DS-GA 1004: Big Data Lecture 6

Big Data Infrastructure and Introduction to Spark

1. Big Data Infrastructure

Hadoop Framework Overview

- **MapReduce**: Processing engine
- **YARN**: Resource manager
- **HDFS**: Storage layer

Evolution of Hadoop

- Hadoop 1.x: MapReduce + HDFS
- Hadoop 2.x: MapReduce + YARN + HDFS + Other engines (Spark, Flink, Hive, Pig)
- Hadoop 3.x: Kubernetes, YARN, HDFS, Cloud storage

YARN Architecture

- Components:
  - Resource Manager (Resource Scheduler + Application Manager)
  - Node Manager
  - Application Master
  - Containers
- YARN acts as the "Operating System" of Hadoop.

Cluster Resource Terminology

- **Container** (not Docker): Abstraction bundling storage, cores, and RAM.

Importance of Data Locality

- **Goal**: Bring computation to where data resides (cheaper than moving data).
- MapReduce splits input into splits; each split maps to HDFS blocks.
- Scheduler uses HDFS block locations to optimize split assignments.

Network Topology Considerations

- **Best**: Node-local execution (data and compute on same node)
- **OK**: Rack-local execution (same rack, low latency)
- **Worst**: Cross-rack execution (higher latency)

HDFS Replication

- Default replication factor: 3
- Placement: Two replicas in same rack, third in different rack.

2. Spark Introduction

Why Spark?

- **Strengths of MapReduce**: Scalability, Fault Tolerance, Commodity Hardware Friendly.
- **Limitations of MapReduce**: Unsuitable for iterative, interactive computations.

Problems with MapReduce for Iterative Algorithms

- Each iteration (e.g., gradient descent) requires full MapReduce cycle.
- High latency due to blocking between map and reduce stages.

Spark's Key Idea: Resilient Distributed Datasets (RDDs)

- **RDD** = Lineage graph of transformations + Data source + Partitioning interfaces.
- **Deferred computation**: Transformations are lazy.

RDD Operations

- **Transformations** (lazy): map, filter, join
- **Actions** (trigger execution): collect, count, reduce, save

RDD Example (Log Processing)

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.filter(_.contains("MySQL")).map(_.split('\t')(3)).collect()
```

Spark Execution Model

- Builds a dependency DAG from transformations.
- Works **backwards** from action to source.
- Caches intermediate RDDs if needed.
- Loss recovery using lineage information.

Pipelining

- Transformations can be pipelined without materializing intermediates.

Partitions and Dependencies

- **Narrow dependency**: Parent partition maps to one child partition (easy recovery, fast).
- **Wide dependency**: Parent partition maps to multiple children (shuffle, slow).

Co-Partitioning

- Preemptively partition datasets by a key to avoid costly shuffles.
- Use `repartition()` or `bucketBy()` in Spark.

Spark DataFrames

- High-level abstraction over RDDs.
- Tables with schemas, columns = RDDs.
- Queryable with SparkSQL.

Important Practices

- Always aggregate (e.g., `reduce`, `count`) before `collect()` to avoid crashing login nodes.
- Understand Spark `explain()` plans to optimize.