

# Sombra: Pagerank and Indexing for Searching the English Wikipedia

## CS560: Programming Assignment 3

Elliot D. Greenlee and Jared M. Smith

April 31st, 2017

### 1 Introduction

We implemented a web portal capable of querying results a ranked version of the Simple English Wikipedia dataset for the third programming assignment in CS560: Advanced Operating Systems. For our backend we designed and implemented a distributed version of the PageRank algorithm based on the AWS cloud computing platform. This scalable system calculates the importance of Wikipedia articles by observing internal references. We named our system after Sombra, a character from Overwatch (<https://playoverwatch.com/en-us/>), a modern team-oriented combat game, because this particular character is a supporting hacker, providing information about which enemies should be prioritized. The ultimate goal of our system is to provide the first iteration of a scalable Wikipedia search tool.



Figure 1: Sombra: An Overwatch Hero

## 2 Design and Implementation

We implemented Sombra as multiple separate portions across a variety of languages, platforms, and programming models. First, preprocessing was done on the Simple English Wikipedia. Next, we applied Google’s PageRank algorithm to this data. We incorporated our prior work, Winston (github link), in order to create an inverted index of the data for querying. Lastly, we adapted our existing webportal to work with the new PageRank system.

Web portal = external user interface. Flask app in python, allows user queries on the pre-generated inverted index file, returning the locations in a separate window.

### 2.1 Preprocessing

**Where is the Code:** The preprocessing code for the inverted index and PageRank is in `src/wikiextractor/WikiExtractor.py` and `src/pagerank/pagerank.scala`

The original dataset for this project came from Simple English Wikipedia (<https://dumps.wikimedia.org/simplewiki/>). Each page is represented in XML in the general format

```
1 <page>
2   <title> Page_Name </title>
3   (other fields we do not care about)
4   <revision>
5     <text> (Page body goes here) </text>
6   </revision>
7 </page>
```

Specific formatting information is also possible in these features, and must be removed. For the inverted index, initial extraction was completed using the Python WikiExtractor tool ([http://medialab.di.unipi.it/wiki/Wikipedia\\_Extractor](http://medialab.di.unipi.it/wiki/Wikipedia_Extractor)). WikiExtractor performs template expansion by preprocessing the whole dump and extracting template definitions. The result of this is a JSON object per page with the URL, the text page cleaned content, the page title, and the page ID. Here is an example of one of these pages, with the body text cut down.

```
1 {"url": "https://simple.wikipedia.org/wiki?curid=1", "text":
  "April\n\nApril is the ... of winter. For example: \"April showers
  bring May flowers.\n\\n\\n", "id": "1", "title": "April"}
```

For PageRank, we used two separate files as the inputs. The file `simplewiki-20170401-pagelinks.sql` lists each page ID and the pages it links to as SQL insert statements, metadata for the Wikipedia dump. This allowed us to ignore pages that are placeholders or redirections, as they do not contain meaningful texts. Additionally, we did not have to worry about considering only terms that are

titles of other pages. This data was converted for the Spark GraphX implementation of PageRank. Later, once the PageRank algorithm was complete, we converted from page ids to page titles using simplewiki-20170401-page.sql, another metadata file.

```
simplewiki-20170401-pagelinks.sql
1 INSERT INTO `pagelinks` VALUES (88452,0,'!!!!',0),(136329,0,'!!!Fuck_You!!!_and_The
2 INSERT INTO `pagelinks` VALUES (66214,0,'1775',0),(69404,0,'1775',0),(146862,0,'17
3 INSERT INTO `pagelinks` VALUES (10089,0,'1934',0),(10090,0,'1934',0),(10091,0,'193
4 INSERT INTO `pagelinks` VALUES (179483,0,'1978_FIFA_World_Cup',0),(179484,0,'1978
5 INSERT INTO `pagelinks` VALUES (46887,0,'2001',0),(48156,0,'2001',0),(48295,0,'200
6 INSERT INTO `pagelinks` VALUES (540584,0,'2015_Nigeria_Entertainment_Awards',0),(2
7 INSERT INTO `pagelinks` VALUES (8607,0,'ANO_2011',0),(342152,0,'ANSES',0),(27021
8 INSERT INTO `pagelinks` VALUES (21602,0,'Ad_Dakhiliyah_Governorate',0),(348050,0,'
9 INSERT INTO `pagelinks` VALUES (206429,0,'Airport',0),(208073,0,'Airport',0),(2130
10 INSERT INTO `pagelinks` VALUES (319259,0,'Alexandra_Stan',0),(457308,0,'Alexandra_
11 INSERT INTO `pagelinks` VALUES (159842,0,'Amblie',0),(159843,0,'Amblie',0),(159844
12 INSERT INTO `pagelinks` VALUES (50151,0,'Ancient_Greek',0),(52439,0,'Ancient_Greek
13 INSERT INTO `pagelinks` VALUES (54559,0,'Annelid',0),(54845,0,'Annelid',0),(54913
14 INSERT INTO `pagelinks` VALUES (8167,0,'April_8',0),(8235,0,'April_8',0),(8236,0,'
15 INSERT INTO `pagelinks` VALUES (340760,0,'Armenia',0),(342271,0,'Armenia',0),(3422
16 INSERT INTO `pagelinks` VALUES (432755,0,'Ask_the_dust_(film)',0),(442842,0,'Ask_t
17 INSERT INTO `pagelinks` VALUES (160226,0,'Auberville',0),(160227,0,'Auberville',0)
18 INSERT INTO `pagelinks` VALUES (253434,0,'Australian_referendum,_1977_(Senate_Casu
19 INSERT INTO `pagelinks` VALUES (467247,0,'Back_to_the_Future',0),(539152,0,'Back_t
20 INSERT INTO `pagelinks` VALUES (153040,0,'Barenton-sur-Serre',0),(153041,0,'Barent
21 INSERT INTO `pagelinks` VALUES (480257,0,'Bay_of_Islands',0),(48391,0,'Bay_of_Isl
22 INSERT INTO `pagelinks` VALUES (152548,0,'Belle_Isle,_Florida',0),(152549,0,'Belle
```

