Assignment 1B

Important Note

In the huge Hadoop class hierarchy, you might notice both org.apache.hadoop.**mapred** and org.apache.hadoop.**mapreduce** classes. The madred classes are the old Hadoop API and should be avoided in new code: always make sure you're looking at the reference (and tutorials) for the mapreduce classes.

Less Important Note

In various docs, you will see different variants of the Hadoop commands, which are basically the old and new ways of doing things. These two commands are exact synonyms:

```
hadoop jar ...
yarn jar ...
```

And so are these:

```
hadoop fs ...
hdfs dfs ...
```

The newer style is to not use the catch-all `hadoop` command, but be more specific about which piece of Hadoop technology you're addressing. I will try to consistently use the newer style.

Most-Viewed Wikipedia Pages

Wikipedia publishes page view statistics *(https://dumps.wikimedia.org/other/pagecounts-raw/)* . These are summaries of page views on an hourly basis. The file (for a particular hour) contains lines like this:

```
en Aaaah 20 231818
en Aaadonta 2 24149
en AaagHiAag 1 8979
```

That means that during this hour, the English Wikipedia page ("en") titled "Aaaah" was requested 20 times, returning 231818 bytes.

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 "20141201-00 67369" (for midnight to 1am on the first of December).

- We only want to report English Wikipedia pages (i.e. lines that start with `"en "`).
- 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.

The filename contains the date and hour for these input files. The one being read in the mapper can be found like this:

```
((FileSplit) context.getInputSplit()).getPath().getName()
```

You will find small subsets of the full data set on the cluster's HDFS at `/courses/732/pagecounts-0`, `/courses/732/pagecounts-1`, and `/courses/732/pagecounts-2` (you can just use those as input directories on the cluster: no need to make your own copies) or you can download the same at http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

JSON Input & Reddit Comments

It is quite common for large data sets to be distributed with each record represented as a JSON *(https://en.wikipedia.org/wiki/JSON)* object, with one object per line in the file(s). That way, input files can be split by line (as the default `TextInputFormat` does for you), and each line is in a well understood and easy to parse format. The input files end up looking like this:

```
{"key1": "value1", "key2": 2}
{"key1": "value3", "key2": 4}
```

For example, this is how the Complete Public Reddit Comments Corpus *(https://archive.org/details/2015_reddit_comments_corpus)* is distributed (with each file BZ2 compressed). For this problem, we will be using (a subset of) it, and determining the **average score in each subreddit**.

Create a `RedditAverage` class for this problem.

You will find small subsets of the full data set on the cluster at `/courses/732/reddit-1` and `/courses/732/reddit-2` (you can just use those as input directories on the cluster: no need to make your own copies) or you can download the same at http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

Parsing JSON

The input to our mapper will be lines (from `TextInputFormat`) of JSON-encoded data. In the mapper, we will need to parse the JSON into actual data we can work with.

We will use the Jackson JSON parser *(http://wiki.fasterxml.com/JacksonHome)* , which is apparently a good one. You will need three JARs: jackson-annotations-2.8.10.jar *(http://repo2.maven.org/maven2/com/fasterxml/jackson/core/jackson-annotations/2.8.10/jackson-annotations-2.8.10.jar)* , jackson-core-2.8.10.jar *(https://repo1.maven.org/maven2/com/fasterxml/jackson/core/jackson-core/2.8.10/jackson-core-2.8.10.jar)* , jackson-databind-2.8.10.jar *(http://repo1.maven.org/maven2/com/fasterxml/jackson/core/jackson-databind/2.8.10/jackson-databind-2.8.10.jar)* . Here is a very quick tutorial on the parts of Jackson we need:

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

ObjectMapper json_mapper = new ObjectMapper();

