Distributed File Systems: Concepts and Examples

by E. Levy and A. Silberschatz (Computing Surveys, 1990)

- Physically distributed computers connected via LAN
- Common file system
- Distributed implementation of a conventional file system
- Client-server architecture
- Software-hardware component
- File operations: create, delete, read, write
- Part of a distributed OS or a networked OS
- Case studies

Issues...

◇ Network transparency

- User mobility
- Performance (overheads)

◇ Fault tolerance

◇ Scalability (high load, user growth, added resources)

Naming and Transparency

- Naming: mapping of logical data to physical blocks of data
- Location transparency
- Location independence
  - stronger
  - dynamic mapping
  - migration/mobility
- Better abstraction
- Share data conveniently

- Naming schemes
  - name is a combination of host name and local name (no transparency)
  - "mount" (NFS): attach remote directories to local name spaces
  - global name structure
  - issue: administrative complexity
  - NSF is the most complex: failures, migration, which machine is allowed to attach which directory to its name space

Naming Implementation Techniques

- Pathname translation
  - recursive lookup
- Structured Identifiers
  - location independent
- Hints
  - location mapping
  - cached location information
- Mount Mechanism
  - joining remote systems
  - mount point: glues together two file systems
  - mount table: redirect name lookups

Semantics of Sharing

- Important since multiple clients share the same file simultaneously
- When data mods are observable by remote clients
- Reads/writes/consistency during a file session
- Application driven (depending on what is available)

- Unix semantics
  - Read sees the effect of all previous writes
  - Writes are visible immediately to those clients (*remote* too who have the file open
  - Share the file pointer across many clients
  - Single file image (delays may occur)
- Session semantics
  - Writes are immediately visible to *local* clients only
  - Already open changes are visible remotely after the file is closed
  - Application must enforce sequential consistency
- Immutable shared file semantics: read only
- Transaction-like semantics
  - locks, so that file session is like a transaction

Remote Access Methods

- Two ways for data transfer
- Remote service
  - via messages
  - network traffic
- Caching
  - best, when locality of reference
  - replacement policy
  - consistency issues
- Hybrid schemes

Implementing Caching

- Granularity of cached data
- Cache location (main memory,local disk?)
- How to propagate mods of cached copies
- How to determine if a client's cached data are consistent

- Cache unit size
  - large is good
  - whole files, blocks
  - performance issues: network
  - *read-ahead*
  - *write-behind*
- Cache location
  - memory *vs* disk
  - advantages/disadvantages

- Modification Policy
  - dirty blocks and flushing of dirty blocks
  - *write-through* to local disk
  - *delayed-write* (may introduce reliability problems)
  - *write-on-close*
  - close relation with semantics sharing
- Cache validation
  - determine if valid
  - client-initiated approach
  - server-initiated approach

Cache consistency

- Cache entire files in sessions
  - simple implementation
- Arbitrate among clients who want to write simultaneously on the same file
  - network traffic
  - expensive (think of back propagation too)
  - Sprite solution: disable caching when file opened by multiple clients in conflicting mode
- Immutable shared file semantics: no problem
- Transaction type semantics: implementation w/ locks

Advantages/Disadvantages of Caching and Remote Service

- Locality of accesses makes caching more attractive
- Remote service becomes expensive (network)
- Transferring big versus small chunks of data
- Disk accesses more efficient if access pattern is known
- Cache consistency: major caching drawback (e.g., frequent writes)
- Caching requires large memories or local disks

Fault Tolerance

- Stateful service: server maintains info on its clients between servicing their requests
  - "Virtual circuit" between client and server
  - Performance
  - Key point: main-memory info kept by the server on its clients
- Stateless service: server does not maintain any info once it finished servicing the request
  - each request identifies the file and position in the file
  - no need to establish and terminate a connection
- Crash?
  - crash at the client side, crash at the server side
  - stateful loses all info, recovery protocol
  - stateless avoids loss of info (by definition)

- Availability
  - available when needed
  - temporal
  - enhanced by replication
  - identifying the server and establishing connection
  - pathname traversal issues (if path is available)
- File replication
  - redundancy
  - think of mirrored disks (and RAIDS)
  - problem with replicas: update
  - preserve consistency
  - some times to preserve atomicity you may end up with infinite blocking
  - *read-only replications* : compromise (those files that are frequently read and rarely modified)