Figure 2: An example from the pagelink.sql metadata file

```

/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `page` (
  `page_id` int(8) unsigned NOT NULL AUTO_INCREMENT,
  `page_namespace` int(11) NOT NULL DEFAULT '0',
  `page_title` varbinary(255) NOT NULL DEFAULT '',
  `page_restrictions` varbinary(255) NOT NULL,
  `page_content` bigint(20) unsigned NOT NULL DEFAULT '0',
  `page_is_redirect` tinyint(1) unsigned NOT NULL DEFAULT '0',
  `page_is_new` tinyint(1) unsigned NOT NULL DEFAULT '0',
  `page_random` double unsigned NOT NULL DEFAULT '0',
  `page_touched` varbinary(14) NOT NULL DEFAULT '',
  `page_links_updated` varbinary(14) DEFAULT NULL,
  `page_latest` int(8) unsigned NOT NULL DEFAULT '0',
  `page_len` int(8) unsigned NOT NULL DEFAULT '0',
  `page_no_title_convert` tinyint(1) NOT NULL DEFAULT '0',
  `page_content_model` varbinary(32) DEFAULT NULL,
  `page_lang` varbinary(35) DEFAULT NULL,
  PRIMARY KEY (`page_id`),
  UNIQUE KEY `name_title` (`page_namespace`,`page_title`),
  KEY `page_random` (`page_random`),
  KEY `page_len` (`page_len`),
  KEY `page_redirect_namespace_len` (`page_is_redirect`,`page_namespace`,`page_len`)
) ENGINE=InnoDB AUTO_INCREMENT=79094 DEFAULT CHARSET=latin1;

/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table 'page'
--

/*!40000 ALTER TABLE `page` DISABLE KEYS */;
INSERT INTO `page` VALUES ('1,0,'April','1,0,0,0,77858293065,'20170304042720','2');
INSERT INTO `page` VALUES ('15536,0,'Account','1,0,0,0,0,54576778707,'2017081642021');
INSERT INTO `page` VALUES ('29346,0,'Harmony','1,0,0,0,0,235278219483,'2017082221414');
INSERT INTO `page` VALUES ('42925,2,'Marx_for_president','0,1,0,0,97062465326,'2');
INSERT INTO `page` VALUES ('5661,0,'SantaMargaritaRiver','1,0,0,0,0,127575327,'2');
INSERT INTO `page` VALUES ('65624,3,'123_166_12','0,0,0,0,0,620662637398,'20160830');
INSERT INTO `page` VALUES ('81833,4,'Compositions_by_lan_Benn','0,0,0,0,0,0,5380099');
INSERT INTO `page` VALUES ('96526,3,'68.9,172.140','0,0,1,0,0,98347759711,'20100301');
INSERT INTO `page` VALUES ('11913,0,'CHU','0,1,0,0,0,30485786889,'20150823122718');
INSERT INTO `page` VALUES ('127562,0,'Wings_(film)','0,1,1,0,0,374518866458,'201409');
INSERT INTO `page` VALUES ('137889,0,'Ensüe la Redonne','0,0,0,0,0,316566515815,'2');
INSERT INTO `page` VALUES ('150571,0,'Fontan','0,0,0,0,0,82991446438,'20151129159152');
INSERT INTO `page` VALUES ('161692,3,'97.89.1.190','0,0,1,0,0,780185336445,'2010031');
INSERT INTO `page` VALUES ('176397,0,'Santa_Margarita_River','0,0,0,0,0,10561175353');
INSERT INTO `page` VALUES ('196615,0,'Sun_Yat-sen','0,0,0,0,0,3214281708795,'2017032');
INSERT INTO `page` VALUES ('204693,0,'The_Chinese_language_family','0,1,1,0,0,85207');
INSERT INTO `page` VALUES ('220158,0,'Georgetown,_Ontario','0,0,0,0,0,1901448851,'1');
INSERT INTO `page` VALUES ('234561,2,'Merlion444','0,0,0,0,0,705724821968,'20130403);
INSERT INTO `page` VALUES ('249732,3,'121.7.39.154','0,0,0,1,0,302453194954,'201006);
INSERT INTO `page` VALUES ('261188,1,'PlayStation_Store','0,0,1,0,0,565724402967,'201303241828);
INSERT INTO `page` VALUES ('274083,0,'1.0.0.1.0,79397326489,'201303241828);

```

Figure 3: An example from the page.sql metadata file

## 2.2 PageRank

**Where is the Code:** The PageRank code for ranking pages is in `src/pagerank/pagerank.scala`

**Where are the Results:** The PageRank results are at `src/pagerank/pagerank.tar.gz`, which can be unzipped with `tar -xzvf pagerank.tar.gz` on a Linux or Mac system or with a Windows unarchiving tool.

**Helper Scripts:** The file `run_tests.sh` will run 25 runs of the algorithm and store the results on disk.

PageRank is an algorithm used on early versions of Google in order to rank the popularity of various pages based on internal references. We implemented a Spark GraphX version of PageRank in Scala using the instructions at <http://mike.seddon.ca/computing-wikipedias-internal-pagerank-with-spark/>. This PageRank algorithm was run on an Amazon Web Services (<https://aws.amazon.com>) cloud computing platform. Rather than running the algorithm for a distinct number of iterations, our implementation runs until convergence to 0.0001, which is the default way Spark's GraphX library does PageRank. We ran the algorithm 25 times for an average of 176 seconds per run, and we recorded the results each time and stored them in our submission directory. Screenshots detailing the exact process can be found below.

## 2.3 Inverted Index

**Where is the Code:** The MapReduce code for building the inverted index is in `src/invertedindex/InvertedIndexBuilder.java`.

**Where are the Results:** The inverted index results are at `src/invertedindex/inverted_index.tar.gz`, which can be unzipped with `tar -xzvf inverted_index.tar.gz` on a Linux or Mac system or with a Windows unarchiving tool.

An inverted index stores mappings from some content to its location in a larger database. Included in this work is the ability to remove stop words based on frequency, but in this assignment we downloaded a stopwords file online and filtered very common English stopwords with the `stopwords.txt` file in the `src/invertedindex` directory. In this assignment, we re-purposed our earlier work from Winston (Programming Assignment 2) to create an inverted index mapping words to their locations in the Simple English Wikipedia. This location data consisted of a document ID, line number, and line index (word position), all of which are needed for the later querying ability. This functionality was performed using a MapReduce function in Hadoop on Erebus, a 400+ GB RAM, 58 core, 50 TB research server and took 83 minutes to build the inverted index for all of the English Wikipedia. Screenshots showing this process are below.

