Big Data Wrangling with Google Books Ngrams

Detailed steps to steup and work with AWS EMR clusters, S3 bucket and Spark notebook

Prepared by - Kulwinder Kaur

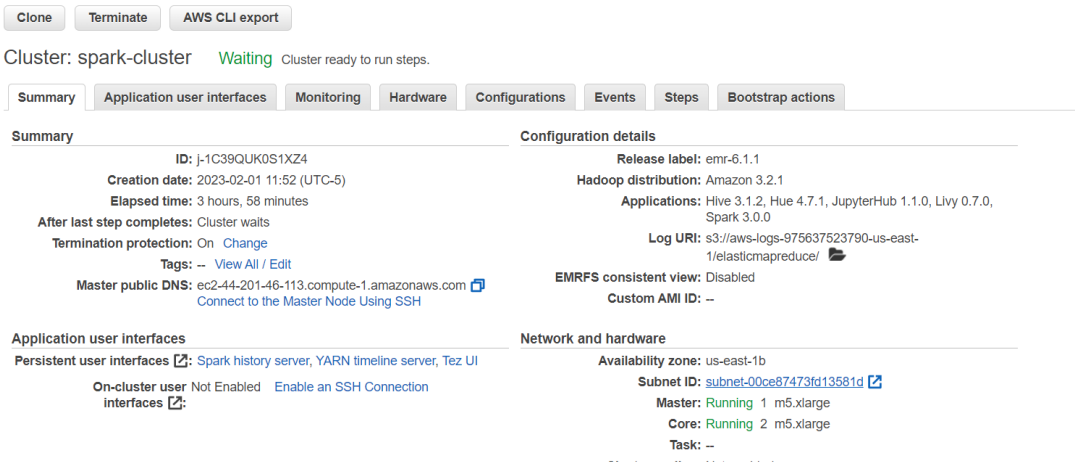
The scope of this data processing and analysis report is to document the workflow involved in filtering and reducing big data of Google Ngrams down to a manageable size, and then do some analysis locally on our machine after extracting data.

The Google Ngrams dataset was created by Google's research team by analyzing all of the content in Google Books - these digitized texts represent approximately 4% of all books ever printed, and span a time period from the 1800s into the 2000s. The Google Ngram Viewer or Google Books Ngram Viewer is an online search engine that charts the frequencies of any set of search strings using a yearly count of n-grams found in printed sources published between 1500 and 2019.

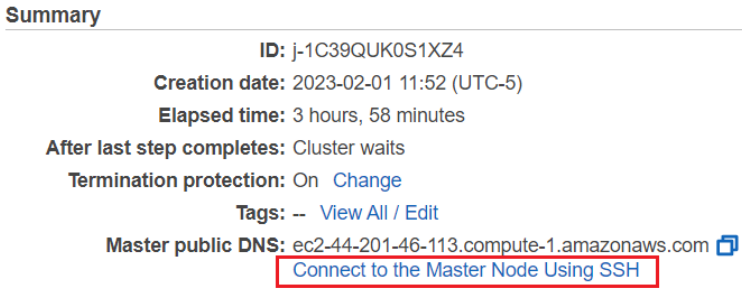
The dataset is hosted in a public S3 bucket as part of the Amazon S3 Open Data Registry. For this assignment, it has been converted to CSV and hosted on a public S3 bucket which may be accessed here: s3://brainstation-dsft/eng_1M_1gram.csv

Creating a new cluster and setting up the head node.

1. Created a new cluster after selecting release version emr-6.1.1 and selecting checkboxes for Hadoop, JupyterHub, Hive, Hue, Livy, and Spark.



