Programming Assignment 3: Estimating the Importance of Wikipedia Articles using the PageRank Algorithm

By: Joseph Dorris and Drew Masters

Introduction

This project is attempting to find the PageRanks of the different wikipedia articles on simple wikipedia using Spark and GraphX.

Data

We chose to use simplewiki-20160305-pages-articles.xml for processing and simplewiki-20160305-all-titles for linking the articles to each other by assigning the article title an index based on its line number.

Experiment

Step 1: Preprocessing

The first step for estimating the PageRank was to preprocess the xml page to make it easy to process. We decided to use the GraphX pageRank function so we could focus on the results and not have to worry about any bugs that might be introduced by our implementation. The GraphX pageRank implementation requires a certain format, an adjacency list, so that it can form the graph. This format can be read in using an edge list file of the format “v1 v2\n” for each directed edge. For this, a python program was written to read in the xml file with the articles as well as a list of the titles of the articles. It would find the title in the xml file, find its associated index in the titles file, and find the associated index in the titles file for whatever links were in the text field. It would then write all of these out to a file called “adlist.txt” creating a list of edges. It was found to have 446,479 edges and 41,197 vertices. This is not all of 390,505 page titles provided by simplewiki-20160305-all-titles and the loss is due to redirection and other factors.

Step 2: Processing

A: set up Azure Spark Cluster

A HDInsight cluster using Spark was created with 5 compute nodes (Figures 1 and 2).

