

# Twitter Search Engine

## Overview

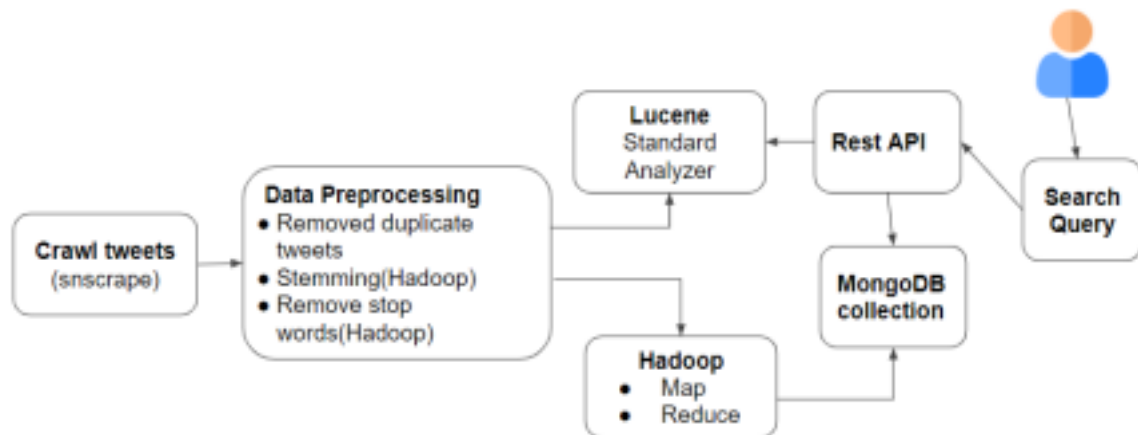
Twitter is a social media platform that has been an excellent platform for data scientists to generate thousands and thousands of Gigabytes of data every day through millions of tweets generated by users. It gives us the freedom to stream and retrieve the data using an official API. Using the free official API, we can retrieve tweets up to seven days back. These tweets give an insight into the popular moods of the users which is an essential insight to many big corporations which use Twitter to work on their marketing strategies.

In our project, we aim to develop an indexer that searches tweets scraped from Twitter efficiently and stores them in a .csv file. Using Snscape we have scraped tweets based on topics such as social, covid, sports, news, trending, movie, etc. Further, we use Lucene and Hadoop respectively to index and search on the basis of the queries entered by users in the UI. The relevant tweets are displayed according to the ranking algorithm on the web interface.

This project consists of the following parts:

1. Scraping tweets (Crawling) (Part A)
2. Indexing the crawled data using Lucene and Hadoop.
3. Searching relevant tweets based on the query from the user.
4. Display relevant results on the web UI.

### a. Architecture



- We used Snscape Python Wrapper to get the tweets using a keyword. Using Snscape, we can scrape millions of tweets. We scraped on tweets on keywords related to various trending topics such as - movies, songs, news, sports, Olympics etc.. We pulled all the attributes from the Tweet Object that we were able to scrape.
- We crawled tweets using Location to get tweets from that particular area. We also used Geocode (Longitude and Latitude) to get the tweets.
- We also used time periods to scrape tweets to avoid duplicates while crawling multiple times on the same keyword.

- The tweets that are geo-tagged are returned. But tweets that are not geo-tagged are also returned if the user has a location set in their profile.

- For Lucene:

- It first tokenizes the input given to it.
- We use the Standard Analyzer to remove stop words, do stemming (to improve accuracy of our model) and make all words lowercase.
- It also provides an automatic ranking based on the Lucene version. -

- For Hadoop:

