Lamport Clocks

Tom Anderson

Last Time

- Primary/backup
  - System behaves as if it is a single server that (almost) never fails.
- Limitations
  - All requests go through the primary
  - System can stall if failure during view change

Lamport Clocks

Can we make sure everyone agrees on the same order of events?

- When impractical to have one node mediate every request

Assume (for now) there are no failures

- Fix with Lab 3

Assume messages are delivered in order sent by each client ("processor order")

- Ex: include a per-client sequence number

Example: Parallel Make

Physical Clocks

- Can we assign every event in a distributed system a unique wall clock time stamp?
- Local clocks aren't perfect
  - Crystals oscillate at slightly different frequencies
  - Typical error is ~ 2 seconds/month
- Synchronize clocks across distributed system?
  - Network messages involve delays
  - Network message delays are variable

Physical Clocks

- Lets assume a network-attached GPS
  - How close can we bound clocks across multiple systems?
- Option 1: client polls the GPS server for current time.
  - How far off will the timestamp be when it arrives back at the client?
- Option 2: repeatedly fetch the GPS time, estimate relative rate of skew of the local clock

Logical Clocks (Centralized implementation)

Send every message to a central arbiter, which assigns an order for all messages.

Problems with centralization?

Space-Time Diagram

Happens Before

Happens before: a->b, ie, b could be caused by a

1. If a and b are in same process, a occurs before b
2. Send occurs before receive
3. Transitivity

Logical clock C: Cj is the value of clock C for process j st if "a happened before b" then C(a) < C(b)

Logical Clock Update

IR1: increment on every event within a process

IR2: put current clock value Ci in each message sent by process i

On receipt at process j, Cj = max(current Cj, Ci + 1)

If apply tiebreaking rule, then we get a total order consistent with partial order.

Example

Globally Ordered Message Delivery

Proposed rule: Always deliver packets everywhere in the same total order, defined by Lamport clock.

Q: When is it ok to act on a message that arrives?

- Can you know that you have received all msgs that might ever arrive with earlier timestamp?

Updated Rule

Proposed rule: Always deliver packets everywhere in the same total order, defined by Lamport clock.

If receive a message with time T, ping neighbors to get their current time (and/or any messages that are before T).

Eliminating Waiting

Can we apply messages without waiting to see if an “earlier” message will arrive?

If temporary inconsistency is ok:

- Keep timestamped log of all operations
- Apply operations in timestamp order.
- If an "older" update arrives later, insert in log and recompute.

Vector Clocks

Each process keeps an array of timestamps, VC[i]

VC[i] is the latest event I know from process i

Revised update rule, at node i:

- Pass vector clock in each message
- Update local clock on message receipt (VC[i]++)
- for all i, VC[j] = max (VC[j], msg's VC[j])

VC defines partial order of events

- don't process an event until you've seen all prior events from that process

Vector Clocks

VC[i] = the latest event I've seen from process i

VC defines partial order of events

- don't process an event until you've seen all prior events (you might receive) from that process

Consistent Snapshot

- Consistent distributed state: one where all events that could have caused an event E are included
  - Even if E happens on one node, and (potentially) causal events happens on other nodes
- Snapshot algorithm
  - Snapshot one node
  - Forward snapshot request to neighbors
  - Process snapshot iff received all events that "happen before" the snapshot request, and then forward

Example

Using Logical Clocks

Can we use logical clocks to implement distributed mutual exclusion?

- Example of state machine replication

Correctness:

- one at a time
- everyone in order of request
- Liveness: (if every holder releases lock) eventually requester gets the lock

Logical Clocks for Mutual Exclusion

Every message timestamped with value of logical clock.

1. To request lock, multicast request Tm to everyone
2. If get a lock request message, put Tm on local queue, ack (once it is ack'ed from everyone, request is in every queue)
3. To release the lock, remove request from local queue, multicast release message
4. If get release message, remove that request from local queue
5. Process i gets lock if its request < everyone else's, and none earlier can arrive
   - Process i has seen a later timestamp from everyone else

Example

Memory

Facebook Storage System

Initially:

- a few front end web servers to do application logic
- a single backend storage server

To scale, add more front ends, more back end servers:

- Each front end pulls data from multiple servers (e.g., one for privacy settings, one for pictures).
- Do users see a consistent view?

Now add some intermediate caches:

- 100+ lookups per page
- 1B+ users: 1M+ front ends, 1M+ caches, 1M+ servers

Facebook Architecture

Memory Models

- In a distributed system, we may have
  - Multiple storage shards (partitions)
  - Multiple cache copies of any stored item
- Memory behavior affects app behavior
  - Memory implementation affects memory behavior

Strong Memory Models

- Linearizable
  - All operations occur in a serial order
  - Consistent with processor order
  - During period of time when op was pending
- Serializable/sequentially consistent
  - Appear as if all operations occur in linearizable order, to the application program
  - Allows compiler optimization to reorder ops

Weaker Memory Models

- Eventually consistent
  - All observers eventually agree on the system state
- Causal ordering
  - All operations respect causal ordering
  - May not be eventually consistent!
- Weakly consistent
  - Might be consistent

Database Consistency

- Snapshot consistency
  - Writes occur in the present
  - Reads may occur in the past
  - Transactions (group of reads) must be at some consistent state (in the past)
- Even weaker reads
  - Semantics are implementation specific
  - Usually for performance optimization
  - Hard to know if app behavior depends on semantics