University of California, Riverside

CS 242 : Information Retrieval & Web Search

Project Report- Part B

Lokesh Koppaka, Abhilash Sunkam, Vishal Lella
{lkopp001, asunk001, vlell001}@ucr.edu

Overview:

The Internet plays a major role in providing a substantial amount of information about all sectors. A huge amount of data is available on the Internet which can be used to get the required information if deployed properly. For every other aspect in day to day to life people rely on this information. Web information can be of various sort like videos, pictures or textual info like audits, remarks and so forth. Learning and gaining knowledge from the web, can be utilized to get profitable productivity thus prospering the exchange.

We see that we have enough information about technologies, tech conferences and various other things related to tech. We use the Twitter data to make a search engine, where the user can search for any tech term and get to know what's currently happening around him.

Architecture:

The architecture of the search engine includes many components which together enable an efficient functioning of the search engine. We used twitter streaming API, to crawl our data. This data was then indexed using Lucene, which was analyzed in the first part. In this part, we use Hadoop, Map-Reduce to index the data. The search engine retrieves the results using the indexing results from both Lucene as well as Hadoop based on our choice. The data is stored in a NoSql database - MongoDB. The indexed results are stored in big JSON which is loaded in memory before doing the query search. We expose two REST APIs on the server so that the client can make a request for searching using particular indexing method (Lucene or Hadoop). The backend APIs are developed using Jersey, a JAVA framework for developing RESTful web services. We used Tomcat for running our server. The client side is done using HTML, CSS, JQuery and we use AJAX call for making a request from a client to server. We show the search results i.e tweets and other details related to the tweets. We also show the locations from where the tweets were originated on a map. Using this search engine, we get to know the current information, events, conferences etc related to tech. The following figure (Fig-1) shows the architecture used in building the search engine.

