NTNU
Kunnskap for en bedre verden

TDT4225 Very Large, Distributed Data Volumes

Solution to Exercise 4, ed. 9. nov 2021

Deadline 5 Nov 2021 16.00 (4pm)

1. **Kleppmann Chap 5**

a) When should you use multi-leader replication, and why should you use it in these cases? When is leader-based replication better to use?

Multi-leader replication is a solution to use when there is geographical distance between the replicas, and possibly when the data being updated / inserted is geo-related, thus, the updates will almost always be in the same region at the same site.
Leader-based replication is the standard replication solution, where you have replication primarily for fault tolerance or read performance. Thus, updates are always performed at the same replica.

b) Why should you use log shipping as a replication means instead of replicating the SQL statements?

Log shipping does not have the problem with concurrency that statement-based replication has. Thus, the log reflects the sequence that the operations are applied at the leader. In addition, if you use logical log shipping, the replicas may have different physical layouts. This problem is only visible when you use a physical log layout, reflecting which blocks that are referred, etc.

c) How could a database management system support read your writes consistency when there are multiple replicas present of data?

If you always route queries regarding certain keys, to the same replica, this will be supported. When reading copy replicas, you might have problems with reading "outdated" data.

2. **Kleppmann Chap 6**

a) Why should we support sharding / partitioning?

This is simply for scalability in very large datasets, or very high query throughput. Big data requires support for sharding /partitioning / scalability, both for storing and processing the data.

b) What is the best way of supporting re-partitioning?

Re-partitioning is tricky. There are multiple ways of doing it. In the textbook by Kleppmann, they recommend to either use a fixed, high number of partitions, like in Riak, Voldemort, ElasticSearch and Couchbase, or you could use dynamic partitioning, like in Dynamo, MongoDB, Hbase, etc. The fixed number of partitions may run out of partitions when the system is extensively scaled, and it requires you to keep a catalog to know where the partitions are located currently. Dynamic partitioning may use hash partitioning, such that a formula may be used instead of a catalog. Dynamic partitioning often use split of one node's data when adding new a new node. Thus, dividing the node's data into two nodes.

c) Explain when you should use local indexing, and when you should use global indexing?

Local indexing is the most used indexing method in distributed systems, since it makes scalability very easy. Just add a new node and make it index the new data inserted into this node. Global indexing requires you to partition the index independently from the data (or documents) itself. Reads may become faster with global indexing, but writes may become slower (they're not local). Updates may be done asynchronously when using global indexes. Thus, it is the profile of the applications which decide which method is best. However, local indexing is the default and usually the best method.

3. Kleppmann Chap 7

a) **Read committed vs snapshot isolation**. We want to compare read committed with snapshot isolation. We assume the traditional way of implementing read committed, where write locks are held to the end of the transaction, while read locks are set and released when doing the read itself. Show how the following schedule is executed using the two approaches:

r1(A); w2(A); w2(B); r1(B);c1;c2;

Read committed: (ul=unlock)
rl1(A); r1(A); ul1(A); wl2(A); w2(A); wl2(B); w2(B); T1 blocks on lock on B; c2; T2 release locks on A and B. rl1(B); r1(B); ul1(B); c1;

Snapshot isolation:
T1 gets its timestamp txid1=1; r1(A); T2 gets its timestamp txid2=2; T2 creates a new version of A and writes it. T2 creates a new version of B and writes it. T1 reads the old version of B. T1 commits. T2 commits.

b) Also show how this is executed using serializable with 2PL (two-phase locking).

rl1(A); r1(A); T2-blocks trying to set write-lock. rl1(B); r1(B); c1; T1 release locks on A and B. T2 may continue: wl2(A); w2(A); wl2(B); w2(B); c2; T2 release locks on A and B.

c) Explain by an example write skew, and show how SSI (serializable snapshot isolation) may solve this problem.

Write skew is when concurrent transactions make trouble for each other, by not having direct

conflicts through dirty reads or writes, but by their aggregate effect on a set of values. The example in the text book is that one doctor should be on call at every point in time, and two update transactions set each of the doctors to not being on call concurrently by reading the values at the same time, but later update these. This may be solved by serializable snapshot isolation by tracking which values are read and that these are changed by another transaction. Thus, the last transaction which tries to commit, will discover that the values it has read are changed. This transaction will then be aborted.

