Storage Systems

NPTEL Course

Jan 2012

(Lecture 35)

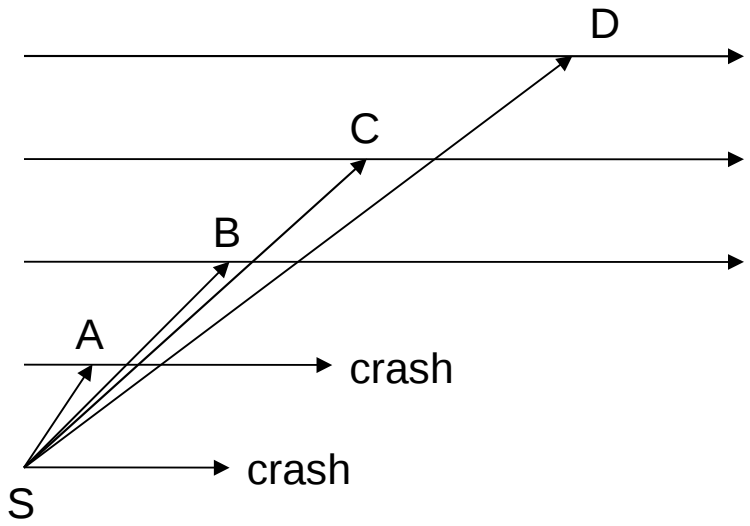
K. Gopinath

Indian Institute of Science

Some Definitions

- static/dynamic membership in a group
  - static: typically mapping betw hw and processes that are restarted on failure
  - dynamic: new processes started, join system; leave system on termination, failure or disconnection
- dynamically uniform: if any process performs some action, all processes that remain operational also perform it => externally visible actions
  - different from commit: in commit, if any process (incl a process that will fail) commits, all (statically defined) processes also commit: recovery on failed process
  - in dynamically uniform: if a process leaves (fails), never rejoins system

DYN uniform



Inspite of crash at sender S, if msg delivered at A, it WILL be delivered at B, C, D

Eg: issuing money from an ATM
Commit, Paxos!

In non-dyn uniform, if S and A crash, it is OK if none of the survivors get the msg

More Definitions...

- non-dynamically uniform: states and actions of processes that subsequently fail discarded; operational part of the system defines system
- failure-atomic multicast: 2 types
  - if *m* sent using a dynamic uniform protocol, when *p* delivers *m*, it also knows that any future execution of the system in which a set of processes remains operational will also guarantee the delivery of *m* within its remaining destinations among that set of processes
  - if *p* receives a non-uniform multicast *m*, *p* knows that if both the sender of *m* and *p* crash or are excluded from the system membership, *m* may not reach its other destinations

More Definitions...

- reliable bcast/mcast: validity, agreement, integrity amongst correct processes
  - if a correct process casts msg *m*, all correct processes *eventually* deliver *m*
  - if a correct process delivers *m*, all correct processes *eventually* deliver *m*
  - for any *m*, every correct process delivers *m atmost* once & only if *m* cast
- uniformity: differs in agreement & integrity wrt correct/faulty processes:
  - if a process delivers *m*, all correct processes *eventually* deliver *m*
  - for any *m*, every process delivers *m* atmost once & only if *m* cast

View Synchrony Model (Birman)

- introduce synchronization mechanism for failure-atomic protocols wrt group membership changes
  - each process at each time instant has a unique view of membership of group
  - processes that proceed together thru 2 consecutive views deliver the same set of msgs betw these views
  - each msg assoc with a view & all send/rcv for a msg occur at processors with that view; send/delivery events considered as a single instantaneous event
- virtual synchrony (VS): use view synchrony to support a execution model for efficient fault-tolerant computing
  - defined interms of an unrealizable “close synchrony” model

