Search System

Parallel Search system using Hadoop Map-Reduce

Deepak Ravishankar RamKumar
#50097970, Computer Science Dept. SUNY Buffalo, Buffalo, New York-14228, US
dramkuma@buffalo.edu

Vijay Manoharan
#50097716, Computer Science Dept. SUNY Buffalo, Buffalo, New York-14228, US
manohara@buffalo.edu

***Abstract*: The paper consist of the key design and implementation details of a Search system using the Hadoop framework. Conclusions are drawn out of the results observed and impact of parallelism over the given problem.**

I. INTRODUCTION

A search system in terms of consumer is referred as a system that finds the results of a query in a huge collection of documents. It is technically referred as an information retrieval system. The objective of the project is to mimic a search system in a distributed network systems and perform search in parallel. Users can query for information about the dataset and the system will fetch results by populating the most relevant file for the query. The metadata and content data will be stored centrally in a distributed environment in a cluster. Initially each node will act as a mapper and fetch results for the particular query in parallel. Once all nodes finish their local computation they will act as a reducer and run a page ranking algorithm to determine the top k pages for a given query. The crux of the project is based on the map reduce algorithm in the Hadoop framework.
The search system is composed of three essential parts:

A. ***Document processing- Indexing****:*
The search system consists of documents over which the query search is done. These documents are parsed,tokenized, indexed and stored to facilitate fast and accurate search result. The index also contains the meta-data of the document collection. The section II explains in detail about the Indexing part.

B. ***Query Processing:***

.The query is parsed and broken into smaller parts called tokens.The tokens are fed into the system to perform the search process. The Query processing is explained in detail in section III.

C. ***Search Process.***
The tokens are searched individually and the matched sub-set of documents are returned. Section IV explains the details of how the search is done.

D. ***Ranking top relevant results.***
Having found the sub-set of documents that match the query requirement, some evaluation is done between the query and each document based on a ranking algorithm used by the system. The search system uses a ranking algorithm that scores the document based on the type and relation with the query. The documents are listed in order of the scores. The section V explains our ranking algorithm.

II. DOCUMENT PROCESSING – INDEXING.

The document collection is a data set obtained from Yelp. This data consists of information about a business, user and user reviews. The data set is classified based on the type of the information of the records and it can be a business record, user record or a review record.

The raw dataset is preprocessed. The initial phase of preprocessing involves normalizing the raw stream of data to a suitable format. The normalized data stream is then divided into smaller records. These documents are then processed to remove meta-tags and other parts of the document that are not required.

The preprocessing step transforms the entire data set into a consistent, well-formed and properly formatted collection. This reduces the complications during indexing and retrieval in the future.

The preprocessed documents are then tokenized by applying various rules as shown figure 2. Tokenization is done to compress the indexable data size and also remove unwanted information from the document. It also transforms every token into an understandable format to the system.

The following tokenization rules are applied.

a. **Accent Rule**: Removes accents from the documents, converts entire document to have uniform encoding. Example UTF-8 encoding.

b. **Capitalization Rule**: Converts the entire document to contain uniform case (lowercase).
c. **Hyphen Rule:** Remove Hyphen '-'from the document that are irrelevant.
d. Number Rule: Remove unnecessary numbers from the document.
e. **Punctuation Rule:** Remove punctuations from the document.
f. **Special character Rule** is applied to remove special characters like '&','#' etc., which does not make any impact to query matching.
g. **Stop word Rule:** The important tokenization rule is the stop word rule, which eliminates many common words that have very little value. By removing these set of words we can reduce the document data size by 40%..

The tokenized document contains only useful data that remains after the above rules are applied.
For example: consider the following document content.

