Storage Systems

NPTEL Course

Jan 2012

(Lecture 34)

K. Gopinath

Indian Institute of Science

Concurrency Control Models

- Distr storage systems: consistency thru txns
  - But trasactions need ordering of writes
- FS, DB, Volume Manager (VM) have diff needs
- FS/DB need control on ordering of writes for serializability: but total ordering (eg. in a SAN) an overkill
- A FS may need stricter guarantees for consistency of metadata (=> ordering) but may not for data
- Similarly, a parallel FS does not need total ordering for non-overlapping upds on a file thru multiple I/O daemons
- Aborts are infrequent in FS as many FS can work quite well with redo-only style transactions
- DB often have long-lived transactions and redo-undo style transactions are necessary

Failures?

- However, designing clustered FS, clustered Volume Managers, etc in the presence of failures very hard
  - Need a way of using total ordering for "control" messages (or a subset of msgs) while not using such ordering for "data" (or majority of) messages
- Timestamp ordering approaches need synchronized clock; otherwise non-causal orderings may result
  - With high speed SANs (2Gbps+), ms accuracy not good
  - other models needed if accuracy/resolution of synchronized clocks not high

Message Synch

- Fly-by-wire control system: correct inspite of process/comm Byz failures
  - redundant lock step (synch) op
  - atomic broadcast of sensor readings to all nodes
  - Internet news: causal order useful
    - reply later than orig msg!
    - cricket scores!
- What about dyn data structures? consistency and reliability:
  - deliver exactly same info to multiple locs
  - virtual synchrony model or GCS

Shared disk filesystems

- Shared storage via a SAN?
  - need a filesystem on top of raw block storage
- A shared disk filesystem allows multiple hosts to concurrently access a shared disk
  - Hosts access data on disk directly via the SAN
  - Filesystem metadata shared and concurrently updated from multiple hosts
  - Filesystem ensures proper synchronization
    - for data access + ensure coherency of caches in nodes
  - There is no single point of failure or bottleneck
    - unlike NFS
  - But need a Distr Lock Manager (DLM)

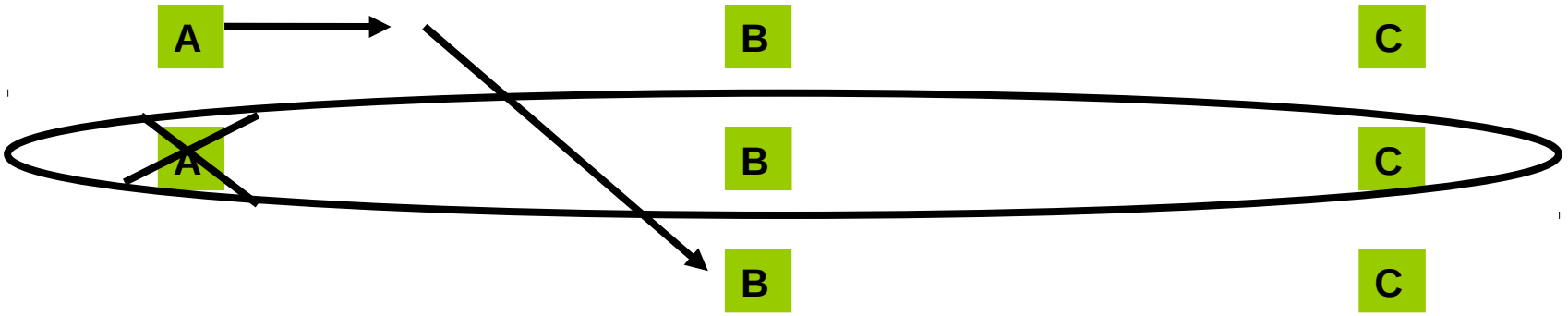
DLM complex!