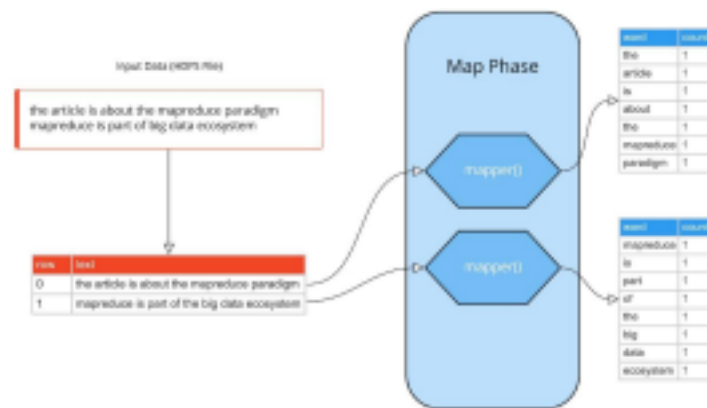
- We first stemmed the crawled data so that words that are derived from one another can be mapped to a central word (root word) or symbol, especially if they have the same core meaning.
- Then the inverted indices obtained at the output of Hadoop are stored in MongoDB.
- We have used the tf-idf ranking algorithm to rank the documents.
- Once the user enters a query on the UI search engine, the relevant documents are pulled from MongoDB through the rest API and displayed in results (according to the ranking).

- For UI

- We are using Angular to build a web application. We show the data in simple table form which contains rank, user\_handle, tweet content, followers of the user, location and date whenever a query is passed.
- The use of Spring-boot makes the API calls easier from the frontend to backend to get the relevant results.
- We have mapped the url by which we distinguish whether the user has chosen lucene or hadoop, a java function thus fetches from the respective lucene or hadoop class.
- The json data can be seen on localhost:8080, we get this data by the http get request from the frontend and display the results.

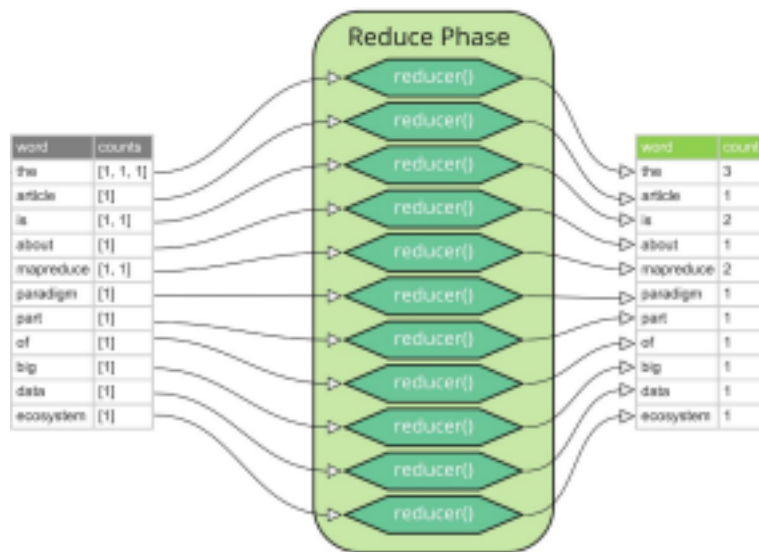
## **b) Hadoop indexing**

We mainly used the MapReduce function of Hadoop. It is a good way to gather all the values of the same key together for further usage. The keys here are the words of the texts or locations of the tweets, and the values are the document IDs, positions of the words in a tweet and so on. In our project, we used one of our computers to implement MapReduce.



## Implementation of Hadoop

1. Our indexing procedure has one MapReduce process. For the map phase, we have two steps: The first one is to analyze the input file which is in a JSON format. 2. Firstly, we analyze the input file which is in JSON format and extract the tweetId and content of the tweet. Each tweet is a JSON string.
3. We emit Word,TweetId for every word in the content of the tweet as the output of the mapper.
4. In the reducer phase, we created a HashMap. In the reducer, we iterate through all the TweetId available for a word and add all of them together. The final output of the reducer phase is a word with its frequency along with TweetId's associated with the word.
5. Finally, we combine the total frequency, document frequency, and index together as the value. They are presented in the following format:  
 "TF:TotalFrequency;DF:DocId1:DocFrequency1;DocId2:DocFrequency2;.....;I:DocId1:Position2; DocId2:Position2.....;" The key and this long value string are subsequently outputted to a file for further processing.



