

CS4121

Home Written HW Programming Homework 1: SQL Programming Homework 2: A Tour of
Apache Spark Programming Homework 3: Tensorflow

Programming Homework 2: A Tour of Apache Spark

Programming Homework 2: A Tour of Apache Spark

Gain a hands-on understanding of Google Cloud Dataproc, Apache Spark, Spark SQL, and Spark Streaming over HDFS.

Due: **April 30th, 2020, 11:59 AM**

Overview

First introduced in [2010](#), Apache Spark is one of the most popular modern cluster computing frameworks.

By leveraging its innovative distributed memory abstraction – Resilient Distributed Datasets (RDDs) – Apache Spark provides an effective solution to the I/O inefficiency of MapReduce, while retaining its scalability and fault tolerance.

In this assignment, you are going to deploy Spark and HDFS, write a Spark program generating a web graph from the entire Wikipedia, and write a PageRank program to analyze the web graph.

You will run those applications using Spark over HDFS in Google Cloud Dataproc.

Finally, you will write a stream emitter using Spark Streaming and a parser using Spark Structured Streaming to play with the recent trend of working with streaming data.

As a well-maintained open-source framework, Apache Spark has well-written official documents; you will find a lot of useful information by simply reading the official tutorials and documents.

When you encounter an issue regarding cluster deployment or write Spark programs, you are encouraged to utilize online resources before posting questions on Piazza.

Learning Outcomes

After completing this programming assignment, students should be able to:

- Deploy and configure Apache Spark, HDFS in Google Cloud Dataproc.
- Write Spark applications using Python with Jupyter Notebook and launch them in the cluster.
- Describe how Apache Spark, Spark SQL, Spark Streaming, HDFS work, and interact with each other.

Environment Setup

You will complete your assignment in Google Cloud Dataproc, which is a managed cloud service for Spark and Hadoop, which allows users to spin up Spark and Hadoop instances quickly.

First, make sure you have followed the [instructions](#) provided by CRF to redeem your credits in Google Cloud.

Then, use the following [link](#), and click on `Enable API` to enable Google Cloud Dataproc. **Note:** you could pin Dataproc on your navigation menu for quick access.

Click on `Create cluster` to create your first Dataproc Cluster. Notice there are three Cluster modes you could choose from: Single Node, Standard, and High Availability. We will start with a Single Node cluster.

Name ?

cluster-e463

Region ?

us-central1

Zone ?

us-central1-a

Cluster mode ?

- Single Node (1 master, 0 workers)
- Standard (1 master, N workers)
- High Availability (3 masters, N workers)

Machine configuration ?

Machine family

General-purpose

Machine types for common workloads, optimized for cost and flexibility

Series


N1

Powered by Intel Skylake CPU platform or one of its predecessors

Machine type

machine type

n1-standard-4 (4 vCPU, 15 GB memory)

 vCPU Memory

4 15 GB

⌵ CPU platform and GPU

Primary disk size (minimum 15 GB) ? Primary disk type ?

500 GB Standard persistent disk

YARN cores ? YARN memory ?

8 24 GB

Autoscaling policy ? (Optional)

☐ Enable autoscaling on the cluster.
This project does not currently have any applicable policy to enable autoscaling in this region. [Learn how to create autoscaling policy.](#)

Component gateway

☐ Enable access to the web interfaces of default and selected optional components on the cluster. [Learn more](#)

⌵ Advanced options

Create Cancel

Equivalent REST or command line

Jupyter Notebook

You will use Python and Jupyter Notebook for this project. If you are unfamiliar with Jupyter Notebook, [this](#) tutorial might be helpful.

Google provides an easy interface to install and access Jupyter notebook when creating Dataproc cluster. Go down to **Component gateway** and select **Enable access to the web interfaces of default and selected optional components on the cluster**. Then click on

Advanced options

Component gateway

☒ Enable access to the web interfaces of default and selected optional components on the cluster. [Learn more](#)

⌵ Advanced options

Go down to **Optional components** and click on . In the pop-up window, choose Anaconda and Jupyter Notebook. Make sure you click on to save your changes.

