# Comparing HiveQL and MapReduce Methods to Process Fact Data in a Data Warehouse

Haince Denis Pen
Data Engineer
Morgan Stanley
New York, NY
haince.pen@gmail.com

Prajyoti Dsilva
Assistant Professor
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India
prajyotidsilva@sfitengg.org

Sweedle Mascarnes
Assistant Professor
Department of Information Technology
St. Francis Institute of Technology
Mumbai, India
sweedalmascarnes@sfitengg.org

*Abstract*– **Today Big data is one of the most widely spoken about technology that is being explored throughout the world by technology enthusiasts and academic researchers. The reason for this is the enormous data generated every second of each day. Every webpage visited, every text message sent, every post on social networking websites, check-in information, mouse clicks etc. is logged. This data needs to be stored and retrieved efficiently, moreover the data is unstructured therefore the traditional methods of strong data fail. This data needs to be stored and retrieved efficiently There is a need of an efficient, scalable and robust architecture that needs stores enormous amounts of unstructured data, which can be queried as and when required. In this paper, we come up with a novel methodology to build a data warehouse over big data technologies while specifically addressing the issues of scalability and user performance. Our emphasis is on building a data pipeline which can be used as a reference for future research on the methodologies to build a data warehouse over big data technologies for either structured or unstructured data sources. We have demonstrated the processing of data for retrieving the facts from data warehouse using two techniques, namely HiveQL and MapReduce.**

***Keywords- data warehouse; data marts; facts, dimensions; big data; hadoop distributed file system; MapReduce; HiveQL; Apache Sqoop***

## I. INTRODUCTION

The main idea of a data warehouse is to act as a centralized repository that store and retrieve historical data for an organization to help in forecasting and strategic decision making. The concept of data warehouse was coined in early 1990s by Bill Inmon, who advocated an enterprise-wide top-down approach to build a data repository to fulfill the needs of an organization. There is another approach proposed by Ralph Kimball, which follows bottom-up approach and dimensional modeling to build data marts which conform to one-another to form a data warehouse [1].

The above approaches work perfectly fine when the source data is structured or in relational format. With the technological boom, data has been ever increasing and is getting difficult to manage every single day. Also, the nature of data has expanded to semi and unstructured data in addition to the structured data. Hence, there has been a more intense need for a rigorous Extraction-Transformation-Loading (ETL) activity.

IBM describes big data in terms of 4 Vs. They are Volume, Velocity, Variety and Veracity. The volume of data generated has been growing tremendously and managing this data is difficult. The velocity at which this data is generated is very high. There is variety in the form of data; structured, unstructured and semi-structured data. And then there is huge veracity in the data collected, which means the data is uncertain or inaccurate.

To manage these different characteristics of data is not possible in a relational data warehousing system, hence the need of a new architecture that manages these characteristics of data and performs all the operations of a traditional data warehouse, like performing analysis and forecasting, providing trends etc.

In this paper, we propose a novel architecture that will become the backbone of a data warehouse over big data platform. We employ the concepts of both data warehousing and data engineering to implement our system. The implemented system acts as a direction for researchers to further address the issue of scalability, complexity, performance, analytics, in-memory representation and several other issues.

For simulating the data warehousing environment, we have collected unstructured data from Twitter as data source. Although, we would like to emphasize on the fact that this paper is not constricted only to twitter and the architecture can be generalized to be used for any other data source under scrutiny, as the architecture is quite flexible.

The data is based on US presidential polls 2016 and specific to the top three presidential candidates namely Hillary Clinton, Jeb Bush and Donald Trump. The system collects data in a point in time from twitter and later calculates the percentage of sentiments of people for their respective candidates. We can run the JAVA API any number of times to collect the data and the system will store it in the data warehouse while preserving the historical data from previous runs.

We will be using one single tweet as the grain element to determine the sentiment of a person towards a candidate at a point in time. We are performing aggregations of data in the fact tables in order to calculate the numbers using two techniques: Hive, which is a SQL like engine developed in JAVA and runs on MapReduce simulating SQL like syntax and functionality; and second technique where we have actually written the custom MapReduce code that performs aggregations to get the same results. Thereby being able to see into the black box and determine what really is happening in the backend; which will further address the issue of size, complexity, analytics, and user performance and code re-usability.