## 2.4 Query

**Where is the Code - Query:** The Python code for querying the inverted

index is in src/searchengine/searchui/searchui/query.py

The final goal of the project was a querying program on top of the internal analysis, which combines the PageRank and Inverted Index generated earlier. Our query program is a Python 2 program re-purposed from earlier work on our Winston system (Programming Assignment 2), which accepts a user-specified query of one or more words. It obtains keywords hits from the the inverted index over all of Wikipedia, and using a our own ranking algorithm, it determines a good display ranking of the hits for later display. The ranking algorithm we developed is defined by this Python code, which returns a list of results in the form of tuples with the word, document name, line number, and word position (where the word is on the line):

```

1 NON_ALPHANUMERIC_CHARS = re.compile('[^a-zA-Z0-9 ]')
2
3 def query(inverted_index, query_string):
4     QueryResult = namedtuple('QueryResult', 'word doc_name line_number
5                               word_number')
6
7     results = []
8     cleaned_query = re.sub(NON_ALPHANUMERIC_CHARS, '',
9                           query_string.rstrip())
10    words_in_query = cleaned_query.lower().split(' ')
11
12    for word in words_in_query:
13        word_entry = inverted_index.get(word)
14        if word_entry is not None:
15            doc_names = word_entry.keys()
16
17            ranks = {}
18            for doc_name in doc_names:
19                rank = pagerank.get_rank(doc_name)
20                if rank is None:
21                    continue
22                else:
23                    ranks[doc_name] = rank
24
25            sorted_ranks = sorted(ranks.items(),
26                                  key=operator.itemgetter(1), reverse=True)
27
28            for ranked_doc_name in sorted_ranks:
29                word_positions = word_entry[ranked_doc_name[0]]
30                for line_number, word_number in word_positions:
31                    results.append(QueryResult(word, ranked_doc_name[0],
32                                              line_number, word_number))
33
34    return results

```

Essentially, this code does the following to get the list of page results:

1. First, we strip the query of non-alphanumeric characters.
2. For each word in the query string:
  - (a) We grab the entry in the inverted index for that word. For example, if we search for "Earth", then the inverted index of Wikipedia will give us a dictionary where the keys are the page titles on Wikipedia where the word "Earth" occurs and the values are dictionaries of line numbers and word number (i.e. the word position or where on the line the word is) of the occurrence of the word "Earth" in the file for the current key.

- (b) Once we have that nested dictionary, for each document name (i.e. page title) where this word occurs:
    - i. We lookup in our pagerank file, which was generated by Apache Spark and our earlier code, the numerical pagerank value for that document name (i.e. page title) on Wikipedia.
    - ii. Then, we build another dictionary of document names to the rank for that document.
  - (c) With this map of document names to ranks, we iterate over the document names we have for the current word in the query string in order of the documents ranked the highest (with the largest numerical page rank) to the documents ranked the lowest:
    - i. At this point, we're going over the results from the inverted index based on our searched word in the query string **in descending order of rank for the page**, and we simply append to the results list we're building up for this query all the results for the document we're on in the ranked documents
3. Finally, we return the custom-ranked results based on the pagerank and the inverted index!

It then returns the relevant location document IDs, line numbers, and line indexes. Currently, our querying functionality is limited to uncomplicated multiple word queries. However, common operations like "and," "or," and "not" would be feasible on our system. Additionally, we have implemented the underlying word-position logic to search for consecutive words using quote specification such as in the example "fare well." In the future this query would only return the lines where these two words appear consecutively.

## 2.5 Web Portal

**Where is the Code:** The web user interface code is in src/searchengine/searchui.

We further improved the query by implementing it in our Python 2 web portal, re-purposed from Winston, and similar to a real search engine. It allows the submission of queries, and then displays the results in a returned page. We built the UI as a Flask App (<https://github.com/sloria/cookiecutter-flask>). Screenshots of the Web UI, search engine are below.

You can run the Web Portal by first install the Python requirements with **pip install -r requirements.txt** then running **honcho start**. Make sure you use Python 2.7.X if you run it. Additionally, you must first have the pagerank.txt file from the pagerank.tar.gz file, the inverted\_index.txt file from the inverted\_index.tar.gz file, and the page\_ids\_to\_titles.csv file from the inverted\_index.tar.gz file in the root of the searchui directory at src/searchengine/searchui in order for the querying program to be able to run queries.

### 3 Screenshots and Process

In the next set of screenshots, we will show you the following:

1. **Setting Up AWS:** Using AWS to setup an Elastic Map Reduce (EMR) cluster with Apache Spark and HDFS to run the PageRank algorithm in a distributed fashion. We did ours with 3 worker nodes (virtual machines on AWS) and 1 master node (the virtual machine on AWS where the driver program for Spark was run from).
2. **Running PageRank on AWS:** Logged into the AWS master node and running 25 runs of the PageRank algorithm.
3. **Extra Credit: Creating the Inverted Index with Winston:** Logged into a large server to compute the Inverted Index for the English Wikipedia based on Winston, our code for Programming Assignment 2.
4. **Extra Credit: Web UI and Custom Ranking:** The final extra credit portion, which is the Web UI using the query function described above.

