Muhammad Abdullah | Roll No: 221980005

Big data Analysis :Assignment 1

Case Study Title: Planning and Managing a Scalable Hadoop Cluster for National Census Data

Part A: Conceptual (20 Marks)

Hadoop Component Roles:

• **NameNode:** This is the master server that stores the metadata of the HDFS. It knows which block of data is stored on which DataNode. For the 2023 census data, it keeps records of all file names, directories, and the location of their blocks.

Explanation: In a Hadoop environment, the NameNode acts like the master brain. It does not store the actual data but stores the file system namespace, meaning it knows where every piece of data is located. This is crucial during data analysis, as the system quickly identifies which nodes to retrieve data from.

• **DataNode**: These are the worker nodes that actually store the data blocks. Each DataNode stores a part of the census data like text files, images, or coordinates.
Explanation: Each DataNode is like a warehouse storing chunks of the entire dataset. These nodes work together to hold large datasets such as high-resolution census images or geographical maps. The distributed nature ensures quick access and redundancy.

• **Secondary NameNode**: It is not a backup NameNode. Instead, it helps the main

NameNode by creating checkpoints (snapshots) of the metadata to reduce the load on NameNode.

Explanation: Contrary to the name, it is not a backup server. Instead, it assists the NameNode by compiling changes in the file system into regular checkpoints. This reduces restart time and prevents memory overload on the NameNode.

• **Rack Awareness**: Hadoop stores block replicas on different racks to make sure that even if one rack fails, the data is still available from another rack. This increases fault tolerance.

Explanation: Hadoop's intelligent placement of data copies across different physical racks ensures that a rack-level hardware failure doesn't affect data availability. This provides fault isolation and improves data availability.

• **Checkpointing**: It is the process of saving the current state of the file system metadata. It helps in restarting the NameNode quickly if it fails.

Explanation: Checkpointing helps in reducing the time it takes to restart the NameNode. In case of failure, the system doesn't need to read the full log, only the latest checkpoint, saving time and computational effort.

2. Why HDFS is Preferred for Large-Scale Projects

HDFS is particularly efficient for batch-processing systems where data is written once and analyzed repeatedly. The large block size (e.g., 256 MB) reduces the number of files the NameNode has to manage. Furthermore, using inexpensive hardware with HDFS makes it accessible for large organizations or governments to manage big data without huge infrastructure investments.

1. Handles Large Data Efficiently:
HDFS can handle files in the range of terabytes to petabytes by breaking them into fixed size blocks (e.g., 256 MB).
These blocks are then distributed across multiple nodes, making large-scale data storage and processing practical.
This makes it ideal for projects like a national census that generate massive amounts of structured and unstructured data.

2. Fault Tolerance:
HDFS stores multiple replicas (default is 3) of each data block on different nodes and racks.
If one node crashes, Hadoop automatically retrieves the data from another replica, preventing data loss.
This ensures uninterrupted access and high reliability in a distributed environment.

3. Horizontal Scalability:
HDFS allows seamless addition of new DataNodes to increase storage or performance without downtime.
This means the system can grow as data grows, simply by plugging in more hardware. It enables organizations to scale based on future demand without redesigning the infrastructure.

4. Cost-Effective:
HDFS is designed to run on commodity hardware, reducing the cost of building and maintaining infrastructure.
It eliminates the need for expensive high-end storage systems typically used in traditional setups.
This makes it accessible even for government or academic institutions with budget limitations.

5. Optimized for Write Once, Read Many:
HDFS is optimized for scenarios where data is written once and read many times, such as log processing or census analysis.
This design reduces complexity in data consistency and increases read performance for analytical workloads.
It's especially useful in projects where historic data is frequently analyzed but rarely modified.

Part B: Analytical Simulation & Numerical Problem Solving (80 Marks)

Q1. HDFS Block Planning and Replication (15 Marks)

Given:

HDFS Block Size = 256 MB

Replication Factor = 3

Data Type Size

Demographic	(TB)	Size (MB)	Blocks (Size ÷ 256 MB)	blocks
Text	100	100 × 1024 × 1024 = 104,857,600 MB		
Images	60	60 × 1024 × 1024 = 62,914,560 MB		
Household Geo-coordinates	20	20 × 1024 × 1024 = 20,971,520 MB		

Total Blocks Before Replication:

409,600 + 245,760 + 81,920 = **737,280**

Total Blocks After Replication (×3): 737,280 × 3 = **2,211,840 blocks**

Total Storage Required (with Replication): Total original data = 100 + 60 + 20 = 180 TB 180 TB × 3 = **540 TB**

104,857,600 ÷ 256 = **409,600 blocks**
62,914,560 ÷ 256 = **245,760 blocks**
20,971,520 ÷ 256 = **81,920 blocks**

By replicating blocks, we ensure fault tolerance at the cost of tripled storage (540 TB)

Q2. NameNode Metadata Analysis (15 Marks)

Each file block has metadata, such as permissions, replication count, and block locations. Since the metadata is small in size compared to the data itself, a NameNode with 32 GB RAM can handle millions of blocks. This analysis confirms that memory planning for NameNode is sufficient and optimized for census-scale workloads.

Assumptions:

Metadata per block = 250 bytes

NameNode RAM = 32 GB = 32 × 1024 = **32,768 MB**

Metadata memory before replication:

Total blocks = 737,280

Metadata = 737,280 × 250 bytes = 184,320,000 bytes
Convert to MB: 184,320,000 ÷ (1024 × 1024) ≈ **175.78 MB**

Metadata memory after replication:

Replicated blocks = 2,211,840

Metadata = 2,211,840 × 250 bytes = 552,960,000 bytes

Convert to MB: 552,960,000 ÷ (1024 × 1024) ≈ **527.34 MB**

Available RAM vs. Required Metadata Memory:

Required: ~**527 MB**

Available: **32,768 MB**

NameNode can efficiently store metadata for all blocks with less than 2% of its available RAM.

Q3. Cluster Size Planning (20 Marks)

Capacity planning is vital to ensure the cluster is neither under-resourced nor excessively provisioned. Adding extra nodes helps maintain the replication factor during node failures. Additionally, forecasting for future expansion (e.g., an extra 120 TB of data) ensures scalability and uninterrupted service.

Each DataNode has **4 TB** usable space.

a) Total DataNodes needed: 540 TB ÷ 4 TB = **135 nodes**

b) For fault tolerance (1 node failure): Add 1 more node → **136 nodes** c) For

additional 120 TB (replicated = 360 TB): 360 ÷ 4 = **90** more DataNodes **Q4.**

Advanced Scenario Simulation (30 Marks)

Hadoop's self-healing mechanism ensures resilience. When nodes go offline, the system marks their blocks as under-replicated and starts replicating them from healthy nodes. MapReduce is tolerant to failure because it relies on the distributed nature of HDFS—tasks can be rescheduled to other nodes with data replicas, minimizing job failure risks.

a) If 5% of DataNodes go offline:

5% of 135 = **6 nodes**
Under-replicated data = 6 nodes * 4 TB = **24 TB**

b) Hadoop Strategy:

Hadoop continuously monitors block health. When nodes return online, it will automatically replicate the under-replicated blocks to maintain the desired replication factor (3).

c) MapReduce Fault Handling:

If a block is temporarily missing due to node failure, Hadoop's JobTracker will try to assign the task to another DataNode that has a replica. If no replica is found, the task waits or fails. However, Hadoop's replication usually prevents such failures.