MAPREDUCE WORKFLOWS

Introduction

MapReduce is a programming model used for processing large datasets in a distributed environment. A *MapReduce workflow* refers to a sequence of MapReduce jobs executed in a pipeline to accomplish complex data-processing tasks.

Why Workflows?

* Single MapReduce job is often insufficient

* Complex analytics require multiple dependent stages

* Output of one job becomes input to another

Components of a MapReduce Workflow

1. Mapper

    * Processes input key/value pairs

    * Emits intermediate key/value pairs

2. Reducer

    * Aggregates values based on intermediate keys

    * Produces final output

3. Driver

    * Configures and submits jobs

    * Controls job flow (job chaining)

4. Intermediate Data

    * Stored in HDFS (default)

    * Can also be passed directly using **in-memory chaining** (e.g., ChainMapper, ChainReducer)

Types of MapReduce Workflows

1. Linear Workflows (Sequential Jobs)

* Job 1 → Job 2 → Job 3

* Most common workflow

* Dependent tasks

Example:

Log Parsing → Sessionization → Aggregation

2. Parallel Workflows

* Multiple jobs run simultaneously
* Final merge after all jobs complete

Used for:

* Comparing multiple datasets
* Parallel feature extraction

3. Iterative Workflows

* Repeated MapReduce steps until a condition is satisfied
* MapReduce is not naturally iterative → used in algorithms like:
* PageRank
* K-Means Clustering

Tools Supporting MapReduce Workflows

Oozie → Workflow scheduler

Azkaban → Job orchestration

Apache Airflow → DAG-based pipelines

Job Chaining Techniques

Driver code:

java

```
Job job1 = Job.getInstance(conf, "Job1");
job1.waitForCompletion(true);
Job job2 = Job.getInstance(conf, "Job2");
job2.waitForCompletion(true);
```

Using ChainMapper and ChainReducer

Efficient method to run multiple mappers and reducers in a single job.

Advantages of MapReduce Workflows

* Scalable
* Fault-tolerant

* Supports large and complex pipelines

* Efficient for batch analytics

2. UNIT TESTING MAPREDUCE USING MRUNIT

2.1 Introduction to MRUnit

MRUnit is a unit-testing framework for Hadoop MapReduce programs.

It helps test Mappers, Reducers, and full MapReduce jobs without running on a cluster.

Why MRUnit?

* Fast testing

* No need for Hadoop cluster

* Detect logic errors early

* Ensures correctness

2.2 MRUnit Architecture

Test Drivers Provided by MRUnit

| Component | MRUnit Class |

| ---------- | ---------- |

| Mapper Test | **MapDriver** |

| Reducer Test | **ReduceDriver** |

| MapReduce Test | **MapReduceDriver** |

2.3 Testing a Mapper with MRUnit**

Steps

1. Create Mapper instance

2. Provide input

3. Define expected output

4. Use `MapDriver` to run test

Example

java

@Test

```
public void testMapper() throws IOException {
   new MapDriver<LongWritable, Text, Text, IntWritable>()
      .withMapper(new WordCount.TokenizerMapper())
      .withInput(new LongWritable(1), new Text("hello world"))
      .withOutput(new Text("hello"), new IntWritable(1))
      .withOutput(new Text("world"), new IntWritable(1))
      .runTest();
}
```

2.4 Testing a Reducer with MRUnit

Example

java

```
@Test
public void testReducer() throws IOException {
   new ReduceDriver<Text, IntWritable, Text, IntWritable>()
      .withReducer(new WordCount.IntSumReducer())
      .withInput(new Text("hello"), Arrays.asList(new IntWritable(1), new IntWritable(2)))
      .withOutput(new Text("hello"), new IntWritable(3))
      .runTest();
}
```

2.5 Testing Full MapReduce Job

Example

java

```
@Test
public void testMapReduce() throws IOException {
   new MapReduceDriver<LongWritable, Text, Text, IntWritable, Text, IntWritable>()
      .withMapper(new TokenizerMapper())
      .withReducer(new IntSumReducer())
      .withInput(new LongWritable(1), new Text("hello hello"))
```

```
        .withOutput(new Text("hello"), new IntWritable(2))
        .runTest();
}
```

2.6 Advantages of MRUnit

* Lightweight
* Isolates logic errors
* No cluster needed
* Reproducible test environment
* Faster development cycle

3. TEST DATA & LOCAL TESTING

3.1 Local Job Runner Mode**

Hadoop has two important modes:

1. Local (Standalone) Mode – runs on a single JVM
2. Pseudo-distributed Mode – simulates cluster on one machine

Local mode is ideal for:

* Debugging
* Unit testing
* Using small datasets

3.2 Test Data Preparation

Types of Test Data

1. Synthetic Data
   * Hand-crafted
   * Predictable
   * Best for unit tests
2. Sample Production Data
   * Small subset from real logs
   * Caught data-dependent errors

3. Edge Case Data

* Empty lines
* Null values
* Strange characters
* Malformed records

3.3 Using Local File System for Testing

Example configuration:

xml

```
<property>
  <name>mapreduce.framework.name</name>
  <value>local</value>
</property>
```

Run MapReduce job locally:

hadoop jar myjar.jar MyDriver input.txt output/

3.4 Debugging Techniques in Local Mode

* Use System.out.println()
* Log4j logging
* Debugging in IDE
* Step-through debugging

3.5 Benefits of Local Testing

* Very fast
* No need for cluster
* Easier debugging
* Better test coverage