#### 3.1 Setting Up AWS

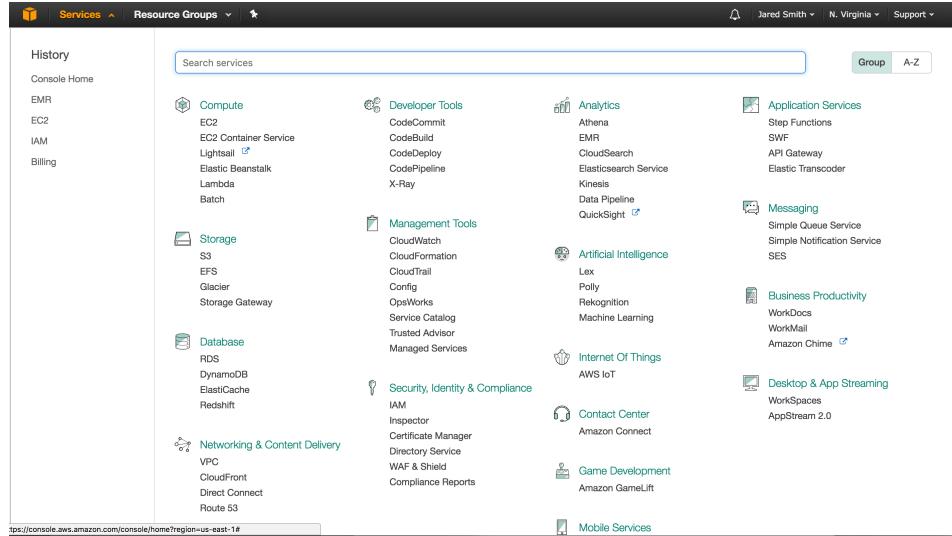


Figure 4: Listing of All AWS Services on the AWS Management Console

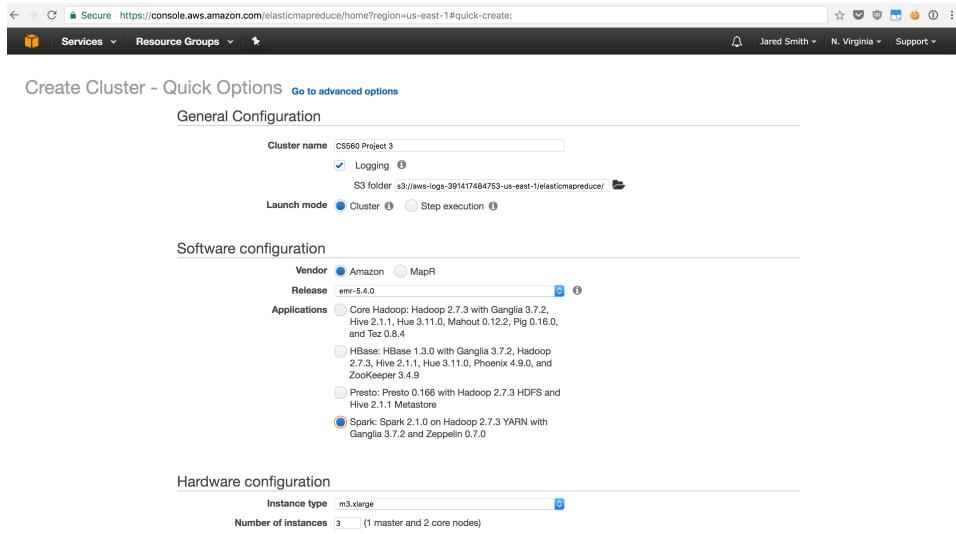


Figure 5: Configuring the AWS EMR cluster, where EMR is Elastic Map Reduce, and is Amazon's pre-built Hadoop/Spark/Kafka/etc. big data service.

Name	ID	Status	Creation time (UTC-4)	Elapsed time	Normalized instance hours
CS560-Project3-3	j-285T9GJHUCYAP	Terminating User request	2017-04-16 19:43 (UTC-4)	2 hours, 55 minutes	96
CS560-Project 3-2	j-2CZ3ANBD06WLQ	Terminated with errors Validation error	2017-04-16 19:41 (UTC-4)	40 seconds	0
CS560 Project 3	j-2AUELKFR821RN5	Terminated User request	2017-04-16 18:44 (UTC-4)	19 minutes	32
cs560project3	j-1R30GFNPNUWE2	Terminated User request	2017-04-16 18:36 (UTC-4)	7 minutes	0
CS560 Project 3	j-3J8GU2J05W0B5	Terminated with errors Validation error	2017-04-16 18:22 (UTC-4)	1 minute	0

Figure 6: We ran into several jobs where the configuration was incorrectly done. This shows the list of prior failed cluster jobs with bad configurations.

The screenshot shows the AWS EMR Cluster Details page. The cluster is named "CS560 Project 3" and is currently "Starting". Key configuration details include:

- Configuration Details:** ID: j-197LZU3YRM6TK, Release label: emr-5.4.0, Hadoop distribution: Amazon 2.7.3, Applications: Ganglia 3.7.2, Spark 2.1.0, Zeppelin 0.7.0, Log URI: s3://aws-logs-391417484753-us-east-1/elasticmapreduce/.
- Network and Hardware:** Availability zone: --, Subnet ID: subnet-0ef91d098, Master: Provisioning 1 m3.xlarge, Core: Provisioning 2 m3.xlarge, Task: --.
- Security and Access:** Key name: aws-pair1, EC2 instance profile: EMR\_EC2\_DefaultRole, EMR role: EMR\_DefaultRole, Visible to all users: All Change, Security groups for Master: --, Security groups for Core & Task: --.

Navigation links at the bottom include Monitoring, Hardware, and Steps.

Figure 7: Starting the AWS EMR cluster

Hardware					
Add task instance group					
Instance Groups					
Filter: Filter instance groups ...	2 instance groups (all loaded)	ID	Status	Node type & name	Instance Type
ig-3MG926JL6X0SP	Provisioning (1 Requested)	MASTER	Master Instance Group	m3.xlarge 8 vCPU, 15 GiB memory, 80 SSD GB storage EBS Storage: none	0 Instances
ig-AZYSL385PB2D	Provisioning (2 Requested)	CORE	Core Instance Group	m3.xlarge 8 vCPU, 15 GiB memory, 80 SSD GB storage EBS Storage: none	0 Instances

Figure 8: This shows the hardware (virtual machines we requested). You can see it says 1 master node and 3 worker nodes.

The screenshot shows the AWS EMR console interface. On the left, there's a sidebar with options like 'Cluster list', 'Security configurations', 'VPC subnets', and 'Help'. The main area is titled 'Cluster: CS560-Project3-3' with a status of 'Waiting'. It shows the cluster is ready after the last step completed. Below this, there are sections for 'Connections', 'Summary', 'Network and Hardware', and 'Security and Access'. The 'Summary' section provides details such as ID (j-285T9GJHUCYAP), Creation date (2017-04-16 19:43 UTC-4), and Hadoop distribution (Amazon 2.7.3). The 'Network and Hardware' section shows the availability zone (us-east-1d), subnet ID (subnet-ef0596b7), and EC2 instance profile (EMR\_EC2\_DefaultRole). The 'Security and Access' section includes information about EC2 instance profiles, EMR roles, and security groups. At the bottom, there are links for 'Monitoring', 'Hardware', and 'Steps'.

Figure 9: The AWS EMR cluster is now online

Hardware					
Add task instance group					
Instance Groups					
Filter: Filter instance groups ...					
ID	Status	Node type & name	Instance Type	Instance Count	
ig-H8T25SLVVKP	Running	CORE Core Instance Group	m3.xlarge 8 vCPU, 15 GiB memory, 80 SSD GB storage EBS Storage: none	3 Instances	Resize
ig-1O2TFP4TGWMKO	Running	MASTER Master Instance Group	m3.xlarge 8 vCPU, 15 GiB memory, 80 SSD GB storage EBS Storage: none	1 Instances	

Figure 10: Running the nodes

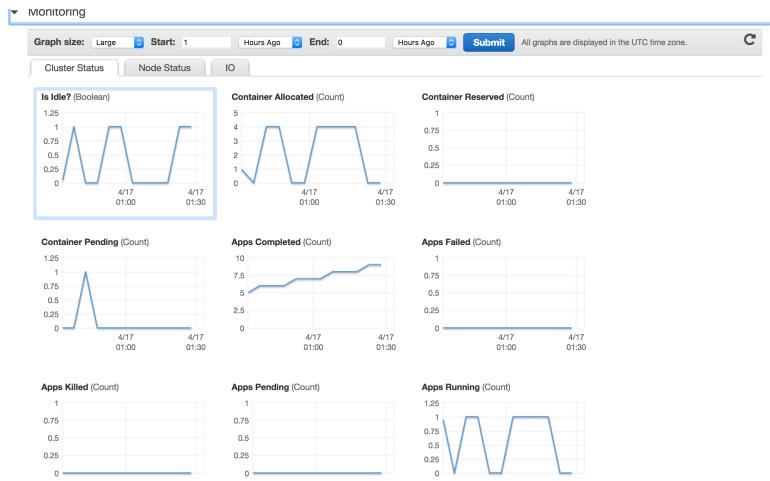


Figure 11: These are some of the graphs for metrics AWS is monitoring on our cluster.

