Cache Consistency Implementation

Tom Anderson

Outline

Last time: consistency models

Now: implementation of consistency models

Setting

- Clients and client caches
  - local copy of some portion of storage data
- Intermediate caches
  - Local copy of some portion of storage data
- Storage nodes
  - Storage may be replicated
  - Storage may be partitioned (sharded)
  - Each item can be have 0, 1, many cached copies

Consistency Models

- Weak
  - not consistent
- Eventual
  - in absence of further writes, all nodes eventually agree on all storage values
- Serializable
  - appears linearizable to application
  - allows compiler optimization
- Linearizable
  - Behaves as if operations applied to a single store

Some Other Models

Read Your Writes
- Every node observes the effect of local writes
- Until value is over-written by some other op

Causal Consistency
- Operations respect Lamport "happens before"
- May not be eventually consistent!

Snapshot Reads (databases)
- Writes always occur in the present
- Reads may occur in the past
- Transactions (group of reads) must be at some consistent state (in the present or in the past)

Operations Take Time

Client sends request; gets back response

When did operation occur?

Operations Take Time

Linearizable:
- System may perform operation at any time between start and completion
- System may select any order for concurrent operations
- If order is important, app/user needs to wait for op to complete before starting next op

Serializable:
- System may perform operation at any time, provided program doesn't depend on the result
- But if program depends on result, must act as if linearizable

Eventual:
- System may perform operation at any time
- Every copy eventually updated

Three Clients Example

Client 1	Client 2	Client 3

put (k1, f(data))
put (done1, true)
	while(get(done1) == false)
	;
	put (k2, g(get(k1));
	put (done2, true)
		while(get(done2) == false)
		;
		rslt = h(get(k1), get(k2))

Initially, done1, done2 = false; put/get interface

Intuitive intent:

client3 should execute h() with results from client1 and client2

waiting for client2 implies waiting for client1

Three Clients Example

Suppose

- Every operation is done in order
- Wait for each operation to complete before moving to the next one
- All keys stored on the same server

Then if done1 is true, k1 has correct value

Do We Need to Wait?

Can we start put(done1) before put(k1) is complete?

Yes, provided:

One server

Server performs operations in order sent, e.g., with client-specific sequence number (“processor order”)

What if Storage is Sharded?

Suppose:

k1 and done1 are stored on different storage nodes

client1 issues put(done1), put(k1) in parallel

Client2 can observe writes out of order

Fix: Issue one write, wait for it to complete, before issuing the next write.

What if Caches?

Cached copy of keys stored locally with clients

- might be out of date with storage node
- If read cached copy, can see old value

What if cached copy of k1 and not done1?

- Might see new (uncached) value of done1
- Might see old (cached) value of k1

Transitivity

Can client3 see client2's writes before it sees client1's writes?

- client3 and client2 may disagree on order of client1 and client2's writes

Suppose system keeps caches up to date by sending every update to every node

- order of arrival might differ on the different nodes

Adve Implementation Rule

Let's assume (wlog) that each process specifies that its own operations happen in some order

- E.g., read A, write B, append C …
- If concurrent, system can choose the order

Serializable if

1. Operations applied in processor order, and
2. all operations to same memory location are serialized (as if to a single copy).

Implementing Single Copy

- Cache invalidation
  - Before every write, locate all copies of data and remove them
  - Apply change to single remaining copy
- Lease: permission for some period of time
  - Ex: lease to use cached copy of some data item
  - Wait until lease expires before applying update (plus clock skew)
  - Or ask client to return lease

Cache Implementations

Coherence Write policy	Weak lease	Strong lease/ Callback
Write-through	DNS, web	AFS
Write-back	NFS	Sprite

Terminology

Weak lease/time to live
- Allow client to use cached copy for some period of time (lease)
- After lease expires, client discards cached copy
- On next use, client fetches the latest version.