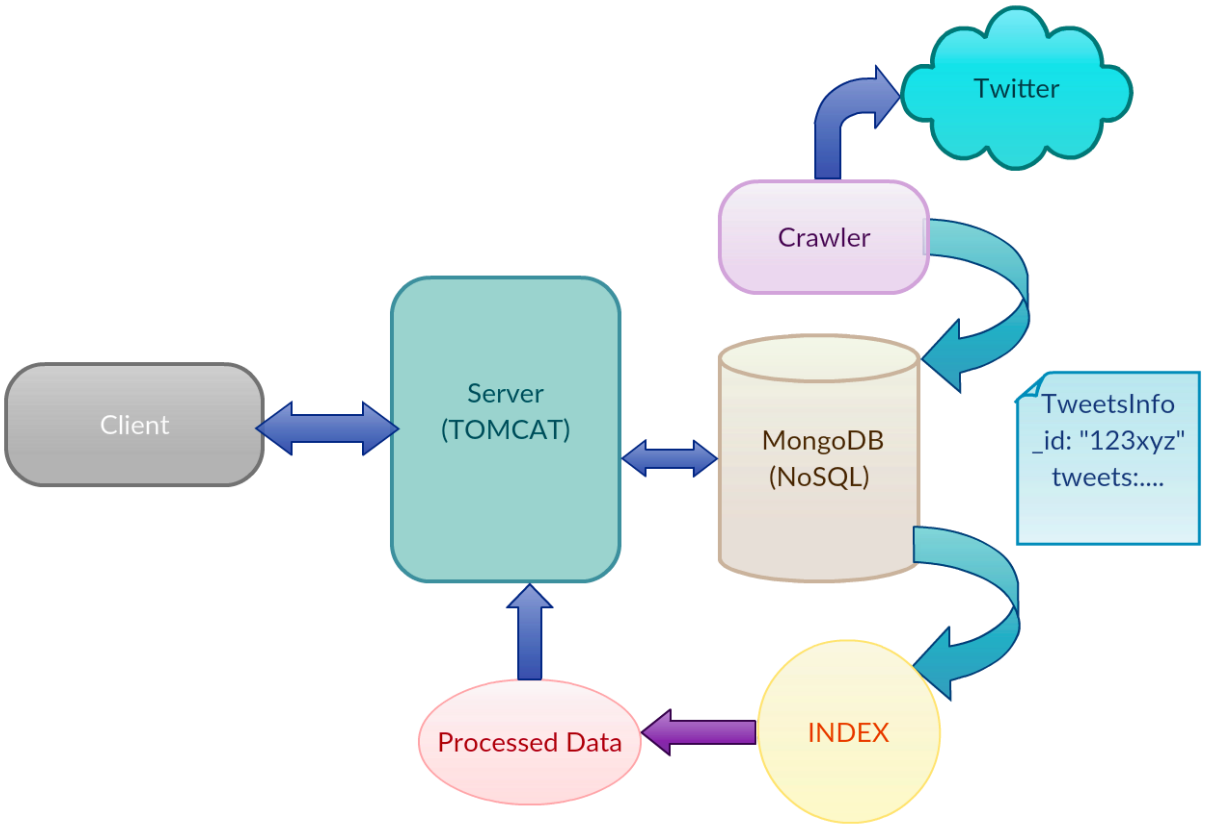

Fig 1: Architecture for building the search engine

Indexing and Searching using Hadoop:

Once the tweets are crawled it is essential to perform indexing for fast retrieval of data. Information Retrieval Systems use the concept of Inverted Index as part of indexing the documents. The use of traditional single machine programs for generating indexing (in this case inverted index) for such huge collection of data with polynomial time complexity is highly inefficient and non-scalable. Hadoop MapReduce programs essentially make its impact in these scenarios by supporting a framework for storing and processing the data in a distributed environment. In this project we have exploited the advantage of HDFS for distributed storing, distributed processing of Map Reduce program to generate an Inverted Index.

Map Reduce Program to generated Inverted Index:

Map Reduce Program is used to generate the Inverted Index. The Mapper takes the responsibility of reading the tweet, hashTag attributes of each tweet instance and generating a key-value pair as <word, {“tf”:3, docId:”123xyz”}> whereas the reducer accumulates the results per word and generates respective posting list.

The Mapper performs the following responsibilities
1. Read tweets, perform preprocessing like stopword elimination, special characters removal, case insensitivity
2. Computes term frequency for each word in the tweet
3. For each word in the tweet outputs key values pairs where key being the word and value being the JSON holding the documentId and the term frequency tf

The Reducer performs the following responsibilities
1. Receives input as key, values generated by the above mapper.
2. Accumulates the results per key(i.e word) to a list(i.e posting list) in decreasing order of the term frequency.


```
{"blockchain":[{"tf":1,"docId":"5c84abb8f8bd5e3078a7ef97"},{"tf":1,"docId":"5c84abc1f8bd5e3078a86f27"}]}
```


Use of hadoop-created index to do the ranking:

The Generated Inverted index is maintained in-memory in the server for fast retrieval of the tweets from MongoDB. When the server receives GET request from the Client, the server reads the search string, splits them into words and uses the in-memory inverted index to get the posting list for the respective words. Once the posting list is retrieved it performs ranking on the retrieved list using the Ranking function tf * idf, tf is retrieved from the posting list itself whereas the idf is computed as $\log_{10}$(total Tweets/ df) where df = the length of the posting list. These posting lists are then maintained in a map with the key being the document-id and value being the ranking score and the map is sorted by the scores. For all the top-ranked documents, the server retrieves the complete tweet instance from the MongoDB using the documentId and returns them as JSON response to the client.

Indexing and Searching using Lucene:

The indexed files generated by Lucene in phase one is used by the server to generate search results. The server reads in the search string, splits them into words and passes the words to Lucene Search which in turn looks into the index file generated by Lucene and returns the relevant documents according to the default ranking function used by Lucene. The returned results from lucene are then returned as JSON response to the client.