## II. MOTIVATION AND RELATED WORK

### *A.* Motivation

To develop a novel architecture which stands as another step ahead to building an ideal data warehouse over big data ecosystem.

Cuzzocrea et al. [2] discussed the current challenges and future research in performing data warehousing and OLAP over big data. The paper shed light on areas of development in data warehousing domain in big data environment.

### *B.* Related work

Cohen et al. [3] discussed the design philosophy, techniques and experience to provide MAD (Magnetic, Agile, and Deep) analytics while designing a VLDB. The paper also focuses on enabling agile design and flexible algorithm development using SQL and MapReduce interfaces. The paper provided various research directions for optimizing query and storage, automating physical design for automated tasks and online query processing for MAD analysis. Herdoutou et al. [4] discussed the Starfish model for designing a system, which is self-tuning and performs analytics on big data. The system is built over hadoop ecosystem and tunes itself to work as per the user requirements. Thuso et al [5] discussed about Hive as an open project, the paper extensively describes the architecture for Hive and explains the query language syntax and its functionality. Although HiveQL is a subset of SQL, and many of the core SQL functionalities are yet to be implemented in it, it has great potential to manage huge amounts of data for reporting and ad-hoc querying, the data being as large as 700 TB. The authors are currently exploring columnar storage and more intelligent data placement to improve scan performance. TPC-H was used as a benchmark while measuring the performance. The authors are also looking at multi-query optimization techniques and performing n-way joins in a single map-reduce job. Floratou et al. [6] have discussed the significance of SQL-on-Hadoop by comparing different SQL like querying platforms over big data like Hive and Impala over diverse frameworks like MapReduce, Tez and file formats like ORC, parquet etc. TPC-H database is used as a standard for benchmarking. Their research concludes that Impala has a significant performance advantage over Hive when the workload fits in-memory. This is due to the fact that Impala processes everything in memory and has very fast I/O and pipelined query execution. Hive runs slower on MapReduce, which it runs faster on Tez; but still it has poor performance as compared to Impala. Kornacker et al. [7] have discussed Impala from user's perspective, explaining its architecture, components, and superior performance when compared against other SQL-on-Hadoop systems. The reason such performance is the massively-parallel query execution engine. Another reason for its fast performance is the way it fetches data from HDFS filesystem, Impala uses an HDFS feature called short circuit local reads which allows it to read the data from local disk at almost 100 mb/s and can go up to 1.2 gb/s. Imapala supports all the popular formats of data storage hence attains more flexibility. This technology is powerful and takes the end-user performance in big data environment to a higher level, and can replace the monolithic analytic RDBMSs.

## III. DATA WAREHOUSING PIPELINE AND MULTI-DIMENSIONAL MODEL

A data pipeline is carefully designed to build the data warehouse over the big data environment.

To create the data warehouse over big data, a pipeline is created. The architecture for the same is explained later on. The data warehouse components are explained as follows:

A multidimensional schema, namely E, is defined by ($F^E$, $D^E$ and $Star^E$) where,

$F^E = \{F1,\ldots, Fn\}$ is a finite set of facts,

$D^E = \{D1,\ldots, Dm\}$ is a finite set of dimensions,

$Star^E = F^E \rightarrow 2^{DE}$ is a function that associates each fact F*i* of $F^E$ to a set of D*i* dimensions, $Di \in Star^E (Fi)$, along which it can be analyzed; where $2^{DE}$ is the power set of $D^E$. [e]

A dimension, denoted as Di being $Di \in D^E$ (abusively noted as D) is defined by ($N^D$, $A^D$, $H^D$) where,

$N^D$ is the dimension name,

$A^D$ is a dimensional attribute, and

$H^D$ is a set hierarchies.

F**act**, $F \in F^E$, is defined by ($N^F$, $M^F$) where,

$N^F$ is the name of the fact,

$M^F$ is a set of measures, and

each associated with an aggregation function $f_i$. [8]

- ***Conversion into a schemata for MongoDB***

We are using MongoDB as an intermediate data repository to store the data as documents within each of its collections that act as dimensions of a data warehouse.

