

Directed Works TD 2: Advanced Hadoop Architecture & MapReduce Paradigms (Solution)

Solution to Exercise 1 (QCM Solutions & Justifications)

Q1 Answer: C

Justification: The Secondary NameNode is **not** a hot standby or backup NameNode. Its sole responsibility is to periodically download the active NameNode's fsimage and edits log, merge them into a fresh checkpoint, and upload it back to reduce NameNode restart time.

Q2 Answer: C

Justification: In YARN (Hadoop v2), resource allocation was decoupled from job execution. The central ResourceManager allocates raw cluster containers, while a dedicated, ephemeral ApplicationMaster is spawned for **each specific job** to monitor task execution and negotiate resources.

Q3 Answer: B

Justification: Mathematically, $\text{Avg}(a, b, c, d) \neq \text{Avg}(\text{Avg}(a, b), \text{Avg}(c, d))$. Because AVERAGE is non-associative, running a naive average Reducer as a Combiner produces mathematically incorrect results. The correct approach is to combine partial (Sum, Count) pairs.

Q4 Answer: B

Justification: Hadoop uses **Speculative Execution** to mitigate stragglers⁸. When it detects a task running significantly slower than average on a degraded machine, it launches an identical duplicate task on a healthy node. The framework takes the result of whichever task completes first and kills the other.

Q5 Answer: B

Justification: By default, Hadoop uses HashPartitioner, which computes $\text{hash}(\text{key}) \% \text{numReduceTasks}$ to deterministically route all intermediate values belonging to the same key to the exact same Reducer node.

Solution to Exercise 2 (Inverted Index Case Study)

1. Function Signatures

- **Map:** (DocID, DocText) -> List(Word, DocID)
- **Reduce:** (Word, List(DocID)) -> (Word, List(DocID: TermFrequency))

2. Mapper Execution Output

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

3. Local Combiner Output (In-Mapper Aggregation)

- **a) Output for Mapper 1 (doc1.txt):**
 - ("hadoop", "doc1.txt:2")
 - ("spark", "doc1.txt:1")
- **b) Network Optimization:** Instead of sending 3 records over the network for doc1.txt, the Combiner aggregates word counts locally and sends only 2 records, reducing network serialization and shuffle traffic by **33%**.

4. Shuffle & Sort Output (Grouped by Key)

- Key "hadoop" → ["doc1.txt:2", "doc3.txt:1"]
- Key "mapreduce" → ["doc3.txt:1"]
- Key "python" → ["doc2.txt:1", "doc3.txt:1"]
- Key "spark" → ["doc1.txt:1", "doc2.txt:2"]

5. Final HDFS Output File

```
hadoop    doc1.txt:2, doc3.txt:1
mapreduce doc3.txt:1
python    doc2.txt:1, doc3.txt:1
spark     doc1.txt:1, doc2.txt:2
```

Solution to Exercise 3 (Meteorological Extremes & Averages)

1. Composite Key & Value Design

- **Composite Input Key (K_2):** (StationID, Year), e.g., "ST_101:2024"
- **Composite Value (V_2):** (Temperature, Count), e.g., (12, 1)

2. Combiner Tuple Strategy

To keep AVERAGE associative across parallel worker nodes, the Combiner must sum partial temperatures and counts locally and emit (PartialSum, PartialCount). The Reducer then

computes: $\text{Global Average} = \frac{\sum \text{PartialSums}}{\sum \text{PartialCounts}}$

3. Execution Trace for (ST_101, 2024)

- **a) Mapper Output for ST_101 (2024 records):**
 - ("ST_101:2024", (-12, 1))
 - ("ST_101:2024", (-4, 1))
 - ("ST_101:2024", (22, 1))
- **b) Combiner Output on Local Node:**
 - Key: "ST_101:2024"
 - Aggregated Value: Sum = $12 + (-4) + 22 = 30$, Count = 3, Min = -4, Max = 22.
 - Emitted: ("ST_101:2024", (Sum=30, Count=3, Min=-4, Max=22))
- **c) Reducer Aggregation & Final HDFS Output:**
 - Total Sum = 30
 - Total Count = 3
 - Average = $\frac{30}{3} = 10.0^\circ\text{C}$
 - Min = -4°C
 - Max = 22°C

Final Output Line in HDFS:

ST_101:2024 Min: -4°C , Max: 22°C , Avg: 10.0°C