Outline for February 20, 2001

1. Greetings and felicitations!
   a. Tuesday, Feb 20 3-4:30: Friday Feb 23 1:10-2:30; go to 1101 Hart Hall to view
2. Writing Policy
   a. Write-through: client writes to file, server immediately updates file written
   b. Delayed write at server: client writes to file, server may hold before updating file; idea is that data may not need to be written at all because client may delete it; problem is crashes loose that data
   c. Delayed write at client: writes sit at client until file is closed, then are flushed to server. Idea is that files are open for a very short time, so this cuts burden on servers
3. Cache consistency
   a. server-initiated: servers inform cache managers when data no longer valid
   b. client-initiated: client cache managers check validity of data before returning it to callers
   c. disallow caching when concurrent-write sharing: file open at multiple clients, and at least one for writing (either server tracks who has file open and how, or lock it)
   d. problem: sequential-write sharing: recently updated file (by one client) is opened for writing by a second client. Second may have outdated blocks in cache (cache timestamps, and compare with real timestamps); first client may not have flushed cached changes yet (server requires clients to flush cache when another client opens file)
4. Availability via replication
   a. Replicate files
      i. Can do only those that must be highly available
      ii. Some attribute data (*e.g.*, protection rights) stored with each replica
      iii. May not reside on same server as containing directory
   b. Replicate volumes (file systems)
      i. Easier to manage (*e.g.*, protection rights associated with volume)
      ii. Need to replicate volume if *any* file on it requires high availability
   c. Replicate filegroups (primary pack; replica called a pack)
      i. Contains subset of files in primary pack
   d. Management: consistency among replicas
      i. Weighted voting scheme
      ii. Agent processes (Locus' current storage site enforces the global synchronization policy)
5. Example: NFS
   a. Architecture: built on RPC and using a virtual file interface à la UNIX (*vnodes*)
   b. Naming: all workstations are (conceptually) clients and servers; in practise, have a few systems designated as file servers (BFS downstairs); discuss file handles; it's stateless
   c. Lack of State: simplifies crash recovery. Handle contains all the info identifying the file, and client kernel tracks file offsets, etc. If client hears nothing, just resend
      i. Server crashes: just restart
      ii. Messages bigger than stateful server would require, but would also require restoring lots of info!
   d. Caching: client caches:
      i. File blocks (on demand, usually 8Kb blocks) as is timestamp of last mod on server. Timeout after some period of time, then must revalidate
      ii. Directory name lookups; flushed when lookup fails and/or new vnode info obtained
      iii. File attributes:90% of all NFS requests to server; discarded after 3 sec (files), 30 sec (directories)
6. Example: Sprite
   a. Architecture: uses a virtual file interface, built on multiple servers
   b. Naming: global tree hierarchy; each subtree is a *domain* and multiple domains may reside on one server; prefix table maps file prefixes to servers (each entry has full name of mount point, server name, and domain name); to locate, refer to prefix table and find longest matching prefix, then go there n(if no entry, broadcast for it); server sends a file token for the file, and this used to reference file. Remote links return contained file names and client iterates.
   c. Caching: in main memory

- i. File blocks: addressed by file token and block number. Query is to cache, then local (if there) or remote; if remote, to cache, then to file
- ii. Delayed writing policy: every 5 sec, cached blocks not modified in last 30 sec are written back to server
- iii. Replacement policy: LRU
- iv. Consistency: server-initiated approach; files open for both reading and writing are not cached and when opened for such, cached blocks are written back before open finished

7. Example: Coda (descendant of AFS)
   - a. Architecture: highly scalable so clients take much of load through caching; local client's disk treated entirely as a cache and clients can operate when disconnected
   - b. Naming: uses volumes; each file, directory has a 92-bit File ID (32 bit volume number, 32 bit vnode number, 32 bit unique identifier). Replication preserves FIDs. Servers have volume location databases. name components mapped to FIDs and this info cached at client.
   - c. Caching: on volume creation, number and location of volume replicas is set (called a volume replication database; set of servers called a volume storage group; set of servers accessible to a client for every cached volume is called accesible volume storage group and is kept track of by client cache manager, called Venus)
     - i. On demand caching, files cached in their entirety on client
     - ii. Obtained from a preferred server (one of the AVSG); if contacted one is not newest copy, because some other member has a newer one, that server becomes preferred server and previous preferred server notified it has stale data
     - iii. Callbacks: set at preferred server, agreement to notify client if file becomes stale; then client invalidates cached file, may get new copy
     - iv. On modification, file sent back to all members of AVSG in parallel (via hardware multicast)
8. Security
   - a. Goals: confidentiality, integrity, availability
   - b. Basic Outline: Foundations, Policies, Mechanisms, Assurance, Human and Operational Issues
   - c. Relationship of policy and mechanism and assurance
   - d. Foundations: ACM model, HRU result
   - e. Policies: BLP (confidentiality), Clark-Wilson (integrity), Chinese Wall (both)
   - f. Mechanism: cryptography
9. Cryptography
   - a. basics (cryptosystems, attacks, codes vs. ciphers, superencryption)
   - b. substitution ciphers (Cæsar cipher, Vigenère cipher)
   - c. transposition ciphers (rail-fence cipher)
   - d. product cipher (DES)
   - e. public key crypto (RSA, DH)
   - f. cryptographic checksums
   - g. key management (Kerberos, PKI)
   - h. digital signatures
   - i. authentication
   - j. Examples: PEM, PGP, IPSec

Static Voting Protocol

Introduction

This is the weighted voting protocol used to ensure mutual consistency among replicas. The rules allow multiple readers and no writers, or one writer and no readers.