"*This is Sample document content, containing àccent data, Capital word, hyphen-word, 131231, punctuation!?, special character such as $ & # and many stop words like at and an am are this that then better etc..*"
After tokenization the document will be:
"sample document content containing accent data capital word hyphen word punctuation special character stop words"

INDEXING. The main phase of document processing is Indexing. Indexing is one of the factor that determines how good a search system is. A well-formed index will have reduced complexity and better performance in finding the relevant document for the query. The indexing phase is carried out by scanning the entire parsed and tokenized document collection. The index consists of a unique key and a payload parameter as value. We are using Term-Document indexing approach. This payload is useful during fetching the results of top k documents for query- *ranking*.
The indexable record is added to buckets based on the first character of the term. There are 36 buckets made for the entire document collection. The buckets are named as the alpha-numberic character. The bucket A will contain only records with term starting with 'a' and bucket Z will contain only terms starting with 'z'. The figure 3 shows the buckets and 4 shows the indexed record inside a bucket.

Let us consider the first record from the figure 2.

"x": "195627^^t^^QND-tvk05oiQUIoL_5jJKw"

Term is 'x' and value is "195627^^t^^QND-tvk05oiQU IoL_5jJKw".
The value is composed of the document-id and the payload, in the given example the document-id is 195627, and the rest are the payload field.

The payload field consist of the following:
- Document type
- Unique identifier of record.
- Score.

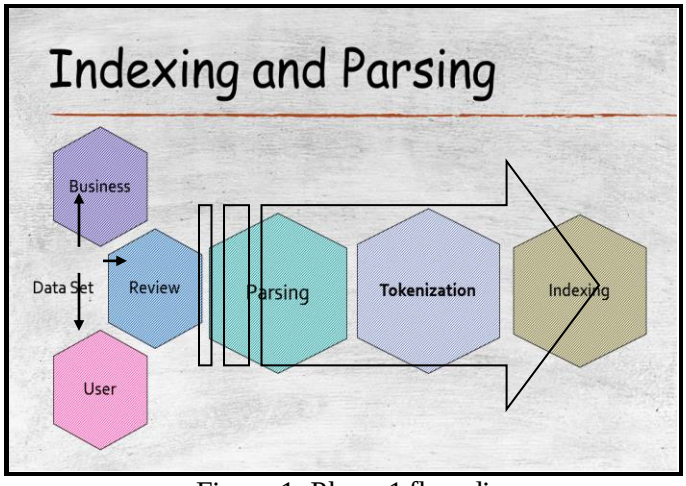


Figure 1. Phase 1 flow diagram.

The above figure1 shows the flow of phase 1, the parsing and processing of document. It starts with separating the data set into types and parsing the raw stream of data. This parsed data stream is applied with different tokenization rules to get a document with tokens, which is finally indexed and stored.

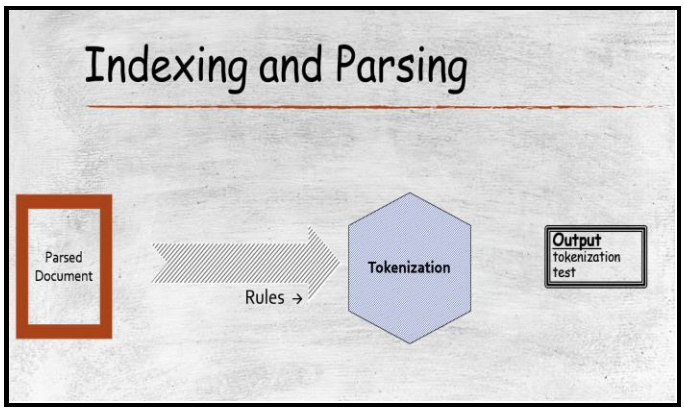


Figure 2. Tokenization.

