YouTube Data Analysis

Implementing PageRank in a MapReduce Programming Model

Sowmya Kudva and Evan Grill
Department of Computer Science & Engineering
University of Nevada, Reno
Reno, Nevada, USA

***Abstract*— Our group analyzed YouTube video metadata collected in 2007 and 2008 to find trends within the data. We utilized Apache Spark to parallelize the analysis. The main focus of our research was in calculating PageRank for a graph of videos with 8.5 million nodes and 170 million vertices. Modifying existing algorithms, we implemented a MapReduce compatible PageRank algorithm. Further analysis included analyzing most popular categories, and videos with high amounts of interactions.**

Keywords—MapReduce, Hadoop, Spark, RDD, PageRank, YouTube

I. Introduction

YouTube, a Google company, has over one billion users generating billions of views and uploading thousands of hours of video every day. YouTube's rapid growth has led to its success as a video streaming platform. With users uploading thousands of hours a day of new content a new problem arises, how to analyze video metadata to improve viewer's experience. Vast amounts of data is not a problem unique to YouTube. Similar companies like Amazon and eBay also have a "big data" problem.

The solution to the big data problem is better data analysis tools. Google's MapReduce programming model, introduced in 2004 [1], tackles the problem of how to parallelize data analysis at scale by implementing a simple two step methodology. First "map" existing data into small, focused tuples. Then "reduce" those tuples into usable data. MapReduce and Machine Learning represent the major approaches to the new field of big data.

Although Google never released their implementation of MapReduce, the programming model itself has been implemented and released as an open source project, Apache Hadoop [2]. Further work has adapted Hadoop to operate on Resilient Distributed Datasets (RDDs) in the form of Apache Spark [3]. This enhancement improves performance on iterative tasks which required repeated access to the same dataset.

Another Google analysis innovation came in the form of PageRank [4]. Developed by, and named for, Google founder Larry Page (with others), PageRank is a popular graph analysis tool that calculates weighted "ranks" for each node of the graph based on the relative weights of the nodes connected peers. The resulting ranks represent the relative influence each node has over its peers with higher values denoting higher influence.

II. Background/Motivation

A. Project Overview

Utilizing a dataset of early (2007-2008) YouTube metadata, provided by researchers at Simon Fraser University [5], our group generated a graph of related videos. This graph was extremely large, having 8.5 million nodes and 170 vertices. Our group utilized a MapReduce PageRank algorithm to calculate PageRanks for the entire dataset. Additionally, we used the MapReduce model to find popular categories and videos using other metrics.

B. Design Comparisons

When choosing our framework, we compared two different implementations of the MapReduce programming mode, Apache Hadoop and Apache Spark.

1) Apache Hadoop: Hadoop's MapReduce framework sits on top of the Hadoop Distributed File System (HDFS). Map and Reduce operations each read from files for their input and write to files for their output. This approach works well for large clusters and for single iteration MapReduce jobs, but seemed ill suited for our use case.

2) Apache Spark: Spark is similar to Hadoop in that it also implements the MapReduce programming model. However, a fundamental difference between the two is that Spark runs its MapReduce operations on RDDs. Initial ingest of data converts the source data (raw text, SQL, NoSQL, etc.) into an RDD kept in memory. From there operations are performed on the RDD entirely in memory. This vastly improves performance (10x) on workloads which feature iterative data access, like the PageRank algorithm that our project uses. For this reason, we chose Apache Spark as our platform of choice.

Apache Spark is built in Scala, a programming language neither of us were familiar with. However, because of its increasing role in data analysis, Apache has developed a Python library for Spark, named PySpark. PySpark is feature complete when compared to the Scala implementation and offers the ease of use of the Python Shell for prototyping.

III. Approach

A. Dataset Description

The dataset we utilized was obtained by researchers from Simon Fraser University [4] in 2007 and 2008. It is organized into rows representing individual videos and columns containing the metadata of the videos. A description of the individual columns can be found in Table 1.

