Assignment 2B

Course Rules

Spark is flexible enough to let you get yourself into trouble. In particular, you can easily do thing not in parallel and lose the ability to do actual big data.

If you use `.collect()` to turn an RDD into a Python list, you **must** have a comment nearby that gives a bound on the size of the data, to indicate that you have thought about the implications of turning it into a non-distributed collection.

Same rules for `.coalesce(1)` or similar operations and destroy any parallelism you have.

```
# keys here are small integers; <=100 elements collected
data = rdd1.reduceByKey(f).collect()
# RDD has ~10k elements after grouping.
rdd3 = rdd2.groupByKey().coalesce(1)
```

Log Correlation

For this question, we will be imagining a situation where we have web server logs to analyse. Dealing with server logs is a fairly realistic task, but publicly available HTTP logs are hard to find because of privacy concerns. There are some old NASA web server logs *(http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html)* available that are big enough to experiment with.

The data we will use is on the cluster HDFS at `/courses/732/nasa-logs-1` and `/courses/732/nasa-logs-2` or can be downloaded at http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

I have a theory: I think that in our web server logs, the number of bytes transferred to a host might be correlated with the number of requests the host makes. I think this is a good theory.

In order to check this theory, I would like to calculate the correlation coefficient *(https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient#For_a_sample)* of each host's number of requests and total bytes transferred. That is, each host making requests will contribute a data point (x, y) where x is the number of requests made, and y is the number of bytes transferred. Then the correlation coefficient can be calculated as,

$$r = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

The formula looks bad, but note that there are really only six sums we need: $n = \sum 1$ (the number of data points), $\sum x_i$, $\sum y_i$, $\sum x_i^2$, $\sum y_i^2$, $\sum x_i y_i$.

That implies an algorithm like:

1. For each host, add up the number of request and bytes transferred, to form a data point (x_i, y_i).
2. Add the contribution of each data point to the six sums.
3. Calculate the final value of r.

Write a Spark program `correlate-logs.py` that loads the logs from the input files, parses our useful info, and calculates the correlation coefficient of each requesting host's request count and total bytes transferred. As before, take arguments for the input and output files. Our output here is really just the value for r. We also care about r^2 because it's meaningful. For debugging if nothing else, let's have a look at all of the values we calculated as well. Output should be like this:

```
r = 0.928466
r^2 = 0.862048
```

You may also want to output the six sums for debugging (and feel free to leave them in your final version).

Hints

We can use a regular expression (with the Python re *(https://docs.python.org/2/library/re.html)* module) to disassemble the log lines.

```
line_re = re.compile(r'^(\S+) - - \[(\S+) [+-]\d+\] "[A-Z]+ (\S+) HTTP/\d\.\d" \d+ (\d+)$')
```

There are a few lines that don't match this regex, and you'll have to ignore them.

After you get the pairs of values (count and bytes, or x and y as we'll start calling them), produce an RDD of the six values you need to sum. Sum those tuples, and you're basically done.

This function adds two tuples of any length pairwise (`add_tuples((1,2,3), (5,6,7)) == (6,8,10)`) and I find very handy:

```
def add_tuples(a, b):
    return tuple(sum(p) for p in zip(a,b))
```

Log Correlation: better formula

Wikipedia tells us *(https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient#For_a_sample)* that this formula may be more numerically stable:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}.$$

Write a program `correlate-logs-better.py` that does the same as above, but using this formula to produce its results. (No algebraic manipulation! Floating point numbers and real numbers are not the same thing. Algebra only works on the second.)

The program will be the same until you produce the (x, y) pairs. Then you need to calculate $\bar{x}$ and $\bar{y}$ (the means of x and y values).

Calculate sums as before, by producing an RDD of the values you need to add and then reducing. The added difficulty here is that you have to include the $\bar{x}$ and $\bar{y}$ values to get there. Then calculate the final results and output as before. (In this case, you'll find the results are the same: we aren't having numerical error problems in the simpler formula.)

Reddit Relative Scores

For this question, we want to know who made **the best** comment on Reddit. But some communities are friendlier than others: we will define "the best" as the comment with the highest score relative to the subreddit's average. That is, we are looking for the largest values of (comment score)/(subreddit average score).

Make a copy of your subreddit average code `relative-score.py`. As before, have input and output directories as arguments on the command line.

The first thing we need to do is to calculate the same RDD of pairs (subreddit, average score). Let's ignore any subreddits with averages 0 or negative.

Now we need the comment data again: make sure you `.cache()` the RDD after applying `json.loads` since that step is expensive and it's where we'll be starting the next calculation.

Form a pair RDD with the subreddit as keys and comment data as values (adjusting variable names as you like):

```
commentbysub = commentdata.map(lambda c: (c['subreddit'], c))
```

Use the .join() *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.RDD.join)* method to get the average scores together with the comments themselves.

We want to report the *author* of the comment, so create an RDD of `comment['score']/average` and `comment['author']`.

Sort by the relative score (descending) and output to files. (Don't coalesce here: you have no reason to believe that the output will be small.)

Get this working on the same data sets as before: `/courses/732/reddit-1` and `/courses/732/reddit-2` or from http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

Go bigger!

Congratulations! You have been given your own cluster and real big data work to do. Your cluster has four nodes with quad core processors, and 4GB of memory per node. (It's not much of a cluster, but a bunch of them have to live inside our cluster so we'll make it work.)