- **MongoDB document object model**

This model defines a framework to hold the specific data in the document format in a NoSQL database. The structure is pre-defined as per the need of the data and the physical schema that is decided upon.

Each of the attribute components are routed to specific document collections and stored as {key: value} pairs.

The structure and mapping of the attributes is in compliance with the physical schema of the data warehouse as shown in Fig. 1.

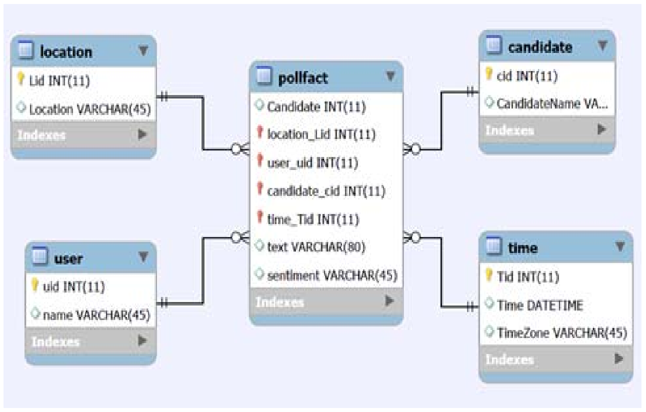
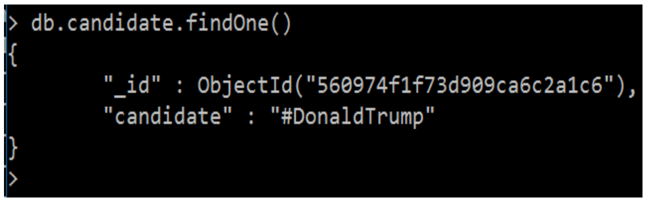


Fig. 1. Schema for data warehouse

The five tables above comprise of the fact and dimensions in the data warehouse. There are five collections with similar document structures in the MongoDB that store the data coming from the source. Fig. 2. shows an example of how a document/tuple is stored in a collection/ table in MongoDB.



```
> db.candidate.findOne()
{
        "_id" : ObjectId("560974f1f73d909ca6c2a1c6"),
        "candidate" : "#DonaldTrump"
}
>
```

Fig. 2. One of the documents from collection Candidate

In case of our dataset, let's assume D as the database schema which holds the collections/dimensions.

$D = \{C_{1,} C_{2,......,} C_N\}$ where,

$C_i$ represents the collections.

Within each collection, $C_i$ = {D1,…,Dx ,…,Dn}, the document Dx could be defined as follows: Dx = {(Attx Id, Vx Id, ……)}

## IV. SYSTEM IMPLEMENTATION

Below section explains the implementation of the system, its technical specification and working.

### *A.* Source Data

The main source for data for this system is extracted from twitter and is in semi structured format. We collected data like the tweet, its user details, the hashtag (to identify presidential candidate), location of origin for the tweet, time-variant information etc. The data is collected through an API written in JAVA and extracted into MongoDB directly. The grain for the data warehouse is each individual tweet in this case. We will be using this component to perform analytics in the later stages and therefore deciding on the grain element is extremely important.

### *B.* Data pipe-line/Architecture

This section will discuss on the data pipe-line/architecture for the system. Each phase in the implementation is marked by the swim lines in the architecture as shown in Fig. 3.

#### 1) Semi structured data

The first step consists of extraction of live unstructured data from Twitter into MongoDB collections using JAVA API. The components of the unstructured data are segregated into specific collections of the NoSQL database. These specific collections will then stand as the cornerstone to populating the actual dimensions and fact in the data warehouse. The unstructured data is specifically extracted b using specific hashtags. In our case, they are the names of the presidential candidates' #HillaryClinton, #JebBush and #DonaldTrump. Once the data is placed into MongoDB collections, it is run through a web classifier that approximates the sentiment for each of the tweet and assigns it a sentiment, which is later updated in the fact collection of MongoDB.