API Description:

Base_URL: /InitiateSearch/webapi/search

1. Hadoop:
   URL : Base URL/hadoop/{query}
   Response Format : application/json

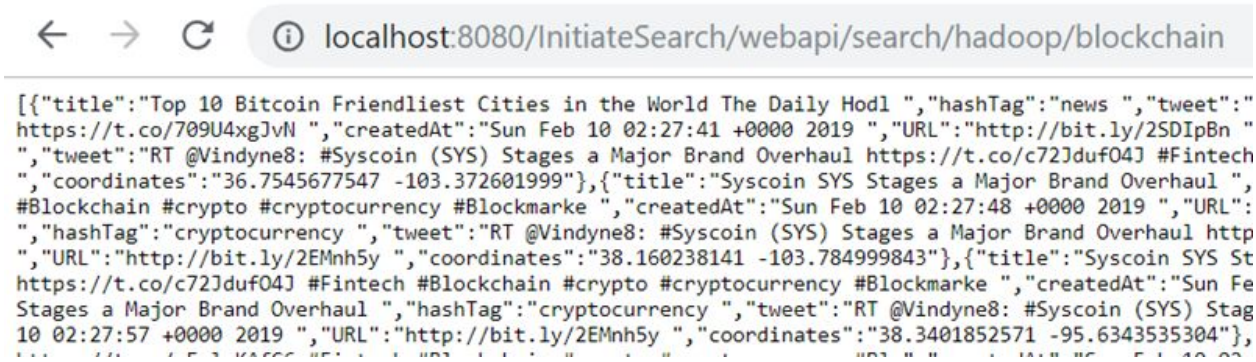

Fig 2: Sample Hadoop Search API Response

2. Lucene:
   URL : Base URL/lucene/{query}
   Response Format : application/json

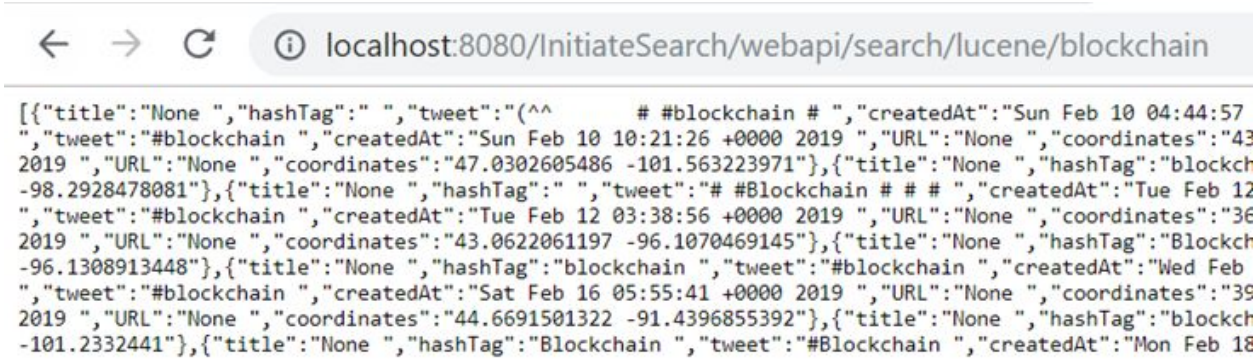

Fig 3: Sample Lucene Search API Response

User Interface:

The web interface has a search bar and a drop down where the user can enter the query word and select either Lucene / Hadoop index in the dropdown and click the search button.

