Spark RDD
Distributed Processing

• Once your data is stored in HDFS, the next step is to process it in parallel
• MapReduce
  ▪ Abstracts your program in two functions, map and reduce
  ▪ Was very limited and very low level
RDD

• Resilient Distributed Datasets
• A distributed query processing engine
• The Spark counterpart to Hadoop MapReduce
• Designed for in-memory processing
Spark High-level Architecture

Developer → Spark Program → Driver → Spark Job → Master node → Executors
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
- Lineage tracking: Keep track of the lineage of each RDD for fault-tolerance
Filter Operation

Similarly, the map (projection) operation

Narrow dependency
GroupBy (Shuffle) Operation

Wide dependency

Similar operation Join
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
- mapToPair
- 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
Java Examples

- Apache Spark homepage
  - [https://spark.apache.org](https://spark.apache.org)

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

```java
JavaSparkContext spark =
    new JavaSparkContext("local", "CS226-Demo");
```

# Hello World! Example. Count the number of lines in the file

```java
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 (response code 200)
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);
Examples

# Count the number of OK lines (response code 200)
# 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);
    }
});
# Count by response code

# Important! Not all transformations and actions are on the getting started guide

```java
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);
```
How Spark RDD works internally
Sample Program

JavaRDD<String> textFileRDD = spark.textFile(inputFileName);
JavaRDD<Long> lengths = textFileRDD
    .map(line => line.length());
long size = lengths.reduce((a,b) => a+b);
RDD Creation

• `sparkContext.textFile("...")`

**RDD**

Partition

- PartitionID: 0 → n-1
- File: Path
- Offset: Long
- Length: Long
- Locations: String[]

- The partitions are defined based on the metadata
- The file is not opened
Transformations

• Transforms one RDD to another
• Does not apply the transformation immediately
• E.g., Map
Map RDD

SourceRDD (textFileRDD)
datatype = A: Class

MapRDD (lengths)
Dependency: RDD
datatype = B: Class
mapFunction: Function<A → B>
**Action: reduce(a+b)**

- Launches the Spark job

```
Action: reduce(a+b)

HDFS

B1
B2
...
Bn

Input Block

TextFileRDD.iterator

Iterator<String>

for each line {MapRDD.f(line)}

Iterator<Long>

long result = iter.next

for each v in iter
  result = ReduceRDD.f(result, v)

return result
```
**Action: reduce(a+b)**

HDFS

- B1
- B2
- ...
- Bn

```
long result = iter.next
for each v in iter
    result = ReduceRDD.f(result, v)
return result
```
Running a complex DAG

```java
PairFunction<String, String, String> lineParser =
    (PairFunction<String, String, String>) line -> {
    String[] parts = line.split("","");
    return new Tuple2<>(parts[0], parts[1]);
};
JavaPairRDD<String, Iterable<String>> input1 =
sc.textFile("file1")
    .mapToPair(lineParser)
    .groupByKey();

JavaPairRDD<String, String> input2 =
sc.textFile("file2")
    .mapToPair(lineParser)
    .filter(record -> record._1.equals("200"));

JavaPairRDD<String, String> input3 =
sc.textFile("file3")
    .flatMap(line -> Arrays.asList(line.split(";"))).iterator())
    .mapToPair(lineParser);

long count = input2.union(input3).join(input1).count();```
DAG Representation

file1 textFile → RDD<String> → mapToPair → RDD<(String, String)> → group → RDD<(String, Iterator<String>)>

file2 textFile → RDD<String> → mapToPair → RDD<(String, String)> → filter → RDD<(String, String)> → U

file3 textFile → RDD<String> →flatMap → RDD<String> → mapToPair → RDD<(String, String)>
Further Reading

• Spark home page: http://spark.apache.org/

• Quick start: http://spark.apache.org/docs/latest/quick-start.html

• RDD documentation: http://spark.apache.org/docs/latest/rdd-programming-guide.html