#### 2) Extraction-Transformation-Loading(ETL) process

One of the most important things to do while implementing a data warehouse is to perform the ETL process. In case of our system, once we had all the

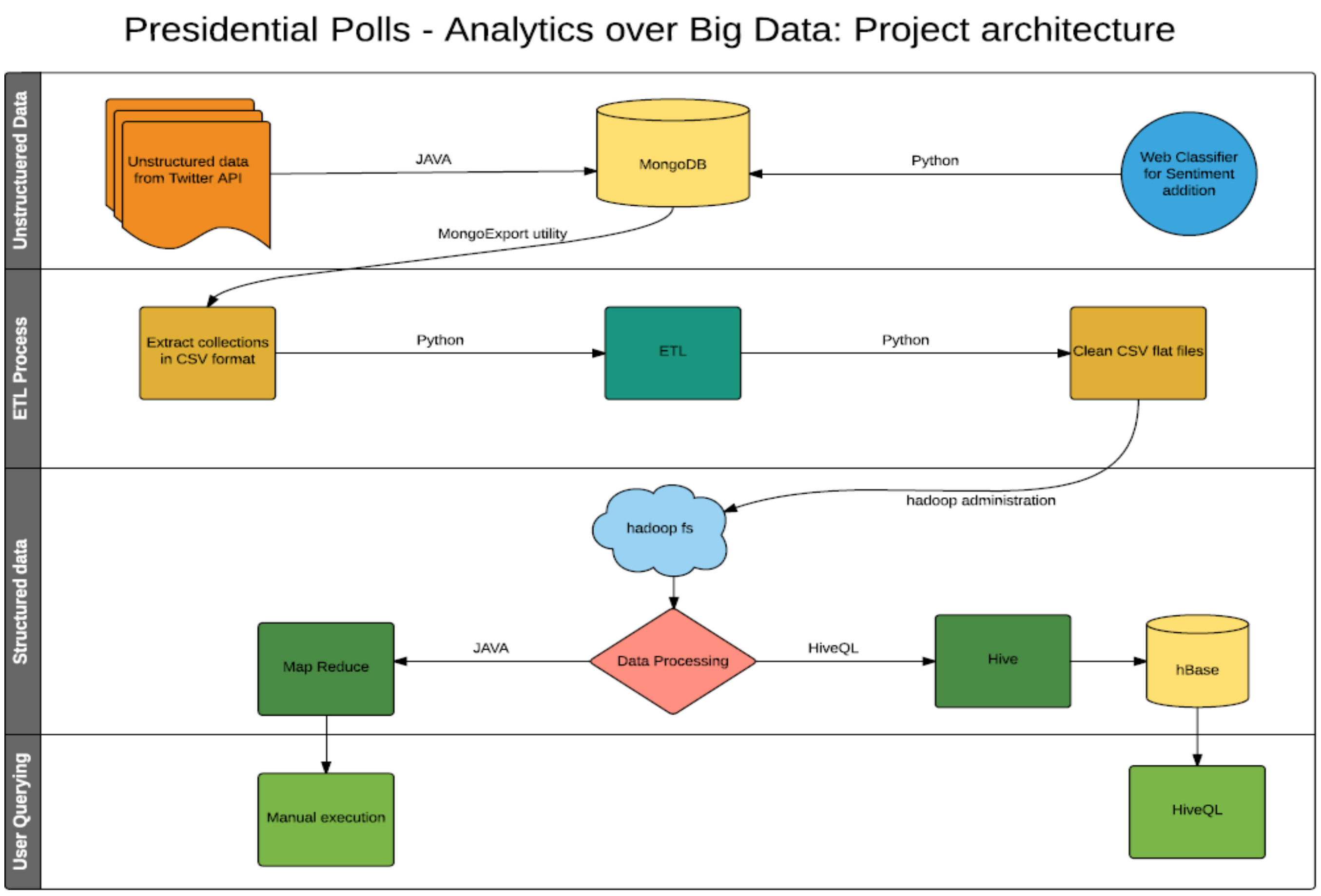
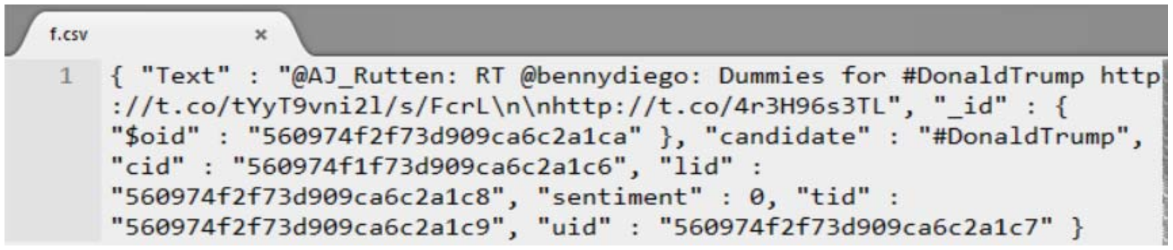


