several times
      - Caching directory information can speed up pathname traversals and avoid the problem of unavailable directories in pathname

## File replication

- Useful redundancy for improving availability
- Multi-machine replication improves performance
- Different replicas of the same file reside on failure-independent machines
- Details of replication must be hidden from users
- Problem: Updates!

## File replication (cont'd)

- Relevant sharing semantics must be preserved when accesses to replicas are viewed as virtual accesses to their logical files.
- If consistency is not an issue, it can be sacrificed for availability and performance
- The choice of consistency might introduce the problem of indefinite blocking
- Updates should be atomic
  - Using a commit protocol (two-phase commit)
  - Can lead to indefinite blocking in case of machine or network failure.

## File replication (cont'd)

- One case consistency can be traded for performance is the replication of location hints
  - Since they are validated upon user, replication does not require maintaining their consistency
- Popular compromise: Read-only replica
  - Files known to be frequently read and rarely modified.
    - Files containing the code of system programs, system data files,...

## Scalability

- Principles:
  - Bounded resources:
    - Service demand from any component of the system should be bounded of a constant which is independent of the number of nodes in the system.
    - The capacity of the server limits the growth of the system
  - A mechanism that relies on broadcasting is not realistic for large-scale systems.
    - Broadcasting is an activity that involves every machine in the system
  - If stateless service is used, a server need not detect a client's crash nor take into precautions because of it.
    - Garbage collection can be used : clients set and reset expiration dates

## Scalability (cont'd)

- Detecting crashes can be very expensive since it is based on polling and time-out mechanisms.
- Network congestion and latency are major obstacles to large-scale systems
- Central control schemes and central resources should not be used to build scalable systems
  - A configuration should be functionally symmetric
  - Administration should be delegated

## Scalability (cont'd)

- A practical approach is clustering
  - System is partitioned into a collection of semi-autonomous clusters
    - A cluster is a set of machines and a dedicated server
  - Clustering complies with bounded resources principles
- Another approach:
  - Lightweight processes

## Lightweight processes

- Servers are supposed to operate efficiently in peak periods
- A single server process is not a good choice
  - A better choice: assigning a process to each client
    - An overhead is multiplexing the CPU among the processes
- Solution: LWPs
  - A thread is a process that have very little non-shared state
  - The abstraction is multiple threads of control associated with some shared resources
  - Extensive sharing makes context switching among peer threads and thread creations inexpensive

## Lightweight processes (cont'd)

- Clients requests accumulate in a common queue and threads are assigned to requests from that queue
- Advantages:
  - I/O request delays a single thread not the entire service
  - Sharing common data structures among the threads is easy.

## Sun NFS

- S/W system for accessing remote files across LANs
- Part of the SunOS operating system
- Set of independent machines with independent file systems
- Sharing is based on client-server relationship
- A machine can be both a client and a server
- No notion of a globally shared file system
  - Sharing only affects the client machine
- Mount operations used for remote accessibility
  - Non-transparent
- Independence of media and heterogeneous environment is achieved through RPC

## NFS services

- Two separate protocols:
  - Mount protocol
    - For mounting files and directories
  - NFS protocol
    - For remote file access
- Protocols are specified as set of RPCs

## Mount protocol

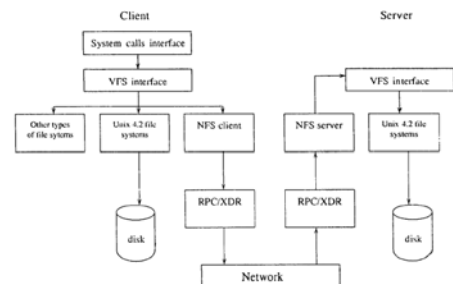


- Cascading mounts are also permitted
- Mount is not transitive
- Mount protocol used to establish initial connection
- Server maintains an export list
- File handles are used for mounting
  - File system identifier + i-node number
- Another list for the client machines and the corresponding mounted directories for admin purposes

## NFS protocol

- Set of RPCs for remote file operations
- No Open or Close operations
  - Servers are stateless
  - Clients list should be treated only as a hint
- Modified data must be committed to the server's disk before the call returns results to the client
- No concurrency control mechanism
  - Users coordinate access outside NFS mechanism

## Implementation



## Architecture

- User interface is Unix file system calls
- VFS layer
  - Separating file system generic operations from implementation
  - Based on file implementation structure called a vnode which contains numerical designator for a file that is network-wide unique
  - Distinguishes local files from remote ones

## Pathname translation

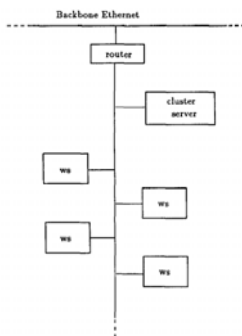
- Breaking the path into component names and doing NFS lookup for every pair of components name and directory vnode
- Lookups are performed remotely by the server
- Directory name lookup cache at the client holds the vnodes for remote directory names
- A server cannot act as intermediary between a client and another server

## Caching and consistency

- One-to-one correspondence between the regular Unix system calls for file operations and the NFS protocol RPCs
- Buffering and caching used for performance
- Two caches:
  - File blocks cache
  - File attributes (i-node information) cache
- Read ahead and delayed-write techniques are used
- Caching unit is fairly large (8kb)

## Andrew

- Development started at CMU in 1983
- Distinguishes between client machines and server machines
- Clients are presented of partitioned space of file names
  - Local name space
  - Shared name space
- Vice:
  - Collection of dedicated servers presenting the shared name space to the clients as identical and location-transparent file hierarchy
- Workstations have local disks
- Clients and servers are structured into clusters
  - Collection of workstations and a representative vice called a cluster server



## Overview (cont'd)

- Server's CPU is the system's bottleneck
  - Heuristic is to off-load work from servers to the clients
  - Whole file caching
    - Cannot accommodate efficiently access for large files
    - User mobility
    - Security
      - Vice interface is the boundary for trustworthiness
      - Authentication and secure transmission functions are part of the communication package
    - Protection
      - Access lists based on recursive group structures
    - Heterogeneity

## Shared name space

- Component units called volumes
  - Are associated with the files of a single user
  - Joined together by a similar process to the mount mechanism
- A vice file or directory is identified by a low-level identifier called fid
  - Fid has equal length components: volume number, vnode number and unifier
  - Fid are location independent
- Volumes need to migrate among disk partitions and servers
- Volume movement operation is atomic
- Read-only replication at the granularity of an entire volume is supported

## File operations and sharing semantics

- No remote interaction is caused by reading or writing files
  - OS on each workstation intercepts file system calls and forwards them to a user level process on that workstation
  - Venus process is used to cache files from vice when they are opened and stores modified copies back on vice when they are closed
  - Venus contacts vice only on open and close
  - Assumes that cached entries are valid unless otherwise notified
  - Callback mechanism is used

## Cont'd

- Andrew implements session semantics for Read and Write operations
- Callback mechanism forces every server to maintain callback information on every client to maintain validity
- Venus also caches contents of directories and symbolic links for pathname translation

## Implementation

- Information in the cache is treated as a hint
- Unix file system is used as low level storage system for both servers and clients
- Venus manages two separate caches
  - Status
  - Data
  - LRU algorithm is used
- Whole file transfer is implemented as side effect of RPC call (one RPC connection per client)