Trending HashTag Recommender

Group – 9

CSC 591-02 Data Intensive Computing

Aneesh Gupta
Chinmoy Jyoti Baruah
Sanket Shahane
05 December 2017

1. Abstract

Use of HashTags is a trend in today's communication. Use of proper hashtags help in popularizing your content in the right channels, and to the people of similar interests. However, many times, users are not aware of the popular hashtags relevant to their content. We have designed a system that can analyze a corpus of recent tweets and filter the most popular hashtags and the tweets corresponding to them. Finally, it will recommend popular hashtags to the users based on content similarity. Since it will analyze fresh tweets at a fixed interval, the recommendations will be relevant as well as trend. Our system is highly available and fault tolerant. To achieve these goals, we have used distributed computing principles in designing our system. Hadoop Distributed File System (HDFS) is used as a data storage mechanism, Apache Spark is used for all the computation jobs, and finally, Apache Solr is used to deliver trending hashtags to the front end in a highly available fashion.

Key Words: HashTag Recommendation, Twitter, HDFS, Spark, Solr, Distributed Computing.

2. Introduction

Hashtags are used in Twitter to target our content to the appropriate audience. Due to the limit of 140 characters on twitter, it is difficult to express yourself fully within those limitations. HashTags are a useful tool for concisely targeting your message to the appropriate audience. The target audiences are people with similar interests and objectives. There are some HashTags that are popular for some topics for example #Vote for politics. Business marketing departments are also using HashTags to reach their target audience. This makes it very crucial for them to use appropriate and trending HashTags, otherwise, their content might get lost in the 500 million tweets generated every day [1].

Studies like [2,3] suggest methods to recommend predefined HashTags based on the hidden topic in the tweet. Though these systems are useful for recommending HashTags and promoting the HashTag culture, these cannot suggest newly emerging and currently trending Hashtags. This is particularly important nowadays because of the rate at which new content emerges and goes viral in a matter of hours to few days.

In this paper, we propose an architecture of a highly available system which incorporates the current trends from twitter data. For this project, we decided to work on the archived twitter data from www.archive.org collected from 2015-03-22 to 2015-03-24 for experimentation and evaluation purposes.

The technology stack we have used is HDFS as a distributed data storage mechanism. Since the size of the data is such that it cannot be stored on a single machine, we had to provide a data-intensive solution using HDFS and Spark running on a cluster of 5 Virtual Computing Lab (VCL) nodes. The remainder of the paper is organized as follows: System Design and Architecture describes our infrastructure setup, Data

Flow pipeline illustrates the flow of the data through our system from ingestion to recommendation, we present the scalability performance of our system in the Results section, and finally in Conclusion and Future Works we discuss the goals that were achieved and provide some suggestions for future enhancements.

3. System Design and Architecture

3.1. Design

The twitter corpus we download from www.archive.org is in the form of JSON (JavaScript Object Notation) files, with every tweet being a JSON object. These JSON files first needed to be cleaned up for further processing by our system. This is because we required only two fields: text and hashtag, from it for further use in our system. The cleanup phase was essential to eliminate unnecessary metadata from the tweet files.

Initially, our approach was to perform the preprocessing cleanup on a single VM and then push only the cleaned files to the HDFS cluster. However, in this approach, the single VM turned out to be bottleneck both regarding processing speed as well as storage capacity. On a machine with Intel i5, 8GB RAM and 128GB SSD, the corpus did not fit, whereas, with a 1TB hard disk, the system would take more than 3 hours to complete the processing. Both limits were unacceptable to us, as we were designing our system with High Availability and Scaling properties in mind.

We, therefore, had to move the data cleaning phase to our cluster. The raw and unprocessed tweets would be uploaded as it is to the HDFS cluster, and then a Spark Job to clean the tweets would be invoked to perform the cleanup and put back the cleaned files back to HDFS in CSV (Comma Separated Values) format. We had planned to then perform aggregations and use KNN (K - nearest neighbors) algorithm with TF-IDF (frequency-inverse document frequency) to provide recommendations to the user. However, we found that Spark Machine Learning Library had no support for KNN algorithm and we had to devise an alternate way to provide the same functionality.

We decided to use Apache Solr, which is an open source enterprise search platform and provides full-text searching and real-time indexing capabilities [4]. Solr indexes the cleaned and aggregated tweet corpus and returned the relevant tweets and hashtags based on user inputted tweet. Solr internally uses BM25, a variation of TF-IF, for document scoring which helps it to return relevant search results [4].