## Hadoop created index to do Ranking

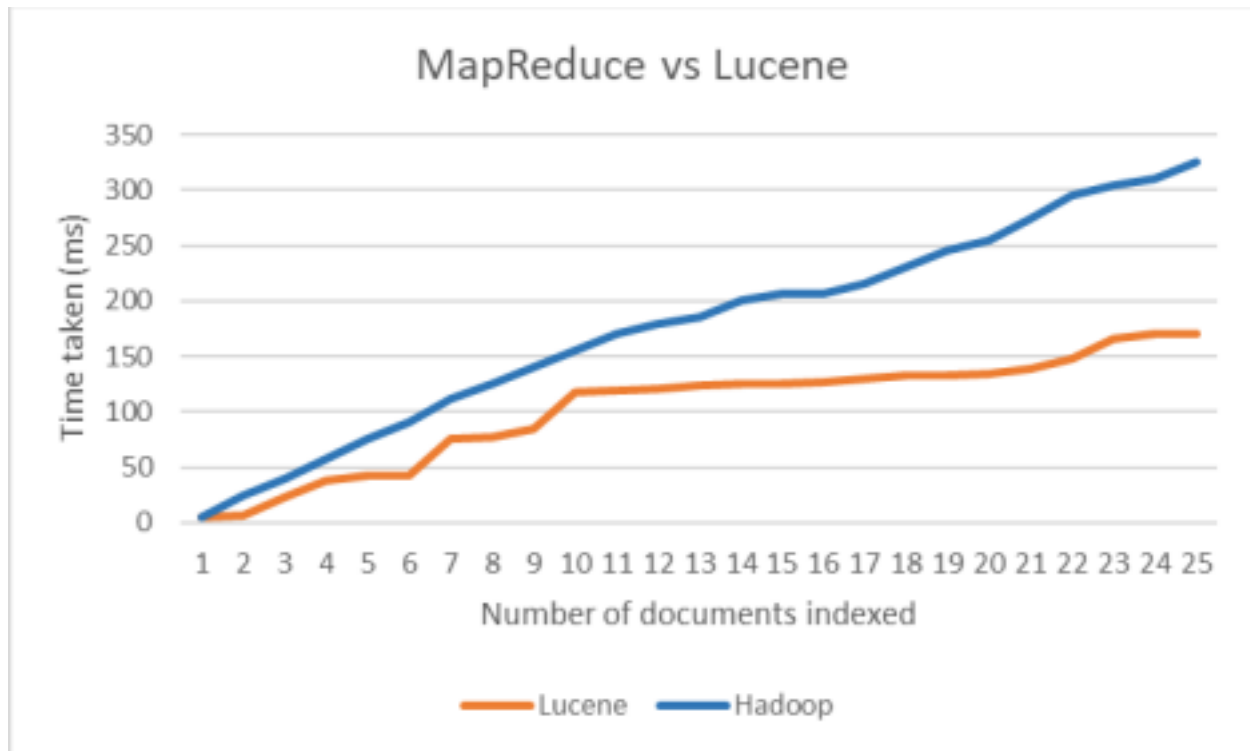
The TF-IDF scores are used to rank the tweets in the following way:

1. First we look for the tweetId.
2. For each tweetId we calculate a TF-IDF score using the following formula:  

$$\text{TF-IDF}_{\text{tweet\_id}, Q} = \text{tf-idf}_{q1, \text{tweet\_id}} + \text{tf-idf}_{q2, \text{tweet\_id}} + \dots + \text{tf-idf}_{qn, \text{tweet\_id}}$$
3. Then the top 10 tweetIds with the highest TF-IDF are selected from the values calculated for each tweetId.

## Lucene Indexing and Ranking

1. Firstly, the QueryParser object is initialized using an analyzer that contains the index and name on which this query is to be run.
2. Then, we create an object of IndexSearcher. Then we create a Lucene directory that points to the location where indexing is to be done. Finally, we initialized the IndexSearcher object that was created with the index directory.
3. We create Query objects by parsing search expressions through QueryParser and perform a search by calling the IndexSearcher.search() method.
4. In the query phase, we decided to rank the results by the "content" value of the user who published this tweet. At first, we use the IndexSearcher to search for the index and record the results of some given words. The results contained information such as the tweet content, published location and followers and so on.
5. The ranked tweets are then displayed on the UI.