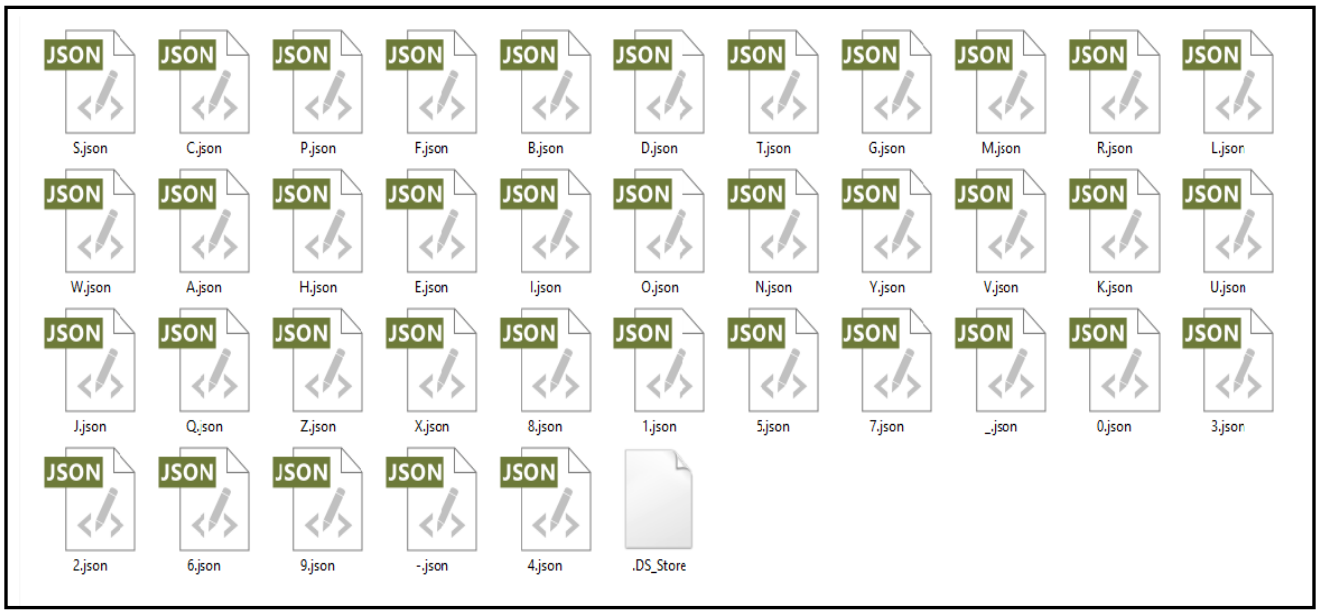


Figure3. Buckets- The dictionary of Index.

Figure 4 Indexed records of Bucket X.

III. Query Processing

When a user searches for a keyword or phrase, the entire query is passed to the query processor. Each term in the query is searched to find the relevant document-id where the term occurs. This phase involves processing the query in order to match the indexed document. Reducing the number of searches will result in decrease in the computational resources and complexity. This can be accomplished by reducing the number of searches done in the query and also number of documents over which search is done.

Reducing the number of searches can be accomplished by removal of irrelevant terms in the query. The entire raw query is first processed and parsed. Following that various tokenization rules are applied to the query and this breaks down the query to individual tokens. Similar tokenization rules were used in the document processing phase.

The later aspect is achieved by writing the index into different files based on some criteria and keeping track about it. We are writing index records to different buckets based on the first character of the term. Thus saving the time in searching the file.