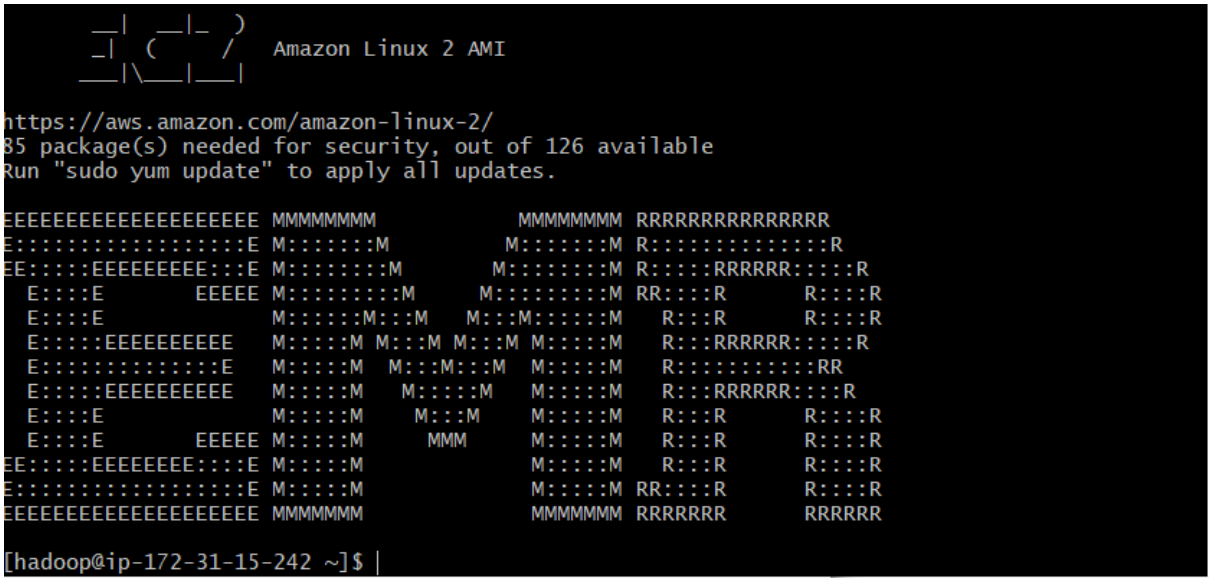
2. Connect to the head node of the cluster using SSH. Go to the summary section of the summary page of the cluster and select ‘Connect to the Master Node Using SSH’.



A new window will open with the SSH command. Copy-paste the command onto your GitBash prompt and remove ' ~/ ' from in front of the pem key.

```
ssh -i ~/kk_aws.pem hadoop@ec2-34-205-135-115.compute-1.amazonaws.com
```

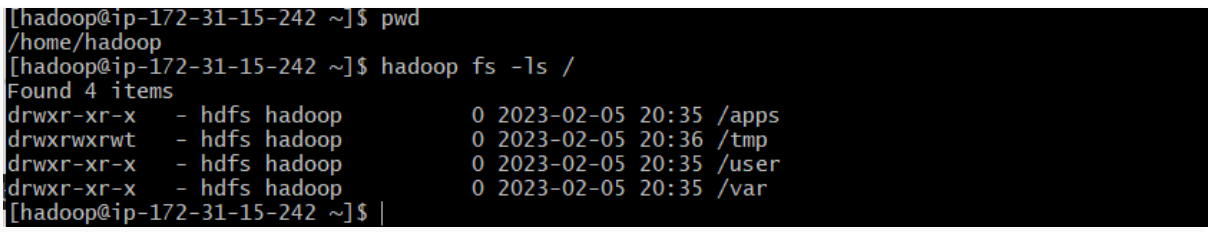
After this, you will be accessing the Hadoop cluster head node.



Copying the data directly from s3 bucket on the Hadoop File System (HDFS) named /user/hadoop/eng_1M_1gram.

3. For this we can use the Hadoop distcp command with source and destination. The public S3 bucket name is s3://brainstation-dsft/eng_1M_1gram.csv which is the source and we can save it to user/Hadoop/ folder

    - Checking the current directory and Hadoop directories.



- Copying directly from s3 bucket using distcp command

```
[hadoop@ip-172-31-15-242 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hado
op/eng_1M_1gram
```

- Confirming that we have the file in our Hadoop folder.

```
[hadoop@ip-172-31-15-242 ~]$ hadoop fs -ls /user/hadoop/
Found 1 items
-rw-r--r--   1 hadoop hadoop 5292105197 2023-02-05 20:49 /user/hadoop/eng_1M_1gram
[hadoop@ip-172-31-15-242 ~]$
```

Reading data from HDF using PySpark (A spark notebook is used)

4. We will create a PySpark notebook from AWS EMR console:
    a. Go to Notebook options> Create Notebook >Fill Notebook name > Leave Default settings > Create Notebook.
    b. Once the Notebook is created, open the notebook in Jupyter.
    c. Open the notebook from terminal and select the kernel PySpark. Our spark notebook is ready and we will be performing some tasks there.

