Assignment 2

About Data Files

In general, you can find the provided input data sets in the cluster's HDFS in `/courses/732/`. If you want to download the (smaller) data sets, they will be posted at https://ggbaker.ca/732-datasets/.

So the smallest word count input set was at `/courses/732/wordcount-1` and could be downloaded from https://ggbaker.ca/732-datasets/wordcount-1.zip.

In general, I probably won't mention these in the assignments, but they'll be there.

MapReduce: Most-Viewed Wikipedia Pages

Wikipedia publishes page view statistics. These are summaries of page views on an hourly basis. The file (for a particular hour) contains lines like this:

```
20160801-020000 en Aaaah 20 231818
20160801-020000 en Aaadonta 2 24149
20160801-020000 en AaagHiAag 1 8979
```

That means that on August 1 from 2:00 to 2:59 ("20160801-020000"), the English Wikipedia page ("en") titled "Aaaah" was requested 20 times, returning 231818 bytes. [The date/time as the first field is not in the original data files: they have been added here so we don't have to retrieve them from the filename, which is a bit of a pain.]

Create a MapReduce class `WikipediaPopular` that **finds the number of times the most-visited page was visited each hour**. That is, we want output lines that are like "20141201-000000 67356" (for midnight to 1am on the first of December).

- We only want to report English Wikipedia pages (i.e. lines that have "`en`") in the second field.
- The most frequent page is usually the front page (`title == "Main_Page"`) but that's boring, so don't report that as a result. Also, "special" (`title.startsWith("Special:")`) are boring and shouldn't be counted.

You will find small subsets of the full data set named `pagecounts-with-time-0`, `pagecounts-with-time-1`, and `pagecounts-with-time-2`.

Starting with Spark: the Spark Shell

See RunningSpark for instructions on getting started, and start `pyspark`, a REPL (Read-Eval-Print 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:

```
>>> sc.version # if it's less than 3.3.0, you missed something
'3.3.0'
>>> numbers = sc.range(50000000, numSlices=100)
>>> numbers
>>> numbers.take(10)
>>> def mod_subtract(n):
        return (n % 1000) - 500

>>> numbers = numbers.map(mod_subtract)
>>> numbers.take(10)
>>> pos_nums = numbers.filter(lambda n: n>0)
>>> pos_nums
>>> pos_nums.take(10)
>>> pos_nums.max()
>>> distinct_nums = numbers.distinct()
>>> distinct_nums.count()
```

You should be able to see Spark's lazy evaluation of RDDs here. Nothing takes any time until you do something that needs the entire RDD: then it must actually calculate everything.

Local vs Cluster

Make sure you can work with Spark (using `pyspark` for now, and `spark-submit` soon) on both your local computer, and on the cluster. Feel free to put an extra `0` on the end of the range size for the cluster.

The RunningSpark page has instructions for both, and this would be a good time to make sure you know how to work with both environments.

Try Some More

See the RDD object reference and try a few more methods that look interesting. Perhaps choose the ones needed to answer the questions below. [❓]

Web Frontends (MapReduce and Spark)

We have been interacting with the cluster on the command line only. 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.

The HDFS NameNode can be accessed at http://localhost:9870/. 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/. 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).

The pyspark shell is the easiest way to keep a Spark session open long enough to see the web frontend. Start `pyspark` on the cluster, do a few operations, and have a look around in the Spark web frontend through YARN.

You can see the same frontend if you're running Spark locally at http://localhost:4040/.

Spark: Word Count

Yay, more word counting!

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

```
from pyspark import SparkConf, SparkContext
import sys

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 >= '2.3'  # make sure we have Spark 2.3+

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

def add(x, y):
    return x + y

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(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-1 output-1
```

Spark: Improving Word Count

Copy the above to `wordcount-improved.py` and we'll make it better, as we did in Assignment 1.

Word Breaking

Again, we have a problem with `wikipedia_popular.py` tokenizing word incorrectly, and upperease/lowercase being counted separately.

We can use a Python regular expression object 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 to **exclude them**.

Spark: Most-Viewed Wikipedia Pages

Let's repeat the first problem in this assignment using Spark, in a Python Spark program `wikipedia_popular.py`. With the same input, produce the same values: for each hour, how many times was the most-popular page viewed?

Spark is far more flexible than Hadoop so we need to pay more attention to organizing the work to get the result we want.

1. Read the input file(s) in as lines (as in the word count).
2. Break each line up into a tuple of five things (by splitting around spaces). This would be a good time to convert he view count to an integer. (`.map()`)
3. Remove the records we don't want to consider. (`.filter()`)
4. Create an RDD of key-value pairs. (`.map()`)
5. Reduce to find the max value for each key. (`.reduceByKey()`)
6. Sort so the records are sorted by key. (`.sortBy()`)
7. Save as text output (see note below).

You should get the same values as you did with MapReduce, although possibly arranged in files differently. The MapReduce output isn't the gold-standard of beautiful output, but we can reproduce it with Spark for comparison. Use this to output your results (assuming `max_count` is the RDD with your results):

```
def tab_separated(kv):
    return "%s\t%s" % (kv[0], kv[1])

max_count.map(tab_separated).saveAsTextFile(output)
```

At any point you can check what's going on in an RDD by getting the first few elements and printing them. You probably want this to be the last thing your program does, so you can find the output among the Spark debugging output.

```
print(some_data.take(10))
```

Improve it: find the page

It would be nice to find out *which* page is popular, not just the view count. We can do that by keeping that information in the value when we reduce.

Modify your program so that it keeps track of the count and page title in the value: that should be a very small change. [❓]

Finally, the output lines should look like this:

```
20160801-000000     (146, 'Simon_Pegg')
```

Questions

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

1. In the `WikipediaPopular` class, it would be much more interesting to find the page that is most popular, not just the view count (as we did with Spark). What would be necessary to modify your class to do this? (You don't have to actually implement it.)
2. An 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?
3. Do the same for `.reduce` and `.reduceByKey`. Which is more like the MapReduce concept of reducing?
4. When finding popular Wikipedia pages, the maximum *number* of page views is certainly unique, but the most popular *page* might be a tie. What would your improved Python implementation do if there were two pages with the same highest number of page views in an hour? What would be necessary to make your code find *all* of the pages views the maximum number of times? (Again, you don't have to actually implement this.)

Submission

Submit your files to the CourSys activity Assignment 2.

Updated Wed Sept. 14 2022, 16:11 by ggbaker.