The scoring function for BM25 [5].

$$Score(D,Q) = \sum_{i=1}^{i=n} IDF(q_i) * \frac{f(q_i,D) * (k_1+1)}{f(q_i,D) + k_1 * (1 - b + b * \frac{|D|}{avgdl})}$$

These design decisions led us to the final architecture of our system which is illustrated and described in the next section.

3.2. Architecture

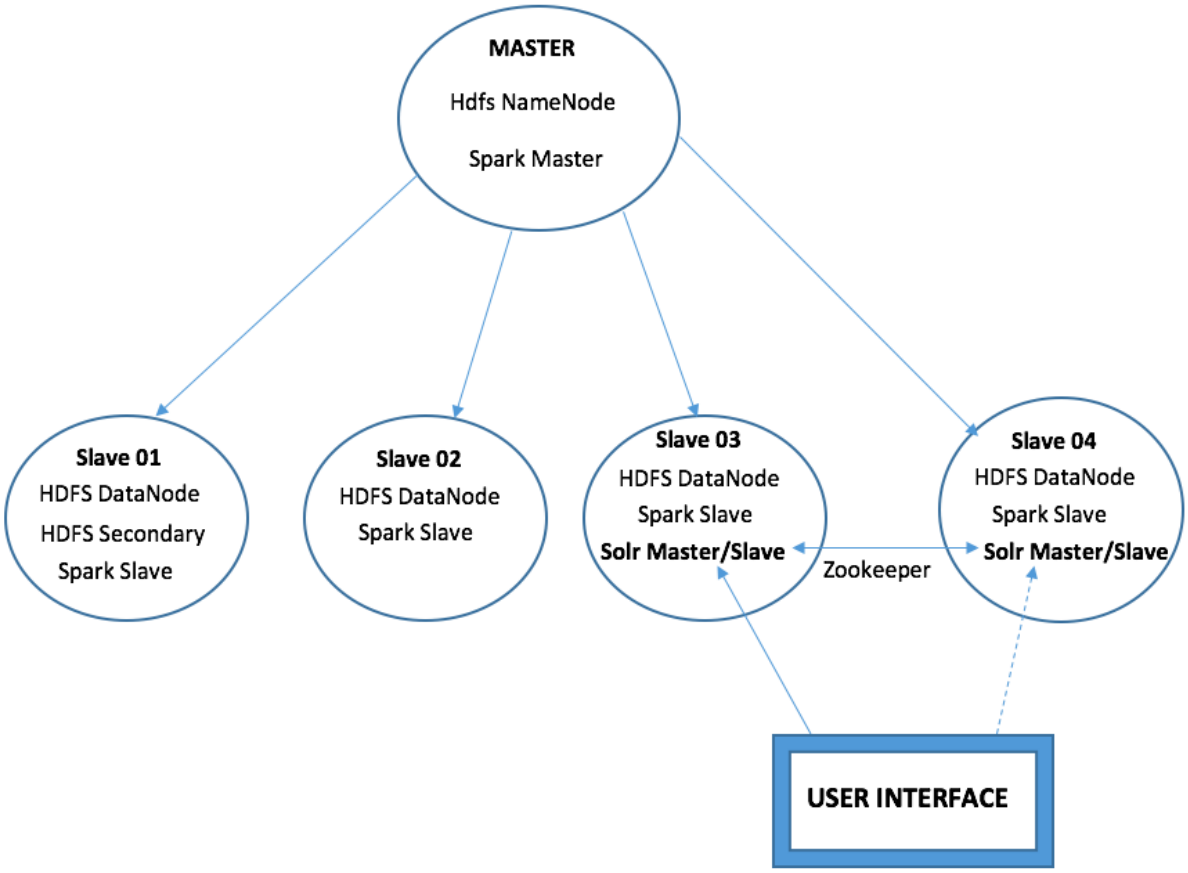

Figure 1: System Architecture Diagram with five node VCL cluster

Figure 1, illustrates the architecture of your system. We have five VCL nodes, each with 4GB RAM, 2 CPU cores and 37.5 GB hard disk space. All have Ubuntu 14.04 running on them. We have one Master node in the system. We have designated it as a Master node because the HDFS namenode as well as the Spark Master/Driver program run on it. All other four nodes are slave nodes, and each has an HDFS data node and Spark Worker running on them. The HDFS Secondary Namenode runs on Slave01 and checkpoints the master's state. We have Solr running on Slave03 and Slave04 as a Leader/follower pair. The Solr instances coordinate with the help of a Zookeeper instance. The User Interface would issue a query against Solr interface to retrieve the results and provide hashtag recommendations to the user based on his tweet.

