Storage Systems

NPTEL Course

Jan 2012

(Lecture 38)

K. Gopinath

Indian Institute of Science

Summary so far

- Device Level
  - Disk Scheduling
- Protocol Level
  - SCSI, iSCSI (block level)
  - NFS (file level)
- Distr System Level
  - Consistency (commit/consensus protocols)
  - Ordering (virtual synchrony, and at file system level)

Also discussed

- Storage Characteristics
  - Choose 2 out of 3 (speed, capacity, cost)
  - Caching
- Naming vs storage
  - Metadata vs data
  - Recursion! (metadata about metadata...)
- Data loss
  - Metadata loss vs data loss
- Interpret bit patterns
  - Long term storage

Scalability

- Networked vs Distributed storage
- Consistency
  - FS, Vol Mgr, DB notions
- Transactions
  - ACID vs BASIC
- Commit Protocols
  - 2-phase commit
  - 3-phase commit
  - Paxos
- CAP theorem
  - Eventual consistency

Need to address

- Distributed Locking in the presence of failures
- Scalability
- Reliability
- Security
- QoS
- Cross layer optimizations
- Archival Storage
- Flash Memory in storage designs and newer Storage Class Memories (SCM)

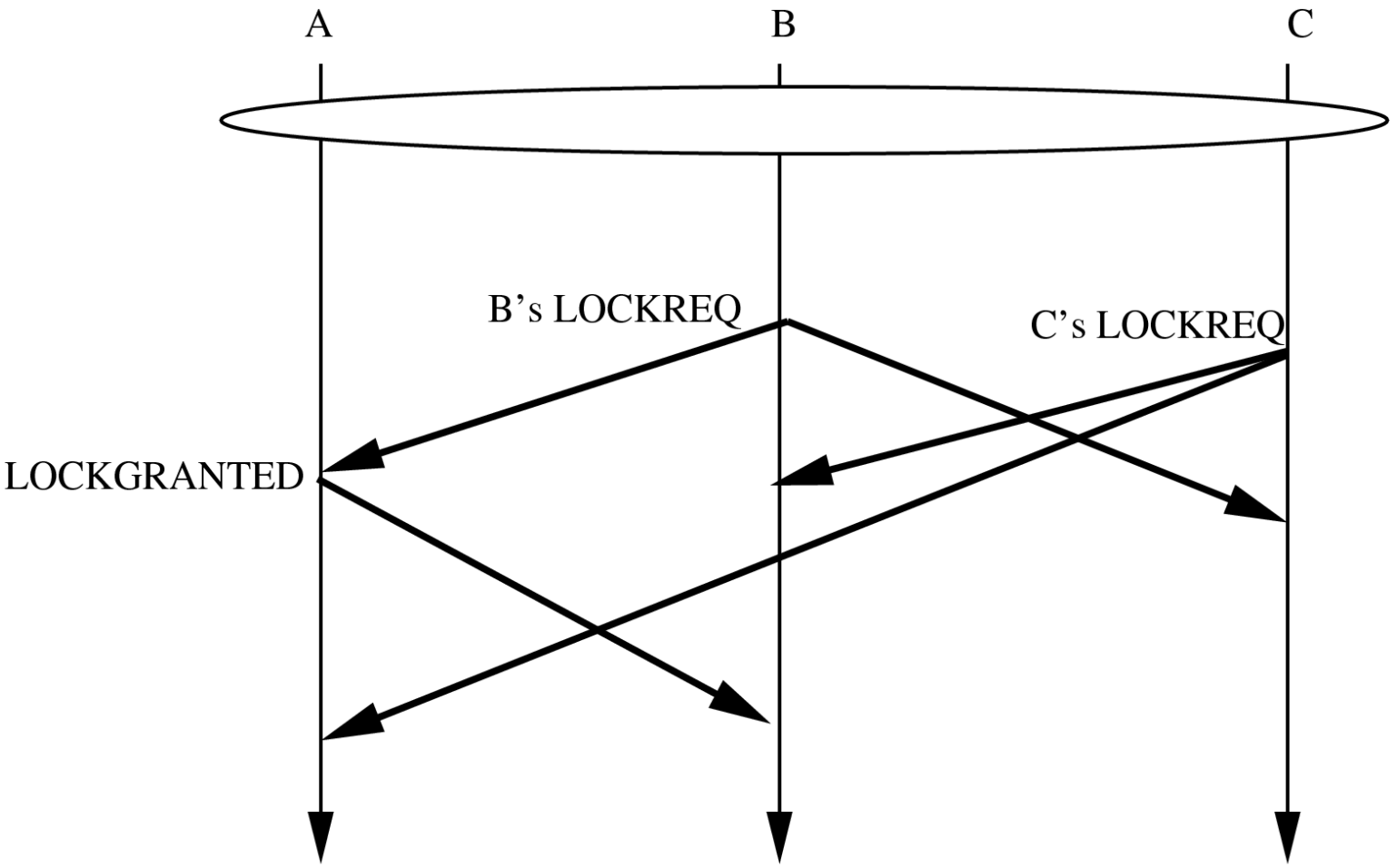
Lock protocol

- Requirements
  - No SPOF, lock state should not be lost
  - The failure of a node should release locks held by it
  - Should minimize network hops
  - No wait for all
  - No total ordering of messages
  - Should distribute lock state

Lock protocol

- A lock requestor does a single multicast for the lock, and expects the lock grant
- When locks are not owned at all, an Initial Lock Server (ILS) can decide this and grant the lock
- The ILS needs to store only lock ids, which is much lesser than complete lock state
  - No need for wait for all
- Locks once granted are cached at the holder, corresp data also cached and written back
  - Enforces cache coherency
  - Unused locks put back to ILS, via an abdicate protocol

C's Request Lost



Lock protocol

- Lock holders queue lock requests if they have locally locked the lock
- Requests are granted in FIFO order to prevent starvation
- When a node is in transit, what happens ?
  - The lock could be unowned at prev owner
  - The lock could be unowned at next owner also!
  - Since lock transit can occur only due to requestors, if we queue at requestor problem solved?