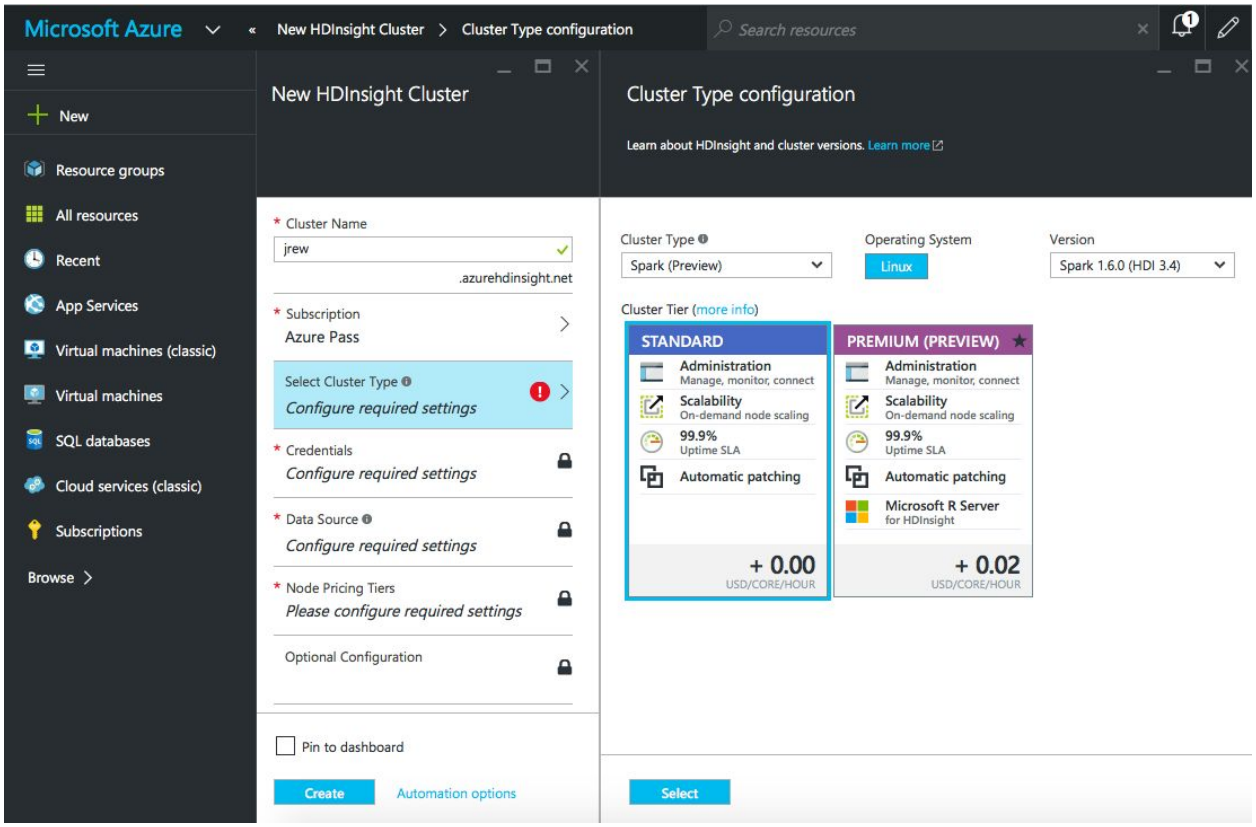

Figure 1

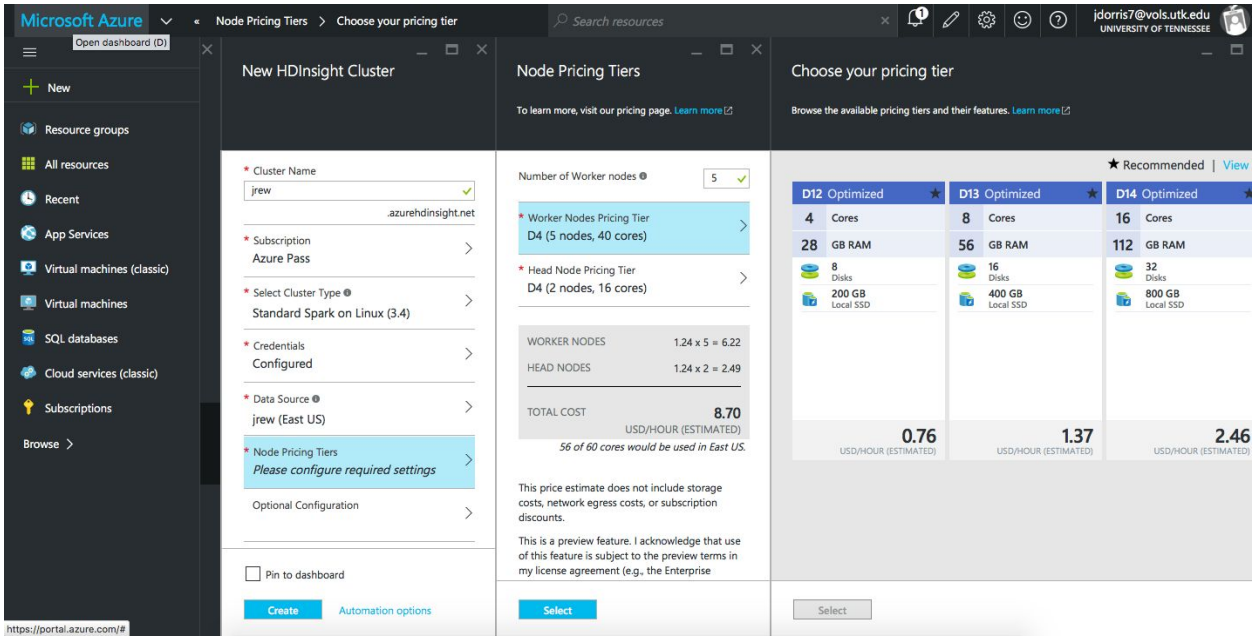

Figure 2

B. GraphX processing

The data was loaded onto the the hadoop file system using the following two commands:

hadoop fs -copyFromLocal adlist.txt ///example

hadoop fs -copyFromLocal titles.txt ///example

A scala script was written to load in the adjacency list and call pagerank for a specified number of iterations. It then joined the pagerank outputs to the original page titles and printed the output. This script is called “process.scala.” This was then run using the spark-shell (Figure 3).

