Data Intensive Computing – Review Questions 1

Adriano Mundo, Riccardo Colella

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

GFS has a consistency model that supports distributed applications, but the state of a file region depends on the mutation's type.
Due to a failed mutation (write or record append) a file region reaches an inconsistent and also undefined state. As a consequence, different clients may see different data at different times.
In the case of a record append failure at any replica, the client retries the operation. Thus, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part.
In the case of a write operation an error can be encountered at any of the replicas. Any error is reported to the client and the write may have succeeded at the primary and an arbitrary subset of the secondary replicas. If the write fails at the primary, it would not have been assigned a serial number and forwarded. Hence, the request is considered to be failed and the region reach an inconsistent state.

2. **Briefly explain what are the differences between GFS and FDS?**

GFS has been designed to provide a scalable distributed file system for large distributed data intensive applications. It provides fault tolerance and delivers high performance to a large number of clients. It is designed around some key observations: component failures are the norm rather than the exception, files are huge, most files are mutated by appending new data rather than overwriting, it is optimised for streaming access.
FDS is a high-performance, fault-tolerant, large-scale datacenter storage designed under the unrealistic assumption that datacenter bandwidth is abundant. FDS returns to the flat storage model: all compute nodes can access al storage with equal throughput, there are no "local" disks. In this way, disk-to-disk bandwidth is extremely high, and it is possible to obtain fast recovery from disks and machine failures.
GFS files are divided into fixed-sized, immutable and unique chunks, which are a single unit of storage. Its architecture is composed by a single master (coordinator), chunkservers (storage) and application client that sends the requests to the master about which chunkservers to contact.
FDS has data stored in logical blobs which are divided into constant sized tracts. Its architecture move data closer to the computing node and there is a process called tractserver which manages every disk and a metadata server which coordinates the cluster. It collects the tract locator table (TLT), which is a list of active tractservers.
In FDS tracts are arbitrary small thanks to the fact that the metadata server is in the critical path only when a client process starts, while GFS require larger chunks to reduce load on the metadata server.

For what regards consistency, there's weak consistency in the GFS' clients because they need to handle garbage entries in files while FDS can provide strong consistency by using chain replication and guarantees updates to all individual tracts.
GFS has a centralized master that keeps all metadata in memory as this approach is a limitation because as the contents of the store grow, the metadata server becomes a centralized scaling and performance bottleneck, while FDS use deterministic placement to eliminate scaling limitation of current blob store metadata servers. Also, it distributes the blob metadata using metadata tracts, which further minimizes the need to centrally administer the store.

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 can assume for contradiction that we have a system that is consistent, available and partition tolerant. So, we can partition our system in P1 and P2 and we do it. Then our client may want to write x in P1. Our system is available, so P1 must respond. However: network is partitioned, so P1 cannot replicate its data to P2. Then our client may want to read from P2, but since the system is available P2 must respond and since the network is partitioned P2 cannot update its value from P1, so it returns x_old. So we have P2 that returns x_old after having the client that already wrote x to P1. This is inconsistent. We assumed a system that was consistent and by contradiction we demonstrated that such a system cannot exist

4. **How does BigTable provide strong consistency?**

BigTable is a distributed storage system for managing structured data which has been designed to scale to a very large size.
BigTable provides a strong consistency model since row operations are atomic, and tablets are served by one tablet server at a time. Thus, only one tablet server is responsible for a given piece of data.
Replication is handled by the GFS layer, which is the file system BigTable is built on. In particular, it relies on a highly available and persistent distributed lock service, Chubby, which consists of five active replicas. Chubby uses the Paxos algorithm to keep its replicas consistent in face of a failure.
Additionally, for some cases, Big Table as a Cloud can also provide strong consistency using the single cluster routing app profile configuration for the read-your-writes consistency (never read data that is older than its most recent writes). This is achieved because all reads and writes are sent to the same cluster.

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

   i. Match a Person called “John Doe”
   ii. Find FRIENDS OF “John Doe”
   iii. Count “John Doe”’s direct acquaintances

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

MATCH (:Person {name: ‘John Doe’}) – [:FRIENDS_OF] -> (p: Person)
RETURN p;

MATCH (:Person {name: ‘John Doe’}) -> (n: Person)
RETURN count(n)
```