Ch 9 – Distributed Filesystem

- Reading Assignment – Summary due 10/18 (CODA paper)
- Goals
  - Network Transparency – you don't know where in the system the file is.
  - High Availability – Users should get same filesystem irrespective of location, and maintenance/failure should cause graceful degradation.

- Overall Architecture
  - Client, Fileserver, client cache/local storage, server cache, name server, cache manager.

- Mechanisms
  - Mounting
    - Binding together different file namespaces to form a single hierarchically structured namespace.
  - Caching
    - Hints
  - Bulk transfer
  - Encryption

Design Issues

- Naming and Name Resolution
  - Single Namespace ? Location Dependent ? Location Transparent ?
  - Context as a component of the namespace ?
  - Name server maps names to stored objects
    - Single centralized server ?
    - Distributed ? – the actual resolution may involve several servers
- Caches
  - Disk vs Memory
    - Disk is larger, simpler memory management, portable systems (volatility?)
    - Memory is faster, client and server caching schemes can be similar, allows diskless clients to participate

- Cache Writing Policy – when should changes made by a client committed at the server ?
    - Write through ? Delayed Write ? Write on Close ?
- Cache Consistency
  - Server Initiated – server informs clients of changes, they can then update or invalidate
  - Client Initiated – client side cache checks with server before returning data to client.
  - Both are problematic and expensive.
  - Caching for concurrent write sharing – don’t cache!
  - Caching for sequential write sharing
    - Outdated file on client.
    - Written blocks not yet flushed.

Sprite FS

- The entire namespace is a tree hierarchy, but is spread over disjoint domains. A server has one or more domains. Each client keeps a prefix table, which maps the topmost directory in a domain(prefix) to the domain and the server it is on.
  - Prefx tables are created dynamically using broadcast, and entries treated as hints.
- To lookup a name, the client finds the longest matching prefix in the table, and sends the remaining path along with the domain token to the corresponding server. This server returns a file token(handle) which is used subsequently.
  - CWD's prefix entry is a part of the state, to make relative filename access faster.
  - What if name sent to remote server again crosses boundary, e.g. on .. or symlinks.

- Caching is done on both client and server side.
- Client Cache stores recently accessed blocks indexed by file token
  - Don’t need to map to disk block, less communication with server.
  - Cache block size is 4K
  - Directories are not cached to avoid inconsistency
- Server Cache also stores file blocks.
- When client requests data, client cache is first checked. Otherwise, request passed to local FS or remote server. Remote Server checks server cache or passes request onto its local FS. No prefetching.
- Writes are delayed. Data unchaged for 30 secs or on cache miss (LRU replacement) is written
  - 20-30% data deleted within 30 secs, 75% files open for < 0.5 sec, 90% for < 10 seconds.
  - No write on close.
  - Cache misses 40% read req, 1% write req. Mostly for large files.

- Cache Consistency is server initiated.
- Concurrent write sharing is avoided
  - When a concurrent write request is received, server instructs current writer to flush data, commits it, and then declares the file uncachable to *all* clients.
  - All requests now flow through the server, which will serialize them.
  - About 1% traffic overhead
- Sequential Write Sharing is overcome by using versions, version number is incremented upon write . Server also keeps track of last client doing write, and requests that client to flush data when it gets the next file open from some other client.

- Virtual Memory and FS cache compete for memory!
  - Cache size is changed in response to memory usage.
  - Cache and VM keep separate pool of free pages, and track time of last access for each block. This is used in negotiation between the systems. Conversion from VM to cache is made difficult by “holding down”.
    - Double caching of executable programs!
- Backing store for VM is the regular filesystem.
  - Process migration is helped
  - Double Caching – avoid by not using client cache for such files.