Fig. 3. Data pipeline/Architecture

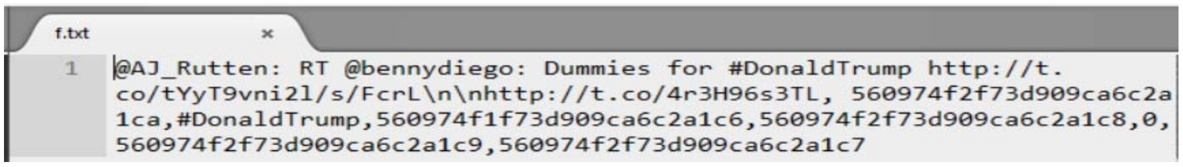
data populated in MongoDB collections, next step was to perform ETL process on that data. We implemented python scripts to perform the cleaning process on the data from collections once it was exported out of MongoDB using the export utility into a CSV file.

The python scripts transform the unclean csv files into flat text files which would be suitable for the big data ecosystem. Fig. 4. dispays small snippet of uncleaned csv exported out of MongoDB. Fig. 5. shows cleaned text file after cleansing/ETL process



```
{ "Text" : "@AJ_Rutten: RT @bennydiego: Dummies for #DonaldTrump http
://t.co/tYyT9vni2l/s/FcrL\n\nhttp://t.co/4r3H96s3TL", "_id" : {
"$oid" : "560974f2f73d909ca6c2a1ca" }, "candidate" : "#DonaldTrump",
"cid" : "560974f1f73d909ca6c2a1c6", "lid" :
"560974f2f73d909ca6c2a1c8", "sentiment" : 0, "tid" :
"560974f2f73d909ca6c2a1c9", "uid" : "560974f2f73d909ca6c2a1c7" }
```

Fig. 4. Unclean CSV file exported out of MongoDB



```
@AJ_Rutten: RT @bennydiego: Dummies for #DonaldTrump http://t.
co/tYyT9vni2l/s/FcrL\n\nhttp://t.co/4r3H96s3TL, 560974f2f73d909ca6c2a
1ca,#DonaldTrump,560974f1f73d909ca6c2a1c6,560974f2f73d909ca6c2a1c8,0,
560974f2f73d909ca6c2a1c9,560974f2f73d909ca6c2a1c7
```

Fig. 5. Clean txt flat file after ETL process

### 3) Structured Data and User Querying

After the cleaning process, the flat files were moved to hadoop fs under Cloudera environment. The next task was to process this data and take out the meaningful trends/patterns out of it.

We employed two techniques to demonstrate the data processing.

