Data Processing with Apache Spark - Comprehensive Exam Notes

1. Processing Massive Data & Parallelism

Key Concepts

- **Problem**: Processing massive datasets requires long runtime
- **Solution**: Parallelism (distributed computing)
- **Benefit**: Speed up computation by distributing work across multiple machines

Why Distributed Computing?

- Single machines have computational and memory limitations
- Distributed systems can handle data that doesn't fit on one machine
- Parallel processing reduces overall computation time

---

2. MapReduce Model

Core Concept

MapReduce is a **preeminent model of parallel computation** with a very simple structure but powerful capabilities.

MapReduce Structure

1. **Map Stage**: Performs simple mapping operations to produce key-value pairs
2. **Intermediate Stage**: Merges key-value pairs per key (shuffle and sort)
3. **Reduce Stage**: Performs aggregation operations per key

Key Characteristics

- **Algorithm perspective**: Very powerful for complex computations
- **Implementation perspective**: Well-suited for computing clusters
- **Data structure**: Uses (key, value) pairs as basic data structure

WordCount Example Workflow

```
Input: "hello world hello"
Map: (hello, 1), (world, 1), (hello, 1)
Shuffle: (hello, [1,1]), (world, [1])
Reduce: (hello, 2), (world, 1)
```

---

3. Relational-Algebra Operations in MapReduce

Basic Terminology

- **Relation**: A table of data
- **Attributes**: Column headers
- **Tuples**: Rows
- **Schema**: Bag of attributes of a relation
- **Notation**: $R[A_1, ..., A_n]$ denotes relation R with schema $A_1, ..., A_n$

Selection Operation

Purpose: Apply condition C to each tuple, output only tuples satisfying C

MapReduce Implementation:

- **Map Function**: For each tuple t in R, test if it satisfies C
  - If yes: produce (t, t)
  - If no: produce nothing
- **Reduce Function**: Identity function (just pass through)

Natural Join Operation

Purpose: Compare tuples from two relations, join if they agree on common attributes

MapReduce Implementation:

- **Map Function**:
  - For tuple (a, b) in R[A, B]: produce b: (R, a)
  - For tuple (b, c) in S[B, C]: produce b: (S, c)
- **Reduce Function**:
  - For key b with values [(R, a), (S, c)]: produce (a, b, c)

Natural Join Example

Tables:

- R: (a_1,b_1), (a_2,b_1), (a_3,b_2), (a_4,b_3)
- S: (b_2,c_1), (b_2,c_2), (b_3,c_3)

Map Output:

- b_1: (R,a_1), (R,a_2)
- b_2: (R,a_3), (S,c_1), (S,c_2)
- b_3: (R,a_4), (S,c_3)

Reduce Output:

- b_1: None (no matching S tuples)
- b_2: (a_3,c_1), (a_3,c_2)
- b_3: (a_4,c_3)

Other Operations

All common relational-algebra operations can be expressed in MapReduce:

- Projection
- Union
- Intersection
- Grouping
- Left/Right/Outer joins

Conclusion: All common SQL queries can be implemented with MapReduce

---

4. Matrix-Matrix Multiplication in MapReduce

Problem Setup

- Matrix M with element m_{ij} in row i, column j
- Matrix N with element n_{jk} in row j, column k
- Product P = MN with element $p_{ik} = \Sigma_j m_{ij} n_{jk}$
- **Constraint**: Number of columns in M = Number of rows in N

Matrix as Relation

- View matrix as relation with three attributes: (row, column, value)

- M[I, J, V] with tuples (i, j, m_{ij})
- N[J, K, W] with tuples (j, k, n_{jk})

Two-Stage MapReduce Solution

Stage 1: Element Pairing

Map Function A:

- For each m_{ij}: produce j: (M, i, m_{ij})
- For each n_{jk}: produce j: (N, k, n_{jk})

Reduce Function A:

- For key j with values from M: (M, i, m_{ij}) and N: (N, k, n_{jk})
- Produce key-value pair: (i, k): $m_{ij} \times n_{jk}$

Stage 2: Aggregation

Map Function B: Identity function (pass through)