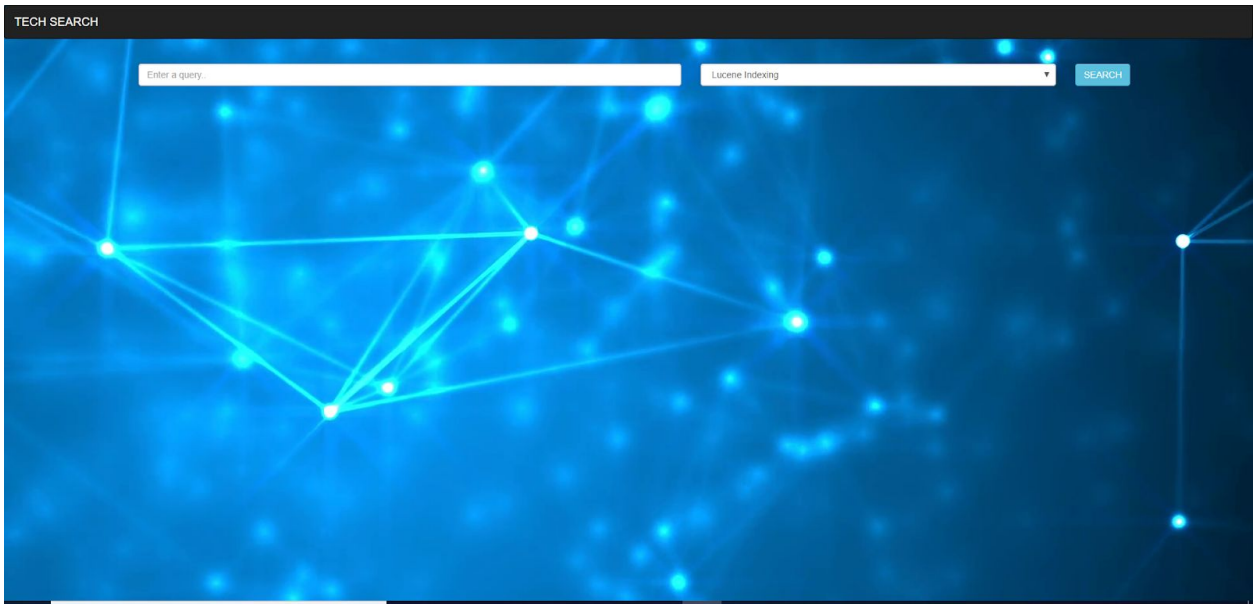

Fig 4: Main page of the tech search page

On searching for the query word, we populate the results retrieved from the server in a table (jQuery Datatable). We also use leaflet maps library to mark the coordinates of retrieved tweets on the map. The column values with URL are returned with anchor tags. This will enable users to navigate to the page quickly and get to know the detailed information.

Another interesting feature in jQuery Datatable is the ability to search for a text within the results in the table easily. We can also sort the Date column in order to get the recent tweets.