Optional components

Select one or multiple components. [Learn more](#)

- ☒ **Anaconda**
Anaconda is a Python distribution and Package Manager with over 1000 popular data science packages. Anaconda becomes the default Python interpreter.
- ☐ **Hive WebHCat**
The Hive WebHCat server provides a REST API for HCatalog. The REST service is available on port 50111 on the cluster's first master node..
- ☒ **Jupyter Notebook**
Jupyter, a Web-based notebook for interactive data analytics. The Jupyter Web UI is available on port 8123 on the cluster's first master node. Python and PySpark kernels are available.
- ☐ **Zeppelin Notebook**
Zeppelin Notebook is a Web-based notebook for interactive data analytics. The Zeppelin Web UI is available on port 8080 on the cluster's first master node.
- ☐ **Druid**
The Apache Druid component is an open source distributed OLAP data store. The Druid component installs Druid services on the Cloud Dataproc cluster master(Coordinator, Broker, and Overlord) and worker (Historical, Realtime and MiddleManager)nodes.
- ☐ **Presto**
The Presto component is an open source distributed SQL query engine. The Presto server and Web UI are available on port 8060 (or port 7778 if Kerberos is enabled) on the cluster's first master node.
- ☐ **ZooKeeper**
The Apache ZooKeeper component is a centralized service for providing distributed synchronization of data.

You could refer to [this](#) doc for more details.

Cluster properties

Components like Spark and Hadoop have many configurations that users could tune. You could change the default values of those settings when creating the Dataproc cluster. Refer to [this](#) doc for a more detailed explanation. For example, if you want to change the default block size of HDFS to 64MB, you could change it by adding a cluster property under **Cluster properties** in the Advanced Options section.

Cluster properties (Optional) ?

Use cluster properties to add or modify configuration files when creating a cluster.

[Learn more](#)

hdfs	dfs.blocksize	67108864	X
+ Add cluster property			

Part 1: Spark and Spark SQL

In this part of the assignment, you will perform 3 tasks

- Get yourself familiarized with the interface and ingesting a Wikipedia database into HDFS
- Parse the Wikipedia database to generate a webgraph of the internal links
- Use the generated graph as an input to a Spark PageRank program to generate the ranks of the internal links.

Task 1: Getting Started (10 Points)

Once you have created the cluster, you have full control over the VM, you could download any package you would need. Click on the cluster name to go to **Cluster details** page. Go to the **VM Instances** tab and click on **SSH**, run the following command to download all files to local HDFS. The files might take a while to transfer. In the meantime, [here](#) is a tutorial for HDFS commands which might be helpful.

```
hdfs dfs -cp gs://assignment2_w4121/enwiki_small.xml /  
hdfs dfs -cp gs://assignment2_w4121/enwiki_test.xml /  
hdfs dfs -cp gs://assignment2_w4121/enwiki_whole.xml /
```

Also, you would need an external package in order to parse XML files. You could run the following command to download it to your VM. You might need to run it on **all** of your VMs if you have multiple machines in your cluster.

```
sudo hdfs dfs -get gs://assignment2_w4121/spark-xml_2.11-0.7.0.jar /usr/lib/spark
```

Now you could go to the **Web Interfaces** tab and use the `Jupyter` link to open Jupyter Notebook Interface.

In this task, we provide you a big Wikipedia database in XML format. It can be found at `/enwiki_whole.xml` in your HDFS.

This input file is very big and you have to use a distributed file system like HDFS to handle it. We have also provided a smaller file `/enwiki_small.xml` for debugging purposes.

The XML files are structured as follow:

```
<mediawiki>  
  <siteinfo>  
    ...
```

```

</siteinfo>
<page>
  <title>Title A</title>
  <revision>
    <text>Some text</text>
  </revision>
</page>
<page>
  ...
</page>
...
</mediawiki>

```

To get a sense of how a Wiki page transfers to a xml file, take a look at the following examples:

- [Apple \(orig\)](#)
- [Apple \(xml\)](#)

Note that the database we provide you is not up-to-date, but they should be pretty close to what you will find in the above examples.

Submit a job

Once you are done with the debugging process on the Jupyter Notebook you could download it as a python file and submit it as a job to run on the cluster.

The screenshot displays the Google Cloud Dataproc console. On the left, a sidebar contains navigation links for Clusters, Jobs, and Workflows. The main area is titled 'Cluster details' for 'cluster-3034'. At the top of this area, there are buttons for 'SUBMIT JOB' (highlighted with a red box), 'REFRESH', 'DELETE', and 'VIEW LOGS'. Below these buttons, a yellow warning icon indicates a recommendation for PD-Standard disks. A set of tabs allows switching between Monitoring, Jobs, VM Instances, Configuration, and Web Interfaces. Under the 'Web Interfaces' tab, there is an 'SSH tunnel' section with a link to create an SSH tunnel. Below that, a 'Component gateway' section lists various services with external links: YARN ResourceManager, HDFS NameNode, MapReduce Job History, YARN Application Timeline, Spark History Server, Tez, Jupyter, and JupyterLab. At the bottom, there is a link for 'Equivalent REST'.