- Scalability
  - Serious issue
  - Andrew file system (most ambitious)
  - Problems
    - network traffic and latency
    - broadcasting
    - growth limited by server capacity
    - detecting crashes (stateful and stateless)
    - trade-off of semantics and network and system load
    - centralized versus clustered designs
  - Lightweight processes
    - group of threads associated with shared resources
    - easy switching
    - essential for servers to be scalable

Case Studies

- UNIX united
- LOCUS
- SUN NFS
- Andrew
- Sprite

UNIX United

- Extends UNIX into a distributed one without modifying kernel
- Software subsystem is added to each of components
- Network transparency
- Naming structures of the component are joined together (and each component is viewed as just a directory)
- Roots of components are assigned a name
- Single virtual root parent
- Relative pathnames
- Logical name space must be mapped onto routing information

- Implementation:Newcastle Connection
- User-level software layer
- User-level programs, connection layer, Unix kernel
- Connection layers through RPCs
- Logical name structure
- Recursive tree structure
- Simple implementation, that provides network transparency without location transparency

LOCUS (UCLA)

- Design requires new kernel (mods are not enough)
- File replication (to increase reading availability)
- Automatic mechanisms to update stale copies of files
- UNIX semantics, file locks, and atomic updates
- Incorporates networking functions (for performance)
- Remote operation protocols for kernel-to-kernel communications
- Name structure
  - disguises both location and replication details
  - replicas at various sites
  - *primary copy* of files – stores complete file
  - other copies may be partial

- File operations
  - *using site*: issues remote requests
  - *storage site*: serves requests
  - *current sync site*: enforces global sync policy
- Opens, Reads, Writes, Closes: are a sequence of the above file operations...
- Unix file system semantics (atomicity)
  - token schemes for synchronization
  - token schemes for cached data
  - contrary to UNIX, may lock parts of files or whole files
- Fault tolerance: reconciliation of multiple copies
  - primary copy always up to date
  - CSS and SS strong cooperation
- Scalability problems though...

SUN NFS

- Across LANs, built on top of SunOS
- View machines as independent, but allows a degree of sharing among them
- No notion of a globally shared file system
- “mount” operations to make a directory transparent
- Design for heterogeneous environment, achieved through RPCs
- Two separate protocols
  - mount protocol
  - NFS protocol (for remote accesses)

- Mount Protocol
  - establishes initial connection between server and client
  - part of the tree that is exported, and the machines that can access it
  - no transitivity
  - cascading mounts
- NFS Protocol
  - RPCs for remote file operations
  - file lookup
  - reads, writes
  - file attributes
  - links and directories
  - stateless philosophy: no concurrency control
  - one-to-one correspondence between regular UNIX system calls and NSF RPCs

- Caching and consistency
  - buffering and caching for performance
  - cached attributes are discarded after a time-out
- In general
  - logical name structure, no global name hierarchy
  - network service, not part of a DOS
  - remote service (in the sense of individual RPCs)
  - stateless, provides fault tolerance
  - sharing semantics: not like UNIX, not clear, since timing dependent

SPRITE (Berkeley)

- Assumes network of workstations with large physical memories
- Caching: key for high performance
- UNIX-like interface
- Location transparency
- Unix semantics
- Unique feature: interplay with virtual memory system
- Uses files to store data and stacks of running processes! (*backing files*)
- Simplifies process migration
- Then, caches *backing files*

- Single file system hierarchy
  - subtrees (*domains*)
  - *prefix table* maps domains to servers on each machine
  - diffuse information via broadcasts
  - dynamic reconfiguration and robustness
  - can keep some files "private"
- Caching and consistency
  - no sequential prefetching for reads
  - but delayed-write policies
  - inspired by diskless workstations
  - server crash
  - but high performance
- Sharing semantics: UNIX like
- Prefix tables: versatile and robust

ANDREW (CMU)

- Very scalable
- Client machines *vs* server machines
- *local name space* and *shared name space* available to clients
- Local disks on workstations
- Clients and servers are structured in *clusters* – hierarchy
- Move work to clients
- Whole file caching
- User mobility (any file in the shared space)
- Security (encryption, authentication)
- Protection (access lists)
- Heterogeneity

- Name space and service model: distinguishes among local and shared name spaces, as well as clients and dedicated servers
- Scalability: whole file caching to reduce server load.
  Penalty: maintain a lot of state data on the servers
- Sharing semantics: simple and well defined
  not undefined like NFS
  not UNIX semantics
  ensure that file updates are visible only after the file is closed
- Component Units and location mapping: complete and replicated mapping at each server