Close/Virtual Synchrony

- **Close** synchronous execution model (infeasible!)
  - Multicast delivered to all group members as a single, reliable ***instantaneous*** event
  - reliable comm (not TCP streams that break unreliably)
  - group addr expansion: membership of group fixed at the delivery of a mcast
  - delivery ordering of concurrent msgs: diff mcasts distinct & ordered same; of related msgs: causal order possible
  - state transfer: at well defined points (eg: a new member join)
  - failure atomicity: mcast a single logical event; failure reporting thru group membership changes that are ordered wrt multicast => atomic mcast
- **Virtual** synchrony: permits asynch executions for which there exists some closely synch execution indistinguishable from the asynch one. => Virtually synchronous process groups

Differences with other models

- transactional serializability:
  - both VS & transactional order-based execution models
  - transactional: focus on isolation of concurrent txns from one another, persistent data & rollback
  - VS: direct cooperation betw group members, failure handling, dynamic reconfig to make progress when partial failures occur
- commit: a form of reliable multicast but also serializability & durability
- multicast delivery: weaker guarantees

Group Membership Service (GMS)

- Behavior depends upon future events
  - suppose a process *p* suspects that process *q* is faulty
  - if *p* itself remains in the system, *q* will eventually be excluded from it
- But cases in which *p* might itself be excluded
  - both *p* and *q* might be excluded
  - system as a whole prevented from making progress if less than majority that participated in previous system view remain operational
  - unfortunately, not clear which case applies until later in the execution when system's future becomes definite
- Good spec: if *p* suspects failure of *q* then *q eventually* excluded from system, unless *p* itself is

Group Membership Protocol for GMS Servers

- on partition: progress only in primary component
  - in non-primary: only safe actions
  - on an eject of *p* from primary: split brain problem if *p* does not know that it has been ejected
  - can use a real clock (synch to epsilon): *p* should detect within delta
  - views should be causally ordered
  - merging
- primary component membership should overlap with that of previous primary
- 2PC if GMS coord live; otherwise 3PC

2PC/3PC Details

- 2PC:
  - 1st phase: list of add/delete events sent to all (incl coord); ack response
  - 2nd phase: coord waits for majority acks
    - if majority, commit upd (incl failures during 1st phase); all upd new view
    - if majority do not respond, wait till comm restored or run a special protocol
    - must prevent a new primary component in which coord not part (impossible!)
- 3PC: new coord if coord fails
  - informs atleast majority about coord failure
  - collect acks and current membership info from all
  - proposes new membership (new add/delete + detected in 1st phase + from old coord). Next as in 2PC

VS Reqs

- system membership takes the form of system views
  - initial system view at system start
  - subsequent views differ by the addition or deletion of processes
- only processes that request to be added to system added
- only processes suspected of failure or that request to leave system deleted
- maj of processes in view *i* must acquiesce in composition of view *i*+1
- starting from an initial system view, subsequences of a single sequence of system views reported to system members; each system member observes such a subsequence starting with the view in which it was first added to the system, and continuing until it fails, leaves the system, or is excluded from the system
- if process *p* suspects process *q* of being faulty, then if the core GMS service is able to report new views, either *q* will be dropped from the system, or *p* will be dropped, or both
- In a system with *synchronized* clocks and *bounded* message latencies, any process dropped from the system view will know *within bounded time*

Impossibility of Synch 2 Clocks!

- Consider 2 nodes P, Q with 2 clocks:
  - Clock_P, say, ideal: Time = *t*
  - Clock_Q: Time = *a*t+b* (*a* skew and *b* offset)
- Delays (asymmetric!)
  - P2Q: *d1*; Q2P: *d2*
- Need to determine *a, b, d1, d2* thru any set of netw pkt exchanges
  - Impossible! (Graham/Kumar'04)
  - From linear algebra...
- However, can determine *a, d1+d2* (roundtrip delay)
  - Offset b cannot be
  - Also sender can predict receiver time when pkt received

API for Clients of GMS

- join(pid, callback) returns (time, GMS list)
  - callback called when membership changed
  - idempotent: if join fails, can issue it again to some other GMS server
- leave(pid) returns void
  - idempotent; fails only if GMS server fails
- monitor(pid, callback) returns callbackid
  - GMS calls callback(pid) if pid fails
  - idempotent; fails only if GMS server fails

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