The following is an example job.

← Submit a job

Job ID**Region** ?**Cluster****Job type****Main python file** ?**Additional python files** (Optional)**Arguments** (Optional) ?**Jar files** (Optional) ?**Properties** (Optional) ?

✕

✕

✕

✕

Labels (Optional) ?

Notice you could specify python file to run from the Google Cloud Storage. Also, when you need to read the XML files, please make sure you have included the jar file. Besides, you could specify the number of cores and memory for driver and executor here.

You could also refer to [this](#) doc to learn more about submitting a job.

Question 1. (4 points) What is the default block size on HDFS? What is the default replication factor of HDFS on Dataproc?

Write a spark program to read in the `/enwiki_small.xml` file as a Dataframe and use `printSchema()` function to print its schema, you could start from something like following. Copy the outputted schema to a **separate txt file** named **schema.txt**. (6 points)

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
df = spark.read.format('xml').options(rowTag='page').load('hdfs:/enwiki_small.xml')
```

Best Practices

- Delete the cluster after you are done for a coding session, do not leave them on overnight. Otherwise you are going burn through credits really quickly.
- Start from small. Start with small files and use one node setup to debug your code. Then try to run your code on bigger files and three-node setup.
- It is recommended to use Jupyter Notebook to debug your code. But do make sure you have shutdown all of your Jupyter notebooks before you submit your job using the web interface.
- Notice it is possible to use Google Cloud Storage

Task 2: Webgraph on Internal Links (50 Points)

You are going to write a Spark program which takes the xml file you just ingested as input (`enwiki_whole.xml`), and generate a csv file which describes the webgraph of the internal links in Wikipedia. The csv file should look like the following:

```
article1      article0
article1      article2
article1      article3
article2      article3
article2      article0
...
```

It is hard and tedious to find every internal link on a page. We have made the following assumptions to simplify the string parsing for you. For each page element, the article on the left column corresponds to the string between `<title>` and `</title>`, while the article on the right column are those surrounded by a pair of double brackets `[[]]` in the `<text>` field in the xml file with the following requirements:

1. All the letters should be convert to lower case.
2. If the internal link contains a `:`, you should ignore the entire link unless it starts with `Category:`.
3. Ignore links that contain a `#`.
4. If multiple links appear in the brackets, take the first one; e.g., take `A` in `[[A|B]]`.

Those assumptions help you filter out some unnecessary links. Note if the remaining string becomes empty after the filtering, you should also ignore them. When we say ignoring a link, we mean it will not show up in the output file of this task.

The two columns in the output file should be separated by a `Tab`. You may assume there are no other `Tab`s in the article name.

Note: It is recommended to use UDF + regular expression to extract links from the documents. Also, try to use the built in functions of spark to sort your results.

You should use the default configuration of Spark, HDFS, unless we specify a different one.

Set the Spark driver memory to 1GB and the Spark executor memory to 5GB to answer Question 2-7.

For the following questions you will need to use the xmls as the input file, and output columns into a CSV file. Separate the columns with a `Tab`.

Question 2. (2 points) Use `enwiki_test.xml` as input and run the program locally on a Single Node cluster using 4 cores. What is the completion time of the task?

Question 3. (2 points) Use `enwiki_test.xml` as input and run the program under HDFS inside a 3 node cluster (2 worker nodes). Is the performance getting better or worse in terms of completion time? Briefly explain.

Question 4. (2 points) For this question, change the default block size in HDFS to be 64MB and repeat Question 3. Record run time, is the performance getting better or worse in terms of completion time? Briefly explain.

Set the Spark driver memory to 5GB and the Spark executor memory to 5GB to answer Question 5-7. Use this configuration across the entire assignment whenever you generate a web graph from `enwiki_whole.xml`.

Question 5. (2 points) Use `enwiki_whole.xml` as input and run the program under HDFS inside the Spark cluster you deployed. Record the completion time. Now, kill one of the worker nodes immediately. You could kill one of the worker nodes by go to the **VM Instances** tab on the Cluster details page and click on the name of one of the workers. Then click on the STOP

button. Record the completion time. Does the job still finish? Do you observe any difference in the completion time? Briefly explain your observations.



Question 6. (2 points) Only for this question, change the replication factor of `enwiki_whole.xml` to 1 and repeat Question 5 without killing one of the worker nodes. Do you observe any difference in the completion time? Briefly explain.

Question 7. (2 points) Only for this question, change the default block size in HDFS to be 64MB and repeat Question 5 without killing one of the worker nodes. Record run time, is the performance getting better or worse in terms of completion time? Briefly explain.

To submit the assignment you will need to use `enwiki_small.xml` as the input file, and sort both output columns in ascending order and save the first 5 rows into a CSV file and name it `p1t2.csv`. Separate the columns with a `Tab`. (38 points for code + output)

Task 3: Spark PageRank (40 Ppints)

In this task, you are going to implement the PageRank algorithm, which Google uses to rank the website in the Google Search. We will use it to calculate the rank of the articles in Wikipedia. The algorithm can be summarized as follows:

1. Each article has an initial rank of 1.
2. On each iteration, the contribution of an article A to its neighbor B is calculated as `its rank / # of neighbors`.
3. Update the rank of the article B to be `0.15 + 0.85 * contribution`
4. Go to the next iteration.

The output should be a `csv` file containing two columns. The first column is the article and the other column describes its rank. Separate the columns with a `Tab`.

Set the Spark driver memory to 5GB and the Spark executor memory to 5GB whenever you run your PageRank program. Write a script to first run Task 2, and then run Task 3 using the csv output generated by Task 2, and answer the following questions. Always use 10 iterations for the PageRank program. When running Task 2, use `enwiki_whole.xml` as input.

Question 8. (2 points) Use your output from Task2 with `enwiki_whole.xml` as input, run Task 3 using a 3 node cluster (2 worker nodes). What is the completion time of the task?

To submit the assignment you will need to use `enwiki_small.xml` as the input file for Task2 and then run your code for Task3, and sort both output columns for Task3 in ascending order

and save the first 5 rows into a CSV file and name it p1t3.csv. Separate the columns with a `Tab`.
(38 points for code + output)

Part 2: Spark Streaming (20 Extra Credits)

In the second part of the assignment, you will learn Spark Streaming as well as Spark Structured Streaming, and write a program for each of them.

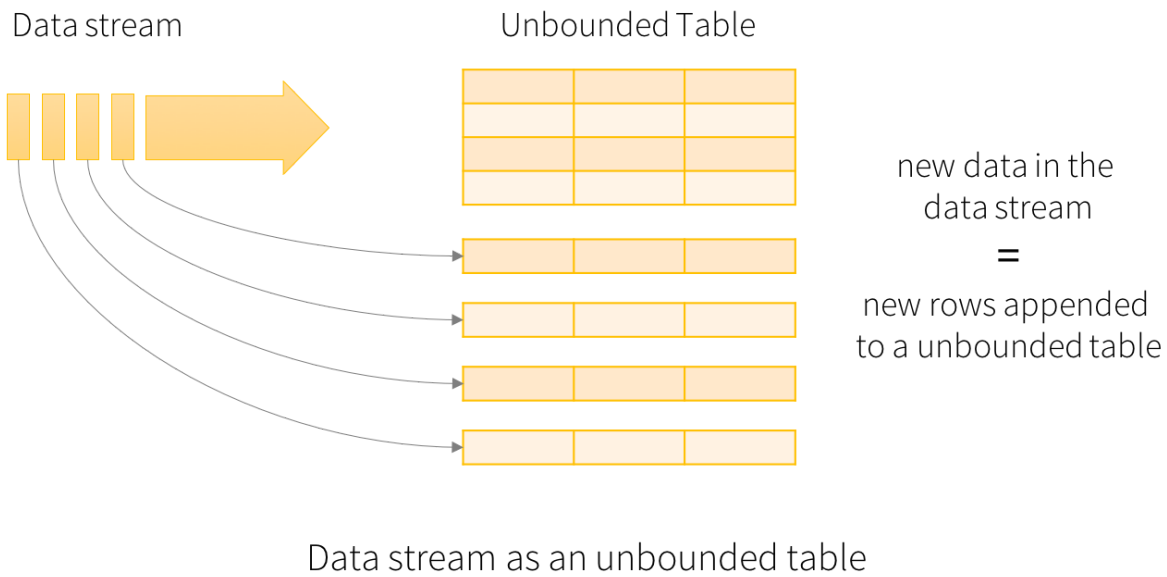
Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.



Internally, Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches. More information about Spark Streaming can be found [here](#).



Spark Structured Streaming is a new high-level API introduced after Apache Spark 2.0 to support continuous applications. It provides fast, scalable, fault-tolerant, and end-to-end exactly-once stream processing without the user having to reason about streaming. The key idea here is to treat a live data stream as a table that is being continuously appended. You are encouraged to read its [programming guide](#).



Task 1: Stream Receiver (10 points)

In this task, you are going to write a stream receiver using **Spark Structured Streaming** to read file source while the file is being generated. The input source should be `csv` files. Your receiver should keep a list of articles whose rank is greater than **0.5** and store the file inside the HDFS. The list should be updated dynamically while the Pagerank program is dumping the output and should be saved inside the HDFS.

The output file should be `csv` whose the format should look like the following. Separate the columns with a `Tab`. To submit, only submit your Jupyter notebook files. (8 points for code)

Note: It might be helpful to create your own csv to test or debug the code. If your stream receiver is running then it should generate output.

