Introduction to Spark

Kenny Lu
School of Information Technology
Nanyang Polytechnic

October 9, 2017

What is Spark?

- A distrbute cluster computing system favoring in-memory computation.
- It was developed intially for batch processing computation like MapReduce

Why Spark?

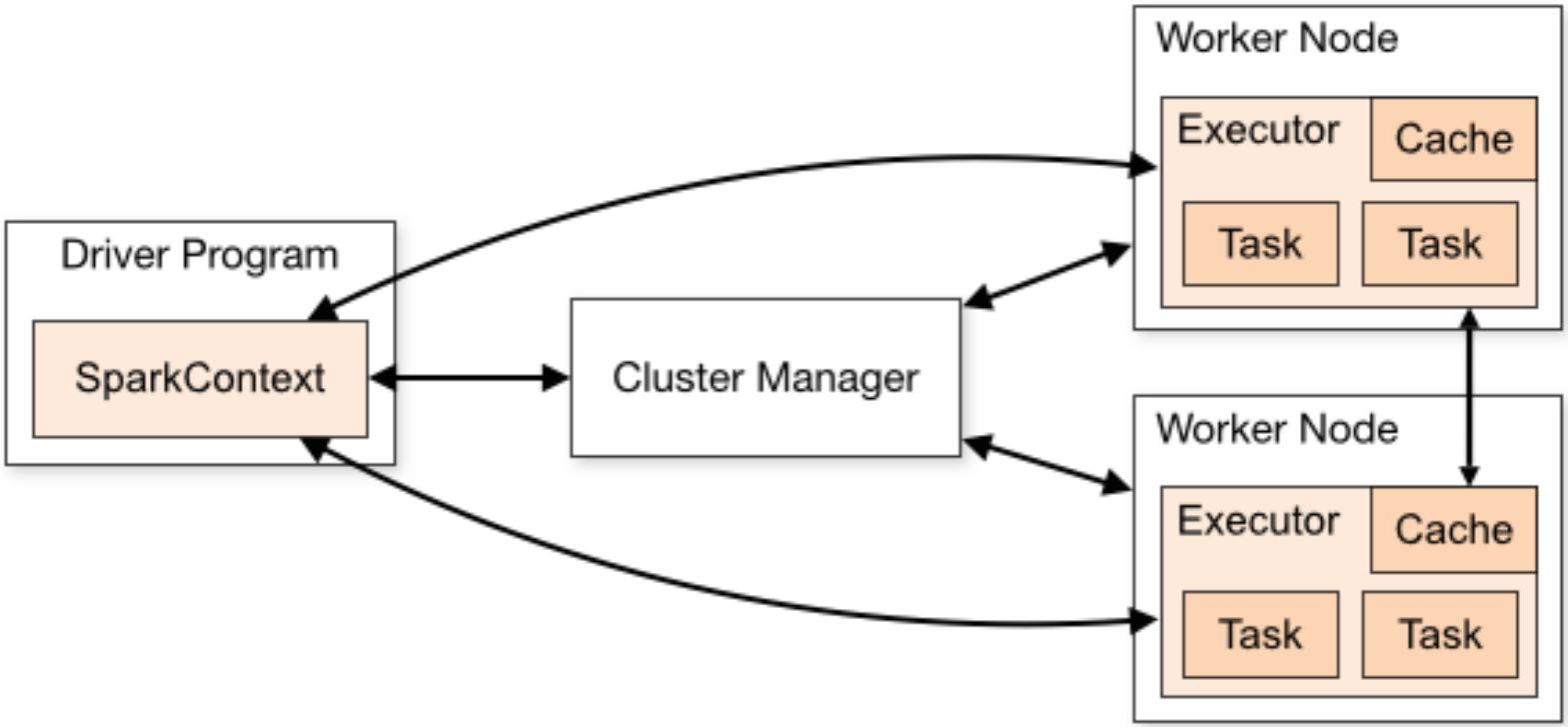
What's wrong with MapReduce?

- it was designed for moderate CPU and low memory systems.
- it relies on disk I/O operations at each intermediate steps.
- Its performance is capped by the disk I/O performance, and symmetric distribution of the Reduce jobs.

Spark comes in assuming our machines are in general more powerful, and RAMs are cheaper.

- it favors in memory computations. Data are loaded from disk and stay in memory as long as possible.
- it uses resillent distributed datasets (RDD) as the abstract data collections.
- it performs better than MapReduce if we have sufficient RAM in the cluster.