admin> spark-shell -i process.scala

The progress could be seen in the stdout and the usage of the nodes could be seen in the dashboard on Azure (Figure 4). The code ran in under 5 minutes.

```
admin_jd@hn0-jrew:~$ spark-shell
16/04/22 19:34:17 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
16/04/22 19:34:17 INFO SecurityManager: Changing view acls to: admin_jd
16/04/22 19:34:17 INFO SecurityManager: Changing modify acls to: admin_jd
16/04/22 19:34:17 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view pe
rmissions: Set(admin_jd); users with modify permissions: Set(admin_jd)
16/04/22 19:34:18 INFO HttpServer: Starting HTTP Server
16/04/22 19:34:18 INFO Server: jetty-8.y.z-SNAPSHOT
16/04/22 19:34:18 INFO AbstractConnector: Started SocketConnector@0.0.0.0:54736
16/04/22 19:34:18 INFO Utils: Successfully started service 'HTTP class server' on port 54736.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (OpenJDK 64-Bit Server VM, Java 1.7.0_95)
Type in expressions to have them evaluated.
Type :help for more information.
16/04/22 19:34:23 INFO SparkContext: Running Spark version 1.6.0
16/04/22 19:34:23 INFO SecurityManager: Changing view acls to: admin_jd
16/04/22 19:34:23 INFO SecurityManager: Changing modify acls to: admin_jd
16/04/22 19:34:23 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view pe
rmissions: Set(admin_jd); users with modify permissions: Set(admin_jd)
16/04/22 19:34:24 INFO Utils: Successfully started service 'sparkDriver' on port 55139.
16/04/22 19:34:24 INFO Slf4jLogger: Slf4jLogger started
16/04/22 19:34:24 INFO Remoting: Starting remoting
16/04/22 19:34:24 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriverActorSystem@10.0.0.1
9:36520]
16/04/22 19:34:24 INFO Utils: Successfully started service 'sparkDriverActorSystem' on port 36520.
16/04/22 19:34:24 INFO SparkEnv: Registering MapOutputTracker
16/04/22 19:34:24 INFO SparkEnv: Registering BlockManagerMaster
16/04/22 19:34:24 INFO DiskBlockManager: Created local directory at /var/tmp/spark/blockmgr-b0239955-eb65-472f-b5d2-76
56a3db44fa
16/04/22 19:34:24 INFO MemoryStore: MemoryStore started with capacity 511.5 MB
16/04/22 19:34:25 INFO SparkEnv: Registering OutputCommitCoordinator
16/04/22 19:34:25 INFO Server: jetty-8.y.z-SNAPSHOT
16/04/22 19:34:25 WARN AbstractLifeCycle: FAILED SelectChannelConnector@0.0.0.0:4040: java.net.BindException: Address
already in use
java.net.BindException: Address already in use
```