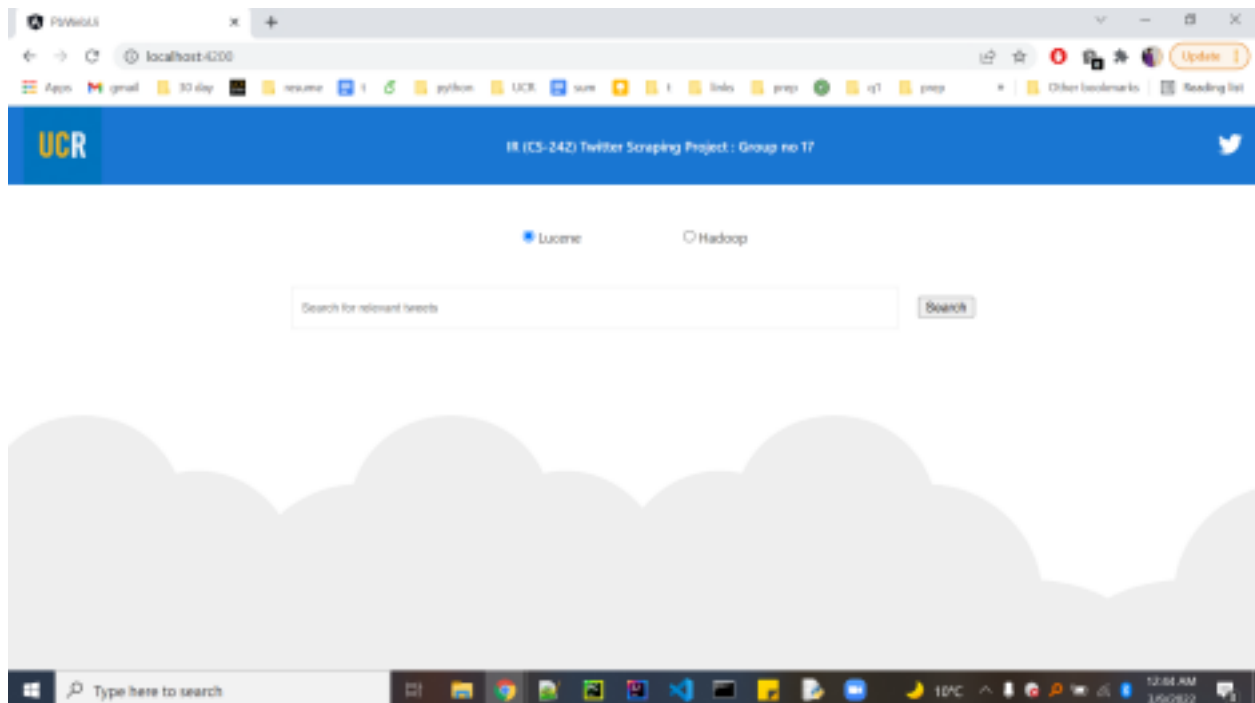
The above graph shows the runtime of Lucene and Hadoop. We can find that Lucene is efficient compared to Hadoop for indexing.

