MapReduce and Apache Spark - Comprehensive Question Bank

Question 1: MapReduce Fundamentals and WordCount Implementation (8 marks)

Part A (3 marks)

Explain the three stages of the MapReduce model and why this architecture is particularly suitable for distributed computing environments. Include the role of key-value pairs in your explanation.

Part B (5 marks)

Consider the following text data distributed across two files:

- File 1: "big data mining techniques"
- File 2: "data mining big datasets"

Manual Calculation Required:

1. Show the complete MapReduce workflow for WordCount
2. Detail the Map phase output for each file
3. Show the intermediate shuffle/sort phase
4. Present the final Reduce phase results
5. Explain how this would be distributed across multiple machines

---

Question 2: Relational Algebra Operations in MapReduce (10 marks)

Part A (4 marks)

Given two relations:

- **Students[ID, Name, Major]**: (101, "Alice", "CS"), (102, "Bob", "Math"), (103, "Carol", "CS")
- **Courses[Major, Course]**: ("CS", "Database"), ("CS", "Algorithms"), ("Math", "Calculus")

Design and trace through a complete MapReduce solution for performing a natural join operation. Show:

1. Map function logic and outputs
2. Intermediate key-value pairs after shuffling
3. Reduce function processing
4. Final join results

Part B (3 marks)

Implement a Python function that simulates the Map phase of the natural join operation above:

```python
def natural_join_mapper(relation_name, tuple_data, join_attribute):
    # Your implementation here
    pass
```

Part C (3 marks)

Explain why the conclusion "All common SQL queries can be implemented with MapReduce" is significant for big data processing. Provide examples of two other relational operations and briefly describe their MapReduce implementation approach.

Question 3: Matrix Operations in Distributed Computing (12 marks)

Part A (6 marks)

Consider two matrices for multiplication:

```
Matrix M (2×3):    Matrix N (3×2):
[1  2  3]          [7   8]
[4  5  6]           [9  10]
                   [11 12]
```

Manual Calculation Required:

1. Represent both matrices as relations with schema [Row, Column, Value]
2. Design the two-stage MapReduce algorithm for matrix multiplication
3. Show the complete execution trace:
   - Stage 1 Map outputs for both matrices
   - Stage 1 Reduce outputs (intermediate products)
   - Stage 2 aggregation results
4. Verify your result matches traditional matrix multiplication

Part B (4 marks)

Implement the first MapReduce stage for matrix multiplication:

```python

```

```python
def matrix_multiply_stage1_mapper(matrix_name, row, col, value, other_matrix_size):
    """
    Args:
        matrix_name: 'M' or 'N'
        row, col, value: matrix element position and value
        other_matrix_size: dimensions of the other matrix
    Returns:
        List of (key, value) pairs
    """
    # Your implementation here
    pass

def matrix_multiply_stage1_reducer(key, value_list):
    """
    Args:
        key: column index for matrix M / row index for matrix N
        value_list: list of (matrix_name, index, value) tuples
    Returns:
        List of ((row, col), product) pairs
    """
    # Your implementation here
    pass
```

Part C (2 marks)

Explain why matrix multiplication requires two MapReduce stages and discuss the computational complexity implications for very large matrices.

Question 4: From MapReduce to Spark DAG Model (9 marks)

Part A (4 marks)

Compare and contrast the MapReduce model with Spark's DAG (Directed Acyclic Graph) model. Address:

1. Architectural differences
2. Performance implications
3. Workflow complexity handling
4. Memory usage patterns

Part B (5 marks)

Consider a data processing pipeline that needs to:

1. Load customer transaction data

2. Filter transactions above $100
3. Group by customer ID
4. Calculate average transaction amount per customer
5. Sort results by average amount

Design Task:

1. Show how this would be implemented using traditional MapReduce (multiple jobs)
2. Illustrate the Spark DAG approach with transformations and actions
3. Explain the lazy evaluation concept and its benefits in this scenario
4. Identify which operations are transformations vs. actions

Question 5: Apache Spark DataFrames and PySpark Implementation (11 marks)

Part A (3 marks)

Explain the relationship between Spark's DataFrame abstraction and RDDs. Discuss:

1. Immutability principles
2. High-level vs. low-level abstractions
3. Performance considerations

Part B (8 marks)

You are given a dataset of flight information with the following schema:

```
flights[FlightID, Origin, Destination, Delay, Distance, Airline]
```

