TDT4225 - Very Large, Distributed Data Volumes

Exercise 4

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 advantegous for when updates need to be performed reliably. This is because if the leader is down for some reason, there will be others to take its place. A common use case is within datacenters where things like performance (local writes), reliability (outages), and network consistency are important. These are all problems that can be resolved by keeping localized leaders in each group of computer nodes (datacenters).

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

Problems occur for:

- Nondeterministic functions like 'NOW()' and 'RAND()'.
- Autoincrementing columns and concurrent updates.
- Statements with side effects, like triggers.
  Therefore, deterministic logs are sent out to followers.

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

This works by keeping track of the specific commit token that is accepted as 'up to date enough' for the replica to be read from. Otherwise, the application layer must decide what to do when the data is not yet up to date enough, e.g. rejecting the read attempt.

2. Kleppmann Chap 6

a) Why should we support sharding / partitioning?

- Scalability over multiple computers for larger datasets.
- Higher bandwidth capacity for high query throughput.

b) What is the best way of supporting re-partitioning?And why is this the best way?(According to Kleppmann).

Hash partitioning using a good hash function can make skewed data uniformly distributed. This usually gives a good load balance (few null values), as long as the hash function is good enough.

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

Global indexing: one big table with index references to smaller partitioned segments of the table. Indexes will be larger, making access slower.
Local indexing: one to one indexing with direct referencing. Results in smaller indexes, meaning access is generally faster.
Therefore, local indexing should be used for very large tables when access times are important.

3. Kleppmann Chap 7

a)Read committed vs snapshot isolation. We want to compare read committedwith 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 releasedwhen doing the read itself.Show how the following schedule is executed using thesetwo approaches:r1(A); w2(A); w2(B); r1(B);c1;c2;

Read commited (write locks are held to the end of the transaction, while read locks are set and released when doing the read itself):

A=0;B=0;

r1(A)=>0; read lock on A is set and released in t1 # read too early (inconsistent state)
w2(A=1); write lock on A is set in t2
w2(B=1); write lock on B is set in t2
r1(B); read rejected because of write lock on B
c1;
c2; write lock on A and B is released in t2

A=1, B=1

Snapshot isolation (use write locks, but reads don't require locks. readers never block writers, writers never block readers)

A=0;B=0;

r1(A)=>0; # read too early (inconsistent state)
w2(A=1); write lock on A is set in t2
w2(B=1); write lock on B is set in t2
r1(B)=>1;

c1;
c2; write lock on A and B is released in t2

A=1, B=1

b) Also show how this is executed using serializable with2PL (two-phase locking).

A=0;B=0;

r1(A)=>0; lock on A is set in t1 # read correctly compared to previous methods
w2(A=1); write rejected because of lock on A
w2(B=1); lock on B is set in t2
r1(B)=>1; read rejected because of lock on B
c1; lock on A is released
c2; lock on B is released

A=0, B=1

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

An example is with the following table:

name	on_call
Alice	true
Bob	true
Carol	false

We try to update the table by setting Alice to false if sum(on_call=true) >= 2 meanwhile doing the same for Bob in a different transaction. This results in both rows being updated at the same time, meaning a write skew occurred:

name	on_call
Alice	false
Bob	false
Carol	false

This effect is caused by a phantom, where a write in one transaction changes the result of a search query in another transaction.

SSI solves this by executing the queries serialized, rather than in parallell. These serialized queries are executed by a stored procedure.

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.

- Your request may have been lost/waiting in queue
- The remote node may have failed/temporarily stopped responding
- The remote node may have processed your request, but the response is lost/delayed

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

A conflict can go unnoticed if two clients write to the same object at the same time. This is because of the network delay being variable between the two clients and the server. Meaning that even though the clocks are synchronized through a server, the server does not know how long it took for the signal to get there. Because of this, errors can occur where one client is thought to have sent the write request before the other, when it actually was the other way around.

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

The problem here is that it relies on two different clocks - the time from lease and the system time. Because system clocks can drift up to 17 seconds in a day, this will stop working completely after less than a day because the time skew will grow larger than 10000 ms, causing the lease to constantly have to be renewed every time it checks for new requests.