After reading the data from HDFS in spark notebook and saving the filtered content back to HDFS file system we can verify using below command if the files got saved or not.

```
[hadoop@ip-172-31-15-242 ~]$ hadoop fs -ls /user/hadoop/spark_output
Found 3 items
-rw-r--r--   1 livy hadoop          0 2023-02-05 21:32 /user/hadoop/spark_output/_SUCCESS
-rw-r--r--   1 livy hadoop         33 2023-02-05 21:32 /user/hadoop/spark_output/part-00000-d
eb4b8fe-49eb-4fe0-a2fa-26527b3b8cea-c000.csv
-rw-r--r--   1 livy hadoop       7305 2023-02-05 21:32 /user/hadoop/spark_output/part-00023-d
eb4b8fe-49eb-4fe0-a2fa-26527b3b8cea-c000.csv
[hadoop@ip-172-31-15-242 ~]$
```

There are two csv files in this directory. We can check the contents of CSV files using -cat command.

```
[hadoop@ip-172-31-15-242 ~]$
[hadoop@ip-172-31-15-242 ~]$ hadoop fs -cat /user/hadoop/spark_output/part-00000-deb4b8fe-49e
b-4fe0-a2fa-26527b3b8cea-c000.csv
token,year,frequency,pages,books
[hadoop@ip-172-31-15-242 ~]$ hadoop fs -cat /user/hadoop/spark_output/part-00023-deb4b8fe-49e
b-4fe0-a2fa-26527b3b8cea-c000.csv
token,year,frequency,pages,books
data,1584,16,14,1
data,1614,3,2,1
data,1627,1,1,1
data,1631,22,18,1
data,1637,1,1,1
data,1638,2,2,1
data,1640,1,1,1
data,1642,1,1,1
data,1644,4,4,1
data,1647,1,1,1
data,1651,1,1,1
data,1674,1,1,1
data,1690,1,1,1
data,1693,1,1,1
```

Collect the contents of the directory into a single file on the local drive of the head node Saving files from HDFS to the local drive.

5. Collecting the contents of the directory into a single file on the local drive of the head node using getmerge and moving this file into a S3 bucket.
   - We merged the files and saved it to home.

```
[hadoop@ip-172-31-15-242 ~]$ hadoop fs -getmerge /user/hadoop/spark_output/part-00023-deb4b8f
e-49eb-4fe0-a2fa-26527b3b8cea-c000.csv /user/hadoop/spark_output/part-00000-deb4b8fe-49eb-4fe
0-a2fa-26527b3b8cea-c000.csv /home/hadoop/merged.csv
[hadoop@ip-172-31-15-242 ~]$ ls /home/hadoop/
merged.csv
[hadoop@ip-172-31-15-242 ~]$
```

   - Move the file from the head node to S3 bucket.
     Using aws s3 command we can move the file from head node to s3 bucket.

```
[hadoop@ip-172-31-15-242 ~]$ aws s3 cp /home/hadoop/merged_file.csv s3://kk-bstn-bucket-jan30
upload: ./merged_file.csv to s3://kk-bstn-bucket-jan30/merged_file.csv
[hadoop@ip-172-31-15-242 ~]$
```

Analysing a subset of Google Ngram data on local machine

6. Created a jupyter notebook.

7. Compare Hadoop and Spark as distributed file systems.

Hadoop and Spark are both open source software utility to manage big data sets. Like Hadoop, Spark splits up large tasks across different nodes. However, it tends to perform faster than Hadoop and it uses random access memory (RAM) to cache and process data instead of a file system.

Hadoop Distributed File System (HDFS): This stores files in a Hadoop-native format and parallelizes them across a cluster. It manages the storage of large sets of data across a Hadoop Cluster. Hadoop can handle both structured and unstructured data. **YARN** is Yet Another Resource Negotiator coordinates application runtimes and resource allocation. It uses **Map reduce** algorithm to process data. It has a **Hadoop Core/Hadoop common** that provides support to all other components it has a set of common libraries and utilities that all other modules depend on.

Spark file system: Spark is structured around **Spark Core**, the engine that drives the scheduling, optimizations, and RDD abstraction, as well as connects Spark to the correct filesystem (HDFS, S3, RDBMS, or Elasticsearch). There are several libraries that operate on top of Spark Core, including Spark SQL, which allows you to run SQL-like commands on distributed data sets, MLLib for machine learning, GraphX for graph problems, and streaming which allows for the input of continually streaming log data.

What are the advantages/ differences between Hadoop and Spark? List two advantages for each.

- Spark is faster in performance than Hadoop as it uses RAM.
- Hadoop is best for batch processing as it uses MapReduce to split a large dataset across clusters for parallel analysis while Spark is suitable for iterative and live-stream data analysis and works with RDDs (Resilient distributed systems) to run operations.
- Hadoop is easily scalable by adding nodes and disks for storage. Supports tens of thousands of nodes without a known limit. However, Spark relies on RAM for computation therefore it is challenging to scale.
- Spark is faster with in-memory processing. Uses MLlib for computations. Hadoop on the other hand is slower than Spark.

Explain how the HDFS stores the data.

The files are split into blocks, typically 64 MB or 128 MB. The blocks are split across many machines at load time. Blocks are replicated across multiple machines. NameNode keeps track of which blocks make up the file and where are they stored. The data is replicated 3-fold across the blocks by default to provide fault tolerance.