3.3. High Availability

We took certain steps to ensure high availability and fault tolerance in our architecture. In case of HDFS, we have the secondary namenode running on a separate node, i.e., Slave01 and not on the same node as Master. The Secondary is responsible for checkpointing master states so that it can recover if it crashes or restarts. By moving the secondary to a separate node, we also ensure that in case the master node crashes the Namenode checkpoints will still be preserved on the Secondary, and it can recover from the same point once it comes back up again.

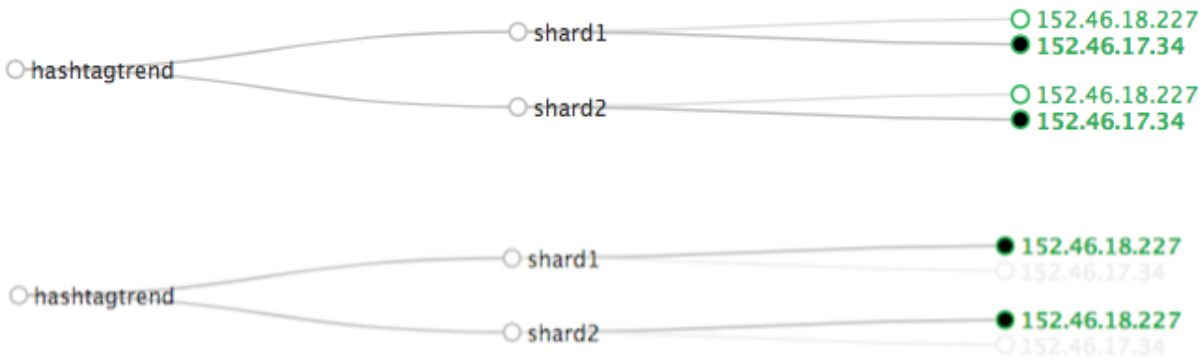

Figure 2: Failover in case of Solr instances

For Solr installation, we have the Leader instance running on Slave03 and Follower instance on Slave04 (or vice versa). The Solr instances use a Zookeeper instance for co-ordination between them. The follower is essentially a hot standby node with Solr shards replicated to it. In case the Leader goes down, it will take over the role of Leader. The query or updates in this design can be processed by either Leader or Follower. If an update query comes to a Follower, he will forward it to the master who will perform the necessary update and sends the change to all the followers (Here we have one follower for now). The failover for Solr cluster is illustrated in Figure 2 as captured from the Solr UI. When the leader is shutdown manually, the follower takes over and becomes the new leader.

Spark for now has no High Availability features in our design. This is because we are using it as a data processing engine for relatively short jobs (~10-15 min). In case a job fails we could always restart it later.

3.4. Data Flow Pipeline

Figure 3 describes the steps involved as the data flows through our system starting from JSON files from the twitter corpus and how it ends up facilitating the Hashtag recommendation to the user. The twitter corpus from archive.org is ingested and distributed onto the Hadoop cluster. Then in Step 1, we run a Spark Cleaning Job on the JSON files to convert them too much smaller CSV files having just two columns Text and Hashtag. The CSV is written back to HDFS in Step 2. In Step 3, we run another Spark Job for aggregation on hashtags to generate Trend numbers. The trend is a score for each hashtag based on how popular it is in the whole corpus. In Step 4, a CSV having three columns Text, HashTag, Trend is written back to HDFS. In Step 5, the CSV with aggregated trend numbers is fetched into Solr and indexed by Text and Hashtag fields. Once indexed it is available for user query through the User Interface. In Step 6, User Interface generates a query request to Solr based on the new tweet entered by the user. In Step 7, Solr returns the recommended hashtags to the user after searching for the new tweet's content in its indexed database and then sorting the relevant search results (i.e., hashtags) by the aggregated trend numbers (popularity score). Here, Steps 1-5 are a one time process while Steps 6-7 (user queries) may be repeated multiple times until next system update with the latest tweet corpus.

