Independent study Report

**Title: Performing Map reduction on Tweeter dataset**

**By Haniel Sceva Manda**

# Problem

In this era of big data where the dataset in any domain just keeps on growing, the developers and analysts are thriving to find the efficient way to segregate the dataset in a compact better way to increase scalability and decrease the time takes to query. This led to the concept of Map reduce where a huge data is divided, segregated into small chunks of datasets so that the data can be easily accessed for analytics or even for web application.

Using this map reduce approach an attempt was made in this project to minimize the size of a dataset substantially. The model is a specialization of the split-apply-combine strategy for data analysis. It is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms.

The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine. As such, a single-threaded implementation of MapReduce is usually not faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with multi-threaded implementations on multi-processor hardware. The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play. Optimizing the communication cost is essential to a good MapReduce algorithm.

The two biggest advantages of MapReduce are:

1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machine instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

2. Data Locality:

Instead of moving data to the processing unit, we are moving processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed issues like Moving huge data to processing is costly and deteriorates the network performance, Processing takes time as the data is processed by a single unit which becomes the bottleneck and there are cases where the master node can get over-burdened and may fail. Now, MapReduce allows us to overcome above issues by bringing the processing unit to the data.

A MapReduce framework (or system) is usually composed of three operations (or steps):

* Map: each worker node applies the map function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of the redundant input data is processed.
* Shuffle: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node.
* Reduce: worker nodes now process each group of output data, per key, in parallel.

The dataset used here to perform MapReduce is a tweeter dataset which consists of three million troll tweets tweeted from different locations by different users over the period February 2012 to May 2019. These data were separated and saved in comma-separated values file (.csv files). In total, there are nine CSV files that include 2,973,371 tweets from 2,848 Twitter handles. At first this data had to be converted into JSON format and then pushed into the MongoDB.

Due to the memory concerns the dataset which is used for this project has been reduced to 214507 documents. This data has been reduced to 195 documents with respected to the Author ID using the map reduction query. This reduced data has been saved into a new collection automatically as soon the data is reduced which is available for future queries and analytics. By using the map reduce query we can further reduce the whole dataset into small compact datasets and save them in different collections. These small segregated datasets have the entire information available, and the time taken to query those huge datasets is bough down efficiently using the map reduce concept.

Here we have a just implemented the map reduce query with respect to the Author ID alone, but this can be used against the region from where the tweets have been made and even separated data with respect to language used etc.,

Querying through all those datasets separately, would be relatively easier and efficient in terms of scalability. Thus, by making it easier for analytics and even to fetch data into frontend in a web application or even data management.

# Software Design and Implementation

## Software Design and NoSQL-Databse and Tools Used

The database used in this project is **MongoDB**. MongoDB is an open-source document database and leading NoSQL database. The very basic feature of MongoDB is that it is a schema-less database. No schema migrations anymore. Since MongoDB is schema-free, your code defines your schema. It is easy to access documents by indexing. Hence, it provides fast query response. The speed of MongoDB is 100 times faster than the relational database. A great advantage of MongoDB is that it is a horizontally scalable database. When we need to handle a large data, you can distribute it to several machines. These are the prime reason to use this dataset.

Initially a part of each CSV files of the original tweeter dataset has been taken and converted into JSON format and then pushed into the Database named ‘NOSQL’, inside the collection called ‘databases’.

For the backend, **NodeJS** has been used as this project has a micro-service architecture. Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server- and client-side scripts. Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript. Developers can create scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task. Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming. These were the prime reasons for choosing the NodeJS as the backend framework for this project.

**ExpressJS** is been used to provied the MVC architecture for the web application and to handle the REST API. Express.js is a NodeJS web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.

**Mongoose** is used to connect to the database, manage relationships and provide schema validations. It is an object data modeling (ODM) library that provides a rigorous modeling environment for your data, enforcing structure as needed while still maintaining the flexibility that makes MongoDB powerful. Mongoose’s main value is that you can define schemas for your collections which are then enforced at the ODM layer by Mongoose. Mongoose.js also has utilities for simplifying Node’s callback patterns that make it easier to work with than the standard driver alone.

Body-parser is a node module which is being used to parses the JSON, buffer, string and URL encoded data submitted using HTTP POST request. CORS is another node moduels which is used in this project to provide an Express middleware that can be used to enable Cross-origin resource sharing.