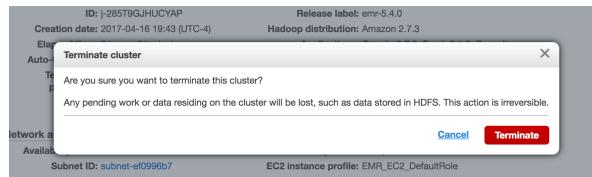


Figure 12: Cluster termination dialog

The screenshot shows the AWS EMR Cluster Details page. The cluster is identified as 'CS560-Project3-3' and is currently 'Terminating' due to a user request. Key details include:

- Connections:** Master public DNS: ec2-52-201-103-137.compute-1.amazonaws.com SSH
- Tags:** None
- Summary:** ID: j-285T9GJHUCYAP, Creation date: 2017-04-16 19:43 (UTC-4), Elapsed time: 2 hours, 54 minutes, Auto-terminate: No, Termination Off protection: None.
- Configuration Details:** Release label: emr-5.4.0, Hadoop distribution: Amazon 2.7.3, Applications: Ganglia 3.7.2, Spark 2.1.0, Zeppelin 0.7.0, Log URI: s3://aws-logs-391417484753-us-east-1/elasticmapreduce/, EMRFS consistent view: Enabled.
- Network and Hardware:** Availability zone: us-east-1d, Subnet ID: subnet-ef0b9867, Master: Running 1 m3.xlarge, Core: Running 3 m3.xlarge, Task: --.
- Security and Access:** Key name: aws-pair1, EC2 instance profile: EMR\_EC2\_DefaultRole, EMR role: EMR\_DefaultRole, Visible to all users: All Change, Security groups for master: sg-4b098534 (ElasticMapReduce-Master), Security groups for slave: sg-5fb49f120 (ElasticMapReduce-Core & Task: slave).

Figure 13: Stopping the AWS cluster

### 3.2 Running PageRank on AWS

We ran PageRank on AWS with 3 worker nodes and 1 master node. Below are the screenshots of us SSH'd into the master node running the pagerank algorithm. It took on average 176 seconds to compute the pagerank.

```

> ssh -i ~/aws-pair1.pem hadoop@ec2-52-201-103-137.compute-1.amazonaws.com [21:36:46]
Last login: Mon Apr 17 01:35:39 2017 from 99-127-194-19.lightspeed.knvltn.sbcglobal.net
[...]
https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
22 package(s) needed for security, out of 45 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2017.03 is available.

[...]
[hadoop@ip-172-31-27-116 ~]$ 

```

Figure 14: We see this once we log into the master node via SSH, which you can get the address for at the top of the cluster status page above.

```
[hadoop@ip-172-31-27-116 sombra]$ hadoop fs -ls
Found 3 items
drwxr-xr-x   1 hadoop hadoop          0 2017-04-17 01:20 .sparkStaging
-rw-r--r--   1 hadoop hadoop    269459986 2017-04-17 00:23 simplewiki-20170401-page.sql
-rw-r--r--   1 hadoop hadoop    269459986 2017-04-17 00:23 simplewiki-20170401-pageLinks.sql
[hadoop@ip-172-31-27-116 sombra]$
```

Figure 15: The list of input files in HDFS that will be fed to the PageRank algorithm.

```
[hadoop@ip-172-31-27-116 sombra]$ spark-shell -i pagerank.scala
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel)
el)
17/04/17 01:36:31 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME,
17/04/17 01:36:51 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://172.31.27.116:4040
Spark context available as 'sc' (master = yarn, app id = application_1492386406944_0010)

Spark session available as 'spark'.
Loading pagerank.scala...
import org.apache.spark.graphx_
import org.apache.spark.rdd.RDD
import java.io...
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@4e3c13b
printToFile: (f: java.io.File)(op: java.io.PrintWriter => Unit)Unit
timestampStart: Long = 1492393014994
page: org.apache.spark.rdd.RDD[String] = simplewiki-20170401-page.sql MapPartitionsRDD[1
] at <console>:35
pageLink: org.apache.spark.rdd.RDD[String] = simplewiki-20170401-pageLinks.sql MapPartit
ionsRDD[2] at <console>:31
pagePattern: scala.util.matching.Regex = \((\d+),0,[^\d]+,\d+\)[,]|
pageRDD: org.apache.spark.rdd.RDD[(String, Long)] = MapPartitionsRDD[4] at flatMap at <co
nsole>:35
pageLinksPattern: scala.util.matching.Regex = \((\d+),0,[^\d]+,\d+\)[,]|
pageLinksRDD: org.apache.spark.rdd.RDD[(String, Long)] = MapPartitionsRDD[5] at flatMap
at <console>:35
(0 + 2) / 2]
```

Figure 16: Running the pagerank algorithm across all 3 worker nodes, which is a Spark Scala job.

```
edges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = MapPartitionsRDD[22
] at rdd at <console>:39
vertices: org.apache.spark.rdd.RDD[(Long, String)] = MapPartitionsRDD[23] at map at <con
sole>:37
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImp
l@3270956
[Stage 5:====>          (47 + 16) / 200][Stage 6:>          (0 + 8) / 200]
```

Figure 17: The page rank is running and Spark is at Stage 5 of processing. Recall the Spark paper describes how Spark uses stages to process the Directed Acyclic Graph of tasks it needs to do.

```
vertices: org.apache.spark.rdd.RDD[(Long, String)] = MapPartitionsRDD[23] at map at <con
sole>:37
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImp
l@73270956
[Stage 6:*****>          (108 + 16) / 200]
```

