1 Spark Primer

Hadoop revolutionized how big data is processed. It did that primarily with HDFS and MapReduce. HDFS provided storage and MapReduce provided compute capability.

Each MapReduce step would pick up data from HDFS, and end up with data in HDFS. This made the system very resilient—HDFS maintains replica copies of blocks in case some machines go bad.

All that reading and writing to disk is slow—especially for intermediate results that are not required to be saved.

Another issue was that in a large job, the results of some steps are not retained, or are used multiple times. There's value in high-level optimization that a sequence of MapReduce jobs just doesn't make easy.

Here enters Spark: it's a compute engine. Instead of working in map-reduce steps, it operates on in-memory data structures called RDDs and more recently Datasets (which are just special kind of RDDs).

Spark attempts to juggle the entire computation in memory, and only flush to disk the output (or intermediate steps if memory is low). This makes Spark fast. Much faster than most map-reduce sequences, especially those with multiple steps.

Spark is also lazily evaluated. What that means is that if nobody asks for a particular output of a computation, then that computation will not be performed (even if it's defined in the code).

With Datasets, Spark got a higher level optimizer. Optimizations that you generally expect out of a relational database are now available within Spark.

2 Before you start

Spark is written in Scala. It's a functional language that's somewhat loosely based on Java. It's not Java. But some of the libraries are shared. I highly recommend installing Scala—mostly because the Spark API is much cleaner in Scala.

This is similar to instaling java on your computer, so head over to `https://www.scala-lang.org/` and grab it.

If you're running debian based linux, you can just run:

```
sudo apt-get install scala scala-doc
```

That said, Spark isn't Scala. You can use Spark from Java or Python.

3 Downloading Spark

Grab the latest version from: `http://spark.apache.org/`

As of this writing, I found: `spark-2.3.0-bin-hadoop2.7.tgz`

You can unzip that into your home folder:

```
tar -xzf ~/Downloads/spark-2.3.0-bin-hadoop2.7.tgz
ln -s spark-2.3.0-bin-hadoop2.7/ spark
```

This created `/home/alex/spark-2.3.0-bin-hadoop2.7`. We also have `/home/alex/spark` symbolic link pointing to the longer path.

Now define a variable `SPARK_HOME` to point to it. At the top of `~/.bashrc`, add:

```
export SPARK_HOME=/home/alex/spark
export PATH="$SPARK_HOME/bin:$PATH"
```

4 Running Stuff

You should now be able to run spark shell:

```
spark-shell
```

From here, you can type in whatever Spark commands you like.

Believe it or not, you can do just about everything you want from this `spark-shell` terminal. Kind of like you can do everything with PostgreSQL from `psql` terminal.

5 Running Fully Distributed Spark

The above section illustrated how to run local spark—it will use all the machine's cores, but won't talk to other machines across the network.

TODO: write up the configuration for this. It's all controlled from files in `SPARK_HOME/conf`, and there are templates for each.

6 Compiling your first program:

I highly recommend working within `spark-shell` to solve problems. It's interactive, iterative, and is an awesome environment to inch your way towards a solution to a business problem.

For production environments such code may not be an option. You might need to create a build that can be deployed on a cluster.

Create a `Test.scala` file (this is a simple app example by googling):

```
import org.apache.spark.sql.SparkSession

object Test {
  def main(args: Array[String]) {
    val logFile = "/home/alex/spark/README.md"
    val spark = SparkSession.builder.appName("TestApp").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
```

```
  }
}
```

We now need to setup the `CLASSPATH`:

```
export CLASSPATH=".:$CLASSPATH:"`find $SPARK_HOME/jars|tr "\n" ":"`
```

This brings all jars from spark folder into `CLASSPATH`. Now compile via:

```
scalac Test.scala
```

This creates a bunch of files:

```
$ ls
Test$$anonfun$1.class
Test$$anonfun$2.class
Test.class
Test$.class
Test.scala
```

You can now create the `jar` with (apparently Spark ignores the manifest file (?)):

```
jar -cvMf Test.jar Test*
```

Then can just kick it off from command line using local machine as cluster:

```
spark-submit --class Test --master local[*] Test.jar
```

If you want to see the output without any logging, just redirect stderr to `/dev/null`

```
$ spark-submit --class Test --master local[*] Test.jar 2>/dev/null
Lines with a: 61, Lines with b: 30
```