4. Kleppmann Chap 8

a) If you send a message in a network and you do not get a reply, what could have happened? List some alternatives.

There are six examples listed in the text book:
1) the request is lost
2) the request is queued and will be processed later
3) the remote node may have failed
4) the remote node is in a bad condition right now, and will respond later
5) the request is processed, but the reply is lost
6) the request is processed, but the reply is delayed

b) Explain why and how using clocks for *last write wins* could be dangerous.

This could be dangerous because clocks are hard to synchronize. Thus, even if the clock tells that this is the latest write, it may not be the latest. Thus, there may exist writes that are the basis for other writes ("happened before"), that may have the newest timestamp.

c) Given the example from the text book on "process pauses", what is the problem with this solution to obtaining lock leases?

```
while (true) {
    request = getIncomingRequest();

    // Ensure that the lease always has at least 10 seconds remaining
    if (lease.expiryTimeMillis - System.currentTimeMillis() < 10000) {
        lease = lease.renew();
    }

    if (lease.isValid()) {
        process(request);
    }
}
```

According to the textbook, the code is relying on synchronized clocks. The expiry time is compared to the local clock, while the lease is provided by a node on the network. The code could also have a pause (some daemon executing, e.g.) between checking the clock and executing the request. Thus, there may even be a takeover by another node of the lease?

5. Kleppmann Chap 9

a) Explain the connection between ordering, linearizability and consensus.

This is not very clear. According to Kleppmann, "there are deep connections between ordering, linearizability and consensus". Linearizability is a recency protocol to ensure that there is a "linear" connection between reads and writes in a distributed system. Thus, to make it appear as one copy of the data. Ordering, like in total order broadcast, is a way of providing consistency in state machine replication. Linearizable systems make an ordering between all operations. Consensus algorithms, like e.g. RAFT, decide a sequence of values which makes them total order broadcast algorithms. Any better explanations??

b) Are there any distributed data systems which are usable even if they are not linearizable? Explain your answer.

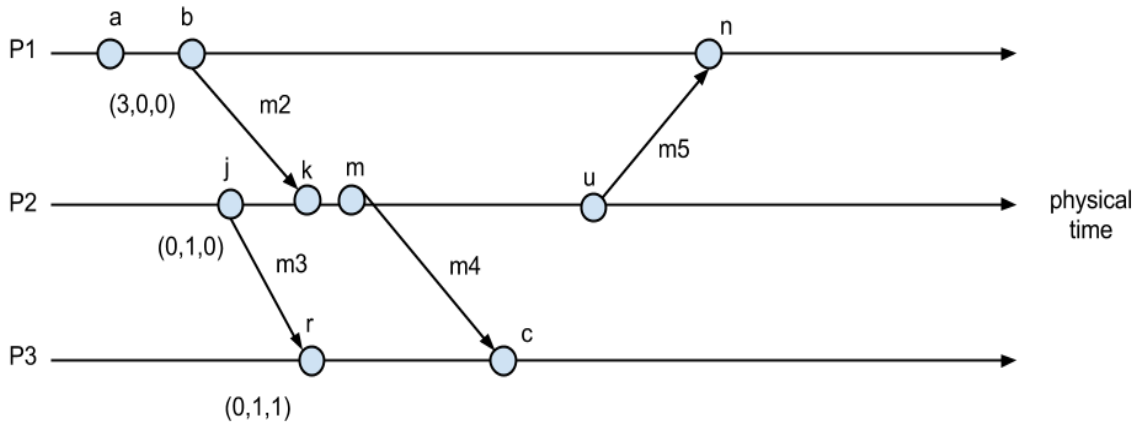
Both multi-leader and leaderless systems are usually not linearizable. They may be useful anyway since they provide services to applications spread around geographically. These systems need to deal with not being linearizable, by providing ways of handling versions and conflicts.

6. Coulouris Chap 14

a) Given two events e and f. Assume that the logical (Lamport) clock values L are such that L (e) < L (f). Can we then deduce that e "happened before" f? Why? What happens if one uses vector clocks instead? Explain.