Figure 18: Now at stage 6

```
vertices: org.apache.spark.rdd.RDD[(Long, String)] = MapPartitionsRDD[23] at map at <con
sole>:37
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImp
l@73270956
[Stage 275:*****>          (172 + 16) / 200]
```

Figure 19: Now at stage 275

```

edges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = MapPartitionsRDD[22]
  1 at rdd at <console>:39
vertices: org.apache.spark.rdd.RDD[(Long, String)] = MapPartitionsRDD[23] at map at <con
sole>:37
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImp
l@73270956
[Stage 1149:> (0 + 2) / 2]

```

Figure 20: Now at stage 1149

```

scala> titleAndPrGraph.vertices.count()
res9: Long = 174803

```

Figure 21: Now we can see the number of vertices in the pagerank generated by Spark the GraphX library on the Wikipedia data.

```

prGraph: org.apache.spark.graphx.Graph[Double,Double] = org.apache.spark.graphx.impl.Gra
phImpl@57e14b
titleAndPrGraph: org.apache.spark.graphx.Graph[(Double, String),Int] = org.apache.spark.
graphx.impl.GraphImpl@42d6568e
timestampEnd: Long = 149393192292
duration: Long = 177298
Duration: 177298 ms (-2 minutes)
United_States: 818.2641259896291
Multimedia: 816.6660942196424
France: 395.8886873670224
Definition: 338.43461234428815
English_language: 319.8528823227395
International_Standard_Book_Number: 284.0803868582328
Country: 250.9338480969472
Geographic_coordinate_system: 234.02814100972452
England: 233.6676732723399

```

Figure 22: The PageRank job finished with top 10 pages of the pagerank

```

#!/bin/bash
for run in {1..25}
do
  spark-shell -i pagerank.java
done

```

Figure 23: We used this script to automate running 25 runs of the pagerank algorithm and collecting results.

```

[hadoop@ip-172-31-27-116 sombra]$ ls results/
pageRank-174-seconds-17 pageRank-174-seconds-17 pageRank-177-seconds-12
pageRank-174-seconds-18 pageRank-175-seconds-19 pageRank-177-seconds-2
pageRank-174-seconds-25 pageRank-175-seconds-20 pageRank-177-seconds-23
pageRank-174-seconds-25 pageRank-175-seconds-21 pageRank-177-seconds-24
pageRank-174-seconds-1 pageRank-175-seconds-22 pageRank-177-seconds-3
pageRank-175-seconds-1 pageRank-175-seconds-4 pageRank-178-seconds-16
pageRank-175-seconds-11 pageRank-175-seconds-5 pageRank-178-seconds-7
pageRank-175-seconds-13 pageRank-175-seconds-6 pageRank-178-seconds-15
pageRank-175-seconds-14 pageRank-176-seconds-15

```

Figure 24: Output files from all iterations where the number in the filename is the number of seconds to finish.

### 3.3 Extra Credit: Creating the Inverted Index with Winston

We acquired the inverted index with the document name, line number, and word position for every word in the English Wikipedia using Winston, our code from Programming Assignment 2. It took 83 minutes to collect all of the data.

To build the inverted index, run the following commands:

- `python WikiExtractor.py --json -it  
abbr,b,big,blockquote,center,cite,em,font,h1,h2,h3,h4,hiero,i,kbd,p,plaintext,s,span,strike,strong,t  
t,u,var ../../data/simplewiki-20170401-pages-meta-current.xml`

Then move the output from wiki extractor to the data directory.

- `./clean_data.py ../../data/text/`
- `number_input.py --i cleaned_pages --o numbered_cleaned_pages`
- `mv numbered_cleaned_pages input`
- `javac -classpath  
"/hadoop/etc/hadoop:/hadoop/share/hadoop/common/lib/*:/hadoop/share/hadoop/common/*:/hadoop/share/hadoop/hdfs:/hadoop/share/hadoop/hdfs/lib/*:/hadoop/share/hadoop/hdfs/*:/hadoop/share/hadoop/yarn/lib/*:/hadoop/share/hadoop/yarn/*:/hadoop/share/hadoop/mapreduce/lib/*:/hadoop/share/hadoop/mapreduce/*:  
/contrib/capacity-scheduler/*.jar" -d ./build *.java`
- `jar cvf InvertedIndex.jar *`
- `mkdir output`
- `hadoop jar InvertedIndex.jar elliotjared.programmingassignmenttwo.InvertedIndexBuilder input output`

Figure 25: This shows the steps we took to generate the inverted index.

```
[20:21:55] > python WikiExtractor.py --links ../../data/simplewiki-20170401-pages-meta-current.xml --filter_disambig_pages --no-templates --json --processes 1 --output -  
INFO: Starting page extraction from ../../data/simplewiki-20170401-pages-meta-current.xml.  
INFO: Using 1 extract processes.  
INFO: 1 April  
INFO: 2 August  
(url: "https://simple.wikipedia.org/wik?curid=1", "text": "April\\n\\nApril is the fourth month of the year\\n\\year, and comes between March\\n\\March and May\\n\\May. It is one of four months to have 30 day\\n\\days.\\n\\nApril always starts on the same day of week as January\\n\\January in leap years.\\n\\nApril's flower\\n\\flowers are the Sweet Pea\\n\\Sweet Pea and Asteraceae\\n\\Daisy. Its birthstone\\n\\birthstone is the diamond\\n\\diamond. The meaning of the diamond is innocence.\\n\\nApril comes between March\\n\\March and May\\n\\May, making it the fourth month of the year. It also comes first in the year out of the four months that have 30 days, as June\\n\\June, September\\n\\September and November\\n\\November are later in the year.\\n\\nApril starts on the same day of the week as July\\n\\July every year, and also starts on the same day as January\\n\\January in leap year\\n\\leap year. April ends on the same day of the week as December\\n\\December every year.\\n\\nApril in a common year\\n\\common year starts on the same day of the week as October\\n\\October, and ends on the same day as July\\n\\July, both of the previous year. April in a common year that follows another one starts on the same day of the week as January\\n\\January of the previous year. April in both a leap year and a year that follows one ends on the same day of the week as January\\n\\January of the previous year. April in a leap year\\n\\leap year starts on the same day of the week as May\\n\\May, and ends on the same day of the week as February\\n\\February and October\\n\\October, both of the previous year.\\n\\nApril starts on the same day of the week as September\\n\\September and December\\n\\December, and ends on the same day of the week as Se
```