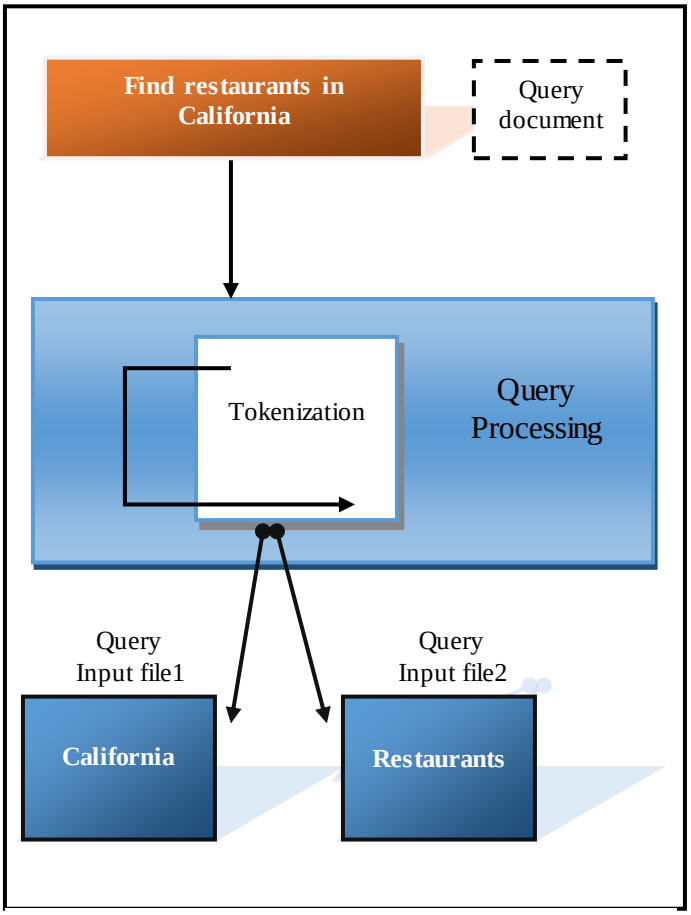


Figure5. The query processing flow diagram.

The following tokenization rules are applied to the query:

a) Accent Rule
b) Capitalization Rule
c) Hyphen Rule
d) Number Rule
e) Punctuation Rule
f) Special Character Rule
g) Stop word Rule

Let us see how these rules impact the performance and reduce the overhead. For an example consider the "hey find some restaurants in California". Now the number of searches for query will be one for each word, summing up to 7.
After applying the rules and removal of unnecessary terms in the query, the final query becomes "California Restaurant". Now the system only needs to search for two terms. On an average the number of searches after query processing reduces by ~2.4 times with query processing.
If the index is not divided into buckets every search involves reading the entire index. If the indexed documents are indexed in some sort of dictionary, the size of a document read is reduced by 20 times. Incorporating the above aspects in the design phase reduces the computation to a greater extent.

The Figure 5 shows how the query processing is done. The input to the query processor is the user search phrase and the output is the list token files. The token files consist of a single token that the mapper will work with.

IV. Search Process

The query terms are searched on the indexed data to find the subset of documents that are relevant. This subset of document is returned as output in an order based on the ranking algorithm. The ranking algorithm is explained in Section V.
Searching the single token:

- The tokenized query is searched for relevant document by taking a single token at a time.
- The tokens are searched in its respective buckets.
- The tokens are matched with the indexed records and a complete list is fetched from the index file.
- The fetched results forms the subset of the relevant document.
- If no indexed record is found for the term in the bucket, a "No matching results" tag is returned.

The tokenized queries are saved into separate files and fed to the mapper. Thus each Mapper can work on a single token. For fast processing the Mappers are run in

parallel. The Mapper searches for the index records stored in the bucket and fetches the value. This values is then emitted to the Reducer. The reduced will get a subset of relevant documents and orders them based on the ranking algorithm.

The following steps are implement for fast processing:

- Mapper runs in parallel, providing scaling during the search.
- The indexed records are stored in buckets, thus reducing the size of search and the computation overhead to find out the bucket to look for.

The figure 8 shows how the relevant documents are fetched. Let take the above mentioned query "Find Restaurants in California", after processing the query will have two tokens "California" and "restaurants". These two tokens will be written into Map input files mapinputfile1.txt and mapinputfile2.txt. Now when the job is initialized by the master node, it will spawn two mappers. Each mapper working on different file will be performing the same operation. The Mapper finds out the bucket to which the query term belongs to and looks for a match. The mapper searches for 'California' in the bucket C and 'restaurant' in the bucket R in parallel. Now the mapper fetches the corresponding entry in the respective bucket and emits the value to the reducer. All these operations are done independently and in parallel. The Reducer then forms relevant document list and scores the list using the ranking algorithm.