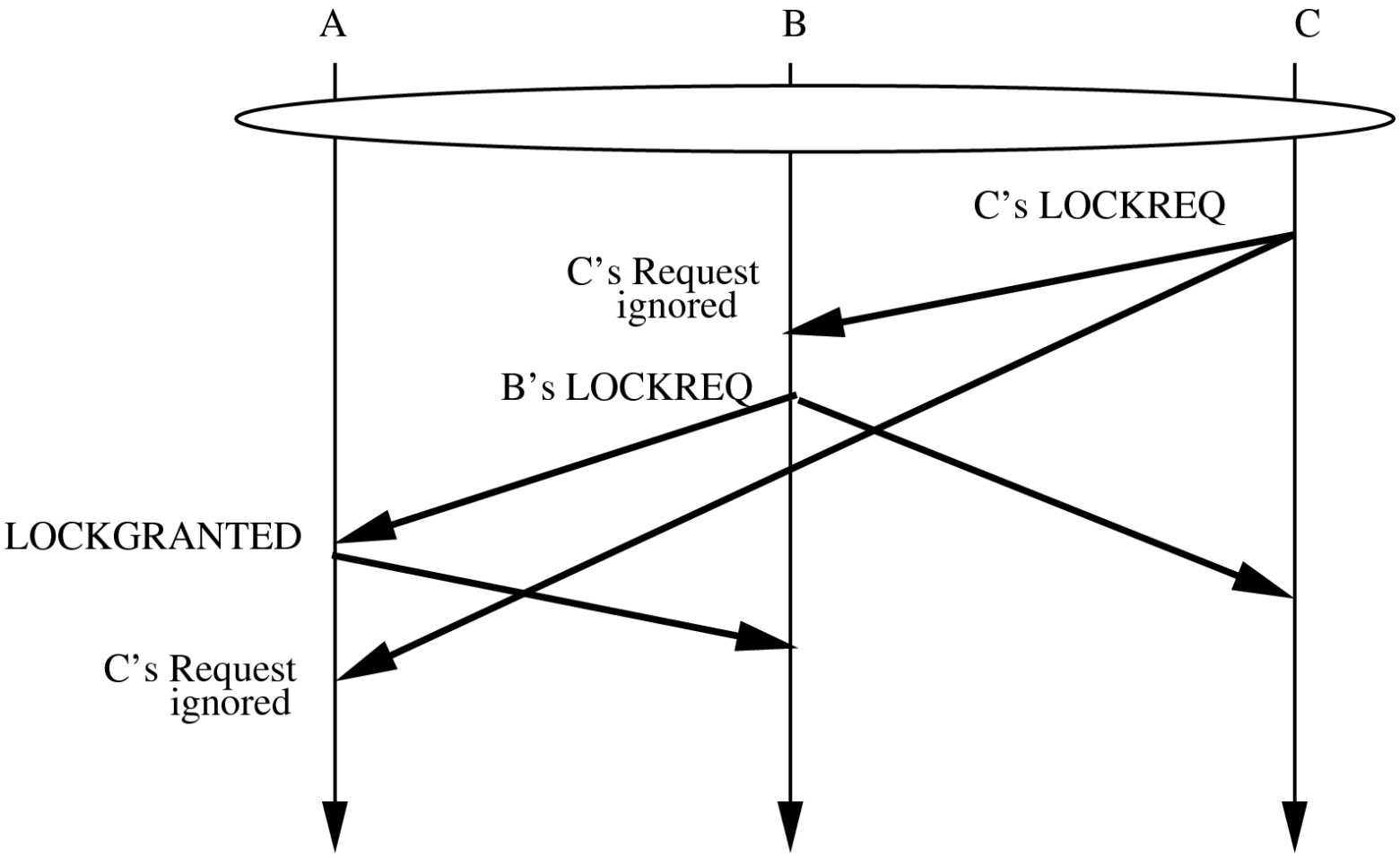
Lock protocol

- The lock requests queued at the requestors may not all be valid..
- With the previous example, one of the nodes (B or C's) lock request will be obsolete
- We flush obsolete requests by having a logical time on each node.
  - Every request that is sent is stamped with the current time
  - Locks carry a timestamp also which says when this lock was last held by a particular node

Lock protocol

- We increment the logical time on the node and stamp this on the lock each time we grant a lock
- If the timestamp on the request is < than the time on the lock, the request is obsolete
- Lock requests with >= timestamp are valid
- New node's timestamp on the locks are set to 0 at all nodes

Need for causal msg delivery



Lock protocol

- On a failure the ILS state is rebuilt by a union of states from all nodes
- On a failure, the gcs flushes all messages which includes the lock grants in particular
- This means when nodes get a new view message, there are no locks on the wire
- This avoids need for a distributed consistent snapshot. All nodes just have to send their states

Reader Writer Locks

- Uses the same underlying scheme as the non reader design
- Need a policy for readers and writers
  - Readers read till a writer comes in
  - A writer granted next
  - Then all waiting readers so far and ....
- To perform the lock protocol activities, a primary reader is elected among the readers.
- This is elected by a writer or by the ILS

Reader Writer Locks

- The code is substantially more complicated
  - The need to handle multiple readers, waiting for readers to drain out
- Recovery is more complicated