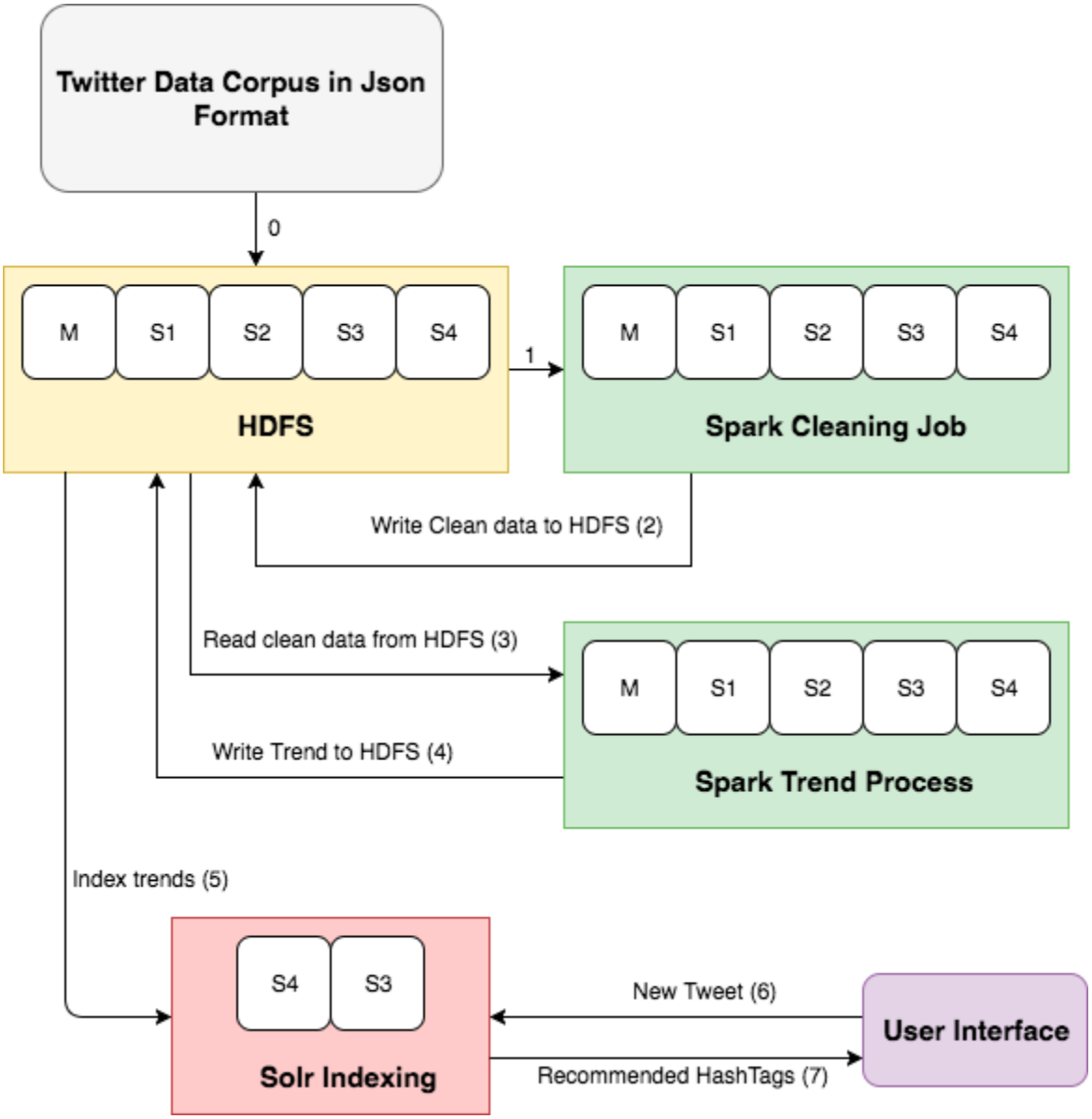

Figure 3: Data Flow Pipeline through the system

4. Evaluation

We evaluated the scalability of our system by testing out the Spark processing jobs. We kept the size of the dataset constant and varied the number of Spark workers from 1 to 5. Increasing the worker nodes from 1 to 2 gave us almost linear scalability of 0.96 on the time dimension. However, adding more worker nodes to the cluster does not maintain the same scalability, it goes down to 0.8, it shows an exponential decrease and eventually flattens out. Figure 4 shows the plot of some spark worker nodes vs. time taken for spark job to complete in absolute terms of seconds while Figure 4 shows the scalability factor achieved as we add each worker node to the cluster. We focus on Spark Jobs since that is the most time-consuming part of our pipeline. Solr indexing additionally takes nine mins each time.

