Project 5: Distributed Computing with Apache Spark

Overview

In this project, you will learn how to use the popular Apache Spark engine to perform a heavy computational task on a large number of machines using the Map-Reduce framework. In particular, you will identify the most frequent book pairs that are reviewed together by the same users on the GoodReads service. Our new "spark" container has the Spark engine preinstalled. We also provide a snapshot of the book lists reviewed by GoodReads users. Your job is to write a simple code on Spark that returns the book ID pairs that are frequently reviewed together.

Development Environment

The main development of Project 5 will be done using a new docker container named "spark" created from "junghoo/spark". Use the following command to create and start this new container:

```
$ docker run -it -p 4040:4040 -v {host_shared_dir}:/home/cs143/shared --name spark ju
```

Make sure to replace `{host_shared_dir}` with the name of the shared directory on your host. The above command creates a docker container named `spark` with appropriate port forwarding and directory sharing.

Our new container has PySpark (v3.0) preinstalled. As before, the default username inside the container is "cs143" with password "password". Once setup, the container can be restarted any time using the following command:

```
$ docker start -i spark
```

Part A: Learning Apache Spark

Writing a program that runs on a large number of machines in parallel can be a daunting task. A number of distributed software infrastructures have been developed to make this task easier. In particular, as we learned in class, Map-Reduce framework asks the programmer to provide just the core computational logic of the given task as a set of Map and Reduce functions. Given these core functions, Map-Reduce framework takes care of the rest, including data distribution, parallel execution, process monitoring, and result shuffling and collection. Apache Spark is popular open-source software that supports Map-Reduce-style programming on a large number of distributed machines.

Once you are inside our `spark` container, you can run the Spark interactive shell by executing the `pyspark` command:

```
$ pyspark
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/loc
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective a
WARNING: All illegal access operations will be denied in a future release
21/02/18 18:18:25 WARN NativeCodeLoader: Unable to load native-hadoop library for you
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(new
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Python version 3.8.5 (default, Jul 28 2020 12:59:40)
SparkSession available as 'spark'.
>>>
```

Inside the Spark shell, you can execute any Python command using Spark API. You can exit from the Spark interactive shell by pressing “Control+D” key.

Now that we know how to start the Apache Spark Shell inside our container, learn how to use it by going over an Apache Spark tutorial on the Internet. For example, the official Quick Start Tutorial provides a ten-minute introduction to essential basics. The file wordCount.py also contains the example code that we went over in the class.

To run the code in wordCount.py interactively in the Spark shell, download the input.txt file into the current directory, make sure that there is no `output` directory, run the Spark shell, and type the following code block of `wordCount.py` in the Spark shell

```
lines = sc.textFile("input.txt")
words = lines.flatMap(lambda line: line.split(" "))
word1s = words.map(lambda word: (word, 1))
wordCounts = word1s.reduceByKey(lambda a, b: a+b)
wordCounts.saveAsTextFile("output")
```

```
$ wget https://oak.cs.ucla.edu/classes/cs143/project5/input.txt
$ rm -rf output
$ pyspark
...
Using Python version 3.8.5 (default, Jul 28 2020 12:59:40)
SparkSession available as 'spark'.
>>> lines = sc.textFile("input.txt")
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> word1s = words.map(lambda word: (word, 1))
>>> wordCounts = word1s.reduceByKey(lambda a, b: a+b)
>>> wordCounts.saveAsTextFile("output")
>>> [Press Ctrl+D and exit spark shell]
$ ls -l output/
total 8
-rwxr-xr-x 1 root root    0 Mar 16 03:18 _SUCCESS
-rwxr-xr-x 1 root root 2682 Mar 16 03:18 part-00000
-rwxr-xr-x 1 root root 2609 Mar 16 03:18 part-00001
$ head -5 output/part-00000
('WASHINGTON', 1)
('President', 1)
('Trump', 3)
('Monday', 2)
('revised', 1)
```

When executed, the code computes the frequency of each word in input.txt file and generates (word, frequency) pairs in the `part-xxxxx` file(s) in the `output` subdirectory.

