TDT4225 – Exercise 4

Håkon Ødegård Løvdal

1: Algebra with filter

	R: 1 000 000	A: 1 000 000	B: 10 000 000
Record Length:		**400B**	**600B**
Fetched B	**600B**	**300B**	**300B**
Volume MB	**600MB**	**400MB**	**6000MB**
NettoVolume MB		**300MB**	**3000MB**

Without filter:

$$n = \left\lceil \frac{VA}{M} \right\rceil = \left\lceil \frac{300}{20} \right\rceil = 15$$

$$V_{nl}^{J} = VA + nVB + VR$$
$$= 400 * (15 * 6000) + 600 = 91000MB = 91GB$$

With filter:

Les A	400 MB
Skriv A’	300 MB
Les B	6000 MB
Skriv B’ 1/3	1000 MB
Les A’ og B’	1300 MB
Skriv A * B	2000 MB
Total:	**11000 MB = 11GB**

2: Parallel algebra

a) Describe the different partitioning methods used in parallel algebra

The book introduces the choices vertical- or horizontal fragmentation. The book does not elaborate much about the vertical fragmentation, so I will elaborate about the horizontal fragmentation here. There are three location selection mechanisms mentioned, and these are:

- Round Robin: This is a very well know algorithm, and I assume a fair knowledge about it. Basically, this algorithm places records in all nodes in a circular manner. Each node gets the same number of records.
- Hashing: The records are places in an address given by a hash formula. The primary key works as an input argument to the hash formula. This is the most convenient way to place the records.
- Value ranges: In this method, a node is given a specific value range to be responsible for. This range is a range within the primary key attributes.

b) Why is hashing a very good method?

Hashing is a preferred method because it makes lookups fast, and in most cases it will distribute the data evenly. This makes it preferable for using with parallel execution. Using a hash-formula for distributing placement of records is efficient. This is because it does not require any form of indexing (the hash formula must be good, to ensure a good distribution).

3: Dynamo

Explain the following concepts/techniques used in Dynamo:

- consistent hashing

I assume a basic knowledge about circular hashing, and do not elaborate this any further. Dynamo does not implement consistent hashing directly. Instead they implement a variant of consistent hashing; rather than mapping a node *n* to a point in the circle, the nodes get mapped to multiple points in the circle. To achieve this, it introduces the concept of "virtual nodes". For the system a virtual node is represented as a node, but each physical node may contain multiple of these virtual nodes. This is done to solve the problem with non-uniform data and load distribution. If a node becomes unavailable, its load may be distributed between virtual nodes. Also it addresses the problem that the generic algorithm is oblivious to heterogeneity of the hardware/performance in nodes.

- vector clocks

In order to capture the causality of objects, Dynamo uses vector clocks. In it is simplest manner this vector clocks are a list of [node, counter]. Each vector clock linked to an object. In other words, the system has an object and its associated vector clock. Keep in mind that each access to data is done on a new and immutable version of the data.

As long as the same node accesses the object, a new object with an incremented counter will be created. This makes it easy to determine whether or not an object descends from an object or not. If two or more nodes tries to update an object the system will branch it out and create two separate objects, with an added vector clock for the node. When a new node wants to read, then all branches will be returned as a part of the context. This returned value is a summary of all the clocks in all branches.

- sloppy quorum and hinted handoff

Dynamo uses "sloppy quorum" to ensure it is available during server failures and unexpected network events. The sloppy quorum means that all read and write operations is done on the first *N* healthy nodes, from a preferences list. These *N* nodes are not always the first *N* nodes found when traversing the consistent hash ring.

The hinted handoff is implemented to ensure that read and write operations do not fail during node failure or network failures. If a node is down during a write operation, then the operation will be sent to another node. This node will have a hint in the metadata, that tells which node it originally should reside within. When the original node is up again, the backup node could send the data to the original node.

- merkle trees

A Merkle tree is a hash tree. All the leaves in the tree are hashes of individual keys, but the parent of leaves are hashes of their children. This makes it possible to check if a root or branches are equal without loading the entire dataset. This is because if the hash values are equal, the leaf values will also be equal.

- gossip-based membership protocol

The gossip-based membership protocol is implemented to ensure that each node contacts a randomly selected peer every second to ensure that the two nodes have a similar and persisted membership change histories. By doing this one avoid having a central registry for node status information.

4: RamCloud

a) How does RamClouds ensure "durability" of data?

RamClouds ensures "durability" by using buffered logging. A copy of an object is stored in DRAM. When an object is modified, all the changes are logged to two or more different servers than the origin server. On these other servers, the logs are stored in DRAM and transferred to disk in an asynchronously manner to ensure maximum disk bandwith.

b) How does Ousterhout argue that RamCloud's potential to support ACID transactions is better than for traditional disk-based distributed databases?

Ousterhout argue that RAMCloud's extremely low latency will give it a benefit. Due to the low latency, it may enable a higher level of consistency than traditional systems of equal, comparable scale. Ousterhout argue that concurrency is determined by the time a transaction uses in the system, and the overall rate transactions arrive in the system. These longer execution times increase the degree of concurrency.

5: Facebook TAO

a) How does Facebook TAO solve the problem that the social graph spans the whole world, and that the data should be close to the user?

In order to solve the problem with global scaling of the social graph, TAO uses a multilayered architecture. Firstly, it has a leader/follower configuration that allows it to scale and handle high workload. Each follower contains cached data and every client request is connected to these followers. Should such an event occur, that the follower does not contain a cached value, a local master is contacted. This master is geographically close, and contains all answers to reads. Write requests are sent to a master. This configuration is again bundled into a *region.* This is to ensure small intra-region latency. Each region master contains a full copy of the social graph.

6: Google Spanner

a) How are TimeStamps used in Spanner's transactions?

Google Spanner uses a time API called TrueTime. TrueTime returns an interval, *TTinterval*, with the earliest and latest possible time for a moment. These time references are generated by GPS and atomic clocks. The reason for doing this is the fact that GPS and atomic clocks have uncorrelated failures. So one is always guaranteed a timestamp. Spanner

can use these timestamps to guarantee that a transaction A happened before transaction B. This means that users can see data in the same order as it happened in the system.