5. Kleppmann Chap 9

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

Linearalizability:

- Recency guarantee on reads and writes.
- Implementations of serializability based on two-phase locking or actual serial execution are linearizable.
- Serializable snapshot isolation is by design not linearalizable. It hides concurrent writes.
- Total order of operations
- Makes a system appear as if there is only one copy of the data, and all operations on it are atomic.

Consensus:

- Replicated systems will have inconsistent values with each other.

- Want nodes to agree with each other.
- Eventual consistency converges to the same values.
- Eventual consistency only has exceptions because of system fault (e.g. high latency/traffic).

Ordering:

- Ordering helps preserve causality, e.g. causal dependency between the question and the answer. E.g. a row must first be created before it can be updated.
- If a system obeys the ordering imposed by causality, we say that it is causally consistent.
- Causality defines a partial order, some operations are incomparable.

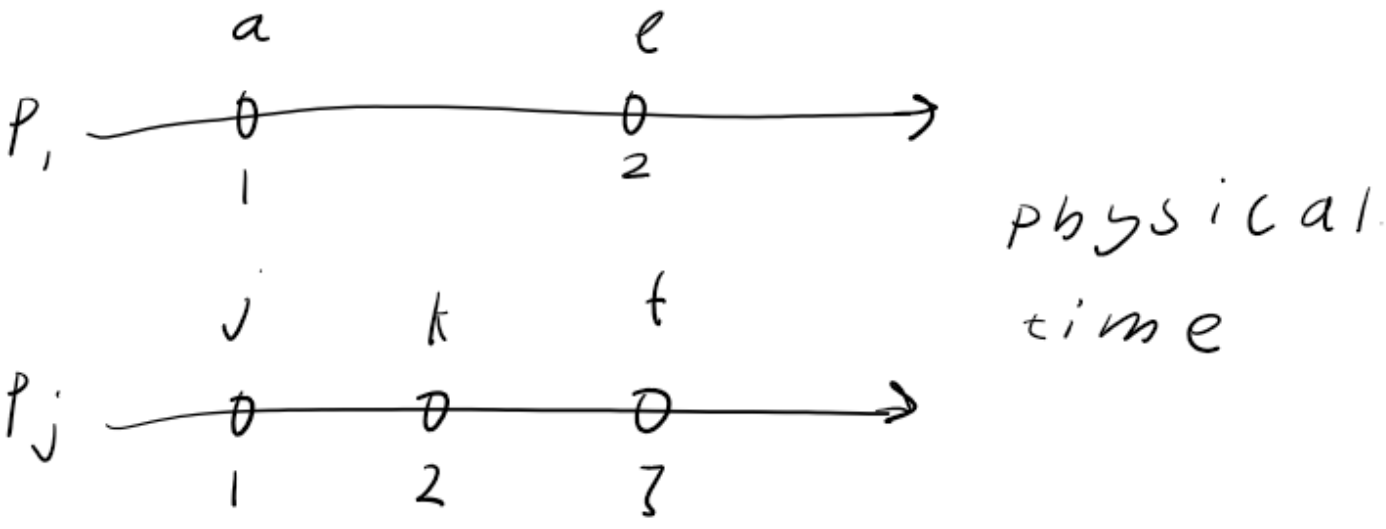
b) Are there any distributed data systems which are usable even if they are not linearizable? Explain your answer.

Netflix is an example of a distributed data system. This does not depend on linearizability because there is no user interaction that require timestamps, and nothing else needs to be synchronized over time. It is simply a database of movies/series that are streamed to a single user at the time.

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.

If e happened before f then we can deduce that L(e) < L(f), but not the opposite. This is because we know there is a chain of events connected between e and f. However, for logical values, we do not know wether or not there is a link between two values. Meaning they could exist in completely different processes with different "event density".



Here, we can see that L(e) = 2, and L(f) = 3 even though they exist in the same physical point in time.