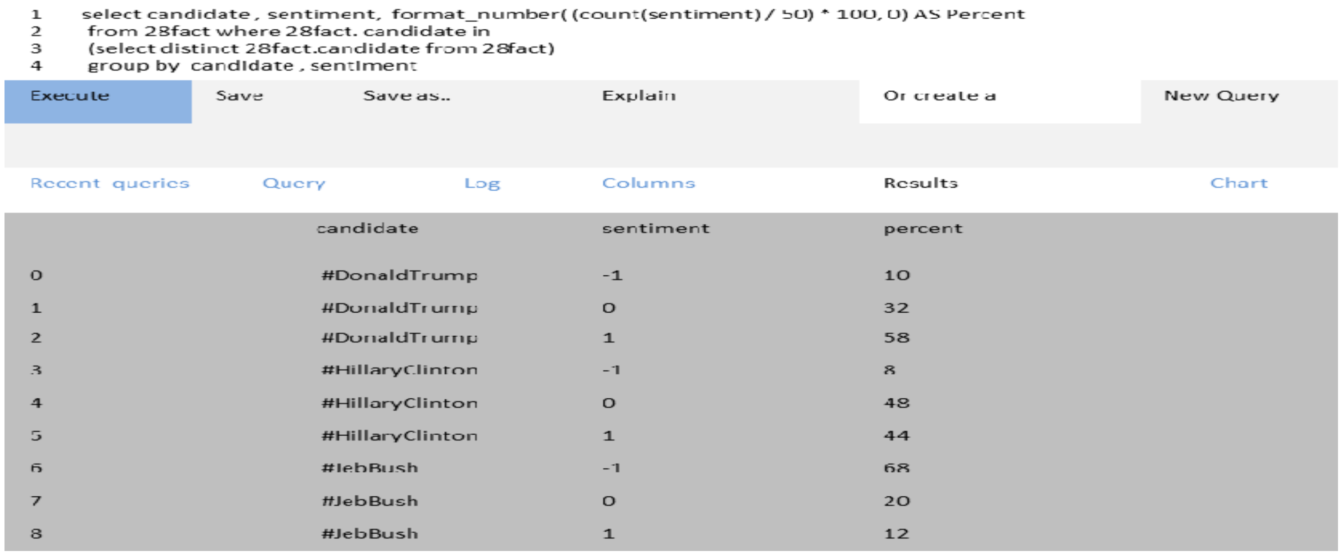
**Using Hive:**

We created tables in Hive as per the physical schema that was designed for the warehouse and populated them with data from Hadoop file system.

Once the data is in place and the warehouse was operational, we ran user queries to perform analytics and got aggregated results from the fact table as shown in Fig. 6.

**Using MapReduce:**

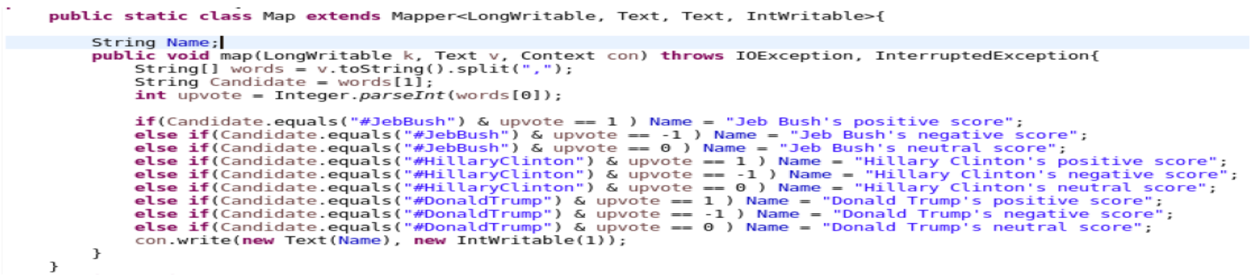
We wrote a manual MapReduce code to process data directly from Hadoop fs and were able to retrieve the same results as the ones from Hive UI. Fig. 7 shows mapper code, followed by Fig. 8. showing reducer code. Fig. 9 depicts result for MapReduce and the results are similar to the ones obtained using Hive



```
select candidate , sentiment, format_number((count(sentiment) / 50) * 100, 0) AS Percent
 from 28fact where 28fact. candidate in
 (select distinct 28fact.candidate from 28fact)
 group by candidate , sentiment
```

| | candidate | sentiment | percent |
|---|---|---|---|
| 0 | #DonaldTrump | -1 | 10 |
| 1 | #DonaldTrump | 0 | 32 |
| 2 | #DonaldTrump | 1 | 58 |
| 3 | #HillaryClinton | -1 | 8 |
| 4 | #HillaryClinton | 0 | 48 |
| 5 | #HillaryClinton | 1 | 44 |
| 6 | #JebBush | -1 | 68 |
| 7 | #JebBush | 0 | 20 |
| 8 | #JebBush | 1 | 12 |