Write through
- All writes are sent through to storage node

Write back
- Writes applied to local copy
- Sent to storage node in background

Weak Lease/Write Through

Domain Name System (DNS)
- Translates domain name (e.g., cs.washington.edu) to IP address
- Clients cache results of name translation, for TTL
- When TTL expires, discard copy; fetch on use

Web browser
- Browser cache holds copy of pages, images, … for TTL
- When TTL expires, revalidate on next use

Semantics?

Weak Lease/Write Back

NFS
- Network file system in wide use
- Clients cache file blocks and name lookups for TTL
- Discard cache copy after TTL
- Clients write changes to local cache, flushed in background to server

Semantics?

Weak Lease

Advantages:
- No state at server
- Can always update state

Disadvantages:
- Consistency model
- Overhead of revalidations
- Synchronized revalidations

Strong Lease

Wait to perform write until all leases have expired
- Need to prevent new leases in the meantime!

Advantages:
- Linearizable!
- Can reclaim lease even if the network fails

Disadvantages:
- If server goes down, clients cannot continue past end of lease
- Writes stall for length of longest lease
- Reads stall until write completes (or writes can starve)

Write Through Cache Coherence

- Server tracks which clients have cached copy
- Before applying update at server:
  1. Send message to all clients with copy
  2. Each client invalidates, responds to server
  3. Server waits for all invalidations, then does update
  4. Then returns to client
- Reads can proceed
  - Whenever there is a local copy
  - Or if no write ahead of it in the queue at the server

Write Through Cache State on Client

- State is per object/cached item
- Transitions?

Invalid Read-Only

Questions

- If write is in progress, can server perform reads/writes to other memory locations?
- If write is in progress at server, is it ok to do read at client?
- Why does server need to wait until invalidation is applied before performing write?

More Questions

- Why does server need to wait until write is applied before returning to client?
- Why does server need to queue incoming requests while write is in progress?
- How much directory state do we need at the server?

Example

- Two concurrent writes to two concurrent readers. Readers have item cached.
- Writers send change through to server; order of operations is the order they reach the server.
- Server uses callback state to invalidate caches.
- Then reader has a cache miss and fetches the value from the server.

Invalidation vs. Leases

- Invalidation makes no assumption about clock synchronization
- But no progress if network fails
  - Possible that client still has cached copy
- Can combine techniques
  - Short lease on entire client cache
  - Client revalidates as a unit as long as it is up
  - When client fails, server can revoke all safely (at lease expiration)

Recovery

NFS server is stateless

- If server fails, can resume immediately
- Every operation is idempotent, can be repeated
- At least once RPC

Cache coherence is stateful

- When server fails, can reconstruct its state from client state
- At most once RPC

Write Back Cache State on Client

- Owned by at most one client (at a time)
- Read-only at any client => not owned by anyone

Invalid

Read-Only

Write-able

Write Back Cache Coherence

- Server tracks which clients have cached copy
- On write miss, client asks server to:
  1. Send message to all clients with copy
  2. Each client invalidates, responds to server
  3. Server waits for invalidations, then returns to client
  4. Client performs write
- Reads can proceed whenever there is a local copy
- Careful ordering of requests at server
  - Enforce processor order, avoid deadlock

Three Clients Example

Client 1	Client 2	Client 3

put (k1, f(data))
put (done1, true)
	while(get(done1) == false)
	;
	put (k2, g(get(k1));
	put (done2, true)
		while(get(done2) == false)
		;
		rslt = h(get(k1), get(k2))

Initially, done1, done2 = false; put/get interface

Intuitive intent:

- client3 should execute h() with results from client1 and client2
- waiting for client2 implies waiting for client1

Question

Is write back always more efficient than write through?

Distributed Shared Memory

- Can run a parallel program across a network of servers
  - Threads communicate through shared memory, not message passing
- Set virtual memory page protection to trigger fault whenever remote operation needed:
  - read to an invalid page
  - write to an invalid or read-only page

Example

Parallel successive mesh approximation
- Update each element based on neighbors
- Repeat until converged

DSM approach
- Put boundary elements in their own pages
- Automatic exclusive when updated
- Automatic fetch of neighbor's boundary pages

Message passing approach
- Explicitly fetch boundary elements from neighbors