File System for Mobile Computing

- Quick overview of AFS
- Identify the issues for file system design in wireless and mobile environments
- Design Options for mobile file system
- Coda File System

Sharing Semantics

- Unix Semantics (Read after write): every operation on a file is instantly visible to all processes.
- Session semantics: a process that intends to write does so to its own copy. After the process closes the file, the changes are then made visible to other processes.
- Close-to-open cache consistency by NFS: clients flush all changes back to the server on each file close, and check for file changes on the server on each file open.

Commonly-used mechanisms and techniques in distributed file systems

- Caching at clients
  - Exploit temporal locality of reference
  - Key issue: the size of cached units. Options: individual pages of files; whole files, large block
  - Where to maintain? In main memory; on disk
  - Cache validation: client to contact server; server notify clients when cached data is rendered stale
  - Spatial locality of a file: read-ahead of file data
- Transferring data in bulk
  - Amortize fixed protocol overheads over many consecutive pages of a file
  - Depend on spatial locality of reference within files for effectiveness

Mechanisms (cont’d)

- Hints
  - Improve performance if correct; has no semantically negative consequence if wrong
  - Most often used for file location information
- Encryption
  - Prevent unauthorized release and modification of information
- Mount points
  - Glue file name spaces into a single, seamless, hierarchically structured name space
  - Two ways: each client individually mounts subtrees from servers; embed mount information in the data stored in the file servers.

Andrew File System

- Goal: distributed file system for large-scale systems
- General principles:
  - Name space: private client name space, & a globally shared and location independent name space
  - Unit of sharing: volume (a variable file set forming a partial subtree of the name space; the basis of disk quotas)
  - Cache coherence: upon each open, contact server to verify the cache to be up to date (AFS-1)
  - Locating server: volume location database (caches volume <-> server mapping)
  - Pathname traversal: client caches directory; client initiates lookup one at a time; client does lookup

AFS (continued)

- General principle (cont'd)
  - ◻ Availability: each pathname directory must be available
  - ◻ State: callback state
  - ◻ Caching:
    - What is cached: status, directory, whole files
    - Client has 2 caches: status cache (kept in VM) to rapidly service stat. Calls; data cache (kept in local disk)
    - Where is it cached: disk
    - Cache size: fixed (64KB)
    - Modifications propagate on close. Modifications to directory are immediately visible
    - No more cache consistency check on open. Cache is assumed to be correct until the client is informed via callback
    - Callbacks: when a client caches a file, the server promises to notify it upon any changes (inconsistency)
    - Callback problems:
      - – Load at server increases if many clients cache same file
      - – Client and server may be out of sync (server crash and recovery)

ACID properties for transactions

- Atomicity: a transaction either commits or aborts. If a transaction commits, all its effects remain; otherwise, effects are undone.
  - Each transaction appears indivisible w.r.t. crashes
- Consistency: a transaction is a correct transformation of the system state.
  - It preserves the state invariants
- Isolation/Serializability: concurrent transactions are isolated from the updates of other incomplete transactions. These updates do not constitute a consistent state.
  - Transactions appear indivisible to each other
- Durability: once a transaction commits, its effects will persist even if there are system failures.

Mobile File Systems

- Requirements:
  - access the same file as if connected
  - retain the same consistency semantics for shared files as if connected
  - availability and reliability as if connected
  - ACID (atomic/recoverability, consistent, isolated/serializablity, durable) properties for transactions
- Constraints:
  - disconnection and/or partial connection
  - low bandwidth connection
  - variable bandwidth and latency connection
  - connection cost

Mobile File system

- Four major aspects of disconnected or partially connected operations:
  - hoarding: what to pre-fetch
  - consistency: what to keep consistent when connectivity is partial
  - emulation: how to operate when disconnected
  - conflict resolution: how to resolve conflicts

Design Options for Hoarding

- Application hints (Coda): application provides a hoarding database
  - ◻ +: can list all files needed
  - ◻ --: cannot predict accurately in advance
  - ◻ --: do not know all system files used
- Prior run (disconnected AFS): application runs the same program
  - ◻ +: no need to explicitly provide hoard database
  - ◻ +: AFS automatically does caching
  - ◻ --: no single run of application typically brings over all files
  - ◻ --: cannot predict all applications
  - ◻ --: tedious to do "cat filename" for each data file u want

Design Options for Hoarding

- Snapshot spying (updated version of coda): look at a time window and hoard all files used in this time window
  - +: works well along with hoarding database
  - -: still has the single run problem
- Semantic distance (Ficus): measure the correlation between file accesses (e.g. the distance btw. file opens and closes) and cluster files
  - +: based on long term user behavior
  - +: independent of a time window
  - --: cannot relate concurrent access
  - ?: do not know how well it works