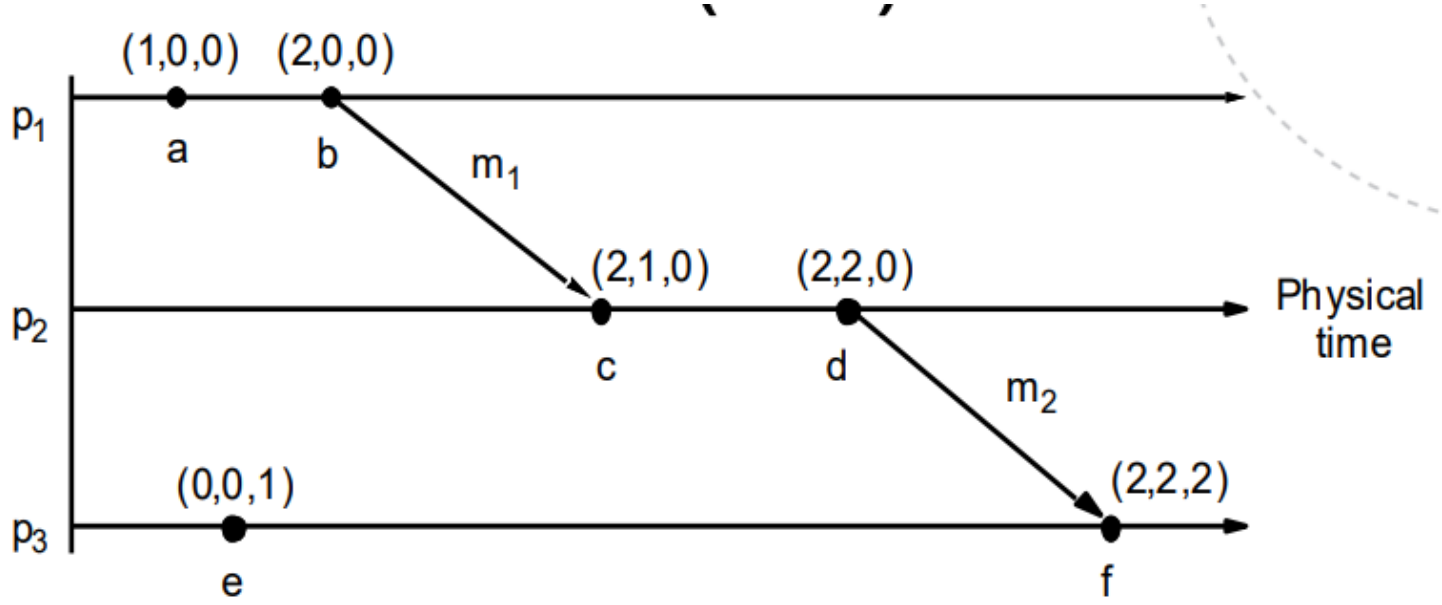
If we use vector clocks instead, we will be able to deduce this:

If e -> e' then V(e) < V(e')

If V(e) < V(e') then e -> e'

Assuming all elements in the vector are lower.

This works because it allows us to keep track of events over different processess, unlike before.



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.

b=(4,0,0)

k=(4,2,0)