JsonNode data = json_mapper.readValue(input_string, JsonNode.class);
System.out.println(data.get("subreddit").textValue());
System.out.println(data.get("score").longValue());
```

See HadoopExternalJARs for instructions on getting these JAR files into your Hadoop jobs.

Passing Pairs

Since we want to calculate average score, we will need to pass around *pairs*: the number of comments we have seen, and the sum of their scores. I have written a quick `Writable` implementation `LongPairWritable` to do this for us. Here is how it is used:

```
LongPairWritable pair = new LongPairWritable();
pair.set(2, 9);
System.out.println(pair.get_0()); // 2
System.out.println(pair.get_1()); // 9
```

Your mapper should be writing pairs of `Text` (the subreddit name) and `LongPairWritable` (the comment count and total score).

Definitely test at this stage. Remember `-D mapreduce.job.reduces=0` to turn off the reducer and see the map output directly.

Reducing to Averages

Write a reducer that takes the mapper's key/value output (`Text` , `LongPairWritable`) and calculates the average for each subreddit. It should write one `Text` , `DoubleWritable` pair for each subreddit. [❓]

The Combiner

At this point, the job should be giving correct output, but the mapper like probably producing lots of pairs like this:

```
canada  (1,1)
canada  (1,9)
canada  (1,8)
canada  (1,1)
```

Those all have to be shuffled to the reducer, but it would be much more efficient to combine them into:

```
canada  (4,19)
```

Remember: the combiner does reducer-like work on chunks of the mapper output, with the goal of minimizing the amount of data that hits the shuffle. The input and output of the combiner must be the same types.

Since our reducer must produce output that is a different type than its input, it doesn't make sense to use it as a combiner. (Go ahead: try it. It should fail complaining about types being wrong.)

Write a combiner (that `extends Reducer` just like a reducer class) that sums the count/score pairs for each subreddit. It should be similar to your reducer, but write long pair values, instead of doubles. [❓]

Multi-Line Input

The input you get isn't always in a format you'd like. For this question, we will imagine that somebody heard about the “file of JSON objects” approach, but missed the part where each JSON object should be encoded *on a single line*, and we ended up with input like:

```
{
  "key1": "value1",
  "key2": 2
}
{
  "key1": "value3",
  "key2": 4
}
```

We have files that aren't one-record-per-line, but at least we note that **each record starts with a '{' as the first character on a line**. Everything between that and the next (or the end of the file) is one string that can be JSON decoded.

Create an `InputFormat` implementation `MultiLineJSONInputFormat` that reads files like this (in a separate `.class` file). That is, it should be an `InputFormat<LongWritable, Text>` that returns file offsets and multi-line strings containing JSON text (not yet decoded: continue to let the mapper do that). See the provided InputFormat skeleton for some help.

Use this as the `InputFormat` for your `RedditAverage` class: once it's working, it should work just as well on the nice single-line inputs as the ugly multi-line ones.

You will find multi-line versions of the data sets from the above problem on the cluster at `/courses/732/redditmulti-1` and `/courses/732/redditmulti-2` or download from http://cmpt732.csil.sfu.ca/datasets/ *(http://cmpt732.csil.sfu.ca/datasets/)* .

Parallel Computation

Sometimes the work you want to do on a cluster doesn't depend on a large set of input data, but is just a big calculation. Let's try that…

We can estimate Euler's consant, e *(https://en.wikipedia.org/wiki/E_(mathematical_constant))* using a stochastic representation *(https://en.wikipedia.org/wiki/E_%28mathematical_constant%29#Stochastic_representations)* . Given a sequence $X_1, X_2, \ldots$ of uniform random variables on the range $[0, 1]$, we can take the smallest number of these such that the total is greater than one:

$$V = \min\{n \mid X_1 + X_2 + \cdots + X_n > 1\}.$$

The expected value $E(V) = e$.

What that means to us: we can use the law of large numbers *(https://en.wikipedia.org/wiki/Law_of_large_numbers)* , sample the values of V a bunch of times, average, and use that as an estimate of e. All we need to do is generate random numbers in $[0, 1]$ until the total is more than 1: the number of random values we needed for that is an estimate of e. If we average over many samples, we will get a value close to e.

Implementation

While it is possible to have MapReduce tasks that take no input from files (by creating an `InputFormat` that gives the mapper values it needs), we will be continuing to use small files to trigger computation.

Create a class `EulerEstimator` that uses java.util.Random *(http://docs.oracle.com/javase/7/docs/api/java/util/Random.html)* to generate random values and estimate e.

- Your input files should be files that contain a single integer on each line (use the default `TextInputFormat`). The integer indicates the number of iterations for your main loop. Get the input path from the first command line argument, as usual.
- In the mapper, repeat the estimation of the random variable the number of times indicated in the input (i.e. `long iterations = Integer.parseInt(value.toString())` and repeat the sampling `iterations` times).
- Use Hadoop counters to count the total number of iterations you have make, and the number of random numbers you had to generate. (More details below.)
- You won't `context.write` any key/value pairs in the mapper, so there is no need for a reducer.
- Set the output format to NullOutputFormat *(http://cmpt732.csil.sfu.ca/hadoop-javadoc/org/apache/hadoop/mapreduce/lib/output/NullOutputFormat.html)* , since there won't actually be any output.

Counters

Hadoop's "counters" are shared variables that hold integers which can *only* be incremented (not even read by the individual tasks). In your mapper, you can add code like this:

