Spark RDD
Where are we?

- Distributed storage in HDFS
- MapReduce query execution in Hadoop
- High-level data manipulation using Pig
A Step Back to MapReduce

- Designed in the early 2000’s
- Machines were unreliable
- Focused on fault-tolerance
- Addressed data-intensive applications
- Limited memory
MapReduce in Practice

Can we improve on that?
Pig

- Slightly improves disk I/O by consolidating map-only jobs
Pig

- Slightly improves disk I/O by consolidating map-only jobs
Pig

- Slightly improves disk I/O by consolidating map-only jobs
Pig

- Slightly improves disk I/O by consolidating map-only jobs
Pig (at a higher level)
RDD

- Resilient Distributed Datasets
- A distributed query processing engine
- The Spark counterpart to MapReduce
- Designed for in-memory processing
In-memory Processing

- The machine specs change
  - More reliable
  - Bigger memory
- And the workload changed
  - Analytical queries
  - Iterative operations (like ML)
- The main idea: Rather than storing intermediate results to disk, keep them in memory
- How about fault tolerance?
RDD Example

FILTER

FOREACH

JOIN

FILTER

GROUP BY
RDD Abstraction

- RDD is a pointer to a distributed dataset
- Stores information about how to compute the data rather than where the data is
- Transformation: Converts an RDD to another RDD
- Action: Returns an answer of an operation over an RDD
Spark RDD Features

- Lazy execution: Collect transformations and execute on actions (Similar to Pig)
- Lineage tracking: Keep track of the lineage of each RDD for fault-tolerance
Filter Operation

Similarly, projection operation (ForEach in Pig)

Narrow dependency
GroupBy (Shuffle) Operation

Similar operation Join

Wide dependency
Types of Dependencies

- Narrow dependencies
- Wide dependencies

**Narrow dependencies:**
Each partition of the parent RDD is used by at most one partition of the child RDD.

**Wide dependencies:**
Each partition of the parent RDD may be depended on by multiple child partitions.

Credit: https://github.com/rohgar/scala-spark-4/wiki/Wide-vs-Narrow-Dependencies
Examples of Transformations

- map
- flatMap
- reduceByKey
- filter
- sample
- join
- union
- partitionBy
Examples of Actions

- count
- collect
- save(path)
- persist
- reduce
How RDD can be helpful

- Consolidate operations
  - Combine transformations

- Iterative operations
  - Keep the output of an iteration in memory till the next iteration

- Data sharing
  - Reuse the same data without having to read it multiple times
Examples

# Initialize the Spark context
JavaSparkContext spark =
    new JavaSparkContext("local", "CS226-Demo");
Examples

# Initialize the Spark context
JavaSparkContext spark =
    new JavaSparkContext("local", "CS226-Demo");

# Hello World! Example. Count the number of lines in the file
JavaRDD<String> textFileRDD =
    spark.textFile("nasa_19950801.tsv");
long count = textFileRDD.count();
System.out.println("Number of lines is "+count);
Examples

# Count the number of OK lines

JavaRDD<String> okLines = textFileRDD.filter(new Function<String, Boolean>() {
    @Override
    public Boolean call(String s) throws Exception {
        String code = s.split("\t")[5];
        return code.equals("200");
    }
});

long count = okLines.count();
System.out.println("Number of OK lines is "+count);
# Count the number of OK lines
# Shorten the implementation using lambdas (Java 8 and above)

JavaRDD<String> okLines =
    textFileRDD.filter(s -> s.split("\t")[5].equals("200"));

long count = okLines.count();
System.out.println("Number of OK lines is "+count);
Examples

# Make it parametrized by taking the response code as a command line argument

String inputFileName = args[0];
String desiredResponseCode = args[1];
...

JavaRDD<String> textFileRDD = spark.textFile(inputFileName);
JavaRDD<String> okLines = textFileRDD.filter(new Function<String, Boolean>() {
    @Override
    public Boolean call(String s) {
        String code = s.split("\t")[5];
        return code.equals(desiredResponseCode);
    }
});
Examples

# Count by response code

JavaPairRDD<Integer, String> linesByCode =
textFileRDD.mapToPair(new PairFunction<String, Integer, String>() {
    @Override
    public Tuple2<Integer, String> call(String s) {
        String code = s.split("\t")[5];
        return new Tuple2<Integer, String>(Integer.valueOf(code), s);
    }
});

Map<Integer, Long> countByCode =
linesByCode.countByKey();
System.out.println(countByCode);
Further Reading

- Spark home page: http://spark.apache.org/
- Quick start: http://spark.apache.org/docs/latest/quick-start.html