  - Reader failure has to be detected by writers or by the ILS
  - If a primary dies, all non-primaries will have to perform duties of primary till ILS elects a primary.

Types of “Distr” FS

- Clustered FS (Posix)
  - CVxFS, GFS2 (RedHat)
- Clustered FS (non-Posix)
  - Ceph, GlusterFS
- Parallel FS
  - PFS, GPFS, NFSv4, Lustre
- GFS, HDFS
- Key Value Stores
  - Cassandra
- Other: ZFS
  - LFS

NoSQL

- Cloud computing has shown that current RDBMs cannot scale and do not have the reqd perf
  - I have not able to find an example of a large-scale Web application that has been able to meet its needs with a single coherent RDBM system (John Ousterhout)
- Column stores
- “NoSQL” systems: Avoid ACID
  - Amazon, Facebook, Google, Yahoo, and Ebay
  - Bigtable: a sparse, distr multi-dimensional sorted map
  - Apache Cassandra — Facebook's dist storage system based on Bigtable data model on Amazon's Dynamo-like structure.
  - Hadoop

Transactional Workarounds for CAP

- “BASE”: No ACID, use a single version of DB, reconcile later
- Defer xact commit until partitions fixed & distr xact can run
- Eventual consistency (e.g., Amazon Dynamo)
  - Eventually, all copies of an object converge
- Restrict transactions (e.g., Sharded MySQL)
  - 1-node xacts: Objects in xact are on the same node
  - 1-object xacts: xact can only read/write 1 object
- Timeline consistency: Object timelines (PNUTS/Yahoo)
  - Reads are served using a local copy; may be stale
  - But application can get current version or any vers>N
  - While copies may lag master record, every copy goes through same sequence of changes
  - Test-and-set writes facilitate per-record transactions