Assignment 2A

In this assignment, we will start working with Spark *(https://spark.apache.org/)* and thus Python.

See PythonSpark for information working with Python and Spark code, and RunningSpark for information on getting your code to go: either on your machine or the cluster.

Web Frontends (MapReduce and Spark)

We have been interacting with the cluster on the command line only. That's probably the best way to submit jobs (where you can get your classpath and command-line arguments just-so), but various Hadoop services present web interfaces where you can see what's happening.

You need some ports forwarded from your computer into the cluster for this to work. If you created a `.ssh/config` configuration as in the Cluster instructions, then it should be taken care of.

If the port forwarding has worked, the URLs described below should now work. If you didn't set up the `.ssh/config` and are on campus, you can access these services at http://master.sfucloud.ca:[port *(http://master.sfucloud.ca:[port)*]

The HDFS NameNode can be accessed at http://localhost:50070/ *(http://localhost:50070/)* . You can a cluster summary, see the DataNodes that are currently available for storage, and browse the HDFS files (Utilities → Browse the filesystem).

Note: Our cluster is set up without authentication on the web frontends. That means you're always interacting as an anonymous user. You can view some things (job status, public files) but not others (private files, job logs) and can't take any action (like killing tasks). You need to resort to the command-line for authenticated actions.

The YARN application master is at http://localhost:8088/ *(http://localhost:8088/)* . You can see the recently-run applications there, and the nodes in the cluster ("Nodes" in the left-side menu). If you click through to a currently-running job, you can click the "attempt" and see what tasks are being run right now (and on which nodes).

Spark Shell

See RunningSpark for instructions on getting started, and start `pyspark`, a REPL (Read-Eval-Print Loop *(https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop)*) for Spark in Python.

You will have a variable `sc`, a `SparkContext` already defined as part of the environment. Try out a few calculations on an RDD:

```
>>> numbers = sc.parallelize([5, 0, 4, -1, -1, 1, 1, -2, 0, 3])
>>> numbers.take(4)
>>> pos_nums = numbers.filter(lambda n: n>0)
>>> pos_nums.collect()
>>> pos_nums
>>> pos_nums.max()
>>> distinct_nums = numbers.distinct()
>>> distinct_nums.collect()
>>> numbers.distinct().map(lambda n: n+10).collect()
```

[Aside: I feel like `.parallelize` is badly named: it would be better called `.createRDD`. Its job is to create an RDD from a Python list (or other iterable).]

You should be able to very obviously see Spark's lazy evaluation of RDDs here: you can see the Spark output when work is actually being done.

Try Some More

See the RDD object reference *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.RDD)* and try a few more methods that look interesting. Perhaps choose the ones needed to answer the questions below. [?]

Word Count

Yay, more word counting!

In your preferred text editor, save this as `wordcount.py`:

```
from pyspark import SparkConf, SparkContext
import sys
import operator

inputs = sys.argv[1]
output = sys.argv[2]

conf = SparkConf().setAppName('word count')
sc = SparkContext(conf=conf)
assert sys.version_info >= (3, 5)  # make sure we have Python 3.5+
assert sc.version >= '2.2'  # make sure we have Spark 2.2+

def words_once(line):
    for w in line.split():
        yield (w, 1)

def get_key(kv):
    return kv[0]

def output_format(kv):
    k, v = kv
    return '%s %i' % (k, v)

text = sc.textFile(inputs)
words = text.flatMap(words_once)
wordcount = words.reduceByKey(operator.add)

outdata = wordcount.sortBy(get_key).map(output_format)
outdata.saveAsTextFile(output)
```

See the RunningSpark instructions. **Get this to run** both in your preferred development environment and on the cluster. (Spark is easy to run locally: download, unpack, and run. It will be easier than iterating on the cluster and you can see stdout.)

There are two command line arguments (Python `sys.argv`): the input and output directories. Those are appended to the command line in the obvious way, so your command will be something like:

```
spark-submit wordcount.py wordcount-2 output-1
```

Improving Word Count: Part 1, as before

Copy the above to `wordcount-improved.py` and we'll make it better, as we did in Assignment1A.

Word Breaking

Again, we have the problem of word breaks being wrong, and uppercase/lowercase being counted separately:

```
$ hdfs dfs -cat output-1/part-"*" | grep -i "^better"
Better 5
better 255
better' 1
better, 27
better," 2
better--grown 1
better--or 1
better. 23
better." 7
better.-- 2
better.--Now 1
better: 1
better; 9
```

[If you're running Spark locally, you can `grep -i "^better" output-1/part*` for the same results.]