## Web Application:

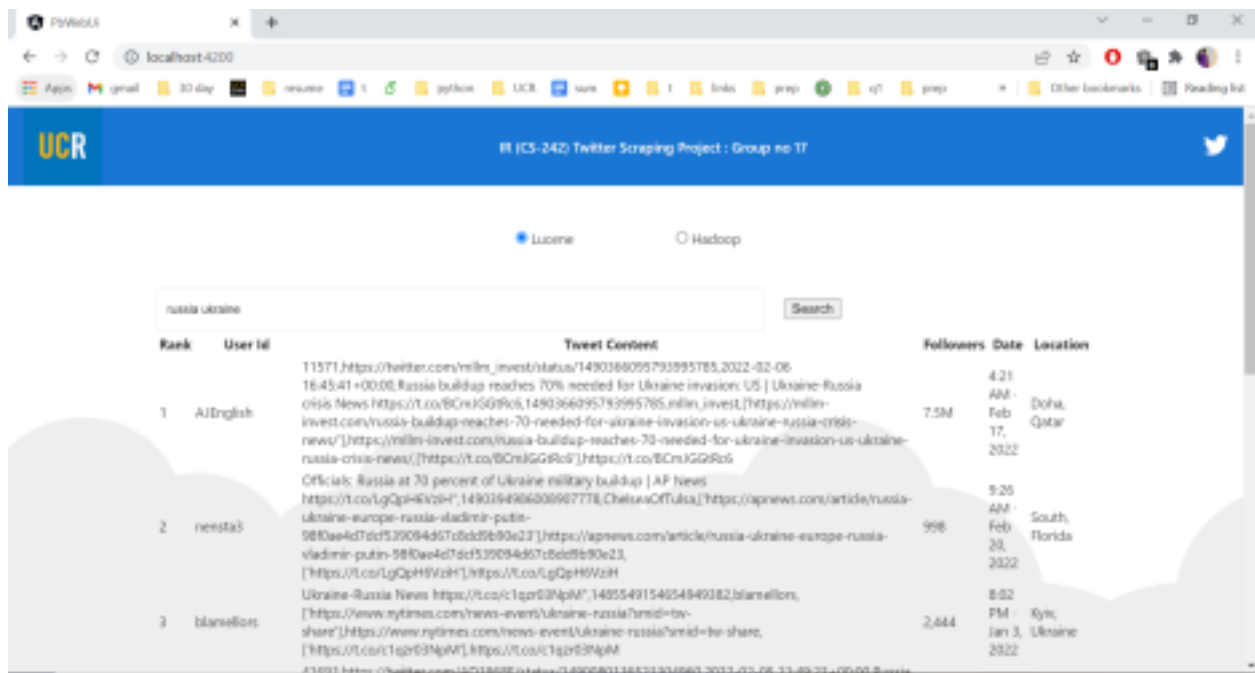
- We used Angular CLI and Spring IO to develop and publish our web application. We used the Spring framework for Rest Apis to get data and show it on the web application using Angular framework as the front end.
- We have used maven and added the dependencies of lucene in pom.xml which lets us run the application.
- The use of simple css and html in angular helped us create the web interface that the user sees for the first time (see Screenshot 1)
- After selecting the indexing type between 'Lucene' and 'Hadoop', the user would enter the query in the search-box and search it using the 'Search' button.
- The query would be passed via the typescript code in frontend via a get API call to localhost:8080 port (or any port where the data is hosted by spring-boot), the data would be a json array that we get from the query file written for each indexing type.

## Screenshots:

1. In the below image we have shown the front page of our Search Engine. The title bar displays our group name and simple link to twitter from the icon. Further, we can either query by "Lucene" indexing or "Hadoop" indexing after typing the query in the search bar and clicking the 'Search' button.