Spark Architecture



A SparkContext is an interface between the Spark Driver Program (application) and the Spark runtime-system

WordCount Example in Spark

Wordcount in Scala

```
val lines = sc.textFile("hdfs://127.0.0.1:9000/input/")
val counts = lines.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://127.0.0.1:9000/output/")
```

Wordcount in Python

```
lines = sc.textFile("hdfs://127.0.0.1:9000/input/")
counts = lines.flatMap(lambda x: x.split(' ')) \
              .map(lambda x: (x, 1)) \
              .reduceByKey(add)
couts.saveAsTextFile("hdfs://127.0.0.1:9000/output/")
```

sc denotes SparkContext

WordCount Example in Spark

Wordcount in Scala

```
val lines:RDD[String] =
    sc.textFile("hdfs://127.0.0.1:9000/input/")
val counts:RDD[(String,Long)] =
    lines.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://127.0.0.1:9000/output/")
```

Recall in Scala `List(1,2,3).map( v => v + 1)` yields `List(2,3,4)`
and `List(List(1),List(2),List(3)).flatMap( l => l )` yields `List(1,2,3)`
An RDD can be seen as a distributed list.

Resilient Distributed Dataset

- RDD is an abstraction over a collection of data set being distributed and partitioned across a cluster of worker machines, mostly in memory.
- Programmers are not required to manage or to coordinate that distributed and partitioned. RDD is fault tolerant.
- RDDs are initialized and managed by the SparkContext.

RDD transformations are pure

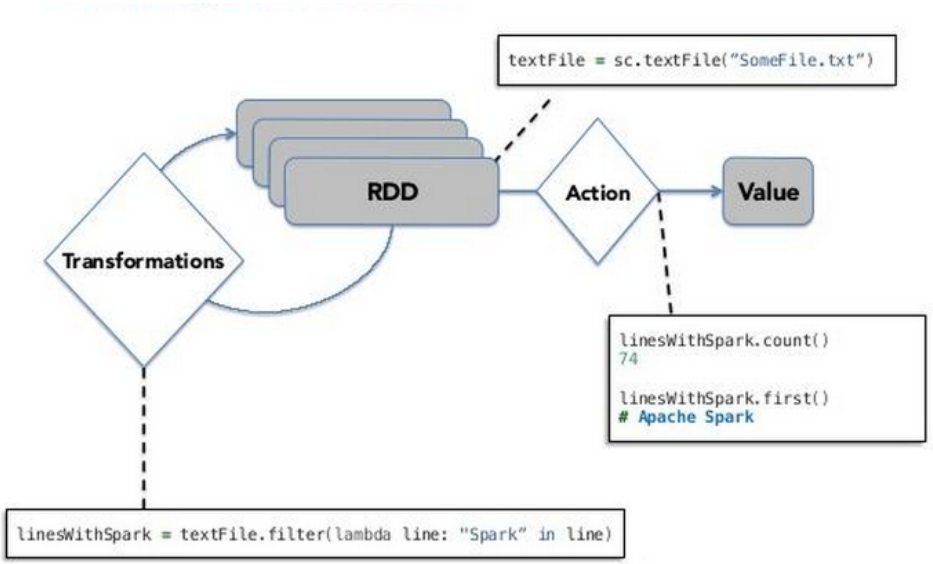


Image adapted from `http://www.hadooptpoint.com`

RDD transformations are pure

Let `r` denotes an RDD,

- `r.map(f)` and `r.flatMap(f)` applies `f` to elements in `r`.
- `r.filter(f)` filters away elements `x` in `r` which `f(x)` yields `false`.
- assuming `r` is a collection of key-value pairs, `r.reduceByKey(f)` will shuffle the pairs and group them by keys. The values grouped under the same key will be reduced by `f`. Data locality is exploit when possible.

RDD transformations are lazy

- Computations do not take place unless the results are needed.
- In memory cache are explicitly created.

Wordcount in Scala

```
val lines:RDD[String] =
    sc.textFile("hdfs://127.0.0.1:9000/input/")
val counts:RDD[(String,Long)] =
    lines.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
counts.persist() // caching
counts.saveAsTextFile("hdfs://127.0.0.1:9000/output/")
val somethingelse = counts.map( ... )
```

RDD transformations are resilient to node failure

Since computations are pure, hence they are deterministic. Final results and intermediate results can be always recomputed.

How to run it?

First start the cluster

```
$ /opt/spark-1.4.1-bin-hadoop2.6/sbin/start-all.sh
```

Run it in the REPL

```
$ /opt/spark-1.4.1-bin-hadoop2.6/bin/spark-shell
 scala> :load Wordcount.scala
```

Or we can type the code in line by line.

Submit to the cluster

Scala

```
$ /opt/spark-1.4.1-bin-hadoop2.6/bin/spark-submit
 Wordcount.jar
```

Python

```
$ /opt/spark-1.4.1-bin-hadoop2.6/bin/spark-submit
wordcount.py
```

It supports R too.