Table 1. Dataset Layout

Column #	Metadata Description
1	Video ID
2	Uploader (Username)
3	Age (Interval between the founding of YouTube and the uploading of the video)
4	Category
5	Length (seconds)
6	Number of Views
7	Average Rating (1.0-5.0)
8	Number of Ratings
9	Number of Comments
10-29	Related Video IDs

The data is divided up into many individual files, first divided by the day on which the collection of data occurred, and then further divided into the depths of the scan. The first day the scan was performed was February 22, 2007. This one scan represents about 9% of the total dataset. The final scan was performed on July 27, 2008. Over that time, data on nearly 8.5 million videos was collected.

B. Analysis Steps

Before we began our work, we developed a workflow for the project. After completion we reevaluated that workflow and present it here.

1) Install Tools: The project required a number of tools be installed including Java, Apache Hadoop, Apache Spark, and Apache PySpark. Fortunately, these tools were available for Mac OS as simple install packages.

2) Configure Environment Variables: The documentation for our tools was very dense and did not outline the configuration process. Intially we struggled getting the tools running but after searching google, we found some short tutorials that highlighted the important environment variables that needed to be configured.

3) Download the YouTube Dataset: The dataset came as a large collection of zip files. We spent the better part of a day downloading, unzipping, and organizing the data.

4) Create a SparkContext Object in Code: The SparkContext object is the element of the code that interfaces with the Spark Framework. It implements all of the analysis methods, like Map, Reduce, and Filter and provides additional formatting methods like Sort to improve presentation.

5) Create an RDD from the Dataset: Once the SparkContext object is created, it can be used to create an RDD from the dataset. Spark offers a method to convert a text file, or files, to an RDD with each row of the text file being converted into a row in the RDD. The method even accepts wildcards (*) in the filename to quickly add a collection of files, like our dataset into a single RDD.

6) Develop a MapReduce Workflow: As this was our first time using MapReduce, we had to spend some time thinking and developing how we wanted to perform our analysis. Each individual analysis had a similar workflow but required different tuples to be passed to and from the Map and Reduce methods.

7) Output the Results: This was the easiest element of the process. Spark has Take and Collect methods that convert the RDD to a multi dimensional list for easy iteration and printing.

C. Preprocessing Required

The dataset included a number of rows which contained limited metadata. Some of the rows included all of the metadata, but no related video IDs. These were not particularly troublesome as they still represented graph nodes, just without any vertices. Additionally, they were useful for the other analyses performed. Other rows contained only a video ID. These rows initially caused errors in the analysis, but once identified we developed a filtering method to remove them prior to analysis.

The data also is processed initially as a single long string for each line of the dataset. Splitting the data was simple and was implemented as the second piece of the preprocessing pipeline. The entire working dataset was filtered and split prior to any analysis.

D. Analysis performed

We used the provided data set to find the following analyses: the highest ranked videos using the PageRank algorithm, the top categories by number of videos uploaded, the top categories by number of views, the highest rated videos, the top videos by number of comments, and the top users by number of videos uploaded.

IV. Evaluation

A. Experimental Setup

For this project we initially utilized consumer grade computing hardware for local execution. One nice feature of Spark is the ability to run execution nodes as threads on a single local machine. This allowed us to run our code locally without having to build and setup a cluster. Our personal machines were both Apple MacBook Pro models with similar specifications including 2.3/2.5 GHz 4-core processors, and 16 GB of RAM.

We later moved our code and data to Amazon's Elastic MapReduce and S3 platforms. On their platform we were able to build clusters of similar specification hardware, but in much greater size. For our in-class demonstration we ran the PageRank calculation on a cluster of 256 cores and 1 terabyte

of RAM. For comparison, PageRank took about 12 hours to run on the entire dataset but took only 5 minutes to run the same calculation on the EMR cluster.

B. Experimental Results

Our project was split into segments, PageRank and the other statistical analyses. We present our findings below.

1) PageRank: Our PageRank algorithm sucessfully calculated an approximated PageRank (we reduced the number of iterations from hundreds to 5 to improve execution time) for the entire 8.5 million videos. Unfortunately, we had trouble getting Spark to sort these results, so obtaining the highest ranked video wasn't possible. This shortcoming was difficult to troubleshoot because the same sorting code worked fine for all of our other analyses but produced a seemingly randomized order for our PageRank calculations. For this reason, the results are not included in this report.