```
article0    rank0
article1    rank1
article2    rank2
...
```

Question 9 (2 points) Start a PageRank program you wrote in Part 1 Task 2 whose input is the link graph generated from "enwiki_whole.xml" and store the output to a directory inside HDFS. Set your stream receiver to read the files generated by the PageRank program. Kill the receiver when the PageRank task is finished. How many articles in the database has a rank greater than **0.5**? You can SSH into the master node and start with 'hdfs dfs -mv' or 'mv' to help you write your program.

Task 2: Stream Emitter (10 points)

In this task, you are going to write a stream emitter using **Spark Streaming** to emit the 'txt' file output generated by the PageRank program to a directory to emulate the case you see in the previous task. Run your stream receiver to catch the stream you are emitting and check if you get the same result as in Question 9.

To submit, include your Jupyter notebook files. (6 points for code)

Question 10 (2 points) Spark Streaming can also be used to send data via TCP sockets. The Emitter in this case will wait on a socket connection request from the receiver, and upon accepting the connection request it will start sending data. Do you think such data server design is feasible and efficient? Briefly explain.

****Questions 11 **** (2 points) How many hours did you spend in this assignment?

Submission Instructions

File name

Each group should submit one zip file to Gradescope. Only one teammember need to submit the file, make sure the team members are assigned to the submission.

Name the file as `<group_num>_assignment2.zip`. For example, `g1_assignment2.zip`.

Content - Code

4 Spark projects from Part 1 Task 2-3 and Part 2 Task 1-2 should goes under folder names `p1t2`, `p1t3`, `p2t1`, `p2t2` accordingly.

Inside each folder, in addition to Jupyter notebook and python files, there should be an additional file named `config` which describes configurations or addition step you did to run the three tasks mentioned below.

You also need to provide a `config` file which describes configurations or addition step you did to run the ****following 3 tasks on a single Spark node ****

1. Use the program in Part 1 Task 2 to take "enwiki_small.xml" as input to generate the graph.
2. Use the program in Part 1 Task 3 to take the graph you just generated and output a rank list of the articles in the dataset.
3. (Optional) Use the stream emitter you wrote in Part 2 Task 2 to emit the rank list output in the previous step to a local directory while using the stream receiver you wrote in Part 2

Task 1 to dynamically read the files and generate the output mentioned in Part 2 Task 1.

Try to be clear about the instructions to run these steps. The purpose of doing this is to check if your program does what we mentioned in the spec. Do not worry whether your program has the lowest completion time.

The structure should be something like

```
g1.zip
├─ p1t1
│   └─ schema.txt
├─ p1t2
│   ├── config
│   ├── p1t2.py
│   ├── p1t2.csv
│   └─ p1t2.ipynb
├─ p1t3
├─ p2t1
├─ p2t2
└─ README (see below)
```

Content - README

Each submission should also include a README to record the answers to the 11 (last 3 optional) questions mentioned in this assignment.

Acknowledgements

This assignment is based on Peifeng Yu and Mosharaf Chowdhury's UMICH EECS598: Advanced Topics on Systems for X.