CMPT 431: Distributed Systems

Assignment 4, Spring 2019

This assignment is to be done in groups of 2!

Note: In this assignment, you can use any programming language of your choice.

Pointers

1. MapReduce tutorial: MapReduce
2. Apache Spark quick start: Spark
3. How to connect and access Hadoop-Cloudera cluster: *Connecting to cluster.*
4. How to compile code in Hadoop: *Compiling code in Hadoop.*
5. How to run your spark code locally (standalone) and on cluster: *Running spark*
6. How to structure your spark code (skeleton): *Spark started code structure*
7. Where to get the dataset: *dataset*
8. How do I view my spark and MapReduce jobs in the web-frontend?*: web frontend*

Additional References:

1. Why Hadoop MapReduce and/or Spark?
2. Getting started with Spark.
3. Essential Hadoop commands.
4. A real-life example of spark in action.

Note that Spark is independent of Hadoop, whereas MapReduce runs on top of Hadoop. YARN is the Hadoop based scheduler.

Q1. MapReduce

Wikipedia publishes page view statistics, which 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

The above 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 file: it has been added here so we don't have to retrieve them from the filename, which is a bit of a pain.

A. **(30 marks)** Create a MapReduce class *WikipediaPopular* that finds the number of times the most-visited page was visited each hour in the dataset. That is, we want output lines that look like “20141201-000000 67369” (for midnight to 1am on the first of December). Report only English pages. Ignore the front page (title == "Main_Page") and “special” (title.startsWith("Special:")).
B. Take a look at the *Web-frontend* to see the distribution of work among the worker nodes and to understand how Hadoop-YARN schedules tasks. You will find small subsets of the full data set named pagecounts-with-time-0, pagecounts-with-time-1, and pagecounts-with-time-2 (use any).

C. **(10 marks)** Run the program once on a single-core processor and once on a multi-core processor. If you can't find a single-core processor, use lower core and compare with higher core. Report the difference in execution time of part A, once with single (low) and once with multi (high) core.

Q2. Spark

A. **(35 marks)** Repeat the problem in Q1, but with Spark, using RDD or DataFrame. With the same input, produce the same output: for each hour, how many times was the most-popular page viewed?

We don't get as much structure with Spark, so we have to figure out how to put things together to get the result we want. It is suggested to proceed as follows:

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 the 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 hints below).

Hints:

- 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))
```

B. Take a look at the *Web-frontend* to see the distribution of work among the worker nodes and to get an understanding of how Spark schedules/optimizes tasks.
C. **(10 marks)** Run the program once on a single (or low) core processor and once on a multi (or high) core processor. Report the difference in execution time of part A.

Q3.

A. **(5 marks)** What are the advantages of Spark over MapReduce?
B. **(5 marks)** How does Spark DataFrame speed up computation over Spark RDDs?
C. **(5 marks)** Do Spark over MapReduce seem to take good advantage of more cores?