2) Top Cateogries by Views: You can see the results for this analysis presented in Table 2. These categories are what we would expect to see when we hear "early YouTube." Music videos, comedy, and video blogs (vlogs) made up a big portion of the content space.

TABLE 2. TOP CATEGORIES BY VIEWS

Rank	Video Category	Views
1	Music	54,799,743,404
2	Entertainment	34,243,145,954
3	Comedy	21,148,944,742
4	Film & Animation	11,843,576,408
5	People & Blogs	10,870,720,833

*3) Top Cateogries by Number of Videos: T*he results for this analysis are presented in Table 3. Here we see a similar list to Table 2, but we've now seen sports take the number five spot. This will correlate with our top uploaders analysis later where sports companies appear multiple times.

TABLE 3. TOP CATEGORIES BY NUMBER OF VIDEOS

Rank	Video Category	Views
1	Music	2,028,807
2	Entertainment	1,892,569
3	Comedy	848,332
4	Film & Animation	716,764
5	Sports	704,799

4) Top Highly Rated Videos: The results for this analysis are presented in Table 4. This list is a little less obvious. We recognized the video in the number one spot, but none of the other videos stuck out as memorable from the period. Because we're working with individual videos now, you'll see that many of the older videos have since been removed, either by the user or by YouTube for terms of service or copyright violations.

TABLE 4. TOP HIGHLY RATED VIDEOS

Rank	Video Title*	Rating	# Ratings
1	Failpictures.net	5.0	67,621
2	mVbXeTHmrI4	5.0	2,492
3	50jJBhmsyuY	5.0	1,794
4	YOU CAN BANK …	5.0	1,471
5	Predicciones de …	5.0	1,035
6	Best Halo Vid Ever	5.0	1,008
7	ui0wD35KSzc	5.0	1,000
8	Scientology: WBM …	5.0	877
9	"LOS SUEÑOS Y …	5.0	626
10	EaVSFlolwZQ	5.0	623

* If the video has since been deleted, the Video ID is reported.

5) Top Commented Videos: The results for this analysis are presented in Table 5. In comparison to the previous list, we were much more familiar with the videos in this list. Two in particular, "Evolution of Dance" and "Chocolate Rain" stand out as viral videos of the period, each having hundreds of millions of views. We would venture to guess that the deleted videos were probably of equal stature for their time.

TABLE 5. TOP COMMENTED VIDEOS

Rank	Video Title*	# Comments
1	kHmvkRoEowc	259,683
2	guitar	258,825
3	uSVRuA8RpkU	244,552
4	-oHivXjiX w	210,672
5	cQ25-glGRzI	186,918
6	Evolution of Dance	182,303
7	Atheist	156,218
8	The $1000 YouTube Experiment	139,865
9	"Chocolate Rain" Original …	129,304
10	x9QNKB34cJo	123,221

* If the video has since been deleted, the Video ID is reported.

6) Top Uploaders: For this statistic, it was very interesting to see how quickly traditional media embraced YouTube. Our data is from the first couple of years of YouTube's life and already the top 10 uploaders are made up of top names in news, music publishing, and sports.

TABLE 6. TOP UPLOADERS

Rank	Username	# Videos
1	expertvillage	8,035
2	universalmusicgroup	4,519
3	CBS	3,418
4	TNAwrestling	1,707
5	AssociatedPress	1,582
6	sonybmg	1,281
7	NBA	1,216
8	ArtisanNewsService	1,167
9	AlJazeeraEnglish	1,159
10	CSPANJUNKIEdotORG	1,079

V. Discussion

A. Challenges

The first challenge that we encountered was getting the toolset installed on our personal machines. Many of our fellow groups had this same trouble. We found that all of the software necessary were available as packages in HomeBrew (https://brew.sh) which made for easy installation. Figure 1 demonstrates the command sequence required. Installation also required setting specific environment variables as indicated in the documentation for the individual tools.

```
> brew install hadoop
> brew install apache-spark
> pip install pyspark
```

Figure 1. Command sequence required to install the toolset for the project.

When we started to implement the PageRank algorithm, most documentation we found implemented the mapper to emit two different types of tuples. Although we tried, we weren't able to figure out how to get Spark to map and output two different tuple types. Luckily, we found an implementation that worked around this limitation and were able to convert it for use in our research.

