Data Intensive Computing - Review Questions 1

Sri & Nik - 5th Sept, 2019

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

When a client tries to modify the existing file, the client gets the lease to all the chunk servers where all the replicas of the particular file are. Then when the data is properly pushed to the buffers of all the chunk servers, the client sends a written request to the primary replica which then serializes and applies the requested modifications. Once this has successfully done, the primary forwards the write request to the secondaries. An inconsistent state is reached in this stage when the secondaries fail to update and left behind in the previous version. When the primary does not receive an acknowledgment from all the secondaries, it sends an error message to the client. The client should request these modifications again to fix the inconsistent state of the file.

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)?

By using UDP and DAT the client knows which node it needs to send a request to in order to get the result of a query. This saves:

- network resources as only 1 requests need to be sent instead of a request to every node
- CPU usage as only a single node needs to do operations
- latency as you don't have to wait on other nodes which will not have the answer anyway

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.

Assume that we have 2 servers, S1 and S2, and a client C. The servers S1 and S2 initially store a piece of information x=0. If the system is consistent, available and partition tolerant, we can assume that in some situations S1 and S2 cannot communicate with each other (due to the partition tolerance).

If C sends a written request to S1 to store x=1, and then a read request for x to S2, there will be an inconsistency. The system, therefore, would only be available and partition tolerant. The system could also throw an error (or have the client wait) due

to the fact that S1 cannot tell S2 about the write request. But then the system would not be available.

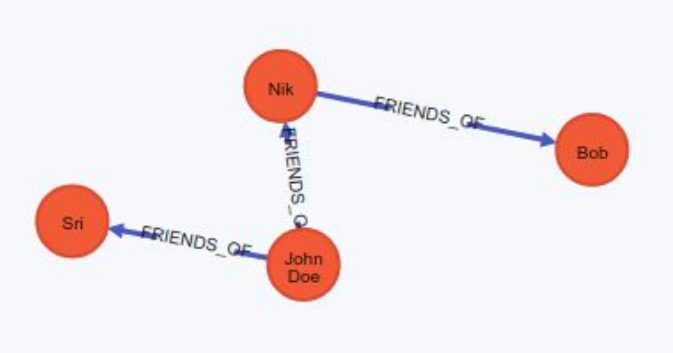
4. How does BigTable provide strong consistency?

BigTable uses Chubby a high available and distributed lock service to maintain consistency across the replicas. Chubby maintains a group of 5 replicas and is active only when the majority of the replicas are available and communicating with each other. One of these active replicas act as the master and handles the requests from the client. Chubby uses the Paxos algorithm to maintain consistency across the replicas in case of any failure[1].

In addition to that, BigTable as a service could provide strong consistency by the 'Single Cluster Routing' app-profile configuration. With this configuration enabled, all the client's requests are forwarded to the same cluster and hence eliminating the chance for inconsistent updates resulting due to the delay in writing huge changes to other clusters[2].

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

```
create (JohnDoe:Person {name:'John Doe'}),
(Sri:Person {name:'Sri'}), (Nik:Person {name:'Nik'}),
(Bob:Person {name:'Bob'}),
(JohnDoe)-[:FRIENDS_OF]->(Sri),(JohnDoe)-[:FRIENDS_OF]->(Nik),
(Nik)-[:FRIENDS_OF]->(Bob)
```



- Match a Person called "John Doe"

```
MATCH (n:Person) WHERE n.name = 'John Doe' return n;
```

n
(0:Person {name:"John Doe"})

- Find FRIENDS OF "John Doe"

```
MATCH (:Person {name:'John Doe'}) - [:FRIENDS_OF] ->
(friend:Person)  return friend;
```

friend
(2:Person {name:"Nik"})
(1:Person {name:"Sri"})

Query took 17 ms and returned 2 rows. Result Details

- Count “John Doe”’s direct acquaintances

```
MATCH (:Person {name:'John Doe'}) --> (x)  return count(*);
```

count(*)
2

Query took 39 ms and returned 1 rows. Result Details

REFERENCES:

1. Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., & Burrows, M. et al. (2006). *Bigtable: A Distributed Storage System for Structured Data*. *Google AI*. Retrieved 5 September 2019, from https://ai.google/research/pubs/pub27898
2. *Overview of Replication | Cloud Bigtable Documentation | Google Cloud*. (2019). *Google Cloud*. Retrieved 5 September 2019, from https://cloud.google.com/bigtable/docs/replication-overview