CODA

- Goals
  - Scalability
    - Clients are loaded to avoid FS bottlenecks.
  - Constant Availability
  - Use with Portable computers
    - Clients local disk is a cache
    - Disconnected operation permitted.
- Naming/Location
  - Namespace is hierarchical and divided into volumes. Volume is set of files/directories on a server
  - Each volume is identified by a unique 92 bit FID
    - Vol. No. + Vnode No. + uniquifier
  - Location information is not explicit in name but stored in volume location database mapping file to FID to location

Operational Semantics

- Single Copy Unix
- AFS-1
  - Open succeeds if latest (F,S)
  - Close succeeds if updated(F,S)
- AFS-2
  - Open succeeds if
    latest(F,S,0) OR (latest(F,S,tau) AND lostcallback(S,tau) AND incache(F).
- CODA
  - Open succeeds if
    If some server reachable then AFS2 else if incache
  - Open fails if servers reachable and conflict or not reachable and not in cache
  - Analogously for close

Replication

- When a volume is created, the number of replicas and the servers storing them are recorded in volume replication database. This set of servers is the Volume Storage Group (VSG). The subset of these accessible by a clients cache system (VENUS) is called Accessible VSG(AVSG)
- Read Once Write All type approach
  - Files are cached on client side on demand. They are obtained from a preferred server in the AVSG. Client verifies with others in AVSG to ensure that Pref. Server has latest data. If not, the server with the latest data is made preferred and AVSG notified about stale data.
  - Callback promises are established between client and preferred server. When notified, a client invalidates data and re-fetches.

- Upon close, a file is transferred in parallel to all servers in AVSG using multicast where available.
- Optimistic Replication requires conflict checks on all server operations
  - State is characterized by update history of an object at a server. This consists of storeids of operations at clients.
  - Coda approximates update history by its length and the Latest storeid (LSID). LSID = clientid+monotonically increasing integer. A CVV (coda version vector) is maintained which estimates (conservatively) update length at every other server.
- Replicas may have
  - Strong Equality (LSIDs and CVVs are equal)
  - Weak Equality (LSIDs same but not CVVs)
  - Dominance/Submission (LSIDs are different, but CVV of one subsumed by the other)
  - Inconsistency

More CODA replication

- Replica States may be transformed by
  - Updates
    - Storing files on closing, creation/deletions etc.
    - 2 Phase protocol: First each site in AVSG checks LSID and CVV of client. If they are equal to or dominate its own LSID and CVV, then it commits the change. Then, CVVs at servers are updated to reflect the clients view of who committed the change.
  - Forces
    - Server-Server Protocol. Basically replays the changes at the dominant site not done at submissive site already.
  - Repairs
    - 2 Phase operation to resolve inconsistencies, or to recover from crashes.
    - Automatic vs. User Intervention
  - Migrates
    - Create a covolume of inconsistent data.

Coda Cache Coherence

- Basic Cache operation involves caching file (and directories) on demand!
- Due to Optimistic replication, the Cache Manager (VENUS) also has to be aware of:
  - AVSG enlargement : missing members of VSG are contacted once every tau seconds. If AVSG is enlarged, callbacks are dropped and next reference to data causes fresh fetch and callback reestablishment.
  - AVSG shrinkage: AVSG members from which data is cached probed every tau seconds. If prefered server is lost then drop callbacks.
  - Problem – since callback only on prefered server, what if my prefered server is not in other clients AVSG
    - Solution: when probing, ask for volume CVV, and compare. Drop callbacks as needed.

Disconnected Operation

- Operate on cached data. As long as data is cached everything is fine
- Cache miss is bad, since there is no way to make it transparent to the user.
  - Avoid! Besides LRU replacement, allow user to tag certain files as “sticky”
- Reintegration: Upon reconnection, replay your operations – force for files, more complex for directories. If there are inconsistencies, co-locate the volume and try to fix.

Sundry stuff

- Please Read section 9.5.5
- Log Structured File Systems (9.6)
  - Basic problem – multiple seeks needed to do read/writes, and this costs time
  - Solution – cache file (directories) in memory. Do all updates in memory. Write once to disk the “log” of all changes, including data and metadata.

NOW/xFS

- Designed to work on workstation clusters (fast connections, trusted environment)
- Serverless architecture
- Uses software RAID, Coperative caching, and LogFS
- Manager Maps, imap (index node), Directories(name to index), stripe map
- Read: Try local cache, contact manager, try finding someone caching it, else read from disk.
- Write: Logging
- Cache Consistency: Token based.