As we discussed earlier, we found that some of the data lacked some of the metadata required for analysis. For some of this data, it simply meant that we had vertices in our graph with no edges. This didn't affect the performance of the analysis as the PageRank algorithm only looks at the connections between individual vertices, thus it never even sees these "rogue" vertices. Other entries in the data had no meta data and required removal. Initially we removed these by hand, but on a dataset this large, that became cumbersome. Evan wrote a simple Python script to filter these lines out, but Sowmya discovered a filter command built into Spark and the group implemented that as a piece of the preprocessing pipeline.

Once we had a working implementation of PageRank and our other analysis, we began testing on the entire dataset. This proved troublesome. Calculating PageRank for a dataset of this size on a single machine is very computationally intensive. The first solution we developed was to reduce the number of iterations of the algorithm, which did improve performance, but running on a four-core Intel mobile processor still took 12-15 hours to calculate. We reduced the size of the data to only the first day's scan, which reduced the size of the graph by over 90%. This reduced the running time of the PageRank analysis to a few minutes, which was much more manageable.

Moving the project over to EMR brought with it it's own challenges. While we had some experience with Amazon Web Services in the past, configuration and setup had shown to have a steep learning curve. That didn't actually seem to be a problem this time around. Instead, we ran into a limitation on the number and types of instances we could include in a cluster. The instances that were available to us in great number were many generations old and quite slow but were incredibly inexpensive. The nodes that were newer, and thus very fast and more expensive, were limited to single instances, or in some cases, locked completely. Amazon's customer service was very helpful though and was quick to increase our limits on the types of instance we were looking at. In the end, our total bill through Amazon was less than $50 for the semester.

Lastly, as discussed in the Evaluation section, we had an issue where we were unable to get Spark to properly sort our PageRank results. The same sorting code worked fine for every other analysis but did not work here. The results returned were in a seemingly random order and made it difficult to discuss. This issue was not resolved by the completion of the project.

VI. Related Work

The analysis of big datasets like the one we used is a very popular field of research as of late. Similar analysis is being performed on the vast amount of unstructured data produced by stock markets around the world. Using the MapReduce programming model, this data can be easily analyzed on a distributed cluster in much less time than was previously possible. From the results, stock market analysts can provide strong recommendations on stock price movement.

Another field of research improved using the MapReduce model include air travel analysis, Using data from millions of flights over decades, researchers can quickly calculate aggregate delays and find problem airlines and airports.

VII. Conclusion and Future Work

We found this project very useful. Neither of us had even heard of MapReduce prior to this class. Through our hands-on work with this project we were able to gain a strong grasp of the basic workflow when handling big data in a MapReduce environment. We also got a good amount of experience working with Amazon Web Services and their Elastic platforms. Both of these pieces of knowledge will be useful to us after graduation when we entire the job market.

While we were developing the project, we considered a number of improvements that could be made if we had more time or were to continue the project. The first such improvement would be to add a graphical user interface (GUI). This would allow an average user to perform on-the-fly queries of the data. We also considered adding significantly more query types. Nearly every permutation of metadata could be queried to obtain meaningful results.

Another improvement we considered was to integrate the YouTube API. This would for queries of current data, fetching of missing metadata like video titles, and direct loading of videos from query results.

VIII. Acknowledgement

The group would like to thank our professor, Dr. Feng Yan for his guidance and support during the semester. His broad coverage of MapReduce, its uses, and its implementations was invaluable to the completion of this project.

Sowmya would like to thank her family who has supported her while she pursues higher education and her team member

Evan, who has been a great guide in building this project into a workable product.

Evan would like to thank his wife, Katie. Without her support, this project would not have been possible.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI '04: 6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–150.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *IEEE Xplore: 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *HotCloud'10: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford, InfoLab, tech., 1999.

[5] X. Cheng, C. Dale, and J. Liu, "Dataset for 'Statistics and Social Network of YouTube Videos,'" *YouTube Dataset*, 2008. [Online]. Available: http://netsg.cs.sfu.ca/youtubedata/. [Accessed: 10-Feb-2019].