Packaging your code

You can interactively develop and debug your code using the Spark shell, but once you finish development, the code needs to be "packaged" so that it can be submitted to and run on a cluster. Packaging your code requires providing additional information on where your code should be run and what will be its "name" so that your task can be easily referenced with the name. In Spark, this is done through "Context". In fact, when we run our code in the Spark shell, a default spark context is automatically created and made available via the variable `sc`. For example, when you executed the following line in the shell

```
lines = sc.textFile("input.txt")
```

`sc` was the variable that references the default context created by the shell.

In your packaged code, however, you have to explicitly create a Spark context for your task. In the provided wordCount.py code, for example, the first few lines create a Spark Context

```
from pyspark import SparkContext
sc = SparkContext("local", "WordCount")
```

which specifies “local” to be where the task should be submitted — in this project, both the Spark shell and your “packaged code” run the task on the same “local” machine, but in a real production environment, you are likely to develop your code in your container, but the packaged code will be submitted to a large production cluster — and names the task as “WordCount”.

Once you created a “packaged” code like wordCount.py, you can “submit and run” your code through the `spark-submit` command:

```
$ rm -rf output/
$ spark-submit wordCount.py
$ head -5 output/part-0000*
('WASHINGTON', 1)
('—', 6)
('President', 1)
('Trump', 3)
('on', 4)
```

The above sequence of commands will delete any output produced in `output/` directory, submit and run the code `wordCount.py`, and show the first 5 lines of the output from your code.

Part B: Computing High-Frequency Book Pairs

GoodReads Book-Review Dataset

Now that you got a basic understanding of Spark, download our book-review dataset into the `/home/cs143/data/` folder:

```
$ mkdir -p /home/cs143/data
$ cd ~/data/
$ wget http://oak.cs.ucla.edu/classes/cs143/project5/goodreads.user.books
```

This dataset was originally downloaded from UCSD Book Graph page and has been preproccessed to have the following format:

```
user1:book1,book2,...,bookn
```

Each line of the file indicates the list of books for which a user wrote reviews. For example, the first line of our dataset

```
$ head -5 goodreads.user.books
1:950,963
2:1072,1074,1210
3:1488
4:1313,1527,1557,1566,1616,1620
5:5316
```

`1:950,963` indicates that the user of user ID 1 wrote reviews for two books whose IDs are 950 and 963.

Writing your code

Now your job is to write a Spark Python code that **outputs the pairs of book IDs that appear frequently together in the users' review lists**. More specifically, the output from your program must consist of lines of the following format

```
((bookid_1, bookid_2), frequency)
```

which means that `bookid_1` and `bookid_2` appear together in the `frequency` number of users' review lists. For example, if your output contains the following line:

```
((536, 1387), 22)
```

it means that 22 users reviewed both books 536 and 1387. **You have to generate one output line per every book pair that appears in more than 20 users' review lists.**

In writing your code, you may find the list of Spark transformation functions helpful. Pay particular attention to map(), flatMap(), reduceByKey() and filter() functions.

Code requirements

Please make sure that your code meets the following requirements:

1. The filename of your code must be `bookPairs.py`
2. Your code should be executable simply by the command `spark-submit bookPairs.py`
3. Your code must take the file available at `/home/cs143/data/goodreads.user.books` as its input data
4. Your code must produce output in the directory `./output`
5. Each line of your output must contain a book pair that appears in more than 20 users' review lists.
6. Each output line must be formatted as `((bookid1, bookid2), count)`.
7. Your code must produce one output line per every unique book pair, ignoring their relative ordering. For example, if book pair (1, 2) appears 30 times in the review list, your code must produce one output, either ((1, 2), 30) or ((2, 1), 30), but not both.

8. The output ordering does not matter as long as your output contains the correct book pairs and their counts.
9. Your code must use only Python libraries/packages that are preinstalled and available in the docker container. Please do not install any other third-party library yourself.