V. RANKING THE DOCUMENT

After fetching the relevant documents that match the query term, the system must present only few results that best matches the query. This is done by the Reducer by using ranking algorithm. The ranking algorithm scores all the relevant documents and orders them. The top k scored documents are returned to user as list of results.
The ranking algorithm depends on the model system uses and richness of the document collection. Most of the search engine/system uses term-document frequency as the primary scoring method.

The rank of the page is affected by:

- Query term-document frequency.
- Query term.
- The type of record
- Payload of the index record.
- Also the term frequency in that document.

The above list shows the order of ranking. The Query term-document frequency is given more score, then followed by the importance of the query term, followed by the type of the record.

The documents with high Query term-document frequency are ranked higher. Query term – document frequency is the number of times the Mapper returns the document-id to the Reducer. This indicates that many number terms in the query match the document. Thus inferring the document is what the user is looking for.

Some important query terms are given more weights and are determined during the query processing. These terms may be name or place. These terms are considered as the wildcard cases in few queried. In the above example "Restaurants in California" the term 'California' is given more weight.

Based on the type of record: The terms that match to the business document is weighted more than the term that belong to a user or a review document type.

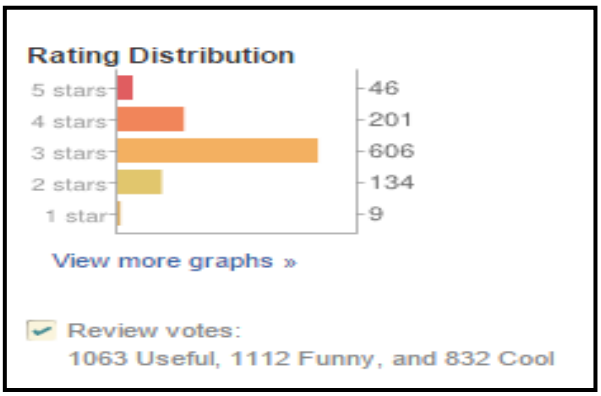


Figure 6.

Figure 7.