**Angular 6.x** is used at the frontend to create website along with HTML and CSS. It is an open-source web application framework. Features like Two-way binding, templating, RESTful API handling, modularization, AJAX handling, dependency injection, etc. makes it the perfect options for a creating a web application. AngularJS provides “Model View Control" architecture which is perfect for dynamic modelling. As we already know, any application is built from a process of combining modules together. These modules work using different logics that are initialized according to individual needs. It has HTML language to build user interface. This leads to a more organized and simplistic UI. The additional elements in the system can be accessed with no need for the DOM to simulate additional elements. The controller doesn't need to manipulate the DOM directly, as this should be done through directives. Directives make up a separate part of the element set which can be used anywhere other than in a web application.

Two components were used to perform map reduce on the click of a button and then to display the reduced data in a table. Initially **POSTMAN** was used to create the HTTP API to get the node perform functions. Later after developing the frontend using angular the REST API request were sent using the angular services.

This entire application forms a **MEAN** stack application with MongoDB as database, ExpressJS to handle the API request, AngularJS at the frontend and NodeJS as the backend.

Once we click the button to perform map reduce an API request is sent to the backend where the map reduced is performed and the new dataset is saved into a new collection. When the button Fetch data is clicked the website is routed to a new page which on page load send a request to backend to fetch the entire collection data of the newly formed dataset from the database. Once fetched this data is displayed in the table.

## Supported Queries

The prime query which reduces the dataset is

**Db.collection.mapReduce( map(), reduce(),**

**{ out: <collection>,**

**query: <document>,**

**sort: <document>,**

**limit: <number>,**

**finalize: <function>,**

**scope: <document>,**

**jsMode: <boolean>,**

**verbose: <boolean>,**

**})**

Where DB is the database which were going to use in the MongoDB, collection is the name of the collection. Mapreduce() is a function which call up the map() function, reduce() function and the at the end we mention what to do with the output of this function or to limit the dataset etc.

**map(){**

**emit (key, value)**

**}**

map() function loops through the entire dataset and maps a value with key and emits the key value pair. In our project the enternal\_author\_id has been used as the key with content of the tweet, region, language and count were sent as value.

**reduce(key, value){**

**return result;**

**}**

reduce() function reduces the key value into a single object by associating all the values with a single key. In the project the we return the reduced value as a document but before we return it we increase the count by one for every match in the external\_author\_id.

**Out: <collection>**

Specifies the location of the result of the map-reduce operation.

**Query: <document>**

Specifies the selection criteria using query operators for determining the documents input to the map function.

**Sort: <document>**

Sorts the input documents. This option is useful for optimization.

**Limit: <number>**

Specifies a maximum number of documents for the input into the map function.

**finalize: <function>**

Follows the reduce method and modifies the output.

**Scope: <document>**

Specifies global variables that are accessible in the map, reduce and finalize functions.

**jsMode: <boolean>**

Specifies whether to convert intermediate data into BSON format between the execution of the map and reduce functions.

**Verbose:<boolena>**

Specifies whether to include the timing information in the result information.

**Collcetion.find({/\* key: value\*/})**

This is another query which has been used to fetch the data matching the key value pair mention. This query filters and send return the data which matches the object which we use to filter. If there is no object present, it’ll return the whole collection. This query has been used to fetch the map-reduced data from the new collection to display it in the frontend.

**Collection.remove({/\*key: value\*/})**

This query is used to remove all the documents which match the object mentioned in the parenthesis. If there is an empty object present with the parenthesis then mongo would drop all the documents present in the collection. In the project we use it to drop the collection before performing the map reduction to stop it from retracting the same keys and mapping them to old dataset.

##### References

1. http://youtube.com/watch?v=
2. https://mongoosejs.com/docs/api.html#model\_Model.mapReduce
3. https://www.youtube.com/watch?v=3slWCHApiFc
4. https://gist.github.com/nherment/1431054
5. http://benbuckman.net/articles/mongodb-mapreduce-nodejs-php/
6. https://stackoverflow.com/questions/34545921/mapreduce-using-node-js-and-mongoose
7. https://mongoosejs.com/docs/api.html#model\_Model.mapReduce
8. https://www.youtube.com/watch?v=x2\_bcCZg8vQ
9. https://www.youtube.com/watch?v=3slWCHApiFc