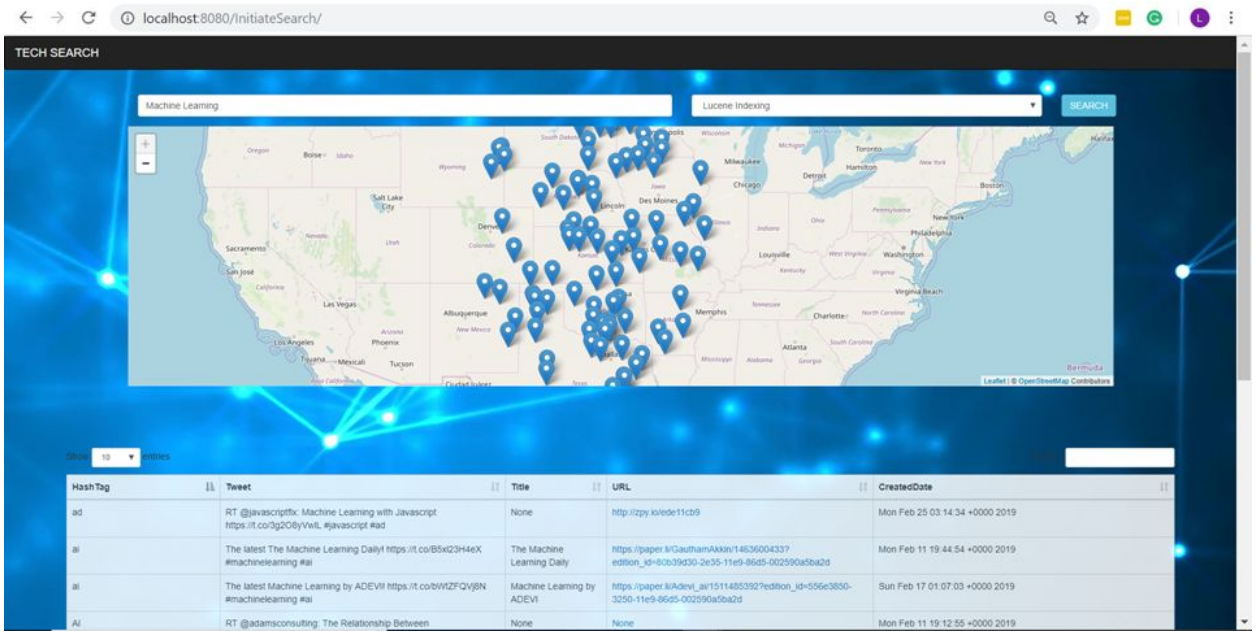

Fig 5: Results displayed for a particular query

Obstacles and solutions:

- Faced heap space problem in reduce phase as we are storing all the tweet attributes in the posting list
    - Resolution: Used MongoDB to store all the tweets and generate documentID. Stored only the document ID and term frequency in the posting list.
- Delay in search due to the repeated load of inverted index every time the search is made.
    - Resolution: Load the inverted index during the initial load of the server. Can optimize more by performing caching.
- During phase 1, we were passing data from crawler to the Lucene indexer as text, later we recognized that this was a bad design since when indexing, we can get the value by checking keys like "Title:". So if a tweet has some content like this, for example, "Book Title: Information retrieval", you may get a parsing error.

Future Scope:

1. Indexing in real time while crawling .
2. As the data increases, we could use sharding to direct the client request to appropriate shard. This would improve the read and write throughput

Comparison Graphs:

