Hadoop/Spark Config

CMPT 732, Fall 2018

Config Objects

There have been config objects around, but we haven't used them much. In MapReduce and Spark:

```
Configuration conf = this.getConf();
Job job = Job.getInstance(conf, "word count");
```

```
conf = SparkConf().setAppName('word count')
sc = SparkContext(conf=conf)
```

```
spark = SparkSession.builder.appName('word count').getOrCreate()
```

The Command Line

We have modified them with command line switches:

```
yarn jar wordcount.jar WordCount -D mapreduce.job.reduces=3 …
```

```
spark-submit --num-executors=4 --executor-memory=4g …
spark-submit --conf spark.executor.memory=4g …
```

Both of these have the effect of modifying the configuration object (and thus the behaviour of the jobs).

In Code

Config options can also be modified in code (where that makes sense, e.g. not the Spark driver memory):

```
Configuration conf = this.getConf();
conf.setInt("mapreduce.job.running.map.limit", 5);
conf.setInt("mapreduce.reduce.memory.mb", 4096);
Job job = Job.getInstance(conf, "word count");
```

```
conf = SparkConf().setAppName('word count') \
    .set('spark.shuffle.compress', False)
sc = SparkContext(conf=conf)
```

```
spark = SparkSession.builder \
    .config('spark.sql.shuffle.partitions', '100') \
    .getOrCreate()
```

End result: these have the same effect.

```
spark-submit --conf spark.io.compression.codec=snappy code.py
```

```
spark = SparkSession.builder \
    .config('spark.io.compression.codec', 'snappy') \
    .getOrCreate()
```

```
spark = SparkSession.builder.getOrCreate()
spark.conf.set('spark.io.compression.codec', 'snappy')
```

Config Options

There are options to tune jobs in many, many ways:

- YARN configuration options
- HDFS configuration options
- MapReduce configuration options
- Spark Configuration options (plus more for Spark+YARN, SQL, etc.)

Spark Context/Session

We have seen both the `SparkContext` object (for RDD operations) and `SparkSession` object (for DataFrame operations).

```
conf = SparkConf().setAppName('example code') \
        .set('spark.executor.instances', 8)
sc = SparkContext(conf=conf)
```

```
spark = SparkSession.builder.appName('example code') \
        .config('spark.executor.instances', 8).getOrCreate()
sc = spark.sparkContext # SparkContext instance already there
```

Both have similar jobs:

- Are singletons: exactly one must exist (but `SparkSession.getOrCreate` will find an existing instance if it's there) to represent the connection to the master.
- Hold configuration for the application.
- Give you access to Spark functionality: creating RDDs/DataFrames, reading files, etc.

Filesystems

In both MapReduce and Spark, we have always accepted the default input filesystem: local files when running locally; HDFS on the cluster.

On our cluster, the default filesystem (Hadoop config `fs.defaultFS`) is `hdfs://nml-cloud-149.cs.sfu.ca:8020`, i.e. our HDFS server. This can be overridden with the path URL.

These are equivalent on our cluster (for MapReduce & Spark):

```
TextInputFormat.addInputPath(job,
    new Path("/user/me/data"));
TextInputFormat.addInputPath(job,
    new Path("hdfs://nml-cloud-149.cs.sfu.ca:8020/user/me/data"));
```

```
sc.textFile('/user/me/data')
sc.textFile('hdfs://nml-cloud-149.cs.sfu.ca:8020/user/me/data')
```

```
spark.read.csv('/user/me/data')
spark.read.csv('hdfs://nml-cloud-149.cs.sfu.ca:8020/user/me/data')
```

There are other URL formats you can access (MapReduce or Spark, input or output):

```
sc.textFile('file:///mnt/share/inputs')
sc.textFile('s3a://s3key:s3secret@bucket/')
```

Or any Hadoop `InputFormat`. Or in Spark DataFrames, one of the data source plugins.

```
df1 = spark.read.format('solr').options(…).load()
df2 = spark.read.format('org.elasticsearch.spark.sql') \
        .options(…).load('index/foo')
⋮
df3.write.format('couchbase').options(…).save(…)
```

Your home directories are shared on all of our cluster nodes. That means that if you really want to work with a local filesystem (not HDFS) file, you can.

```
spark.read.text('file://home/me/data/small_input_file.txt')
⋮
df.write.csv('file://home/me/results/small_output')
```

… as long as you get the permissions set properly on the directories.

Spark 2.4

Spark 2.4 was just released (2018-11-02). Some notable new features…

- A bunch of new functions, mostly to work with `ArrayType` and `MapType` values in DataFrames.
- Eager evaluation for DataFrames: the option to turn off lazy evaluation for easier debugging (but worse performance).

- More Pyspark+Pandas: user-defined aggregation functions and user-defined window functions.
- Pyspark structured streaming `ForeachWriter` and `.foreach` and `foreachBatch`.
- Spark DataFrame image input.

Available on our cluster: `module load newspark`