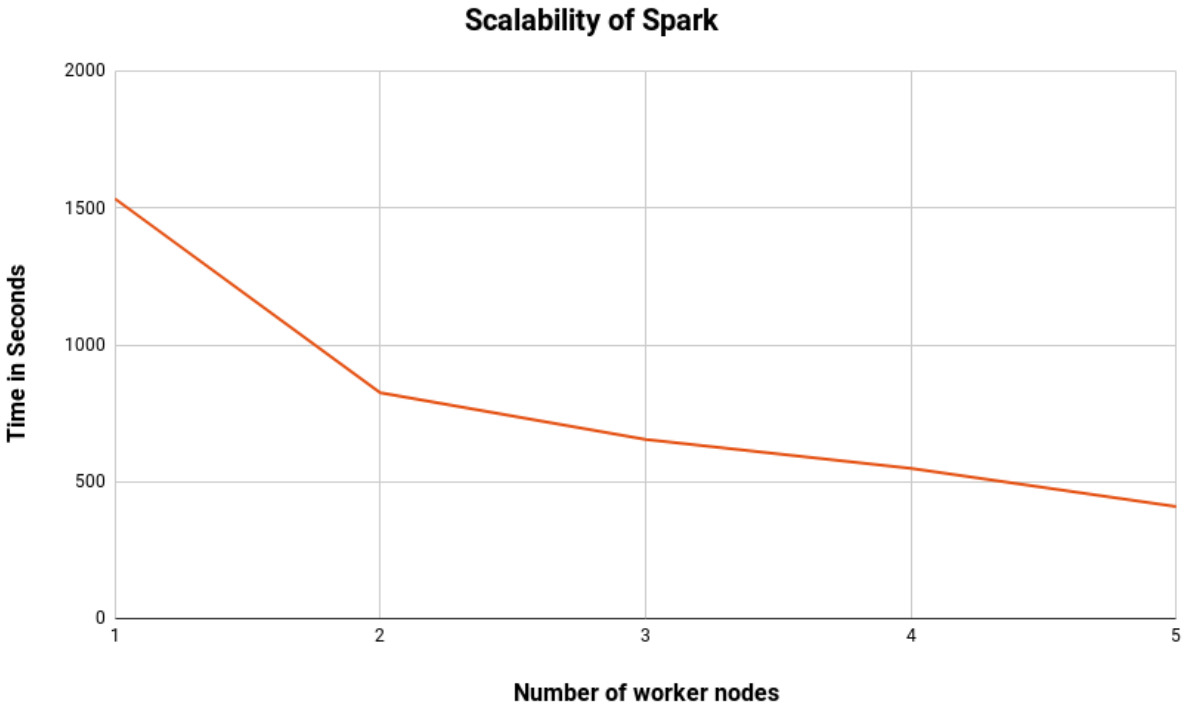

Figure 4: Scaling properties of Spark with increasing number of nodes

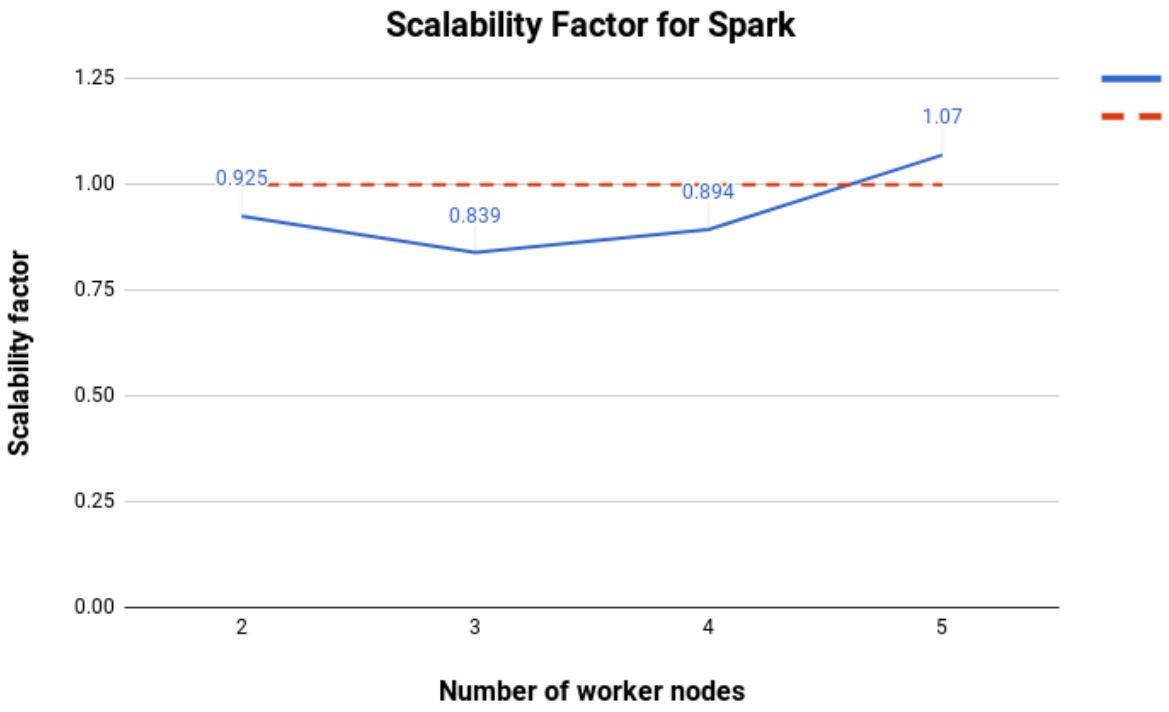

Figure 5: Variation of Scalability Factor on incremental addition of nodes

The most surprising result in Figure 5 is the increase in the scalability factor when adding the 5th Node. It is 1.07. After investigating the results, we found out that when the 5th spark worker is added, it gets added on the same node on which the Spark Driver is running and at this moment decreasing the communication overhead of the overall system and at the same time it also shares the workload of the remaining workers.

5. Related Work

A method using k-Nearest Neighbors (kNN) and Naive Bayes proposed by [6]. Their idea is to use inverse document frequency (IDF) and get the top 3 words in the tweet with meaningful information. kNN is used to find the score of each HashTag based on the probability estimated from Naive Bayes. This method relies on kNN algorithm which does not scale well with increasing data size. One property of kNN is that it is a lazy learner, i.e., it requires all the input data to be present in the memory to make predictions for a new incoming tweet. Our method, however, does not have such requirements due to use of Solr instead of plane kNN searching.

Recommending hashtags based on similar users or similar tweets was proposed by [7]. Based on the TF-IDF vectorizing, they select the top 'n' users similar to the current user using cosine similarity. New hashtags are added to the user's list for use. However, in this approach, the current trends are not analyzed and does not recommend recent trending HashTags. Also, our system does not rely on the user's behavior.

6. Conclusion

