Data Intensive Computing - Review Questions 1

Daniele Montesi, Francesco Staccone

1. Explain how a file region can be left in an inconsistent state in GFS?

   In GFS, failed mutations (writes or record appends) lead to chunks that are undefined and inconsistent. As a result of this inconsistent state of the file region, different clients may see different data at different times.
   For what concerns the writing mutation, the inconsistency might happen after the primary replica performs the request and forwards it to all secondary replicas. In case of errors in any of the replicas mutations, the modified region is left in an inconsistent state and the client request is considered to have failed.
   For what concerns the record append mutation, it follows a similar control flow of the write mutation. If a record append fails at any replica, the client retries the operation. As a consequence, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part, leaving the region in an inconsistent state.

2. Briefly explain how HopsFS uses User Defined Partitioning (UDP) and Distributed Aware Transaction (DAT) to improve the system performance (e.g., CPU usage, network traffic and latency)?

   With the aim of improving the performance of the file system operations, HopFS uses User Defined Partitioning to distribute data across different database nodes: the namespace is partitioned such that the metadata for all immediate descendants of a directory reside on the same database shard for efficient directory listing. This allows Partition Pruned Index Scans, meaning that scan operations are localized to a single database shard, reducing network load, latency and CPU usage.
   Moreover, HopFS uses Distributed Aware Transaction to choose which Transaction Coordinator handles which file systems operations: the transaction is started on the database shard that stores all/most of the metadata required for the current file system operation, reducing network load and latency even more.

3. Show with an example that in the CAP theorem, we can have only two properties out of Consistency, Availability, and Partition Tolerance at the same time.

   We prove the CAP theorem with a *Reductio ad absurdum*.
   Let's assume that we have a system that assures property of Consistency, Availability and Network Partition. This means that if I read from any node, I should be able to receive the most recent value (Consistency) with no waiting time (Availability) and errors (Network partition). However, assume that I'm performing a Write to a node X1 and a subsequent read to a node X2 for the same resource. Since the network is partitioned, I won't receive immediately the most recent value at node X2, being the network partitioned. But that means that the aforementioned system is not consistent.

4. How does BigTable provide strong consistency?

   BigTable provides strong consistency assigning a tablet to only one tablet server. A tablet server manages thousands of tablets.
   During the period of update or replication process, data is locked to ensure that no other processes are updating the same data. In case of failures, a tablet can be assigned to no server and the services become unavailable to the client until a new tablet server assignment is performed.
   Moreover, the consistency is assured by the replication that is managed by the underlying distributed file system GFS.

5. Write a simple query in Cypher for each of the following requests:

   - Match a Person called "John Doe"

     ```
     MATCH (n:Person {name: 'John Doe'})
     RETURN n;
     ```

   - Find FRIENDS_OF "John Doe"

     ```
     MATCH (:Person {name:'John Doe'})-[:FRIEND_OF*]->(p:Person)
     ```

```
RETURN p;
```

- Count ”John Doe” ’s direct acquaintances

```
MATCH (:Person {name:'John Doe'})-[:FRIEND_OF*1]->(p:Person)
WITH count (p) AS directAcquaintancesCount
RETURN directAcquaintancesCount;
```