Fig. 6. Analytics performed using Hive (Fact Aggregation)

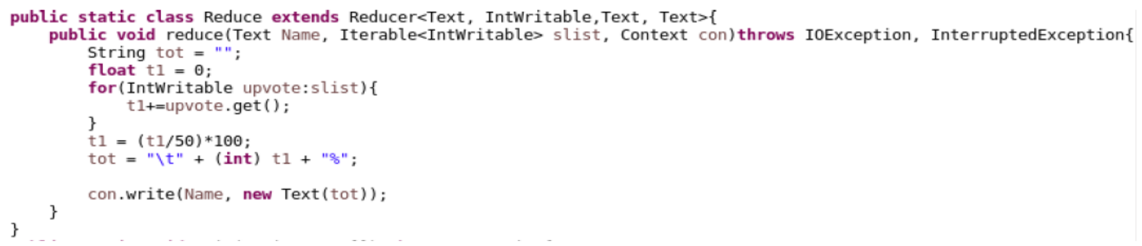


```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>{

    String Name;
    public void map(LongWritable k, Text v, Context con) throws IOException, InterruptedException{
        String[] words = v.toString().split(",");
        String Candidate = words[1];
        int upvote = Integer.parseInt(words[0]);

        if(Candidate.equals("#JebBush") & upvote == 1 ) Name = "Jeb Bush's positive score";
        else if(Candidate.equals("#JebBush") & upvote == -1 ) Name = "Jeb Bush's negative score";
        else if(Candidate.equals("#JebBush") & upvote == 0 ) Name = "Jeb Bush's neutral score";
        else if(Candidate.equals("#HillaryClinton") & upvote == 1 ) Name = "Hillary Clinton's positive score";
        else if(Candidate.equals("#HillaryClinton") & upvote == -1 ) Name = "Hillary Clinton's negative score";
        else if(Candidate.equals("#HillaryClinton") & upvote == 0 ) Name = "Hillary Clinton's neutral score";
        else if(Candidate.equals("#DonaldTrump") & upvote == 1 ) Name = "Donald Trump's positive score";
        else if(Candidate.equals("#DonaldTrump") & upvote == -1 ) Name = "Donald Trump's negative score";
        else if(Candidate.equals("#DonaldTrump") & upvote == 0 ) Name = "Donald Trump's neutral score";
        con.write(new Text(Name), new IntWritable(1));
    }
}
```

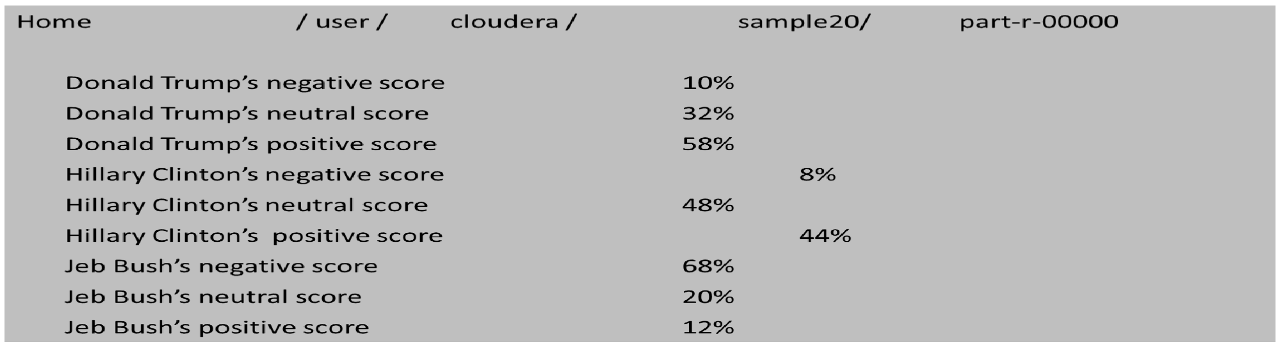
Fig. 7. Mapper code



```
public static class Reduce extends Reducer<Text, IntWritable,Text, Text>{
    public void reduce(Text Name, Iterable<IntWritable> slist, Context con)throws IOException, InterruptedException{
        String tot = "";
        float t1 = 0;
        for(IntWritable upvote:slist){
            t1+=upvote.get();
        }
        t1 = (t1/50)*100;
        tot = "\t" + (int) t1 + "%";

        con.write(Name, new Text(tot));
    }
}
```