While certainly not as complete as the Java `BreakIterator`, this will give us a Python regular expression object *(https://docs.python.org/3/library/re.html)* that we can use to split the string into words:

```
import re, string
wordsep = re.compile(r'[%s\s]+' % re.escape(string.punctuation))
```

Apply `wordsep.split()` to break the lines into words, and convert all **keys to lowercase**.

This regex split method will sometimes return the empty string as a word. Use the Spark RDD filter method *(https://spark.apache.org/docs/2.2.0/api/python#pyspark.RDD.filter)* to **exclude them**.

Unicode Normalization

The Python `unicodedata` module *(https://docs.python.org/3/library/unicodedata.html)* has a function `normalize`. Use it to get your keys into **NFD normal form**, as before.

Now the output data should be pretty close to what we got from MapReduce.

Improving Word Count: Part 2, output fiddling

But MapReduce isn't our gold-standard of beautiful output.

The `sortBy` and `map` calls in the `outdata = wordcount.…` line aren't really required: they are just there to make the output more like we saw in MapReduce. **Remove them** (so the whole line is `outdata = wordcount`) to see what Spark would output without any cleanup.

More Output

The sorting and output formatting was nice. Let's put it back:

```
outdata = wordcount.sortBy(get_key).map(output_format)
```

Put this in a directory "by-word" in your output directory. That is, save to `output + '/by-word'`

We would also like to see the output of the `wordcount` sorted by frequency in "by-freq". In your program, also output the word count results (same format) **sorted by frequency**: most frequent first; sorted alphabetically within equal frequencies.

Caching

Since you are now using the value in `wordcount` twice, it must be calculated twice: once when you output "by-word" and then again to output "by-freq".

Spark doesn't keep RDDs laying around since even simple code creates a lot of them. In the original word count, there were five. If we're very verbose, we can see them:

```
text = sc.textFile(inputs)
words = text.flatMap(words_once)
wordcount = words.reduceByKey(operator.add)
words_sorted = wordcount.sortBy(get_key)
outdata = words_sorted.map(output_format)
```

...and if we're not very verbose, they don't get names but still exist:

```
outdata = sc.textFile(inputs).flatMap(words_once) \
    .reduceByKey(operator.add).sortBy(get_key) \
    .map(output_format)
```

It doesn't matter if we assign a variable to each RDD: they are still RDDs that are created (at least conceptually). If Spark kept all of them in memory, things would get bad very quickly.

The default is that Spark will throw away an RDD after its results have been used. If we want to keep the values around because **we** know we'll need them again, they can be *cached*.

Use the `.cache()` method to avoid calculating the `wordcount` RDD twice. You probably won't see a noticeable speed difference with such a small data set. (Maybe I'm seeing a fraction of a second when I run locally on my machine with the wordcount-3 *(http://cmpt732.csil.sfu.ca/datasets/)* data set?)

Reddit Average Scores

As before, we want to use data from the Complete Public Reddit Comments Corpus *(https://archive.org/details/2015_reddit_comments_corpus)* , and calculate the average score for each subreddit. You can use the same data from `/courses/732/reddit-1` and `/courses/732/reddit-2` or you can download the same at http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

Name your program `reddit-averages.py` and as before, take command-line arguments for the input and output directories.

Some hints:

- The built-in Python JSON module *(https://docs.python.org/3/library/json.html)* has a function `json.loads` that parses a JSON string to a Python object.
- Your keys and values should be as before: subreddit as key, and count, score_sum pair as value.
- Write a function `add_pairs` so `add_pairs((1,2), (3,4)) == (4, 6)`. That will be useful.
- There is no separate combiner/reducer here: just reduce to pairs and divide in another step. (Spark does the combiner-like step automatically as part of any reduce operation.)

Since we're taking JSON input, let's produce JSON as output this time. You can use `json.dumps` to convert a Python object (a tuple in this case) to JSON text. So, our output lines will look like this:

```
["xkcd", 4.489060773480663]
```

Questions

In a text file `answers.txt`, answer these questions:

1. An RDD *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.RDD)* has many methods: it can do many more useful tricks than were at hand with MapReduce. Write a sentence or two to explain the difference between `.map` and `.flatMap`. Which is more like the MapReduce concept of mapping?
2. Do the same for `.reduce` and `.reduceByKey`. Which is more like the MapReduce concept of reducing?

Submission

Submit your files to the CourSys activity Assignment 2A.

Updated Thu Sept. 21 2017, 09:55 by sbergner.