Figure 26: Running WikiExtractor to get the page data as JSON to pass to our inverted indexer.

```
[2018:55]
> head -n 1 wikit_00
("url": "https://simple.wikipedia.org/wiki?curid=1", "text": "April\nApril is the fourth month of the year, and comes between March and May. It is one of four months to have 30 days.\nApril always starts on the same day of week as July, and ends on the same day of the week as December. Additionally, it starts on the same day of the week as January in leap years.\nApril's flowers are the Sweet Pea and Daisy. Its birthstone is the diamond. The meaning of the diamond is innocence.\nApril comes between March and May, making it the fourth month of the year. It also comes first in the year out of the four months that have 30 days, as June, September and November are later in the year.\nApril starts on the same day of the week as July every year, and also starts on the same day as January in leap years. April ends on the same day of the week as December every year.\nApril in a common year starts on the same day of the week as October, and ends on the same day as July, both of the previous year. April in a common year that follows another one starts on the same day of the week as January of the previous year. April in both a leap year and a year that follows one ends on the same day of the week as January of the previous year. April in a leap year starts on the same day of the week as May, and ends on the same day of the week as February and October, both of the previous year.\nApril starts on the same day of the week as September and December, and ends on the same day of the week as September, both of the next year as a common year. April starts on the same day of the week as June, and ends on the same day of the week as March and June, both of the next year as a leap year.\nApril is a spring month in the Northern Hemisphere and an autumn/fall month in the Southern Hemisphere. In each hemisphere, it is the seasonal equivalent of October in the other.\nIt is unclear as to where April got its name. A common theory is that it comes from the Latin word \"aperire\", meaning \"to open\", referring to flowers opening in spring. Another theory is that the name could come from Aphrodite, the Greek goddess of love. It was originally the second month in the old Roman Calendar, before the start of the new year was put to January 1.\nQuite a few festivals are held in this month. In many Southeast Asian cultures, new year is celebrated in this month (including Songkran). In Western Christianity, Easter can be celebrated on a Sunday between March 22 and April 25. In Orthodox Christianity, it can fall between April 4 and May 8. At the end of the month, Central and Northern European cultures celebrate Walpurgis Night on April 30, marking the transition from winter into summer.\nPoets use \"April\" to mean the end of winter. For example: \"April showers bring May flowers.\"\\n\\n", "id": "1", "title": "April"}]
```

Figure 27: Sample output from Wikiextractor

```
[jms@erebus invertedindex]$ time hadoop jar InvertedIndex.jar elliotjared.programmingassignmenttwo.InvertedIndexBuilder input output
17/04/18 12:40:41 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/04/18 12:40:41 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/04/18 12:40:41 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
```

Figure 28: Now we're running the inverted index

```
[jms@erebus invertedindex]$ time hadoop jar InvertedIndex.jar elliotjared.programmingassignmenttwo.InvertedIndexBuilder input output
17/04/18 12:40:41 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/04/18 12:40:41 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/04/18 12:56:38 INFO input.FileInputFormat: Total Input files to process : 124002
17/04/18 12:56:39 INFO mapreduce.JobSubmitter: number of splits:124002
17/04/18 12:56:39 INFO mapreduce.JobSubmitter: Submitting token for job: job_local1691128810_0001
17/04/18 12:56:39 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/04/18 12:56:39 INFO mapreduce.Job: Running job: job_local1691128810_0001
17/04/18 12:56:39 INFO mapred.LocalJobRunner: OutputCommitter set in config null
17/04/18 12:56:39 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/04/18 12:56:39 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/04/18 12:56:39 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/04/18 12:50:42 INFO mapreduce.Job: Job job_local1691128810_0001 running in uber mode : false
17/04/18 12:50:42 INFO mapreduce.Job: map % reduce %
17/04/18 12:50:55 WARN mapred.LocalJobRunner: job_local1691128810_0001
java.lang.OutOfMemoryError: GC overhead limit exceeded
    at java.util.Hashtable$Entry.clone(Hashtable.java:1261)
    at java.util.Hashtable.clone(Hashtable.java:550)
    at org.apache.hadoop.conf.Configuration.<init>(Configuration.java:714)
    at org.apache.hadoop.mapred.JobConf.<init>(JobConf.java:442)
    at org.apache.hadoop.mapred.LocalJobRunner$Job$MapTaskRunnable.<init>(LocalJobRunner.java:244)
    at org.apache.hadoop.mapred.LocalJobRunner$Job.getMapTaskRunnables(LocalJobRunner.java:99)
    at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:544)
17/04/18 12:50:56 INFO mapreduce.Job: Job job_local1691128810_0001 failed with state FAILED due to: NA
17/04/18 12:50:56 INFO mapreduce.Job: Counters: 0

real    10m16.062s
user    18m26.663s
sys     2m50.530s
```

Figure 29: However, sometimes we had not enough memory issues for the inverted index. We increased the memory by raising the maximum heap size for the JVM and it took off fine from there.

```

2017-04-18 16:14:23,668 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:init(1091)) - kvstart = 26214396; length = 6553600
2017-04-18 16:14:23,668 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:createSortingCollector(403)) - Map output collector class = org.apache.red.MapTask$MapOutputBuffer
2017-04-18 16:14:23,668 INFO [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner (LocalJobRunner.java:statusUpdate(618)) -
2017-04-18 16:14:23,871 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:flush(1462)) - Starting flush of map output
2017-04-18 16:14:23,871 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:flush(1484)) - Spilling map output
2017-04-18 16:14:23,871 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:flush(1485)) - bufstart = 0; bufend = 144; bufvoid = 104857600
2017-04-18 16:14:23,871 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:flush(1487)) - kvstart = 26214396(104857584); kvend = 26214380(10485752
= 17/6553600
2017-04-18 16:14:23,871 INFO [LocalJobRunner Map Task Executor #0] mapred.MapTask (MapTask.java:sortAndSpill(1669)) - Finished spill 0
2017-04-18 16:14:23,872 INFO [LocalJobRunner Map Task Executor #0] mapred.Task (Task.java:done(1099)) - Task@attempt_local2006851354_0001_m_105843_0 is done. And is in
ss of committing
2017-04-18 16:14:23,872 INFO [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner (LocalJobRunner.java:statusUpdate(618)) - map
2017-04-18 16:14:23,880 INFO [LocalJobRunner Map Task Executor #0] mapred.Task (Task.java:sendDone(1219)) - Task 'attempt_local2006851354_0001_m_105843_0' done.
2017-04-18 16:14:23,880 INFO [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner (LocalJobRunner.java:run(276)) - Finishing task: attempt_local2006851354_0001_
2017-04-18 16:14:23,880 INFO [LocalJobRunner Map Task Executor #0] mapred.LocalJobRunner (LocalJobRunner.java:run(251)) - Starting task: attempt_local2006851354_0001_m
2017-04-18 16:14:23,880 INFO [LocalJobRunner Map Task Executor #0] output.FileOutputCommitter (FileOutputCommitter.java:<init>(123)) - File Output Committer Algorithm