Fig. 8. Reducer code



```
Home                 / user /      cloudera /              sample20/         part-r-00000

     Donald Trump's negative score                  10%
     Donald Trump's neutral score                   32%
     Donald Trump's positive score                  58%
     Hillary Clinton's negative score                      8%
     Hillary Clinton's neutral score                48%
     Hillary Clinton's  positive score                     44%
     Jeb Bush's negative score                      68%
     Jeb Bush's neutral score                       20%
     Jeb Bush's positive score                      12%
```

Fig. 9. Output from MapReduce code

### *C.* Significance of Data Processing

The ultimate goal of this system is to have a scalable data warehouse capable of providing reports, and analytics based on data aggregation and facts collected.

The above two techniques stand as the cornerstone to achieve these features of a data warehouse, while making it more scalable, and giving it the ability to provide faster results using MapReduce.

While Hive created MapReduce jobs in the backend by converting its SQL like query syntax into Jobs, it can be slow at times and the user has no control over updating it. The disadvantage of this being that the jobs might take time to complete and there might be a delay in response back to the user about query result.

On the other hand, using a manual MapReduce code, gives the user an ability to actually see and modify the working of the code, thereby finding ways to make it more efficient. By optimizing the code to get results quickly will help in improving the overall end-user performance.

## V. CONCLUSION / FUTURE SCOPE

The paper implemented a system that would be a new research direction to building more efficient data pipelines / architectures for the research community. This novel approach is general and can be applied to almost any corpus of data, with specific steps to be taken care of at each stage in the pipeline. The code base used to implement the system is generic and can be re-used as and when required. By implementing the system, we have addressed the issues in big data warehouse research directions spanning analytics, scalability, end-user performance and visualization. We have successfully implemented a data mart in big data ecosystem from unstructured data.

The future work to this paper would be to implement a pipeline to build a structured data mart in the big data ecosystem and migrate the data from relational data warehouses to this new system. And later connect the data marts from different sources using fact tables so there is communication between the two. The idea here is to have a way where the data from legacy systems and big data systems could communicate with one another to provide powerful results and help in better decision making by building them on the same environment.

## REFERENCES

[1] Kimball, Ralph, "*The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*", John Wiley & Sons, 1998.

[2] A. Cuzzocrea, L. Bellatreche, & Y. Song, I, "Data warehousing and OLAP over big data: Current Challenges and Future Research Directions", In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*, pp. 67-70. ACM, 2013.

[3] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, & C. Welton, *"MAD skills: new analysis practices for big dat*a", Proceedings of the VLDB Endowment, vol. 2, no. 2, pp. 1481-1492, 2009.

[4] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, & S. Babu. *"Starfish: A Self-tuning System for Big Data Analytics."* In *CIDR*, vol. 11, pp. 261-272. 2011.

[5] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, & R. Murthy, "Hive-a petabyte scale data warehouse using hadoop", In Data Engineering (ICDE), IEEE 26th International Conference on , pp. 996-1005, March 2010.

[6] A. Floratou, U.F. Minhas & F. Özcan, "Sql-on-hadoop: Full circle back to shared-nothing database architectures", Proceedings of the VLDB Endowment, vol. 7, no. 12, pp. 1295-1306, 2014.

[7] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, & M. Yoder, *"Impala: A modern, open-source SQL engine for Hadoop",* In Proceedings of the Conference on Innovative Data Systems Research (CIDR'15) ,2015.

[8] M. Chevalier, M. Malki, A. Kopliku, O. Teste, & R. Tournier, *"Implementing Multidimensional Data Warehouses into NoSQL",* In 17th International Conference on Entreprise Information Systems (ICEIS), Barcelona,Spain,2015.