Figure 3: spark-shell

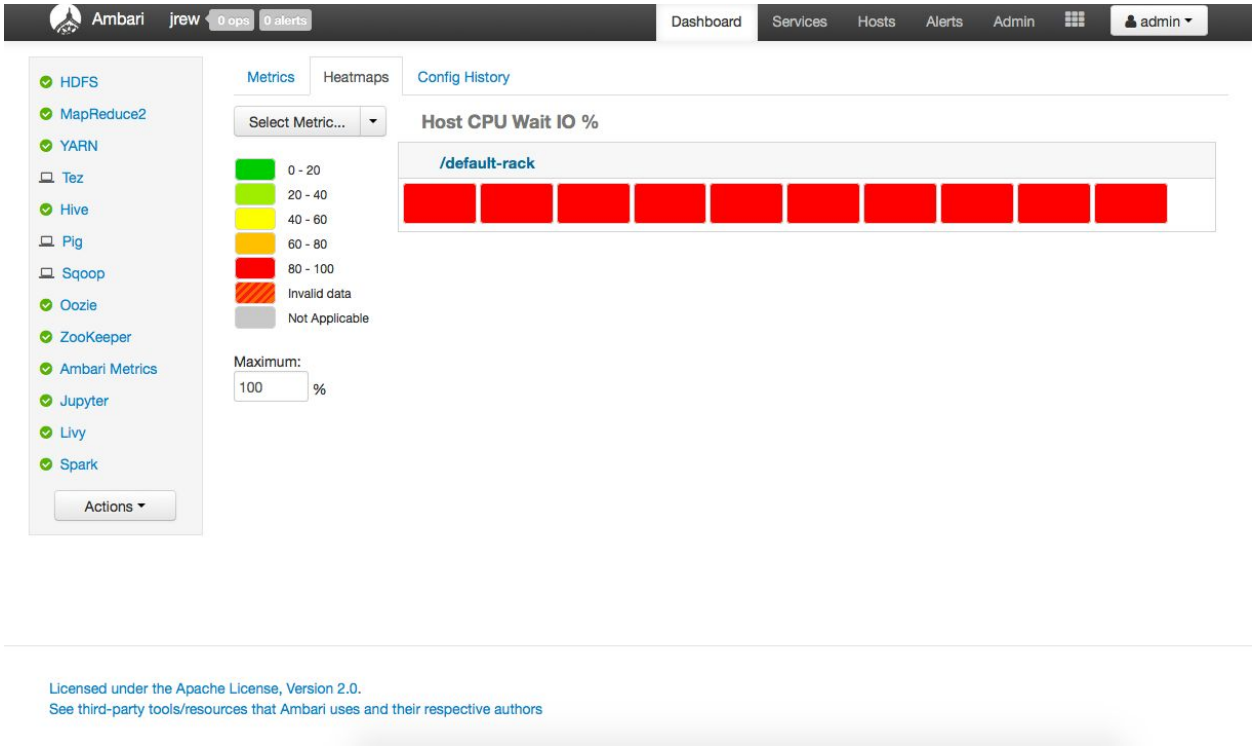

Figure 4: GraphX running on all nodes

Step 3: Post-Processing

The output of the above steps provides the wikipedia page title and its associated PageRank to stdout. This was then copied to a text file “result.txt.” This is then made to be 2 columns by removing ‘(‘, ‘)’, and ‘,’ with vim. This was then sorted using “sort -k 2 -g -r result.txt > sorted.txt” which has the final wikipedia articles sorted based on their PageRank.

Results

The top 20 pages after running graphx pageRank for 25 iterations

1. France 145.9304684849193
2. Departments_of_France 115.88453400207966
3. Communes_of_France 88.15089902914092
4. Regions_of_France 61.803216855211986
5. Animal 44.25579809110248
6. List_of_Internet_top-level_domains 40.32922695159465
7. United_States 36.630486988937214
8. Municipality 35.43299525441297
9. Plant 27.065871506573473
10. Belgium 25.67574521674622
11. Germany 24.507072293940855
12. England 23.08544267932019
13. City 22.78688085851196
14. Human 22.509074801458816

15. Country 21.172720693485356
16. Commune_in_France 20.797399997293713
17. United_Kingdom 19.98179206511936
18. India 19.72889772897683
19. Europe 19.00210317890577
20. Mammal 18.726551706125495

The top 20 after running 100 iterations.

1. France 145.9305527635898
2. Departments_of_France 115.8845385701194
3. Communes_of_France 88.15090015308583
4. Regions_of_France 61.80321879226894
5. Animal 44.25646494193374
6. List_of_Internet_top-level_domains 40.32927196311952
7. United_States 36.630713007402264
8. Municipality 35.43299919767273
9. Plant 27.0662804490448
10. Belgium 25.67575718646581
11. Germany 24.507132083794307
12. England 23.085521972499045
13. City 22.786945014936176
14. Human 22.50944858750671
15. Country 21.17285039330131
16. Commune_in_France 20.797400164740225
17. United_Kingdom 19.981885284440605
18. India 19.729004353310863
19. Europe 19.00223519817424
20. Mammal 18.72686235132462

Conclusion

The sorted pageRanks did seem to be the pages that were expected. Countries are mentioned in many different topics so it makes sense that they would be high on the list. There are many different animals and plants species so these also make a lot of sense to be ranked high. The results were checked by looking on google for other calculated wikipedia page ranks for the entire website- https://www.nayuki.io/res/computing-wikipedias-internal-pageranks/wikipedia-top-pageranks.txt And the results are similar but not the same. This could have been due to the fact that there are differences between the simple wikipedia and regular wikipedia and also maybe more iterations were needed for convergence. To check whether more iterations were needed, it was run again with 100 iterations but the results were very similar.