```

Figure 30: Here is the job in the middle of it running.

```

2017-04-18 18:24:16,998 INFO [main] mapreduce.Job (Job.java:monitorAndPrintJob(1418)) - map 100% reduce 100%
2017-04-18 18:24:28,802 INFO [main] mapreduce.Job (Job.java:monitorAndPrintJob(1429)) - Job job_local1623027122_0001 completed successfully
2017-04-18 18:24:30,980 INFO [main] mapreduce.Job (Job.java:monitorAndPrintJob(1436)) - Counters: 30
  File System Counters
    FILE: Number of bytes read=103425034928712
    FILE: Number of bytes written=4243540172279
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
  Map-Reduce Framework
    Map Input records=1117655
    Map output records=1117655
    Map output bytes=16811965
    Map output materialized bytes=19791364
    Input split bytes=16464685
    Combine input records=0
    Combine output records=0
    Reduce input groups=369368
    Reduce shuffle bytes=19791364
    Reduce input records=1117655
    Reduce output records=369368
    Spilled Records=2235310
    Shuffled Maps =124014
    Failed Shuffles=0
    Merged Map outputs=124014
    GC time elapsed (ms)=219164
    Total committed heap usage (bytes)=2856376129290240
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=107060430
  File Output Format Counters
    Bytes Written=14117342
real    83m37.288s
user    17m16.502s
sys     15m57.737s

```

Figure 31: Finally, it's completed in 83 minutes.

### 3.4 Extra Credit: Web UI and Custom Ranking

We also ran a few basic queries through our web portal for the extra credit portion to show functionality. Here are those results.

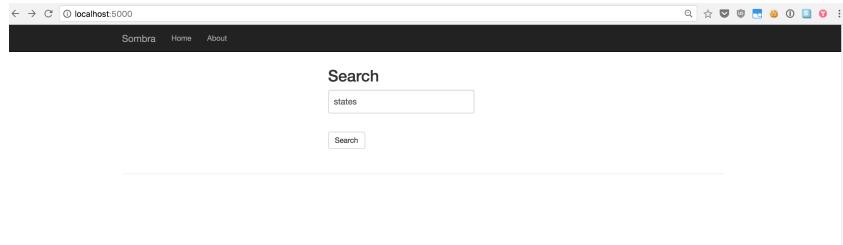


Figure 32: Here is the main search page, which is borrowed from the extra credit for Programming Assignment 2.

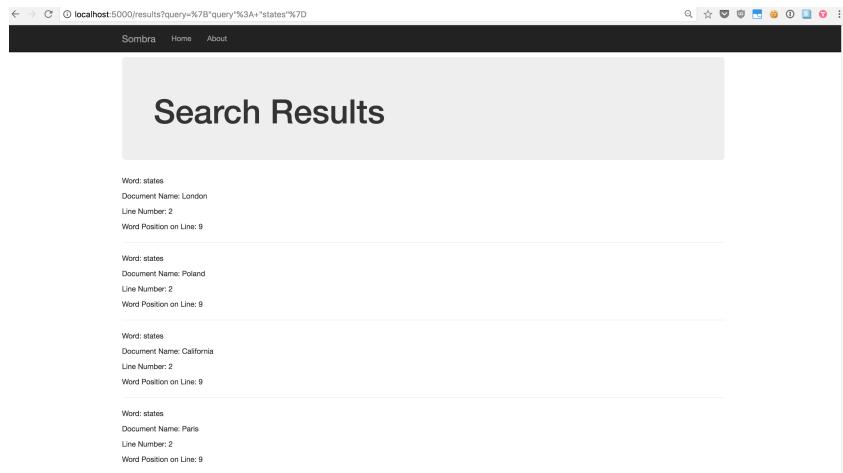


Figure 33: Here is the search results page for the query "state", showing the highest ranking results at the top based on the pagerank and the underlying results based on the inverted index.

## 4 Test Results

As part of our testing process, we ran both the PageRank algorithm by itself, and the querying and webpage functionality as a whole. For the PageRank test, we produced a sorted list of Simple English Wikipedia pages based on their PageRank values in descending order on three virtual machine instances. This test took 176 seconds to converge on average across 25 runs. The top results for one iteration are shown below, and one set of the full results are available at `/results/pagerank-176-seconds-15`. This ranked list makes perfect sense for our dataset. An English speaking country and Wikipedia creator, the United States, tops the list, followed up by other countries. All of these would be linked to in many cultural references across simple pages.

```

United States: 910.3641259896291
Multimedia: 816.660094219424
France: 395.8886873670224
Definition: 338.43461234428815
English_language: 319.852882327395
United_Kingdom: 313.98604609382176
International_Standard_Book_Number: 284.0803868582328
Country: 250.93384509694194
Geographic_coordinate_system: 234.02814100972452
England: 233.6676732723399
Japan: 231.42446809588768
Association_football: 230.94232313662533
Wikimedia_Commons: 229.64370286147533
Germany: 216.65665828527426
Europe: 214.7473503397607

```

Figure 34: Top 15 results of PageRank with article title and value.

## 5 Challenges

We faced several challenges while building Sombra.

- The largest challenge for this project was the switch to using Amazon Web Services. The benefits of this system are its high effectiveness once setup, and commonality in the industry. Most individuals in the technology industry has, does, or will use AWS at their company. However, setup for AWS is challenging, usually requiring multiple extensive courses to master.
- On the inverted index side, we struggled with garbage collection and available memory, especially with the small file sizes used in this project, which required many configuration changes. Hadoop works well on larger file sizes, but not the smaller files sizes. This was eventually remedied by upping the JVM memory limit on the heap for the garbage collector.
- One challenge we overcame quickly was the preprocessing steps for the inverted index. Wikiextractor, an open source technology, really saved the day with parsing and sanitization of the Wikipedia markup, something that usually would have taken a large amount of time to implement ourselves.

## 6 Conclusion

In summary, we built Sombra, a web portal capable of querying results a ranked version of the Simple English Wikipedia dataset. To support this web portal, we implemented a scalable Spark PageRank algorithm, and incorporated our earlier work on Winston, which ran an inverted index calculator. This allowed for queries to return word location by page, ranked by importance using the PageRank algorithm. Our implementation was successful, allowing for quick researching of Wikipedia entries in Simple English, a system that would be useful for anyone non fluent in the English language.