Data Intensive Computing - Review Questions 1
Deadline: Sep. 6, 2025

Group AKA
Ahmad Al Khateeb
Aleksandra Burdakova
Kusumastuti Cahyaningrum

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

In GFS, a file region can be left in an undefined state when concurrent successful mutations occur to the same file region (to the same chunk). Another scenario is when a primary replica crashes mid-write, or a secondary replica fails to apply updates consistently, and replicas diverge temporarily.

2. The GFS paper emphasizes "treating failures as the norm". What specific design choices embody this philosophy?

a. GFS is designed with the assumption that component failures are normal. Components include disk, memory, network, and power. These components shall be constantly monitored, and possible faults shall be detected and handled frequently.
b. Replication is handled by dividing files into 64 MB chunks, where each chunk is replicated across different machines (chunkservers) and racks.
c. The centralization of the master is only valid for storing metadata (namespace, file-to-chunk mappings), maintaining operational logs, and scaling throughput, without having to handle the operations from clients. However, it also helps increase the fault-tolerance by detecting missing or corrupted replicas and automatically trigger re-replication from healthy replicas, while continuously rebalancing the replicas to spread the load.
d. Detecting stale replicas (when a chunkserver misses updates, and the replica becomes stale), and garbage-collecting these replicas.
e. Optimization of large and append-heavy workloads by focusing on huge files with concurrent appends and mostly sequential reads.
f. Relax strict file system semantics: extend and adapt the standard interface to boost performance and scalability.
g. Build in fault tolerance: use replication, checksumming, monitoring, and automatic repair for reliability.

3. Using one example show that in the CAP theorem, if we have Consistency and Partition Tolerance, we cannot provide Availability at the same time.

Example (BigTable -> guarantees Consistency over Availability):
Suppose we have two nodes, X and Y, in a distributed database.
X network partition occurs, so X and Y cannot communicate.

To maintain consistency, read/write operations are possible to only be done on one of X or Y (since they have inconsistent data, and can't communicate, but the system continues to function since partition is tolerated). So, one of the two nodes will be unavailable, in that case if the other node performs an update on data, consistency is broken, or the write-operations are rejected, and in this case the system is unavailable (in the sense that it's not possible to perform write-operations).

4. In the BigTable, what role do SSTables play in enabling efficient reads?

Storing data as persistent, ordered, immutable key-value pairs, simplifying lookups.
Using indexes, caching, and Bloom filters to minimize disk seeks.
Support mapping of complete SSTables into the memory without disk seek.
Supporting locality groups and compression to reduce unnecessary I/O.
Applying compactions to limit the number of SSTables read during queries.

5. The Cassandra paper emphasizes decentralization with no master node. What challenges does this design create compared to BigTable's single master?

Data partitioning & location: Cassandra must use consistent hashing and gossip protocols, which can cause uneven (non-uniform) data/load distribution.

Failure handling: Failure detection and recovery are decentralized, making coordination harder than BigTable's master-managed recovery.

Load balancing: Redistributing data across nodes is more complex without a master to oversee balance.

Metadata management: Schema and metadata updates require distributed coordination, often with external tools.

Overall complexity: Decentralization demands sophisticated distributed algorithms, while BigTable simplifies this with a central master.

6. Compare BigTable's "tablet" model with Cassandra's "consistent hashing" partitioning. Which model better supports community-controlled clusters, and why?

Comparison:

Feature	BigTable (Tablet Model)	Cassandra (Consistent Hashing)	Why Cassandra Supports Community-Controlled Clusters

Architecture	Centralized master manages tablets	Decentralized, peer-to-peer, masterless	No single point of control; all nodes are equal
Data Partitioning	Tablets assigned by master	Consistent hashing distributes data across nodes	Nodes can autonomously manage their data ranges
Failure Handling	Master coordinates recovery	Gossip protocol and replication handle failures	System continues despite node failures without central control
Scalability	Master assigns new tablets; adding nodes requires coordination	Nodes can join/leave dynamically with localized impact	Supports incremental, community-driven scaling
Administrative Control	Centralized; master performs metadata and load management	Distributed responsibilities across nodes	No central authority needed; shared governance possible
Best Use Case	Tightly controlled environments	Community-controlled or peer-managed clusters	Decentralization and autonomy fit collaborative environments

The model that better supports community-controlled clusters is Cassandra, because:

1. Decentralized governance: no single master controlling the cluster.
2. Fault tolerance: system continues even if some nodes fail.
3. Scalability & autonomy: nodes can be added/removed independently.
4. Shared responsibility: management is distributed, not centralized.

Why not BigTable? Its single master makes it better for centralized, tightly controlled environments, not community-managed ones.

7. Imagine an NGO storing health data in Cassandra. How would you configure replication and consistency policies to balance patient privacy, availability, and sustainability?

Privacy:

- Use datacenter-aware replication to keep data within compliant regions and protect patient privacy.

- Use encryption technology to protect patient data (client-to-node and node-to-client encryption).
- Use role-based access control to the data - grand minimum amount of data to the user that really needs it.

Availability:

- Leverage Cassandra's decentralized architecture, multi-datacenter replication, and Gossip Accrual failure detection to ensure fault-tolerant, continuous access.
- Commit logs guarantee durability during node failures.

Consistency:

- Writes: Use QUORUM to ensure most replicas confirm changes, maintaining data integrity.
- Reads: Use QUORUM for critical data to get up-to-date results; lower consistency (e.g., ONE) can be used for non-critical queries to improve latency.

Sustainability:

- Run on commodity hardware and scale incrementally with consistent hashing.
- Optimize replication and consistency levels to balance resources.
- Rely on compactions and decentralized operations to reduce operational overhead.