

TDT4225 Very Large, Distributed Data Volumes

Solution to Exercise 4

D

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.

2. Kleppmann Chap 6

- a) 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.

- b) 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.

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.

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.

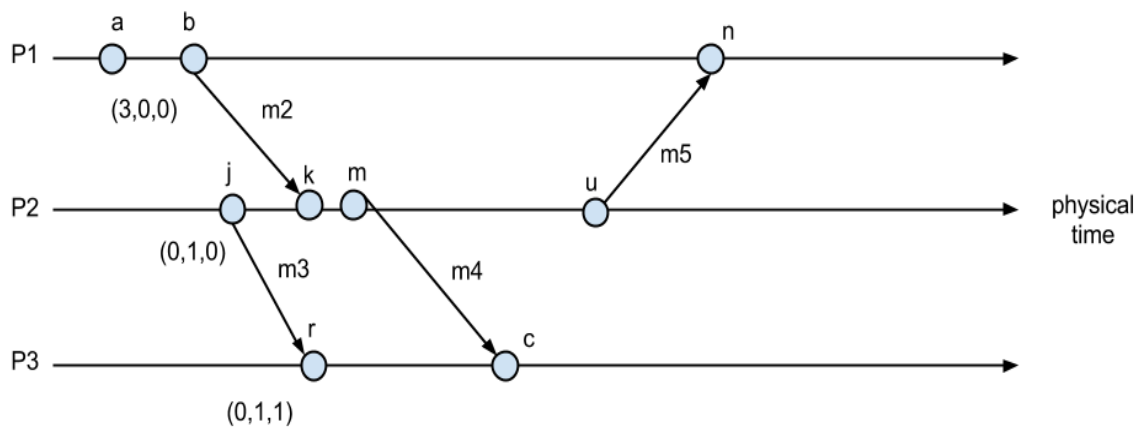
Both multi-leader and leaderless systems are usually not linearizable. They may be useful anyway since they provide services to applications spread around geographically. These systems need to deal with not being linearizable, by providing ways of handling versions and conflicts.

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.

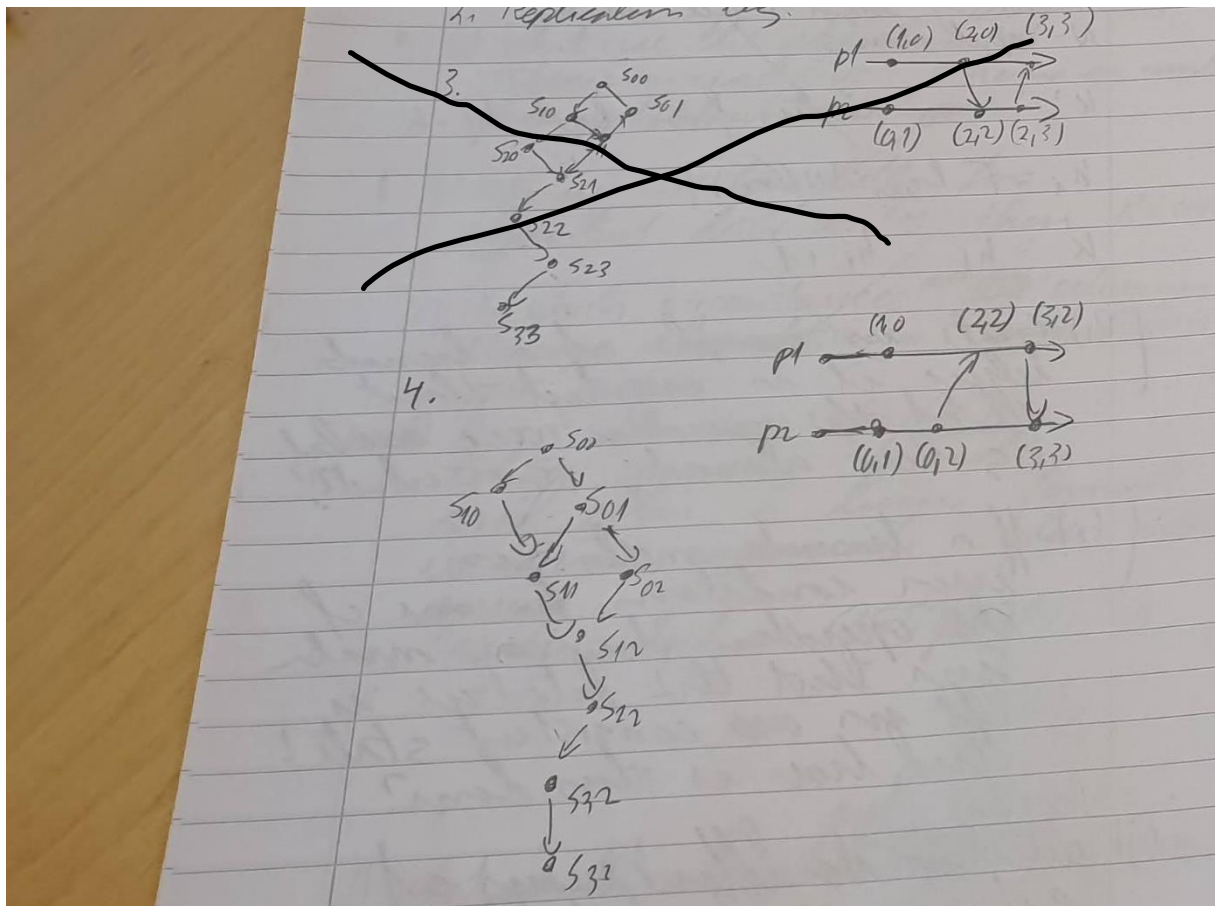
With Lamport logical clocks, this cannot be deduced. Assume that two processes p_1 and p_2 only have local events. The first event at p_1 will have lower Lamport logical clock value than the second event at p_2 . Still there is no "happened before" since two local events at different processes cannot 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)



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 elected by nodes transforming 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.