The Hashtag Recommender System demonstrates all qualities that any recommendation system should possess. We were able to process 54 GB(2.5 days) of data in a distributed manner and recommend the most trending hashtag for the searched word/phrase.

Our system provides fault tolerance by using secondary/backup nodes for HDFS and Apache Solr. The time taken for completion of the Spark job is approximately 10-12 mins, and Solr indexing took approximately ten mins on the whole data.

For future work, we can use Apache Kafka to push in streaming data into Apache Spark and store it in the HDFS cluster after processing it. This system could be built as a browser plugin which could be used to recommend trending hashtags on any social media sites. Also, hashtag analytics can be built on the existing system by using Banana - which is data visualization which uses Solr for data analysis and display.

7. Acknowledgements

We would like to thank Dr. Vincent Freeh and Liang Dong for their guidance and support in developing this project.

8. References

[1] Twitter Usage Statistics, http://www.internetlivestats.com/twitter-statistics/
[2] Sriram B, Fuhry D, Demir E, Ferhatosmanoglu H, Demirbas M. *Short text classification in twitter to improve information filtering*. In SIGIR'10. New York: ACM. 2010.
[3] Garcia Esparza S, O'Mahony MP, Smyth B. *Towards tagging and categorization for micro-blogs.* 2010.
[4] Apache Solr Wiki, https://wiki.apache.org/solr
[5] BM25 Wikipedia Page, https://en.wikipedia.org/wiki/Okapi_BM25
[6] Dovgopol R, Nohelty M. *Twitter hash tag recommendation.* CoRR; 2015. vol. abs/1502.00094.

[7] Kywe SM, Hoang TA, Lim EP, Zhu F. *On recommending hashtags in twitter networks.* In Social Informatics. Berlin: Springer; 2012. p. 337–50.
[8] Hadoop, http://hadoop.apache.org/
[9] HDFS, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[10] Apache Spark, https://spark.apache.org/
[11] Apache Zookeeper, https://zookeeper.apache.org/