- Availability of system now depends upon FT of DLM
  - Failure of any node in cluster should not cause lock state to become inconsistent or lost.
- Need to handle concurrent events - local, remote lock requests, failures, join of nodes
  - Stream protocols, like TCP, provide only a point to point bit pipe
    - A “broken” TCP stream noticed by the other peer based on most recent activity, and timeouts may be long (minutes)
    - Lack of consensus amongst set of peers wrt a single node
      - Each peer may judge a node as dead at different times
      - Differing perceptions can comprise consistency
  - Using unreliable datagrams means no ordering among messages and with respect to failures
    - Basic message retransmission and flow control issues complicate the lock protocol code
- Distr Consensus alg needed for agreement on membership&c

- Split Brain
  - Consider a system with 2 nodes A and B connected thru multiple diff connections
    - One thru netw (system netw) N1
    - One thru SAN (storage netw) N2
  - If a live node A is considered dead thru N1 by B but N2 connection to SAN with A is still good, B in dilemma
    - Cannot know status of A's connection with SAN
    - If assume that A is really dead, inconsistent SAN upd
- Mexican Shootout
  - To be really sure, B has to “kill” A if it cannot talk to A thru N1
  - But this is symmetric. A can also decide the same and kill B! *System unavailable!*

Ordering betw msgs & reconfigurations

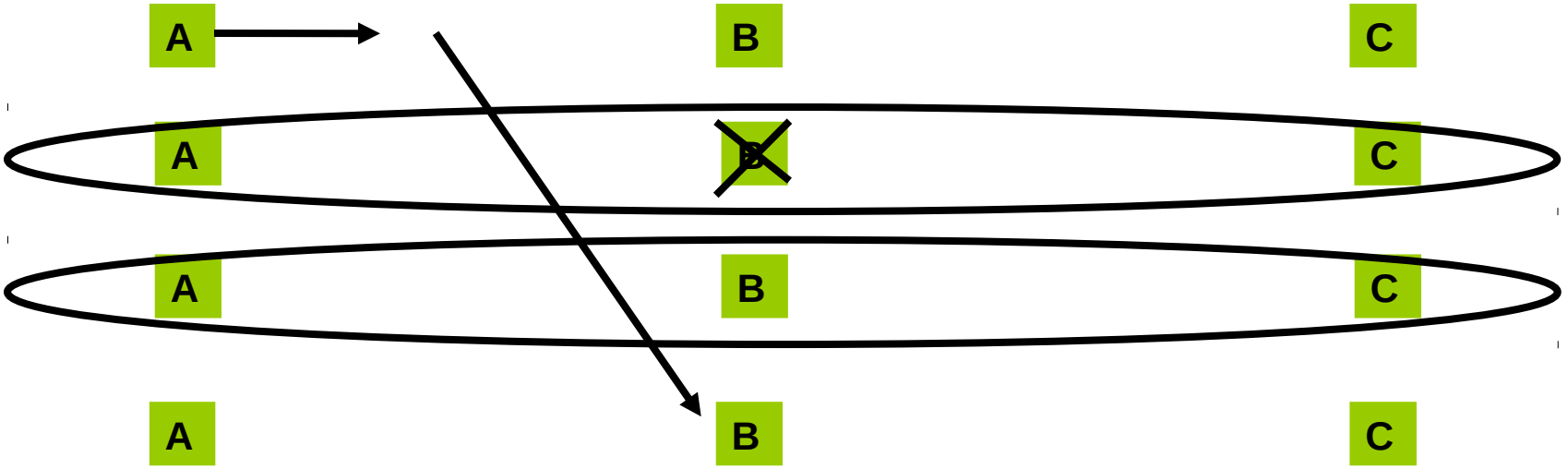
- No guarantees about ordering of data msgs and reconfigurations
- In a concurrent system, this can lead to loss of integrity
  - A's record locks might be forcibly released in a reconfiguration
  - When update is received later, an unlocked record will be updated!!!



- Note that node A could have recovered fast and joined back in the system, before the earlier message was delivered to B

