TDT4225 9. Des 2020 Solution (version 11. Des 2020)

1. a)
   User(<u>userid</u>, name, …)
   Activity(<u>actid</u>, userid, starttime, endtime, acttype)
   Trackpoint(<u>trpid</u>, actid, seq, x, y, altitude, timereg)
   Touristsite(<u>tourid</u>, x, y, sitetype)
   Visit (<u>tourid, actid</u>, visittime)
   There are some foreign references in the tables above.

   b)
   For MongoDb it is convenient to have a User document containing all activities and Trackpoints as they are "owned" by the user. It is questionable if it is wise to have Trackpoints as a part of each activity, as there may be very many trackpoints per activity. However, a Touristsite should be its own document, meaning that there will be a two-way reference between Activity and Visit. This is tricky in MongoDb and may be solved in multiple ways. We could have manual references (ids) from Activity to Touristsite, so that to compute which users that are visiting a certain site, requires a manual join by iterating through all Users and Activities.

   c)
   The problem here is the situation with the two-way relationships as described in b). It probably requires a lot of computation every time you need to know who has visited a certain site.

Cited from a very clever student: Some of the disadvantages that come with using a document model for this type of data includes:

- Referring to a nested item dirrectly is difficult. It is doable but requires complicated syntax.
- If the data were to be structured to "imitate" a relational database, with one separate collection each for Users, Activities, Trackpoints and TouristSites, then joining collections together is poorly supported. It is especially troublesome with many-to-many or many-to-one (like we have in our example) relationships in the data. A typical approach is to solve this at the application-level, which just moves the complexity somewhere else and does not solve the problem by itself.
- Once there is enough data, the collection size can exceed the treshold (e.g. MongoDB has a maximum document size of 16 megabytes). Thus we cannot add data to our collection anymore. For this reason, doing nested items can be a bad idea since the collection will quickly grow. In an effort to "distribute" the data across several collections by having separate collections for Users, Activities, Trackpoints and TouristSites, you have to deal with the point above.

2. a)
   31 00000000010000010

33 00000000000000001
34 00000000001111000
63 00000011100000000
73 11000000000000000
87 00111000000000000
88 00000000000001100

b)
31 9,1,6,1
33 16,1
34 10,3
63 5,3
73 0,2
87 2,3
88 14,2

3. a) According to Kleppmann p. 156
   1. Take a consistent snapshot
   2. Copy the snapshot to the follower node
   3. Ship the changes to the follower (log shipping)
   4. Catchup of log

   b) “Happens-before”: Use version vectors or version clocks (Dynamo). They will reflect which nodes know about changes from other nodes. Many students reference the algorithm in Kleppmann, p. 190.

   c) See Kleppmann p. 179- All students got this right.

4. You could use a service like Zookeeper or Etcd. These are services that collect and distributes changes in “meta-data” to programs/processes which are connected to the fault-tolerant service management system. See p. 372 in Kleppmann, Membership services. Dynamo uses a gossip-based protocol for the same purpose.

5. a) This is similar to the doctors-on-call example, thus, *write skew*.
   b) T2 reads the updated A before T1 commits, thus, *dirty read*.
   c) Seems like both transactions try to increment A, thus*, lost update*.
   d) Seems like both transactions try to set A and B to the same value, but they end with A=3 and B=2. Thus*, dirty write.*

6. a) read-committed
   rl1(A); r1(A); ul1(A); rl1(B); r1(B); ul1(B); wl2(A); w2(A); rl2(B); r2(B); ul2(B);
   wl2(B); w2(B); try rl1(B) have to wait; c2; ul2(A); ul2(B); wakeup T1;
   rl1(B); r1(B); ul1(B); c1;

   b) snapshot isolation
   T1 gets timestamp TXid1=1;
   r1(A); r1(B);

T2 gets timestamp TXid2=2;
T2 writes new version of A with timestamp TXid2;
r2(B); T2 writes new version of B with timestamp TXid2;
r1(B) reads old version of B.
c1;
c2;

7. a) p. 341 in Kleppmann:
   Linearizability: total order of operations
   Causality: partial order of operations. “Happens-before” for some of the operations.

   b) Kleppmann p. 330-31: Locking and leader election. Constraints and uniqueness guarantees. Cross-channel dependencies. Some students say that flight reservations is a good example. However, in Kleppmann this is said to not need linearizability, as a double booking is solved by some money, bonus points, free beers, etc 😊

8. a) Dybvik p. 37. Leveled compaction, universal compaction and FIFO compaction. Leveled compaction is the standard one, where MemTables are flushed directly to a Level0 SSTable. Thus, there are multiple Level0 SSTables. L0 SSTables are merged into L1. L2 are merged into L2, and so on. Each level is 10 times larger than the previous layer.

   b) Dostoevsky has addressed this issue. They discovered that too much work goes into merging the different layers of SSTables. Thus, they have a tunable merging activity where point lookup cost is traded with space amplification and range lookups. They say that it is only necessary to merge the largest level and SSTables. Some students mention write stops and write stalls as described by Dybvik. Some students also mention Warlo’s auto tuner. I’ll notify Hans-Wilhelm about your statements 😊

9. Sloppy quoroms and hinted handoff. Dynamo paper.

10. Coulouris says Lamport says we cannot synchronize clocks perfectly across a distributed system. We cannot in general use physical time to find out the order of any arbitrary pair of events occurring within it.
    Kleppmann: Incorrect clocks easily go unnoticed.
    Some students mention Google spanner as an exception.

11. a || m
    b || n
    c || n
    d || n
    d || o
    d || p
    e || q

12. The situation is that probably there will be local data that is updated at each site. Thus, the other regions might have very few updates to data that “belongs” to other sites. Exceptions to this will probably be “shared data”, which again probably is only updated from an adm

site. Kleppmann: The simplest approach is to avoid conflicts. Let all updates to a particular item go through the same leader.