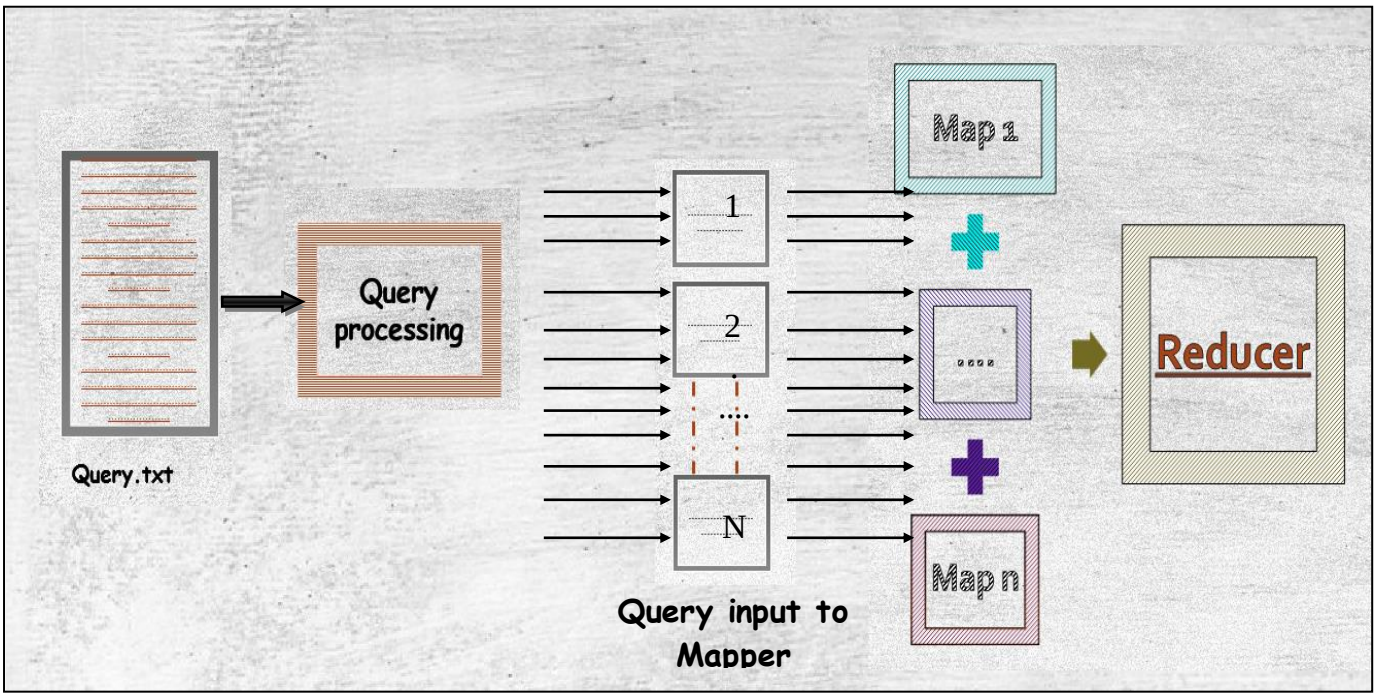
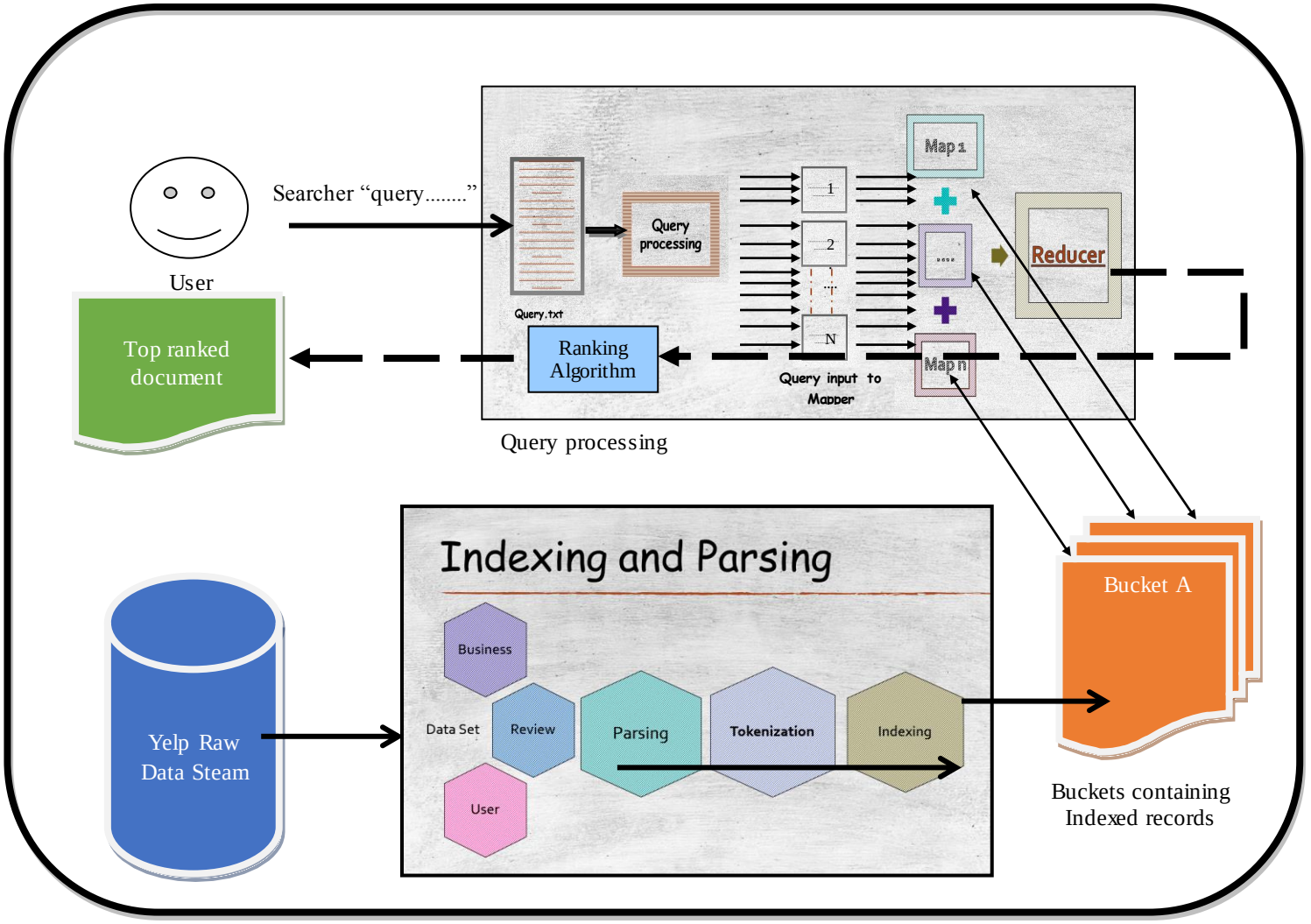


