Advanced Topics in Computer Systems, CS262B

Prof Eric A. Brewer

Disconnect Operation in Coda

February 10, 2004

I. Disconnected Operation in Coda

Basic idea: extend techniques for caching and replication to support *disconnected* operation, where a work-station is cut off from some or all servers for a while.

Primary usage scenarios:

- network partition and/or server crash,
- portable computers.

Approaches to concurrency control:

- Pessimistic:
  - lock data before using to ensure access to it (note: a *lease* is a lock that times out)
  - or only allow updates from a subset of all clients (e.g. from a majority partition)
- Optimistic:
  - no locking; allow updates from all clients
  - resolve conflicting updates when they are discovered

Pessimistic approach provides strong consistency at the expense of availability.

Optimistic approach provides availability at the expense of strong consistency.

For disconnected operation must use an optimistic approach.

Observe: a pessimistic approach can still be implemented on top of an optimistic one if desired.

Server replication:

- first-class servers handle sharing among clients, provide backup, perform conflict detection, etc.
- clients read from best available server, write updates to all available servers.
- clients poll to maintain membership about the AVSG (available volume storage group). Why did the designers prefer this over having servers maintain availability information?
  - principle of putting as much processing on clients as possible.
  - AVSG is potentially different for every client.

- client compares server states and tells conflicting servers to reintegrate.
  What might go wrong with this approach?
  - inconsistent file copies may not get resolved for long periods of time if they don't appear in any client's AVSG when that client tries to access them.

Client states: hoarding (normal) -> emulation (disconnected) -> reintegration -> hoarding

Hoarding:

- based on access patterns
- database provided by user
- recording mode available to construct database entries
- hoard walk to detect changes. Could become a performance problem. Why?
  Takes O(size of client cache) to perform. As client disks get big, so does hoard walk time.
- why is Coda's hoarding approach imperfect?
  Can't guarantee that you've seen/mentioned all external dependencies of a program.

Emulation:

- Client cache must provide services normally provided by server.
- Note: whole-file caching wins in this environment.
- Client cache must log what has changed so that it can update the servers during reintegration. Log compression is crucial. Why?
  Otherwise log size is a function of the total number of client updates instead of the size of the client's mutated files.
- Client cache must keep log in stable storage. Why?
  Log must survive client crashes/shutdowns (unlike traditional client caches).

RVM:

- covered later in the course
- bounded persistence: compromise between persistence of updates and performance of updates.
- note: RVM still guarantees a consistent cache state even if updates are lost due to bounded persistence intervals.

Reintegration:

- Main issue is how to resolve conflicts.
- Done with each server in AVSG. What happens if you get conflicting results from different servers?
  Basically the same thing happens as when servers are found to conflict during normal operation. However, the disconnected operation log and data are stored onto a server for

later reintegration.

- Conflicts in ordinary files are punted to the user.
- Directory conflicts can mostly be resolved automatically. Approach is similar to one used by Locus: merge additions and non-conflicting modifies and deletes.

Supporting measurements:

- reintegration time: small examples only, so hard to extrapolate. Probably not a problem given the log compression techniques they have/propose. Has not been a problem so far.
- client cache size: tens of megabytes appear to be enough.
- likelihood of conflicts: < 0.5% in all cases for periods up to a week in length. Has not been an issue in practice. Why might this change in the future?
  - clients may get more aggressive about sharing once they know they can rely on conflict detection.
  - groupware and other multi-user forms of sharing are becoming more prevalent.

Key features of paper:

- Support disconnected client hosts by allowing cached data to be used as if it were weakly-consistent replicated data.
- Let anyone modify data; detect conflicting updates when replicas reintegrate.
- Maintain a hoard database to avoid not having needed files when disconnected.

Some flaws:

- only W/W conflicts detected; R/W conflicts not handled.
- no client/client sharing possible.
- server consistency may be too lazy (esp. in light of the lack of support for R/W conflict detection).
- volume-atomic reintegration is too coarse.

A lesson: Optimistic approaches to problems have the potential to provide significantly greater availability *if* the common usage case does not incur conflicts *and* if conflict detection is reliable.