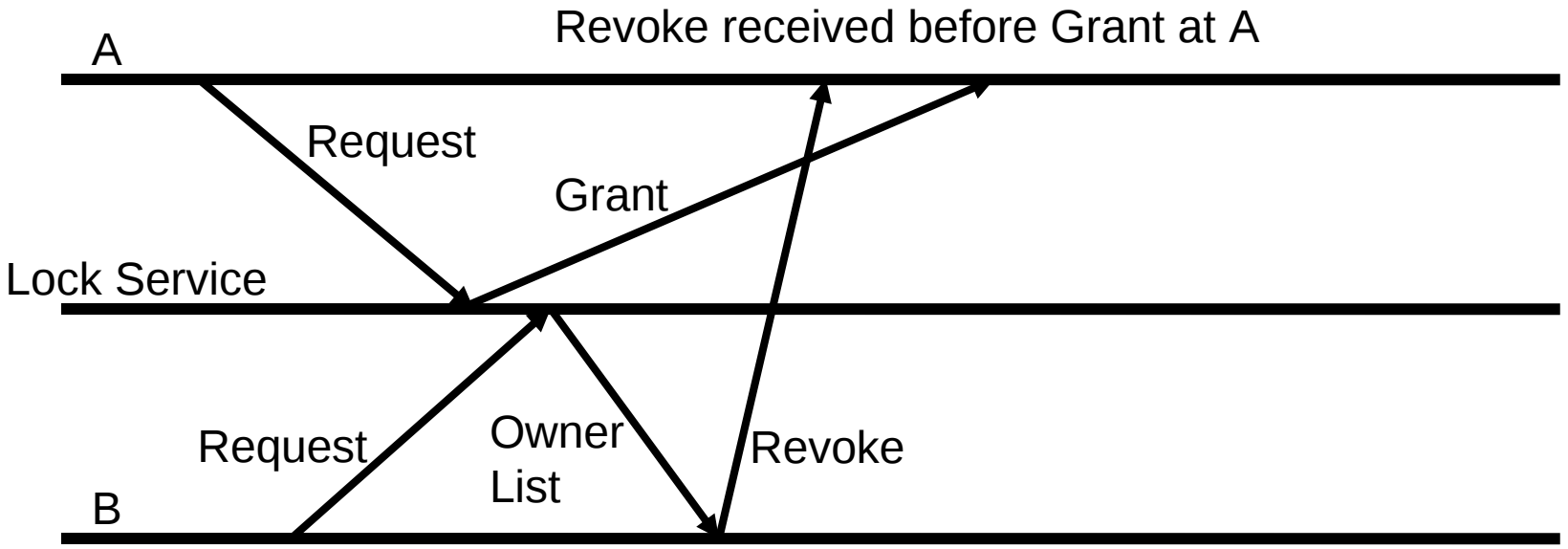
Ordering between msgs & reconfigurations

- Instead of sender failing, receiver could fail and recover, and receive message in new incarnation.
- A might have requested for an object named X in the previous incarnation. In the new incarnation X could be something else!!!



Lack of message ordering

- In an old FS with a lock manager built without message ordering



Could have been avoided by use of causal message ordering since a causal relationship exists between grant and revoke

What is required?

- Reasoning the behavior of concurrent systems difficult
- Failures add further complications, as node failures and message ordering may not be preserved
- Some earlier “serverless” FS built without even FIFO ordering: easy to get into deadlocks
  - Should provide atleast basic FIFO msg ordering to simplify system design
- Need “timely” agreement among processes about membership in system
- Need ordering among msgs with varying levels of strictness - FIFO, CAUSAL, TOTAL.
- Need a strong messaging+membership system (the group communication system, GCS) which DLM can use
  - If lock protocol runs inside kernel, so should GCS.

Ordering between msgs & reconfigurations

- What is needed is that all msgs sent in a view should be delivered in that view itself – not some before failure and some after failure
  - Thus message delivery should be atomic w.r.t failures
  - "*virtual synchrony*" model
- When a failure happens all messages sent in current membership must be flushed out of system before new membership (view) installed

GCS model

- Integrates messaging and membership
- Membership detected by heartbeats
- Each msg associated with view in which it is sent
- A message is delivered in that view only - view delivery will be delayed if need be
- Retransmission, flow control all handled by GCS - interface is asynch
- The membership list is ordered and delivered to the appls in the same order for all members
  - Msgs can be ordered "Fifo", "Causal", "Total"
  - Node gets msgs in real time order but appls in order reqd/specified

GCS model

- When an app joins a group, it is sent a new view msg
- GCS then controls appl with data and view msgs
- On a failure/join detection, a *group unstable* msg sent, to let the app know group not stable
- Appl should stop further messages and when done should tell GCS
- GCS flushes all messages "floating around"
- After all messages flushed, the new view delivered
  - Note that if an appl keeps sending msgs, new view may never get delivered

Summary

- Introduced message ordering problem in a distr system in the presence of failures
- Helpful if appl can depend on a higher level model to simplify state of system in presence of failures