Figure.8 Flow Diagram Query processing.

Below- Figure 9. The Complete Search System



Query: "Restaurants in California"

Output for top 10 documents

```
1) Document Id      464133
neighborhoods     ["South Los Angeles"]
city     Los Angeles
latitude     34.025176
review_count     2
full_address     2912 S Figueroa St
South Los Angeles
Los Angeles, CA 90007
stars     2.0
type     business
url     http://www.yelp.com/biz/carls-jr-restaurants-los-angeles-11
schools     ["University of Southern California"]
name     Carl's Jr Restaurants
categories     ["Restaurants"]
photo_url     http://s3-media1.ak.yelpcdn.com/assets/2/www/img/924a6444ca6c/gfx/blank_biz_medium.gif
state     CA
business_id     V8nh9DR3ePaEoneIvLvXpQ
open     true
longitude     -118.277385


2) Document Id      474013
neighborhoods     ["South Los Angeles"]
city     Los Angeles
latitude     34.0266092
review_count     29
full_address     525 W 28th St
South Los Angeles
Los Angeles, CA 90007
stars     3.0
type     business
url     http://www.yelp.com/biz/grinder-restaurants-los-angeles
schools     ["University of Southern California"]
name     Grinder Restaurants
categories     ["American (Traditional)","Restaurants"]
```

```
9) Document Id      469348
neighborhoods     []
city     Claremont
latitude     34.0966296
review_count     53
full_address     333 W Bonita Ave
Claremont, CA 91711
stars     3.5
type     business
url     http://www.yelp.com/biz/full-of-life-village-baker-claremont
schools     ["Harvey Mudd College"]
name     Full of Life Village Baker
categories     ["Food","Bakeries","Coffee & Tea","Sandwiches","Restaurants"]
photo_url     http://s3-media2.ak.yelpcdn.com/bphoto/9MvSy6yz0zB-0C7FkhxxMw/ms.jpg
state     CA
business_id     YI_atw2AZqGEfjn9_ZJ5nw
open     true
longitude     -117.7183628


10) Document Id      471451
neighborhoods     ["South Los Angeles"]
city     Los Angeles
latitude     34.0247747
review_count     13
full_address     3021 S Figueroa St
South Los Angeles
Los Angeles, CA 90007
```

Query: “search for Stephanie”

```
Search Results for " search for stephanie? " :
473485
94694
120866
100605
71228
47182
105943
116498
109427
117256
```