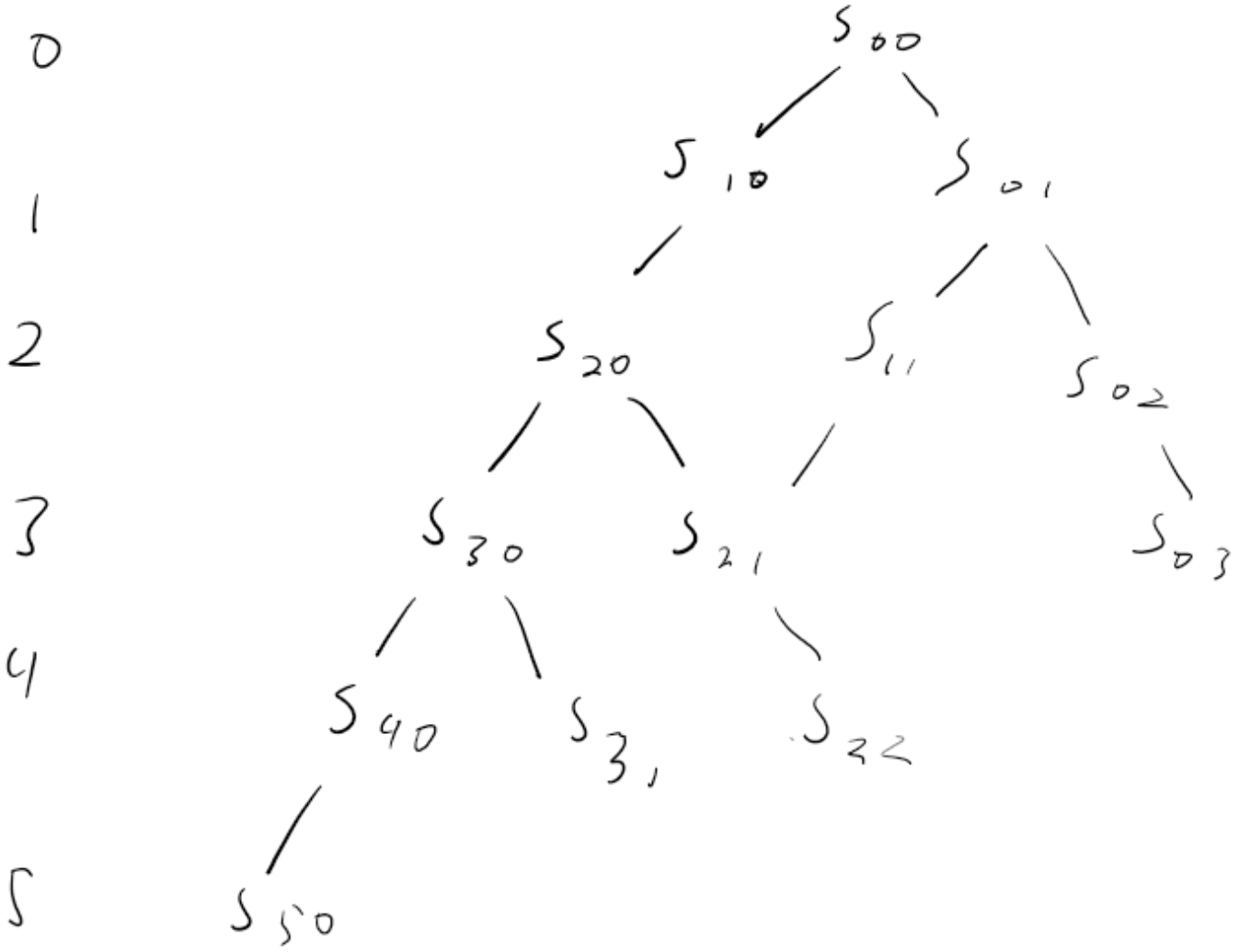
m=(4,3,0)

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 S00. (Sxywhere x is p1's state and y is p2's state)



7. RAFT has a concept where the log is replicated to all participants. How does RAFT ensure that the log is equal on all nodes in case of a crash and a new leader?

Whenever a new leader takes over, the following points are considered:

- Old leader may have left entries partially replicated
- No special steps by new leader: just start normal operation
- Leader's log is "the truth"
- Will eventually make follower's logs identical to leader's
- Multiple crashes can leave many extraneous log entries

A commited log from a leader will stay in the logs of all furure leaders. This guarantees that even the last logs of the old leader will eventually be replicated to all the followers.

8. Dynamo

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

- consistent hashing:

    - Hahing technique used so that when a node is changed in the hash table, only nearby nodes are affected. In contrast, typically, the whole hash table needs to be rearranged if the hash function uses the modulo operator.
- vector clocks:
    - Same data structure as used in task 6a. This is used to keep track of the order of events in a distributed system, where each process can happen at different systems.
- sloppy quorum and hinted handoff:
    - Sloppy quorum and hinted handoff is a technique used to handle failiure of nodes. If a node A needs to be written to and is currently unavailable, the message will be sent to a nearby node instead with a "hint" in the metadata that the message is addressed to node A. When A is back again, the nearby node will then pass the message to A. This hides/abstracts away down time.
- merkle trees:
    - Data structure, specifically a hash tree, is a tree where every leaf node is labelled with the hash value of a data block, while non-leaf nodes are labelled with the hash value of its child nodes. This allows for efficient and secure verification of large amounts of data.
- gossip-based membership protocol:
    - Process of peer-to-peer computer communication based on the way gossip or viruses spread. This works by randomly selecting another computer and sharing information. The rate of expanding network knowledge will be exponential, similar to how viruses grow.