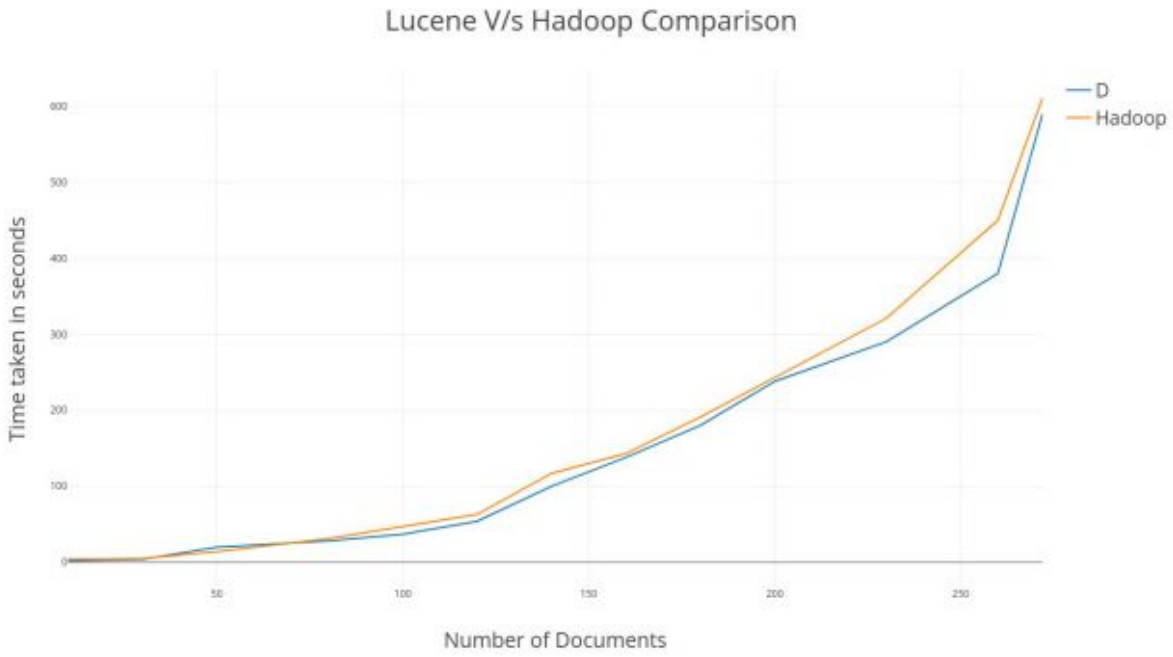

Fig 6: Indexing Performance

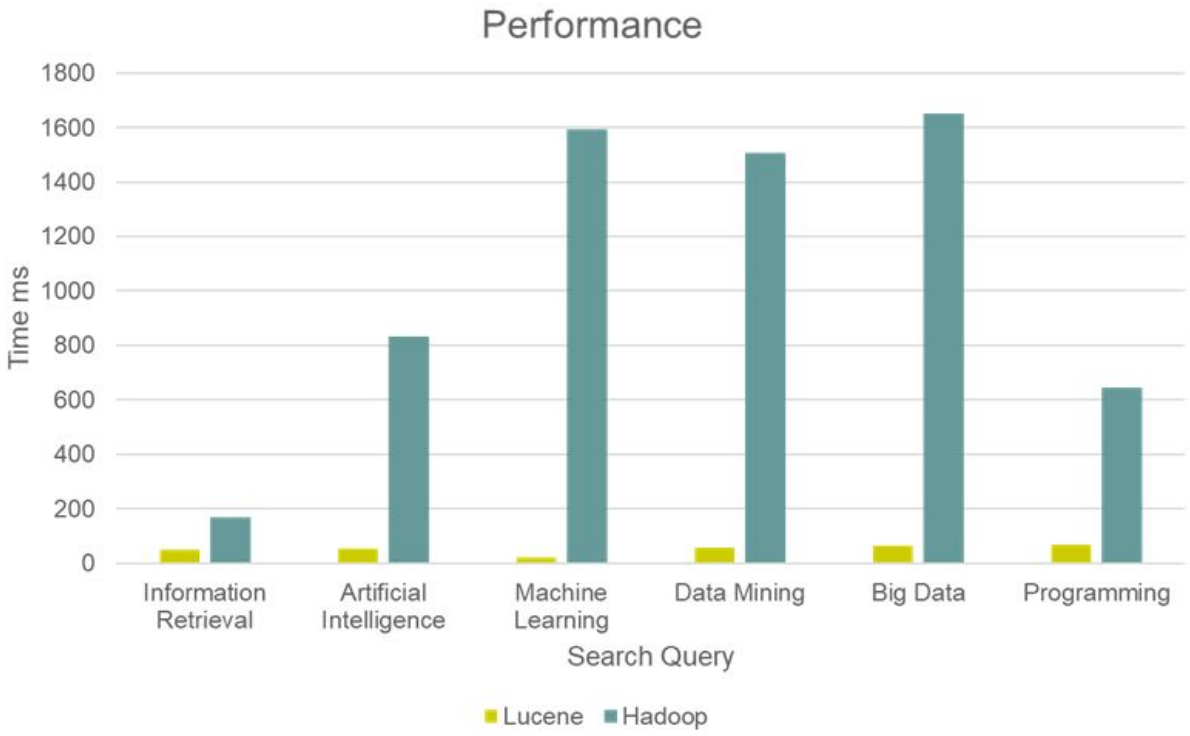

Fig 7: API Performance on various queries

Individual Contribution:

Lokesh Koppaka

1. Primary Contributor for Tweet indexing using Lucene
2. Implemented API's for Lucene and Hadoop
3. Implemented Map-Reduce program for inverted index generation
4. Performed initial system level configurations
5. Worked on the project report

Abhilash Sunkam

1. Wrote code for crawling tweets from Twitter Streaming API
2. Developed front end of the website
3. Helped in developing APIs for Lucene and Hadoop
4. Worked on Map Reduce index generation for Hadoop
5. Researched the frameworks and libraries for building the search engine
6. Worked on the project report

Vishal Lella

1. Learnt how Twitter Streaming API works
2. Helped in deciding the hashtags to seed
3. Helped in visualizing the indexed file using luke
4. Learnt how lucene works
5. Worked on Map Reduce index generation for Hadoop
6. Worked on the project report

References:

- https://datatables.net/
- https://www.vogella.com/tutorials/REST/article.html
- https://stackoverflow.com/
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- https://leafletjs.com/examples.html
- https://www.vogella.com/tutorials/ApacheTomcat/article.html