**Apache Spark – Advanced Concepts & Performance Tuning Lab Manual**

**Lab 1: Spark Architecture Deep Dive**

**Objective:**

Understand the roles of Driver, Executors, and the Shuffle mechanism.

**Steps:**

1. **Start Spark Shell or Notebook**
2. pyspark
3. **Create a simple DataFrame and observe stage breakdown**
4. df = spark.range(1, 1000000).withColumn("squared", col("id") \* col("id"))
5. df.groupBy((col("id") % 10).alias("group")).count().show()
6. **Open Spark UI (default:** [**http://localhost:4040**](http://localhost:4040/)**)**
   * Observe the DAG and stage plans.
   * Identify shuffle boundaries (wide vs narrow dependencies).

**Lab 2: DataFrame API Essentials**

**Objective:**

Familiarize with basic DataFrame operations.

**Steps:**

1. **Load data from CSV**
2. df = spark.read.option("header", True).csv("data/transactions.csv")
3. **Explore schema and data**
4. df.printSchema()
5. df.show(5)
6. **Apply filters, aggregations**
7. df.filter(col("amount") > 1000).groupBy("category").agg(avg("amount")).show()

**Lab 3: Cache vs Checkpoint**

**Objective:**

Compare use-cases and effects of cache and checkpoint.

**Steps:**

1. **Perform an expensive transformation**
2. transformed\_df = df.withColumn("log\_amount", log(col("amount") + 1))
3. **Cache and checkpoint**
4. transformed\_df.cache()
5. transformed\_df.checkpoint()
6. transformed\_df.count()
7. **Check Spark UI for storage and DAG truncation**

**Lab 4: Spark UI Deep Dive**

**Objective:**

Understand Spark UI components for job diagnostics.

**Steps:**

1. **Run a job and open Spark UI**
2. **Review tabs:**
   * **Jobs:** Identify stages
   * **Stages:** View task distribution
   * **SQL:** Review logical and physical plans
   * **Storage:** See cached datasets
   * **Executors:** Review memory usage

**Lab 5: RDD vs DataFrame**

**Objective:**

Contrast RDD and DF performance and usage.

**Steps:**

1. **Create RDD and DF**
2. rdd = spark.sparkContext.parallelize(range(1, 1000000)).map(lambda x: (x, x\*x))
3. df = spark.range(1, 1000000).withColumn("squared", col("id") \* col("id"))
4. **Compare performance**
5. rdd\_sum = rdd.map(lambda x: x[1]).sum()
6. df\_sum = df.select(sum("squared")).show()

**Lab 6: Parquet Partition Discovery**

**Objective:**

Read partitioned Parquet data and observe pruning.

**Steps:**

1. **Load partitioned data**
2. df = spark.read.parquet("data/sales\_data")
3. **Check partition columns**
4. df.printSchema()
5. **Use filters and check physical plan**
6. df.filter("year = 2023").explain(True)

**Lab 7: Broadcast Join Hint**

**Objective:**

Use broadcast hint to optimize small-large joins.

**Steps:**

1. **Create small and large DFs**
2. small\_df = spark.read.csv("data/store.csv", header=True)
3. large\_df = spark.read.csv("data/transactions.csv", header=True)
4. **Broadcast join**
5. joined = large\_df.join(broadcast(small\_df), "store\_id")
6. joined.explain(True)

**Lab 8: Reading from External Sources (Blob/ADLS)**

**Objective:**

Load data from Azure Blob and ADLS Gen2.

**Steps:**

1. **Set Spark config for Blob/ADLS**
2. spark.conf.set("fs.azure.account.key.<account>.blob.core.windows.net", "<key>")
3. **Read data**
4. df = spark.read.csv("wasbs://<container>@<account>.blob.core.windows.net/data.csv", header=True)

**Lab 9: Advanced UDF Profiling (Optional Stretch Lab)**

**Objective:**

Profile performance of UDFs and optimize.

**Steps:**

1. **Define a simple UDF**
2. def compute\_score(x): return x \* x
3. spark.udf.register("score\_udf", compute\_score)
4. df.withColumn("score", expr("score\_udf(amount)"))
5. **Compare with pandas\_udf**
6. @pandas\_udf("double")
7. def fast\_score(x): return x \* x
8. df.withColumn("score", fast\_score(df["amount"]))
9. **Review Spark UI for performance differences**

**Lab 10: Pipeline Build and Optimization**

**Objective:**

Construct and optimize a pipeline to achieve 2× speedup.

**Steps:**

1. **Build a transformation pipeline**
2. df = spark.read.csv("data/big\_txns.csv", header=True, inferSchema=True)
3. df = df.withColumn("amount\_log", log(col("amount") + 1))
4. df = df.groupBy("category").agg(avg("amount\_log"))
5. **Add cache and repartition**
6. df = df.cache().repartition(4)
7. df.count()
8. **Measure execution time and compare before/after optimizations**