Example results

Writing and debugging code on a large dataset is often time-consuming and difficult, so when you develop code, it is a good idea to work on a smaller dataset than your real dataset. This way, you can iterate over and improve your code more quickly. Also, your code will produce smaller outputs that is easier to investigate and verify.

To help you debug your code, here are a few example outputs on subsets of our dataset.

1. When your code is run on the first 1000 lines of our dataset, it should produce the following book pairs and counts:

Book Pairs	**Count**
(536, 1387)	22

2. When your code is run on the first 3000 lines of our dataset, it should produce the following book pairs and counts:

Book Pairs	**Count**
(613, 939)	21
(1000, 66)	27
(1000, 1116)	32
(1000, 1117)	33
(1116, 66)	28
(1116, 1117)	28
(1117, 66)	27
(1386, 536)	53
(1386, 1387)	58
(1387, 536)	57
(1471, 1473)	21
(1525, 1526)	34
(1604, 1605)	22
(12710, 1525)	22

Book Pairs	Count
(12710, 1526)	22

Note: You can take the first k lines of a file using the Unix `head` command. For example, the following command

```
$ head -1000 goodreads.user.books > goodreads.1000
```

takes the first 1000 lines of the file `goodreads.user.books` and save it into the file `goodreads.1000`.

What to Submit

For this project, you need to submit **a single zip file** named `project5.zip` that has the following packaging structure.

```
project5.zip
 +- bookPairs.py
 +- README.txt (optional)
```

Each file or directory is as follows:

1. `bookPairs.py`: this is the main Python code that you wrote to compute the frequent book pairs. This code should be executable simply by typing "`spark-submit bookPairs.py`".
2. `README.txt` includes any comments you find worth noting.

To help you package your submission zip file, we have made a packaging script p5_package, which can be run like the following:

```
$ ./p5_package
zip project5.zip bookPairs.py
  adding: bookPairs.py (deflated 47%)
[SUCCESS] Created '/home/cs143/project5/project5.zip'
```

(You may need to use "chmod +x p5_package" if there is a permission error.)

When executed, our packaging script will collect all necessary (and optional) files located in the same directory as the script and create the `project5.zip` file according to our specification that can be submitted to CCLE.

Testing Your Zip File

To ensure the correct packaging of your submission, we have made a grading script p5_test for Project 5, which can be executed like:

```
$ ./p5_test project5.zip
```

(You may need to use "chmod +x p5_test" if there is a permission error.)

You **MUST** test your submission using the script to minimize the chance of an unexpected error during grading. When everything runs properly, you will see an output similar to the following from the script:

```
$ ./p5_test project5.zip
Executing your Spark code.....
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/loc
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective a
WARNING: All illegal access operations will be denied in a future release
21/02/19 16:47:17 WARN NativeCodeLoader: Unable to load native-hadoop library for you
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/02/19 16:47:18 INFO SparkContext: Running Spark version 3.0.1
21/02/19 16:47:18 INFO ResourceUtils: ================================================
21/02/19 16:47:18 INFO ResourceUtils: Resources for spark.driver:

...

21/02/19 16:48:51 INFO ShutdownHookManager: Shutdown hook called
21/02/19 16:48:51 INFO ShutdownHookManager: Deleting directory /tmp/spark-5dcbfcc0-ce
21/02/19 16:48:51 INFO ShutdownHookManager: Deleting directory /tmp/spark-034d45e3-c8
21/02/19 16:48:51 INFO ShutdownHookManager: Deleting directory /tmp/spark-034d45e3-c8
((1525, 12710), 99)
((1386, 1402), 59)
((6410, 12698), 29)
((995, 7503), 21)
((1200, 1402), 25)

SUCCESS! We finished testing your zip file integrity.
```

The test script will run your `bookPairs.py` code and display the first 5 lines in the output from your code on the screen. Make sure that the produced output matches what you expect from your code.

Submitting Your Zip File

Visit GradeScope to submit your zip file electronically by the deadline. In order to accommodate the last minute snafu during submission, you will have 1-hour window after the deadline to finish your submission process. That is, as long as you start your submission before the deadline and complete within 1 hour after the deadline, you are OK.