Options for Hoarding (Cont'd)

- Application context: create working sets for each application, and load all the files for desired applications
  - +: solves the single run problem
  - +: works well for application/system files
  - --: still cannot predict which applications user will run
  - --: do not work well for data files (e.g. emacs will create a working set of all previous data files unless pruned carefully)

Options for Consistency Management

- Optimistic consistency during disconnection: cannot contact server during disconnection, so assume NO conflict
  - +: allows disconnected operation
  - --: causes potential conflict
- Conservative consistency policy: lock the file before disconnection (or when callback recall fails)
  - +: prevents conflicts
  - --: if a portable caches files and disconnects, backbone users cannot access file (I.e., requires full connectivity among clients currently caching a file)

Consistency Management

- Conservative on shared files, optimistic on private files: monitor file sharing activity at server (how many users perform concurrent read/write sharing of a file), and be conservative for read/write shared files
  - +: reduces conflict while improving access to unshared files
  - --: cannot accurately monitor files, particularly when servers are replicated and network may be partitioned
- Replay: when the communication pipe is thin, replay the actions (commands) rather than keep files consistent
  - +: good for actions that create large files (e.g. gcc)
  - --: significant processing overhead
  - --: almost impossible to keep environments and context fully consistent

Consistency Management

- Block-by-block consistency: when the communication pipe is thin, fetch/update critical blocks on demand
  - +: works well in conjunction with (3), can propagate changes to files which are known to be shared more often
  - --: needs optimistic concurrency as backup
  - --: does not use application semantics
- Application-dependent consistency: application imposes structure on file and specifies which parts of the file need to be kept consistent
  - +: semantic consistency users bandwidth intelligently
  - +: can work for both aware and unaware applications
  - --: complex mechanisms to handle unstructured files
  - --: per-user mechanism, since different applications may potentially use different templates on the same file

Emulation

- Goal: client emulates whatever basic distributed file system is being adapted to support disconnection
- Tasks involved:
  - ◻ Create files
    - Can either create temporary or permanent file ids
    - Can hoard the directory structure in order to reduce potential conflicts
  - ◻ Maintain logs:
    - For session semantics (1), log only file-mutating closes
    - For replay semantics (4), log all operations
    - For block-by-block semantics (5), log all writes
    - For application-dependent semantics (6), log only writes which need to be kept consistent
    - Compress logs by deleting entries upon unlink, overwrites, etc.
  - ◻ Propagate logs:
    - Upon partial or total reconnection, propagate log back to server
    - Can prioritize updates to propagate

Conflict Detection

- Detect write-write conflicts: detect conflict conservatively when the same base version of the file has been updated concurrently during disconnection by both the portable and the backbone
  - +: simple
  - --: inadequate in some cases
- Detect read-write and write-write conflicts: provide serializability
  - +: satisfies a key requirement for transactions
  - --: complex

Conflict detection and resolution

- Application specific conflict detection procedures: application provides the rules to detect conflicts and merge updates when files have been updated concurrently
  - +: works well for structured files
  - --: cumbersome and difficult for unstructured files
- Ownership: owner has the "correct" copy in case of conflict
  - +: simple semantics
  - --: some external resolution still needs to be performed by user who is notified that his/her changes have been discarded

Conflict resolution

- Application specific conflict resolution procedures: application provides the rules to resolve conflicts
  - +: can automate fully
  - --: very difficult to write applications in such an environment without adequate library support
- Multi-level read-write: need to introduce multi-level read/write semantics
  - All reads and writes are provisional until they have been propagated and conflicts have been resolved
  - Notion of provisional vs. committed operations

Main Features for Coda

- Main goals: availability and scalability
- Disconnected operation for mobile computing
- High performance thru client side persistent caching
- Server replication
- Security support
- Continued operation during partial network failures in server networks
- Good scalability
- Application transparent adaptation
- Well defined semantics of sharing, even in the presence of network failures

Trickle Reintegration

- A mechanism that propagates updates to the servers asyn., while minimally impacting foreground activity
- Deferring the update propagation to servers, Conceptually similar to write-back caching
- Make write-disconnect state permanent
- Keep log optimization thru an aging window

Mobile File Systems

- Options:
  - hoard, cache or prefetch part/whole files: what to cache, where to cache, what grain to cache at
  - a variety of consistency semantics: optimistic, conservative, application-dependent, block-by-block,
  - relaxed properties (e.g. only isolation) for transactions
  - reconciliation upon reconnection: when to propagate updates, which updates to propagate given limited/partial connection capability, how to resolve detected conflicts
  - application-level hints or directions for caching/consistency/partial consistency: use of semantics for validation, caching and consistency
  - profiling for hoarding/caching
  - reservation of system resources, and loss profiles to arbitrate between applications during conflict