A Crash Course on Apache Spark

Yesheng Ma

February 28, 2017

Overview

1. Introduction
2. Scala Syntax
3. Spark Essentials
4. Simple Quiz

Distributed System?

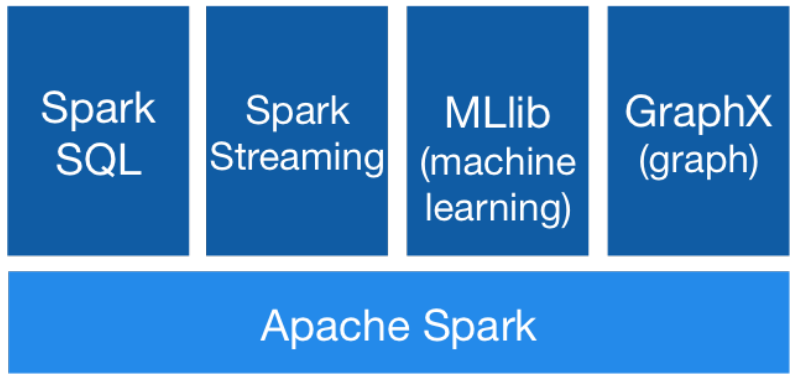
Three key components:

- Distributed file system: GFS, HDFS, Amazon S3
- Cluster management: Mesos, YARN
- Distributed computing: MapReduce, Spark

What is Apache Spark

- A general-purpose cluster computing framework
- An abstraction over Hadoop MapReduce
- Functional programming/lazy evaluation
- Founded in 2009 at AMPLab, UC Berkeley

Spark Stack: a unified engine



- Spark SQL (SQL on Spark)
- Spark Streaming (stream processing)
- MLlib (machine learning)
- GraphX (graph processing)

Scala: OO+FP

```
1 val x = "hello, world"
2 var y = 1 to 10
3 def inc(x: Int) = x + 1      // method def
4 def inc = (x: Int) => x + 1 // lambda expr
5 (1 to 10).map(x => x + 1) // method call with f
6 (1 to 10).map(_ * 10)    // placeholder, type inference
```

Example: Word Count

Doing WordCount is extremely simple in Spark: 2 lines of code

```
1 val file = sc.textFile("file:///opt/spark/README.md")
2 val wc = text.flatMap(l => l.split(" "))
3     .map(w => (w, 1))
4     .reduceByKey(_ + _).collect
```

Programming Model

In Spark, we write programs in terms of transformation on datasets. The abstraction is called Resilient Distributed Dataset(RDD).

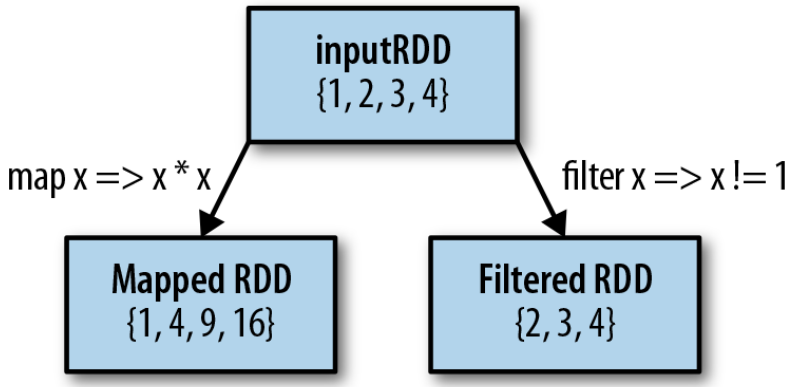
We can build(register) an RDD in two ways:

- From Scala collection, e.g. List, Array
- From file system, e.g. local fs, HDFS

Operations on RDD

We can treat RDDs as immutable data structures in functional programming:

- Transformations: map, filter, flatMap, reduce, fold etc
- Actions: take, first, saveAsText etc



Operations on RDD(cont'd)

About transformations and actions:

- transformations only register the metadata of computation
- actions trigger the computation and leads to *side effects*

Idea of lazy evaluation!

Under the hood: DAG

Spark keeps track of the transformations on an RDD and generates a *directed acyclic graph* of execution.

Then Spark schedules tasks on cluster based on optimized DAG execution plan, which is similar to compiler optimization.(Demo)

Operations on RDD(cont'd)

Do be careful when passing functions to RDD transformations: the serialization overhead can be large(pass the whole closure)!

```
1  class PassFunction {
2    val hello = "hello, world"
3    def op(x: Int): Int = x * 2 + 3
4    def main(args: Array[String]): Unit = {
5      val conf = new SparkConf().setAppName("Test")
6      val sc = new SparkContext(conf)
7      val data = sc.parallelize(1 to 100)
8      val result = data.map(op).collect
9      sc.stop()
10   }
11 }
```

Computation Reuse

We might want to cache the result of a computation and later computations can take advantage of that:

- rdd.persist(PersistLevel)
- rdd.cache()

More advanced techniques

- Broadcast variable: every worker knows the value of the broadcast variable

```
// on driver
val broadcastVar = sc.broadcast(Array(1, 2, 3))
// on executor
broadCastVar.value()
```

- Accumulator variable:

```
// on driver
val accum = sc.longAccumulator("long accum")
// on executor
accum.add(1)
```

Quiz

1. Does the dataset need to fit in memory in order to be computed?
2. Do I need to have Hadoop installed in order to run Spark?
3. How can I get the largest number in an RDD of type Int?

The End