We can simulate such a situation by invoking Spark on our Cluster like this:

```
spark-submit --conf spark.dynamicAllocation.enabled=false --num-executors=4 --executor-cores
```

When you start the relative scores code like this, everything should go fine. [I admit it: the last question was just a ruse to get you here.]

Make sure you are connected so you can access the web frontends as at the start of Assignment2A. You'll want them in a minute…

It's time for some big(ger) data. I have created another data set on the cluster's HDFS at `/courses/732/reddit-4/` (it's too big to really post for download, but it's on the cluster login node at `/home/bigdata/reddit-4.zip` if you want to SCP when you're on campus). In small groups, run the above algorithm on it, with our cluster and the above `spark-submit` command. (If you all do this individually at the same time, you'll overwhelm our cluster.)

While you're waiting, have a look at the YARN frontend (port forwarded at http://localhost:8088/ *(http://localhost:8088/)*). Click on your job's ApplicationMaster link. [If you're off campus, you'll have to edit the URL and change “nml-cloud-199.cs.sfu.ca” to “localhost”.] You should be able to see the beautiful Spark UI. Have a look around: you've got some time to kill.

When you're bored with that, kill the application since it's not going to finish any time soon. You can either go to the YARN frontend, click the application and then “Kill Application”, or `yarn application -kill <APPLICATION_ID>`.

We have failed to actually process big data with this implementation.

Make it faster! (or, “Use technology to solve our problems.”)

That was too slow.

Part of the problem here is Python: it's not a language that was designed to go fast. Most people (university faculty members included) believe that it can't be fast. These people have missed what has happened to compiler technology in the last few years. Just-in-time compilers *(https://en.wikipedia.org/wiki/Just-in-time_compilation)* are now a thing: an amazing thing.

PyPy *(http://pypy.org/)* is a Python implementation that includes a just-in-time compiler. On our cluster, you can turn it on:

```
module load pypy3
```

In case you also use *module load* to setup spark, ensure that you load the pypy module before that. See “Spark and PyPy” on the RunningSpark page for a few more details if you're interested.

Now run your code on the `reddit-4` data set. It should go faster now. And eventually start failing.

Kill it again.

Your program (if it's doing the same thing as mine) is failing tasks because it's running out of memory in the executor at (or near) the `.join`.

Memory isn't infinite! (or "Be clever to solve our problems.")

What we're trying to do here is actually a little dumb: the data set we're failing on only contains data for three subreddits: the RDD containing the averages-by-subreddit is tiny. Then we're joining it against a large RDD, making many copies of that information.

Make a copy of your `relative-score.py` called `relative-score-bcast.py` and make the changes below…

Since it's relatively small, could just turn it into a Python object (probably a dict with `dict(averages.collect())`) and send that to each executor. We can take the average scores RDD, `.collect()` it, and use sc.broadcast() *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.SparkContext.broadcast)* to get it sent out (it returns a Broadcast object *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.Broadcast)*).

Once you broadcast a value, it is sent to each executor. You can access it by passing around the broadcast variable and any function can then get the `.value` property out of it.

To calculate the relative score, we map over the comments: write a function that takes the broadcast variable and comment as arguments, and returns the relative average as before. Make sure you're **actually using the broadcast object**: you should have to access it as `averages.value` inside the function that runs on each executor.

This technique is a *broadcast join*. Now you should actually be able to run it on the `reddit-4` data set on the cluster. [❓]

Let's be clear: on your little 4 × 4 GB cluster, you could have yelled "I need more RAM", spent a bunch of money, caused a headache for the operations people, and made your original program run. Or you could have been just a little bit clever.

Go Bigger With Word Count!

We have one more input set for word counting: text extracted from the Project Gutenberg DVD *(http://www.gutenberg.org/wiki/Gutenberg:The_CD_and_DVD_Project)* , which totals 1.9GB when GZIP compressed. It's on the cluster as `/courses/732/wordcount-4`.

Try your word count code from last week on it (either version, but make sure you have the `.cache()` right if you're using the "improved" version). Feel free to do this in small groups, just to keep the cluster load down, but answer the questions individually.

While the job is running, keep an eye on the stages and executors in the YARN/Spark web frontend so you can see how things are going.

Your code likely takes more than 5 minutes, but could be much faster (about half of that). Diagnose the problem by looking at the data and the way the calculation progresses in the Spark web frontend.

Fix the program so it processes this input faster. (big hint *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.RDD.repartition)*) (We're not asking for the code: just answer the question below.) [❓]

Questions

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

1. Under what conditions will the broadcast join be faster than an actual join?
2. When will the broadcast join be slower?
3. What was the problem with the input data `wordcount-4`? How did you fix the wordcount program so it handled this input and processed it more quickly?

4. Note that MapReduce also had something very much like broadcasting in its DistributedCache. Also, Spark has Accumulators *(https://spark.apache.org/docs/2.2.0/api/python/pyspark.html#pyspark.Accumulator)* which are very similar to MapReduce's counters. This isn't really a question, I just couldn't find anywhere else to make this point.

Submission

Python code should be beautiful and readable. There will be some “style” component to the marking from now on.

Submit your files to the CourSys activity Assignment 2B.

Updated Wed Oct. 04 2017, 11:14 by sbergner.