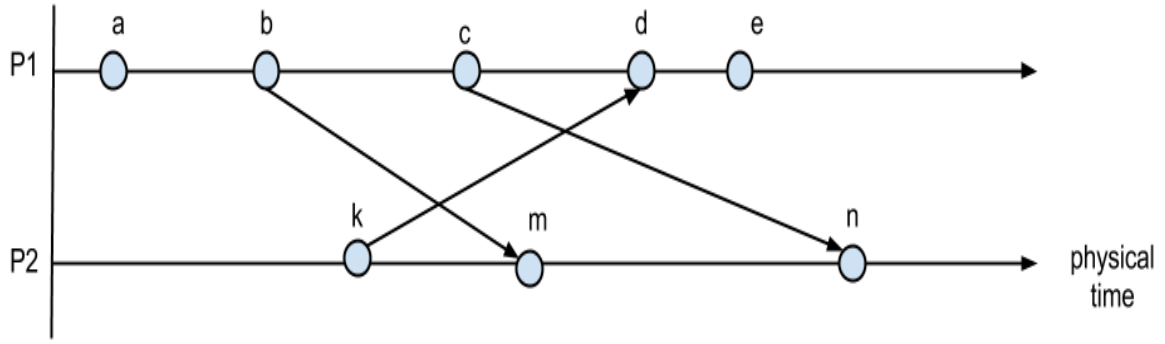
With Lamport logical clocks, this cannot be deduced. Assume that two processes p1 and p2 only have local events. The first event at p1 will have lower Lamport logical clock value than the second event at p2. Still there is no "happened before" since two local events at different processes cannot influence each other (without any communication). However, this is possible with vector clocks. This is because we with vector clocks explicitly count how many events at other processes that a given event at a process can have been influenced by.

b) The figure below shows three processes and several events. Vector clock values are given for some of the events. Give the vector clock values for the remaining events.



Solution: b (4,0,0) , k (4,2,0), m (4,3,0), c (4,3,2) , u (4,4,0), n (5,4,0)

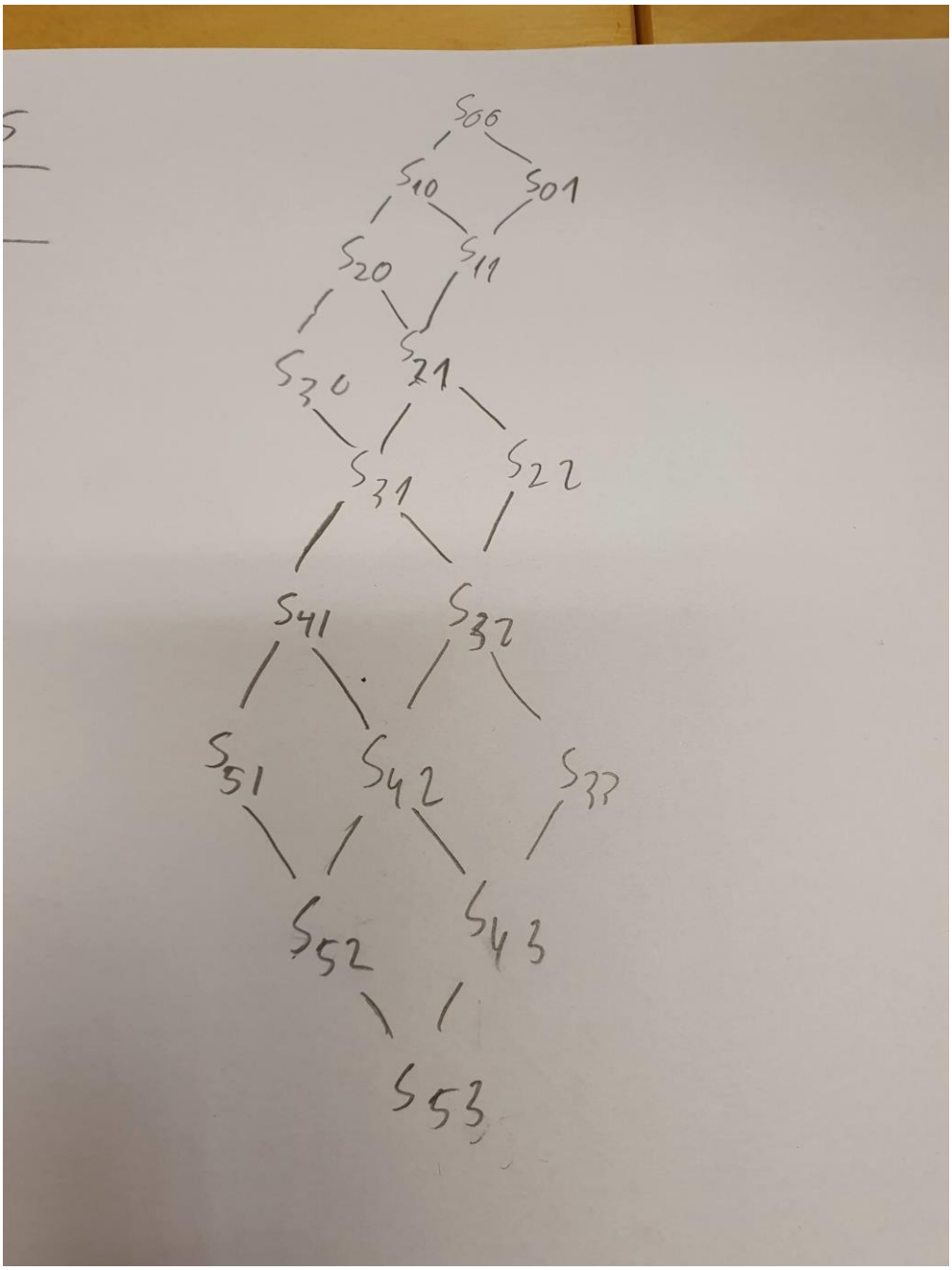
c) The figure below shows the events that occur at two processes P1 and P2. The arrows mean sending of messages. Show the alternative consistent states the system can have had. Start from state S_{00}. (S_{xy} where x is p1's state and y is p2's state)