Notation

- n sites
- S_i site; L_i corresponding lock manager (usually a process on the same site); when L_i grants a LOCK_REQUEST for a file, it locks the file locally.
- N_i version number of replica at site S_i
- V_i number of votes assigned to replica at site S_i
- r read quorum; a read is allowed if r read votes are accumulated
- w write quorum; a write is allowed if w write votes are accumulated
- v votes total

Voting Algorithm

In what follows, we require that $r + w > v$ and $2w > v$.

1. S_i issues a LOCK_REQUEST to L_i
2. When LOCK_REQUEST granted, S_i sends a VOTE_REQUEST message to all other processes
3. When S_j receives a VOTE_REQUEST message:
   a. S_j issues a LOCK_REQUEST to L_j.
   b. If LOCK_REQUEST granted, S_j returns N_j and V_j to S_i
4. S_i waits for some timeout period, then determines if it has a quorum. Let P be the set of sites from which replies have been received and let $Q = \{ s \in P \mid N_j = \{ \max \{N_k \mid k \in P \} \}$
   a. For a read request, if $V_r = \sum_{s \in P} V_s \geq r$, then S_i has a read quorum
   b. For a write request, if $V_w = \sum_{s \in Q} V_s \geq w$, then S_i has a write quorum.
5. If S_i does not obtain the desired quorum, it issues a RELEASE_LOCK to L_i and all L_s from which it has received votes (that is, all L_s with $s \in P$).
6. If S_i obtains a lock, it checks that its local copy is current (and if not, obtains a current copy).
7. If S_i requested a read:
   a. The local copy is read.
   b. S_i issues a RELEASE_LOCK to L_i and all L_s from which it has received votes (that is, all L_s with $s \in P$).
8. If S_i requested a write:
   a. The local copy is written.
   b. S_i updates N_i
   c. S_i sends all the updates and N_i to all sites in Q
   d. S_i issues a RELEASE_LOCK to L_i and all L_s from which it has received votes (that is, all L_s with $s \in P$).
9. If S_i receives an update, it performs the update on its local copy.
10. If L_i receives a RELEASE_LOCK, it releases the local lock.

Guarantees

- None of the obsolete copies are updated due to a write quorum (see 6).
- There is a subset of replicas that are current and whose votes total to w (see 7).
- There is a non-null intersection between every read quorum and every write quorum (from the relationship

among r, w, and v).

- The write quorum w is high enough to disallow simultaneous writes on two distinct subsets of replicas (see the relationship among r, w, and v).

Example

There are four sites. S_1, S_2, and S_4 have 1 vote, and S_3 has 2 votes. S_1 wants to read a file. For this file, $N_1 = 1$, $N_2 = 2$, and $N_3 = N_4 = 3$. Assume $r = 3$ and $w = 3$; then $3 + 3 > 5$ and $2 \times 3 > 5$.

S_1 issues LOCK_REQUEST to L_1
L_1 grants LOCK_REQUEST
S_1 broadcasts VOTE_REQUEST to S_2, S_3, S_4
S_2 receives VOTE_REQUEST, issues LOCK_REQUEST to L_2
L_2 grants LOCK_REQUEST, so S_2 returns $N_2 = 2$ and $V_2 = 1$ to S_1
S_3 receives VOTE_REQUEST, issues LOCK_REQUEST to L_3
L_3 grants LOCK_REQUEST, so S_3 returns $N_3 = 3$ and $V_3 = 2$ to S_1
S_4 receives VOTE_REQUEST, issues LOCK_REQUEST to L_4
L_4 grants LOCK_REQUEST, so S_4 returns $N_4 = 3$ and $V_4 = 1$ to S_1
After timeout, S_1 computes $V_1 + V_2 + V_3 + V_4 = 1 + 1 + 2 + 1 \geq 3 = r$, so S_1 has a read quorum. It checks that its copy is current; it sees it is not as $N_1 < N_3$, and obtains a current copy from S_3. It sets N_1 to 3.
S_1 reads the local copy
S_1 sends RELEASE_LOCK to L_1, L_2, L_3, and L_4

Now suppose S_1 wants to write. The protocol above is the same until the timeout. Then:
After timeout, S_1 computes $V_1 + V_2 + V_3 + V_4 = 1 + 1 + 2 + 1 \geq 3 = w$, so S_1 has a write quorum. It checks that its copy is current; it sees it is not as $N_1 < N_3$, and obtains a current copy from S_3.
S_1 writes the local copy
S_1 sets N1 to 4
S_1 sends the updates and $N_1 = 4$ to S_2, S_3, and S_4
S_1 sends RELEASE_LOCK to L_1, L_2, L_3, and L_4

Now let $r = 2$ and $w = 4$. Suppose S_1 wants to read but S_2 and S_3 do not respond. Then:
After timeout, S_1 computes $V_1 + V_4 = 1 + 1 \geq 2 = r$, so S_1 has a read quorum. It checks that its copy is current; it is not, as $N_1 < N_4$, and obtains a current copy from S_4.
S_1 reads the local copy
S_1 sends RELEASE_LOCK to L_1 and L_4

Now suppose S_2 wants to read, but S_1 and S_4 do not respond. Then:
After timeout, S_2 computes $V_2 + V_3 = 1 + 2 \geq 2 = r$, so S_2 has a read quorum. It checks that its copy is current; it is not, as $N_1 < N_3$, and obtains a current copy from S_3.
S_1 reads the local copy
S_1 sends RELEASE_LOCK to L_1 and L_3

Suppose S_1 wants to write but S_3 does not respond. Then:
After timeout, S_1 computes $V_1 + V_2 + V_4 = 1 + 1 + 1 < 4 = w$, so S_1 does not have a write quorum.
S_1 sends RELEASE_LOCK to L_1, L_2, and L_4