Storage Systems

NPTEL Course

Jan 2012

(Lecture 36)

K. Gopinath

Indian Institute of Science

Ordering semantics

- none
- FIFO: if a process casts *m* before *m′*, no correct process delivers *m′* before *m*
- causal: if cast of *m* precedes *m′*, no correct process delivers *m′* before *m*
  - *e* precedes *f (*Lamport*)* iff
    - a process executes both *e* and *f* in that order, or
    - *e* is the cast of some msg *m* and *f* is the delivery of *m* at some process, or
    - there is an event *h* such that *e* precedes *h* and *h* precedes *f*
- total: if at a correct process *p*, *m* delivered before *m′*, then *m* will be delivered before *m′* at all destinations they have in common

Logical Clocks

- Associate a counter with each process and msg in system
  - LT_p: logical time for process *p* (*p*'s counter)
  - LT_m: logical time of message *m* ("logical timestamp of *m*")
- Maintaining counters for a new event:
  - if $LT_p < LT_m$, process *p* sets $LT_p = LT_m + 1$
  - if $LT_p \geq LT_m$, process *p* sets $LT_p = LT_p + 1$
  - for other events, process *p* sets $LT_p = LT_p + 1$
- For a more accurate clock with static number of processes, vector clock

Vector Clocks

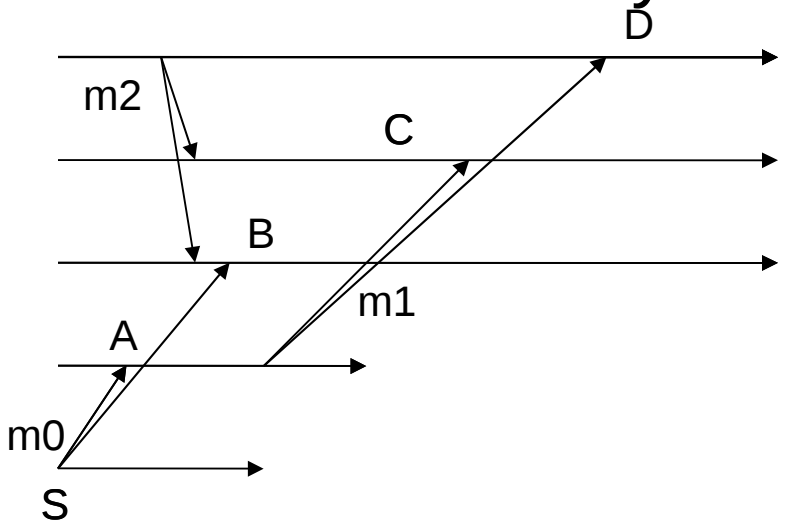
- A vector clock is a vector of counters, one per process in set
- Vector time for an event *e*: for each process in vector, how many events occurred at that process causally prior to when *e* occurred
  - if VT of msg *m* is [4,5,6], 4 events happened causally prior to sending of *m* at process *p*, 5 at *q* and 6 at *r*
- VT_p and VT_m*:* vector times associated with process *p* and msg *m*

  Given a vector time VT, VT[*p*]: entry in vector corresp to process *p*
- a count of the number of events that have occurred at *p*

  Updating vector clock:
- Prior to performing any event, process *p* sets $VT_p[p] = VT_p[p]+1$
- Upon delivering a msg *m*, process *p* sets $VT_p = max(VT_p, VT_m)$

Causally/Totally Ordered

- m0 and m1 causally ordered but m2 none such



- Total order: since m2 delivered before m0 at B,
  - if m0 also delivered to C, then
  - it should be later than that of m2
- Weak/strong: whether dyn uniform?

Orthogonal Choices

- atomicity (does not specify delivery order but same everywhere)
- delivery order (FIFO, causal, total)
- uniform
- timed (to be delivered within a bounded time)

Probs with Causal Model

- replicated db with 2 copies of bank acct x on 2 sites
  - user deposits 20: mcast "add 20 to x"
  - bank adds interest: mcast "add 10% to x"
  - both not causally related
  - may do it in diff orders!
- P publishes X_0, X_1,...
  - Q subscribes to P and computes f(X_0), f(X_1),...
  - causal order: guarantees that f(X_0) received after X_0
  - But may still make mistakes:
    - X_0, X_1, v=f(X_0),... & thinking v is f(X_1)
- need addl tags

Ordering Models for Storage

- Consider Echo distr FS model (ACM TOCS May'94 Mann et. al.)
  - replication (servers+disks), caching (==single-copy equiv), global naming (single-system view), distr security
  - Coherent write-back caches for both files+dirs thru ordered write-behind
    - write-behind: written back after a fixed time
    - write-back: written back after an unbounded time
  - Large caches that are transparent except on faults/crashes
- Avoid NFS drawbacks such as
  - incoherent caches (some NFS do close-on-sync but not dirs)
  - unlinking open file problem
  - applns can write even if no space avlbl on server

Echo

- FS data changed only by a write that is logically done at a distinct point in time
  - A fault causes some writes to be discarded =>undone at a distinct point in time
- On network partition, a client blocks or is returned error on timeout (NFS hard/soft)
- readops (stat, open, rd, ls, lookuppn, ...)
- writeops(wr, create, mkdir, rename, fsync, ...)

Echo

- Ordered & stable writes needed if writes can be discarded at any time
  - write requested by one client and observed by another: write should be stable
  - writes on same obj should be stable in logical order
    - overwrites (length preserving) by one client can be reordered as an "opt"
    - overwrite failure-atomic if only one block modified
  - fsync on dir and files should make them stable
  - forder: constrains ordering of write
    - forder(f1, f2,...): any pending ops on f1, f2,... logically performed before any ops ordered after forder
      - an “update” to each of its arguments: like “touch” in makefiles
    - returns immediately unlike fsync

Echo Model

- Define 2 relations: -> (data dep) and => (partial order for stable writes)
  - => a subset of ->
  - both -> & => transitive
- o1->o2 if o1 is a write, o1 & o2 have an operand in common, o1 performed logically before o2, o1 not discarded when o2 performed
- o1=>o2 if o1->o2 and o1 & o2 writes but not overwrites
- if o1=>o2 and o1 discarded implies o2 discarded
- if o1->o2 and o1 & o2 on diff clients, o1 stable when o2 performed
- if o1=>o2 and o2 stable implies o1 stable
- if fsync(f) successful, f is stable