Output:

```
1) Document Id      473485
neighborhoods     ["Westwood"]
city     Los Angeles
latitude     34.0654733
review_count     3
full_address     UCLA Medical Center
200 Medical Plaza
# 530
Westwood
Los Angeles, CA 90095
stars     3.5
type     business
url     http://www.yelp.com/biz/stephanie-smooke-m-d-los-angeles
schools     ["University of California - Los Angeles"]
name     Stephanie Smooke, M.D.
categories     ["Professional Services"]
photo_url     http://s3-media1.ak.yelpcdn.com/assets/2/www/img/924a6444ca6c/gfx/blank_biz_medium.gif
state     CA
business_id     TjqU4LDziEuYBMeyxQMGfw
open     true
longitude     -118.446099


2) Document Id      94694
user_id     BLmRIoETvx6ZiegiIWh0tw
average_stars     3.61328125
name     Stephanie W.
review_count     512
votes     {"cool":2142,"useful":2600,"funny":1549}
type     user
url     http://www.yelp.com/user_details?userid=BLmRIoETvx6ZiegiIWh0tw


3) Document Id      120866
user_id     0Dq3T5EId_WRUtic7wZzBg
average_stars     4.10552763819095
name     Stephanie H.
review_count     199
votes     {"cool":1910,"useful":2140,"funny":1568}
type     user
url     http://www.yelp.com/user_details?userid=0Dq3T5EId_WRUtic7wZzBg
```

The above figures 6 & 7, show the special scoring parameter. The special parameter are different for different document type.

- For Business document: the special parameter is "Stars", this is the rating that the business gets from the user. Levels of star indicates that the business is good and can be recommended or not. The star can be in range from 0 to 5, where 5 being the highest rating and 0 being the lowest rating.
- For User document, the number of review votes determines the score. Greater the number of votes, higher score is given.
- For the Review document, the number of useful reviews voted by other users determines the score.

VI. Results and Performance

A. Storage

Tokenization reduces the size of the document that is indexed. Without tokenization the size of the index will be 3-6 times more. Tokenization not only removes the term that have low significance but also reduces repetitions. For example, the word 'computer', 'Computer' and 'COMPUTER' will be mapped to three different indexed records by meaning the same. Thus saving close 50% of the storage space.
Also increasing the performance by pushing most of the needed data to the main memory and reducing the latency in accessing the data from the disk.

B. Search performance

If the search done without tokenization uses more resources, after tokenization the resources reduces by 3-6 times.
The resources are:

- CPU processing
- Storage during search

C. Index performance

Index was carried in serial and then in parallel. When the 1 million documents indexed serially in 1 hour 50 minutes. The same document set indexed parallel using multi-threads in 22minutes.
Efficient indexing reduces the overhead in computation of:

- Faster retrieval.
- Sorted indexes- Binary search can be performed on the sorted

Query	CPU Time secs	Accuracy
Restaurants in california (top 10)	0.064	90%
Restaurants in california (top 100)	0.0822	83%
Stephanie	0.086	95%

The page 8 and 9 contains the result of search.

VII. CONCLUSION

We developed a complete search system making use of the map reduce algorithm in the Hadoop framework. This project gave us exposure to big data and the problems that come with it. We also explored other parallel paradigms such as multithreading. We were able to analyze the execution time for the given query by parallelizing it on multiple levels.

VIII. Future Work

- Our immediate goal is to improve the search efficiency and to scale more document collection. Some improvements that can improve the efficiency are caching (HDFS caching),
- Query Correction
- Provide Suggestions to users, based on the previous searches.
- Incorporate NLP for supporting complicated queries.
- Cluster the data based on the type to speed up the query result.
- Stemming can future reduce the size of the index.

References

[1] Wikipedia article for indexing: http://en.wikipedia.org/wiki/Search_engine_indexing.

[2] Hadoop : map-reduce:

- http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- https://developer.yahoo.com/hadoop/tutorial/module4.html

[3] JSon : http://www.json.org/java/

[4] Yelp Data Set : http://www.yelp.com/

[5] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze : " An introduction to Informational Retrieval" edition 2009 Cambridge UP.