

## **Directed Works TD 2: Advanced Hadoop Architecture & MapReduce Paradigms**

### **Learning Objectives:**

- Evaluate the architectural mechanisms of Hadoop (YARN resource allocation, NameNode metadata checkpointing, and Speculative Execution).
- Design and trace parallel MapReduce algorithms for non-trivial data structures (Inverted Indexes and Multi-Metric Aggregations).
- Analyze the mathematical and algorithmic constraints governing Combiners and Partitioners in distributed pipelines

### **Exercise 1: Multiple Choice Questions (QCM), System Architecture & Mechanics**

*Select the correct answer for each question and justify your choice in 1–2 sentences.*

#### **Q1. Role of the Secondary NameNode in HDFS:**

What is the primary function of the Secondary NameNode in a standard Hadoop HDFS cluster?

- ☐ It acts as a real-time hot standby that automatically takes over if the primary NameNode crashes.
- ☐ It executes MapReduce tasks whenever DataNodes run out of memory.
- ☐ It periodically merges the edits log with the fsimage file to prevent the edit log from growing too large.
- ☐ It stores duplicate copies of actual data blocks across different racks.

#### **Q2. Architectural Evolution: MRv1 vs. YARN:**

In Hadoop v2 (YARN), which component inherited the application lifecycle and job monitoring responsibilities previously handled by the single JobTracker in MRv1?

- ☐ The ResourceManager
- ☐ The ApplicationMaster (per job)
- ☐ The NodeManager
- ☐ The NameNode

#### **Q3. Mathematical Constraints on Combiners:**

A developer attempts to compute the Average Salary per Department using a local Combiner. They configure the Reducer class to run directly as the Combiner. Why will this yield incorrect results?

- ☐ Combiners cannot process numerical values in Hadoop.
- ☐ The AVERAGE operation is non-associative and non-commutative, causing arithmetic bias when applied to partial counts.
- ☐ Combiners run only on the Master node, where data is not available.
- ☐ The Reducer signature (Key, List(Value)) is incompatible with the Mapper output.

#### **Q4. Handling Slow Nodes via Speculative Execution:**

During a MapReduce job execution on 100 nodes, 99 nodes finish in 2 minutes, but 1 node is running extremely slowly due to a degraded hard drive controller ("straggler"). How does Hadoop handle this?

- ☐ It aborts the entire job and throws a HardwareException.
- ☐ It launches a duplicate ("speculative") task on another node and accepts whichever task finishes first.
- ☐ It pauses the 99 fast nodes until the slow node catches up.
- ☐ It dynamically reduces the HDFS block size of the slow node.

#### **Q5. Partitioner Logic & Routing:**

In a MapReduce job configured with 4 Reducer tasks, which component determines exactly which Reducer will receive a given intermediate key-value pair (K2, V2)?

- ☐ The NameNode using block location tables.
- ☐ The HashPartitioner evaluating  $\text{Math.abs}(\text{key.hashCode()}) \% \text{numReduces}$ .
- ☐ The Combiner running locally on the DataNode.

- The InputFormat splitting mechanism.

## Exercise 2: Algorithmic MapReduce – Inverted Index for Search Engines

**Scenario:** A web search engine indexes unstructured text files stored in HDFS to enable fast document retrieval. To do this, it builds an **Inverted Index**: a data structure mapping each unique word to a list of Document IDs where the word appears, along with its occurrence frequency in each document (e.g., word  $\rightarrow$  ["doc1.txt:2", "doc2.txt:5"]).

You are given **3 input documents** processed across 3 parallel mappers:

- **Document 1 (doc1.txt):** hadoop spark hadoop
- **Document 2 (doc2.txt):** spark python spark
- **Document 3 (doc3.txt):** hadoop python mapreduce

1. **Function Signatures:** Write down the formal type signatures for the Map and Reduce functions:
  - Map:  $(K_1, V_1) \rightarrow \text{List}(K_2, V_2)$
  - Reduce:  $(K_2, \text{List}(V_2)) \rightarrow \text{List}(K_3, V_3)$
2. **Mapper Execution:** Show the exact intermediate  $(K_2, V_2)$  key-value pairs emitted by each of the 3 mappers. (Assume the InputKey  $K_1$  is the Document ID/File name and  $V_1$  is the document text).
3. **Local Combiner Design:**
  - A) If we use an in-mapper combiner or a local Combiner function, show the compressed output generated on Node 1 (doc1.txt) before sending data over the network.
  - B) Explain how this reduces network I/O during the Shuffle phase.
4. **Shuffle & Sort Phase:** Show the grouped lists  $(K_2, \text{List}(V_2))$  delivered to the Reducer tasks.
5. **Final Output:** Write down the exact final key-value pairs written back to HDFS.

## Exercise 3: Multi-Metric Aggregation – Meteorological Extremes & Averages

**Scenario:** A climate research institution collects daily weather logs from thousands of IoT weather stations. Each line in the dataset contains: StationID, Year, Month, Day, Temperature\_Celsius

### Sample Input Dataset Split:

ST\_101, 2024, 01, 15, 12  
ST\_102, 2024, 01, 15, 25  
ST\_101, 2024, 01, 16, -4  
ST\_101, 2024, 01, 17, 22  
ST\_102, 2024, 01, 16, 30

For every unique (StationID, Year) pair, compute:

1. The Maximum Temperature recorded.
2. The Minimum Temperature recorded.
3. The Average Temperature across all readings.

### Questions:

1. **Composite Key & Value Design:** Define the exact (Key, Value) structure emitted by the Mapper so that a single MapReduce job can calculate all three metrics simultaneously.
2. **Combiner Tuple Strategy:** To compute the average correctly across a cluster using a Combiner, why must the Combiner emit a tuple (Sum, Count) rather than a single number?
3. **Trace Execution:**
  - a) Show the emitted pairs from the Mapper for StationID = ST\_101 and Year = 2024.
  - b) Trace the Reducer execution and write the final HDFS output line for (ST\_101, 2024).