```
context.getCounter("Euler", "iterations").increment(iterations);
context.getCounter("Euler", "count").increment(count);
```

And when the code is run, you will see the values of all counters in the output like this:

```
$ ${HADOOP_HOME}/bin/yarn jar euler.jar EulerEstimator e-inputs
⋮ [many things happen]
        Euler
                count=271832537
                iterations=100000000
⋮ [a few more counters]
```

This implies $e \approx 2.71832537$ which is pretty close to the correct value of $2.718281828\ldots$.

Pseudocode

The logic for the mapper is basically this:

```
iter = [integer from input line]
count = 0
for i from 1 to iter
    sum = 0.0
    while sum < 1
        sum += [random double 0 to 1]
        count ++

iterations_counter += iter
count_counter += count
```

Randomness

The implementation of the `Random()` constructor (in OpenJDK's Random.java *(http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/Random.java#l104)* at least) uses the current time to seed the random number generator: it is certainly possible that we will be starting tasks on different nodes at the exact same time, leading to pseudoreplication.

Update your `Mapper.map` method so it seeds the random number generator with a value you know to be unique (with very high probability). Hint: you can get the input filename *(https://stackoverflow.com/questions/19012482/how-to-get-the-input-file-name-in-the-mapper-in-a-hadoop-program)* (and then its `.hashCode` is an integer), and the first argument to `map` is the position of the line within the file (offset from beginning in bytes, a `LongWritable`).

MapReduce Input and Tasks

Try your MapReduce job on two sets of input from euler-1.zip *(http://cmpt732.csil.sfu.ca/datasets/euler-1.zip)* and euler-2.zip *(http://cmpt732.csil.sfu.ca/datasets/euler-2.zip)* . One is one file with 100 lines, and the other is 100 files each with one line. Both sets of input imply $\mathbf{3 \times 10^9}$ iterations of the test, so they represent the exact same calculation.

Try both sets on the cluster. How long does each take to complete? Why should they be different? [❓]

You can learn things about completed jobs like this: (start and finish times in milliseconds after some epoch)

```
yarn application -list -appStates FINISHED
yarn application -status application_1111111111111_0001
```

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. What would be necessary to modify your class to do this? (You don't have to actually implement it.)
2. Was there any noticeable difference in the running time of your `RedditAverage` with and without the combiner optimization?
3. For the given sample inputs to `EulerEstimator`, how long did it take each take to complete on the cluster? Why would they be different?

Submission

Submit your files to the CourSys activity Assignment 1B.

Updated Wed Sept. 06 2017, 15:57 by ggbaker.