Some Issues On Supporting Data Access In Mobile Computing

A Presentation for CS791m (Mobile Computing)

Ping Xuan, Subhabrata Sen, Oscar J. Gonzalez-Gomez, Jesus Fernandez
(The Operating System Group)
Dept. of CS, UMass

Fall 1996

Data Access

- Overview
- Problems
- Solutions
  - Availability
    - * Caching
    - * Prefetching
    - * Hoarding
  - Consistency
    - * Eventual consistency
    - * Update conflict detection

Data Access

The Ultimate Goal:

- provide efficient and transparent support to access to shared data, in a mobile environment.
  - availability
  - correctness/consistency
  - efficiency
  - transparency

Constriants: Mobile elements

- resource-poor (relative to static elements)
- more prone to loss, destruction, and subversion
- must operate under a much boarder range of networking conditions

Simple idea for solutions:

- can't totally rely on remote resources
- Should utilize the available resource wisely

Problems

Common to distributed systems, but more severe:

- low BW, long response time
- variable BW and latency
- unequal resources among hosts

New problems due to mobility:

- both location and connections are not static
- long, frequent disconnections

Overall, mobile system looks like a scaled-up version of distributed system when connections are on; but new issues do arise when the connections are not on.

Solutions

Dynamically reconfigure resources in response to mobility

- dynamical replication of data (and service)

Disconnected operations

- increase data availability
- reduce network traffic, latency, etc.
- reduce power
- reduce server burden

Availability

- Notebooks may have large storage, strong computing power, but low network capability. Even PDA's can benifit from caching.
- Latency of remote operations degrades as the distance between hosts increases.
- Client movement cannot be constrainted, although patterns of movement may exist.

Some new thinking on file-system design is needed.

The central idea: caching

Server-based vs. Client-based?

As expected, the cache coherence problem becomes bigger - a tradeoff between availability and consistency

Caching

What to cache?

- Is simple LRU sufficient enough?
- the SEER Predictive Caching System
  - the semantic distance between files
  - definition of a reference
  - some implementation issues

Hoarding

- Cache misses are more expensive in mobile environment.
  Even so when disconnected.
- New idea: voluntary, client-initiated disconnection
- How to do hoarding?
  - do nothing
  - user-provided information
  - snap-shot spying
  - access tree-based

Prefetching

- Online strategy: compared to hoarding
- Old idea:
  - prepaging
  - prefetching of files
  - prefetching of database objects
- What to fetch?
  - access tree (semantic structure), or
  - probabilistic modeling of user behavior

Consistency

More challenges on data consistency.

- "Global communication" is costly.
- Assumption: simultaneous sharing (other than read-read) is rare.
  (what if this assumption is not valid?)

"Lazy" or weak consistency are more likely to achieve high availability.

- Many algorithms for managing strong data consistency by atomically updating all available copies do not work well in a partitioned network
- Server-initiated callbacks for cached data invalidation have similar problems.
- Read-any/write-any?

Eventual Consistency

Reach eventual consistency while minimizeing assumptions about communication characteristics

- Server propagates write among copies using an "anti-entropy" protocol
- Reconciliation
- Anti-entropy ensures that all copies are "converging" towards the same state
- Eventually reach identical state if no new updates
- An example: pair-to-peer anti-entropy:
  - each server periodically selects another server
  - will reach identical state *if* they have the same writes performed in the same order

Update conflict detection

System support for detection of update conflicts

- Non-semantics-based detection
  - version vector
  - simple timestamps
- Dependency checks (on each write)
  - dependency set
  - If queries (run at server) do not return the expected results
  - Application-specific resolution of update conflicts:

Case study: Replica Management

- the model of operation
- assumptions
- the approach
  - Server-based writes
    - use primary-secondary replication scheme; clients communicates only with primary
    - write-back caching with the server, not the client
    - N-fault-tolerant replicated updates before purge client cache
    - Forced write (when cache full, to avoid blocking)
  - Interface
    - loose-write
    - strict-write

Case Study (continued)

- – Currency Tokens (CT)
  - ∗ a client holds CT are guaranteed to find the current version, by loose-read in this sequence
    - · client cache
    - · primary server
    - · secondaries
  - ∗ Potential Consistent Writer (PCW)
    - · client has performed strict-read, and the reading process has write permission. (has both the desire and ability of a consistent update)
  - ∗ client receive CT if it is the only PCW, or there is no PCW for the file.
  - ∗ if strict-read performed without CT: every server contacted and returns its copy; notify PCWs.
  - ∗ Use version vectors to tag files

Case Study (continued)

- – Revocation of CTs
    - ∗ First PCW arrives
    - ∗ Second PCW reading the file: neither should have CT.
    - ∗ Primary unable to contact client for a long time
  - – Blocking
    - ∗ write may block - purge not received and cache full
    - ∗ strict-read blocks :
      - · if holding CT: only when local, primary, and N or more secondaries not reachable
      - · if no CT: blocked if any server unreachable, or any PCW inaccessible.
- compared to Coda: Coda is lazy: user wait only at open and close - here only open (close/write to be picked up later); Coda allow read/write as long as any replica is available - here strict-read may be not allowed.
- compared to Echo: use both read and write tokens - bad for sharing