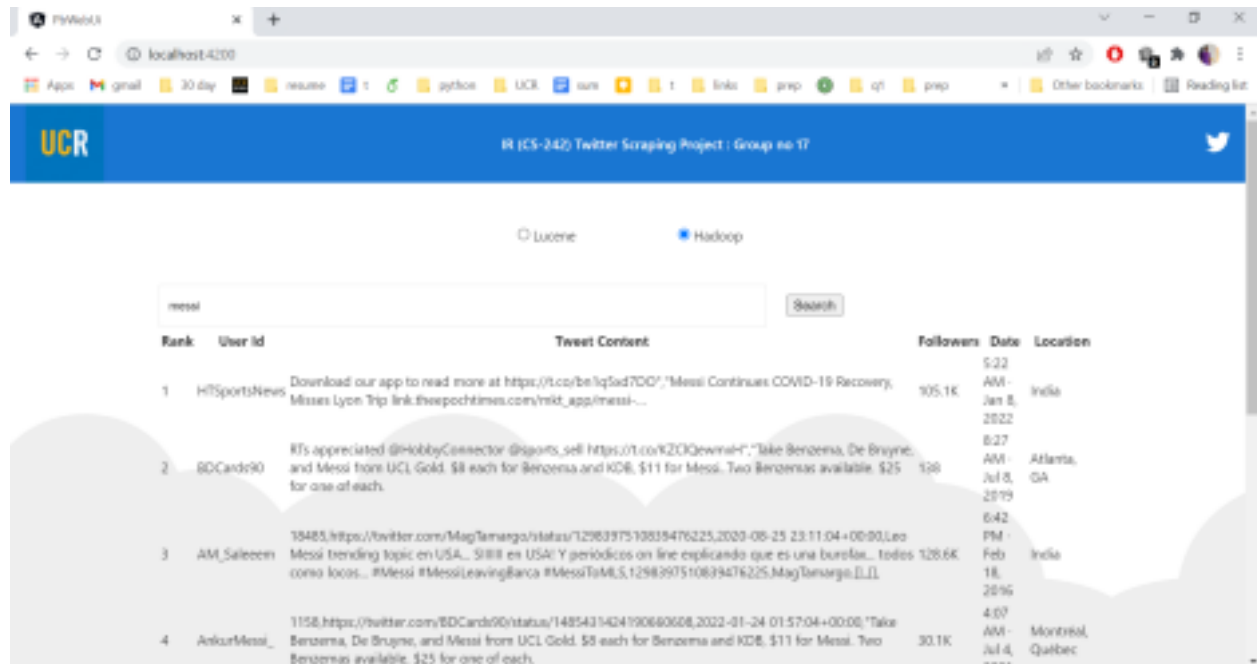


2. In the below image we show results on query 'Russia Ukraine' which shows the **rank**, **tweet content**, **followers**, **date** as well as **location** of respective search result from the indexed Lucene data.



3. In the below image we show results on query 'messi' which shows the rank, tweet content, followers date as well as location of respective search result from the indexed

Hadoop data.



Rank	User Id	Tweet Content	Followers	Date	Location
1	HTSportsNews	Download our app to read more at <a href="https://t.co/be1q5ad700/">https://t.co/be1q5ad700/</a> , "Messi Continues COVID-19 Recovery, Misses Lyon Trip <a href="http://link.thepointtimes.com/mkt_app/messi...">link.thepointtimes.com/mkt_app/messi...</a>	105.1K	5:22 AM - Jan 8, 2022	India
2	BDCard90	RTs appreciated @HobbyConnector @isports, sell <a href="https://t.co/9ZCQewmH">https://t.co/9ZCQewmH</a> , "Take Benzema, De Bruyne, and Messi from UCL Gold. \$8 each for Benzema and KDB, \$11 for Messi. Two Benzenas available. \$25 for one of each.	138	6:27 AM - Jul 8, 2019	Atlanta, GA
3	AM_Saleem	18485, <a href="https://twitter.com/MagTamarago/status/1296397510839476225">https://twitter.com/MagTamarago/status/1296397510839476225</a> , 2020-08-25 23:11:04+00:00, Leo Messi trending topic on USA... Still en USA: Y periódicos on line explicando que es una burlesca... todos 128.6K como locos... #Messi #MessiLeavingBarca #MessiToMLS, 1298397510839476225, MagTamarago, [L]	128.6K	6:42 PM - Feb 18, 2016	India
4	AnkurMessi	11158, <a href="https://twitter.com/BDCard90/status/1485431424190660608">https://twitter.com/BDCard90/status/1485431424190660608</a> , 2022-01-24 01:57:04+00:00, "Take Benzema, De Bruyne, and Messi from UCL Gold. \$8 each for Benzema and KDB, \$11 for Messi. Two Benzenas available. \$25 for one of each.	30.1K	4:07 AM - Jul 4, 2022	Montreal, Quebec

## Limitations of the system

1. Indexing of contents of all tweets may be difficult as some of the tweets crawled contain text in languages other than English. However, our code deals only with English text.
2. The code uses a single thread to extract data that lacks parallelism.
3. Since Hadoop is stored in a single disk, this doesn't guarantee efficient performance.
4. We have limited data with respect to all the topics so this project just depicts the miniature version of a full-scale version.

## Obstacles and Solutions

1. Getting our code to work on Hadoop was a major challenge on the bolt server.
2. It was difficult to install and set up Hadoop on our local machines.
3. The format of output files had to be managed to display it in the user interface.

## Deployment Instructions

In order to deploy the code, you need to have Lucene and Hadoop installed on your system

1. The data crawled with snsrape is copied to the bolt server in the 'data' directory.
2. The Lucene indexing file is run by
  - a. `javac twitterindex.java`

- b. java twitterindex
- 3. After the Lucene indexing runs the indexed files are written in the '**result**' directory.
- 4. After the Hadoop indexing runs the indexed files are written in the '**output**' directory.
- 5. This data can be stored in any database and made to connect with the frontend.
- 6. Run the following files to run the complete project :
  - a. Install spring-boot with web dependencies, setup maven as well in your machine.
  - b. Run the complete DemoApplication project. (This would show the data on the mapped API on the particular url)
  - c. Run 'ng serve' command after loading the angular project '**web\_app**' by installing the angular dependencies.

## References

- 1. <https://Lucene.apache.org/>
- 2. [https://Hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://Hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)