Coding Task - Implement PySpark operations:

```python
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, count, desc, col

def analyze_flight_data(spark, flights_df):
  """
  Implement the following analyses:
  1. Find average delay by airline
  2. Count flights by origin airport
  3. Identify top 5 routes (origin-destination pairs) by frequency
  4. Calculate delay statistics for flights over 1000 miles

  Args:
    spark: SparkSession object
    flights_df: DataFrame with flight data

  Returns:
    Dictionary containing results of all analyses
  """
  # Your implementation here
  # Remember to use appropriate transformations and actions
  # Show lazy evaluation in practice
  pass
```

Additional Requirements:

1. Clearly identify which operations are transformations vs. actions
2. Explain the execution order due to lazy evaluation
3. Discuss how this would be distributed across a cluster

Question 6: Applied MapReduce Algorithm Design (10 marks)

Part A (6 marks)

Design a complete MapReduce solution for implementing a k-Nearest Neighbors (kNN) classifier for movie recommendation. Given:

- Training dataset: Movies with features [Genre, Rating, Year, Duration] and labels [Recommended/Not Recommended]
- Unknown movie to classify: ("Action", 8.5, 2023, 120)
- k = 3

Design Requirements:

1. Define the distance metric for movie similarity

2. Design the Mapper function with detailed logic
3. Design the Reducer function for finding k nearest neighbors
4. Explain the final voting mechanism
5. Discuss how a Combiner could improve performance

Part B (4 marks)

Implement the core distance calculation and mapping logic:

```python
import math

def calculate_movie_distance(unknown_movie, known_movie):
    """
    Calculate distance between movies using appropriate metrics
    Args:
        unknown_movie: (genre, rating, year, duration)
        known_movie: (genre, rating, year, duration, label)
    Returns:
        Distance value
    """
    # Your implementation here
    pass

def knn_mapper(movie_id, movie_features, unknown_movie):
    """
    MapReduce mapper for kNN classification
    Args:
        movie_id: identifier for known movie
        movie_features: (genre, rating, year, duration, label)
        unknown_movie: movie to classify
    Returns:
        (key, value) pair for reducer
    """
    # Your implementation here
    pass
```

Bonus Challenge Question: Real-World Application (5 marks)

You are tasked with processing streaming e-commerce transaction data to detect fraudulent activities in real-time. The system needs to:

1. Process transactions as they arrive
2. Maintain running statistics per user

3. Flag suspicious patterns (unusual amounts, frequencies, locations)
4. Generate alerts for potential fraud

Discussion Points:

1. How would you adapt MapReduce concepts for streaming data?
2. What are the limitations of traditional MapReduce for this use case?
3. How does Spark Streaming address these challenges?
4. Design a high-level architecture using Spark's streaming capabilities

---

Answer Guidelines and Marking Rubric

Theoretical Questions (30-40% of total marks)

- **Conceptual understanding**: Clear explanation of MapReduce stages, DAG model, lazy evaluation
- **Comparative analysis**: Understanding trade-offs between different approaches
- **Real-world application**: Connecting concepts to practical scenarios

Manual Calculations (25-30% of total marks)

- **Step-by-step process**: Complete workflow demonstrations
- **Accuracy**: Correct intermediate and final results
- **Clarity**: Well-organized presentation of calculations

Coding Implementation (30-35% of total marks)

- **Functional correctness**: Code produces expected results
- **Best practices**: Proper use of PySpark functions and transformations
- **Code organization**: Clear structure and appropriate comments
- **Framework understanding**: Correct identification of transformations vs. actions

Problem-Solving Process (10-15% of total marks)

- **Algorithm design**: Logical approach to complex problems
- **Optimization considerations**: Understanding of performance implications
- **Scalability awareness**: Recognition of distributed computing challenges

---

Study Preparation Tips

Key Concepts to Master

1. **MapReduce Workflow**: Map → Shuffle → Reduce pipeline

2. **Key-Value Paradigm**: Understanding data flow through transformations
3. **Relational Operations**: Selection, join, projection in distributed context
4. **Matrix Operations**: Two-stage approach for complex computations
5. **Spark Advantages**: DAG model, lazy evaluation, in-memory processing
6. **DataFrame Operations**: Transformations vs. actions distinction

Practice Exercises

1. Trace through MapReduce workflows with different data sets
2. Implement common SQL operations using MapReduce logic
3. Design distributed algorithms for various computational problems
4. Practice PySpark DataFrame manipulations
5. Analyze performance implications of different design choices

Common Exam Pitfalls to Avoid

- Confusing transformations with actions in Spark
- Incomplete MapReduce workflow descriptions
- Missing intermediate steps in manual calculations
- Not considering distributed computing constraints
- Ignoring lazy evaluation implications in code design