Solution:

S00 -> S10 or S01
S10 -> S20 or S11
S20 -> S30 or S21
S30 -> S31
S01 -> S11
S11 -> S21
S21 -> S31 or S22
S31 -> S41 or S32
S41 -> S51 or S42

S51 -> S52
S22 -> S32
S32 -> S42 or S33
S42 -> S52 or S43
S52 -> S53
S33 -> S43
S43 -> S53



7. **RAFT log replication**
   The leader replicates the log by sending AppendEntriesRPCs. This also functions as heartbeats. Thus, the leader will send AppendEntriesRPC also in case of no new log. If the leader crashes, a new leader will be eleceted by nodes transomforming from follower to

candidate. By randomizing the timeouts, one new leader will win before the others even have started to send out RequestVoteRPCs.The new leader holds the truth and will use its log. Other nodes (followers) may get their log truncated or extended depending on the situation.

8. Dynamo

a) Explain the following concepts/techniques used in Dynamo:

- *consistent hashing*: Technique to load balance (partition) data in a ring of computers. Both data and computers are hashed into the ring. A node (computer) in the ring has responsibility to store / manage the data with hash values from the previous node in the ring to this node. The idea is to load balance among many computer in an easy manner.
- *vector clocks*: A technique to see the update history of an object. Each node which updates the object puts a trace in the object (nodeNo, seqNo). By using the vector one may see the causality of different versions of the object, i.e. which updates are based on previous updates. This may be used to «tidy up» objects which have been updated concurrently by different nodes.
- *sloppy quorum and hinted handoff*: All reads and writes are done to/from the first N «healthy» nodes in the preference list of the object, i.e., if there are some «unhealthy» nodes they will be skipped. A «hinted handoff»-node will take over the responsibility from the failed node. When the failed node recovers, it will get its (new) data copied from the hinted handoff-node.
- *merkle trees*: «hinted handoff» manages temporary errors, Merkle-trees handle more permanent error by comparing data, in order to correct errors in data values. The nodes in the Merkle trees contain hash values based on the nodes further down in the tree. Each level in the tree may easily be compared to see if there are some differences in the values below in the tree. This reduces the amount of data which will be sent between nodes to the restore the replicas.
- Gossip-*based membership protocol*: To make use of nodes in a distributed system, we must know if a node is alive or dead, elsewise we sometimes need to wait for a long time for a reply from a dead node. Additionally, Dynamo needs to know when to do hinted handoff and it needs to add and remove nodes. Thus, a membership protocol is necessary to tell which nodes are up and which are down. In Dynamo they use a gossip-based protocol, which sends requests to other nodes at randomized points in time and they tell which changes in the node availability they have seen.