Reduce Function B:

- For key (i, k): sum all associated values
- Result: (i, k): v where v is the element in row i, column k of P = MN

---

5. MapReduce to DAG (Directed Acyclic Graph)

Evolution from MapReduce

- **Traditional MapReduce**: Simple two-step model (Map → Reduce)
- **DAG Model**: Orchestration of any steps forming a directed acyclic graph
- **Limitation**: MapReduce requires storing intermediate results in HDFS rather than memory

Apache Spark Solution

- Implements DAG-based workflow system
- Keeps intermediate results in memory when possible
- More efficient than traditional MapReduce for complex workflows

---

6. Apache Spark Architecture

Distributed vs Single-Machine Computing

- **Single Machine**: All data and computation on one computer
- **Distributed**: Data and computation spread across multiple machines
- **Spark Advantage**: Operates on distributed data as if it's on a single machine

DataFrame Abstraction

- **Definition**: High-level structured abstraction representing a table of data
- **Similarity**: Like Pandas DataFrame but supports distributed computing
- **Key Feature**: Data may be distributed across different locations transparently

---

7. Spark Operations: Transformations and Actions

Two Types of Operations

Transformations

- **Purpose**: Create another DataFrame/RDD
- **Example**: Creating a new row, filtering data, joining tables
- **Characteristic**: Lazy evaluation (not executed immediately)

Actions

- **Purpose**: Produce a computational result
- **Example**: Count rows, collect results, save to file
- **Characteristic**: Trigger execution of transformations

Spark Application Structure

- **View**: DAG of transformations and actions
- **Execution**: Only when action is performed (lazy evaluation)

---

8. Lazy Evaluation

Concept

Spark doesn't evaluate DataFrame/RDD until it has to (when an action is performed)

Benefits

- **Optimization**: Spark can optimize entire computation pipeline

- **Efficiency**: Avoid unnecessary intermediate computations
- **Resource Management**: Better memory and CPU utilization

Example Flow

```
df.filter(...).select(...).groupBy(...).count()
     ↑          ↑          ↑          ↑
Transformation Transformation Transformation Action
   (lazy)       (lazy)       (lazy)    (executes all)
```

---

9. Spark's DataFrame API (PySpark)

Key Features

- High-level API for structured data processing
- Built on top of Spark's RDD (Resilient Distributed Dataset)
- SQL-like operations for data manipulation
- Comprehensive API with extensive functionality

API Reference

- Complete documentation available at Spark's official documentation
- Covers all DataFrame operations and methods
- Examples and use cases provided

---

10. Summary & Key Takeaways

MapReduce Model

- **Strength**: Powerful computation model for massive data processing
- **Framework**: Hadoop's MapReduce implementation
- **Limitation**: Limited to two-stage processing, disk-based intermediate storage

Spark's DAG Model

- **Advancement**: DAG model with series of transformations and actions
- **Advantage**: Rich set of APIs, in-memory processing
- **Efficiency**: Lazy evaluation and optimization
- **Flexibility**: More complex workflows than traditional MapReduce

Practical Applications

- **Data Processing**: ETL operations, data cleaning, aggregations
- **Analytics**: Complex queries, machine learning pipelines
- **Big Data**: Processing datasets too large for single machines
- **Real-time Processing**: Stream processing capabilities

Study Tips for Exam

Key Concepts to Remember

1. **MapReduce workflow**: Map → Shuffle → Reduce
2. **Relational operations**: Selection, Join implementations
3. **Matrix multiplication**: Two-stage MapReduce approach
4. **Spark advantages**: DAG, lazy evaluation, in-memory processing
5. **DataFrame operations**: Transformations vs Actions

Practice Problems

1. Design MapReduce for common SQL operations
2. Trace through natural join examples
3. Understand matrix multiplication steps
4. Identify transformations vs actions in Spark code
5. Explain lazy evaluation benefits

Important Formulas

- Matrix multiplication: $p_{ik} = \Sigma_j m_{ij} n_{jk}$
- Natural join condition: Common attributes must match
- Key-value pair structure in MapReduce operations