Big Data Normalization for Massive Databases

Suresh Varma Penmasta
Department of CSE
Adikavi Nannaya University
Rajahmundry, A.P, India

K. B. V. Brahma Rao
Department of MCA
B. V. Raju College
Bhimavaram, A. P, India

R Krishnam Raju Indukuri
Department of MCA
B. V. Raju College
Bhimavaram, A. P, India

***Abstract*— Data normalization is a technical database operation performed by a database analyst with the assistance of normalization tools; the goal is to associate similar forms of the same data item into a single data form. Based on how close the association of these various data permutations of the "same" data item are, the data variants can be normalized as a first normal, a second normal, or a third normal form, with the third normal form representing the loosest association of two data forms. This paper reviews the theoretical and experimental for retrieving the required data from large databases in computational complexity with respect to Comparison of CPU time taken for data retrieving from the database before and using normal forms, Elimination of data redundancy and Less volume of data stores in Main Memory.**

***Keywords*— Big Data, MPP, database, normalization, analytics, Map/Reduce, Broadcast Join, HDFS, commodity hardware**

I. INTRODUCTION

"Information explosion is the rapid increase in the amount of published information and the effects of this abundance of data. As the amount of available data grows, the problem of managing the information becomes more difficult, which can lead to information overload." [1]

Big Data analytics is rapidly becoming a commonplace task for many companies. For example, banks, telecommunication companies, and big web companies, such as Google, Facebook, and Twitter produce large amounts of data. Nowadays business users also know how to monetize such data. For example, various predictive marketing techniques can transform data about customer behavior into great monetary worth. The main issue, however, remains to be implementations and platforms fast enough to execute ad-hoc analytical queries over Big Data. Until now, Hadoop has been considered a universal solution, but it has its own drawbacks, especially in its ability to process difficult queries, such as analyzing and combining heterogeneous data, and performing fast ad-hoc analysis.

To store digital data is not sufficient, its needs to be queried as well. But with such huge volumes of data, there is a need to look at query algorithms from a different perspective. For instance, algorithms need to be storage-aware in order to load and retrieve data efficiently. There is tremendous amount of research being undertaken towards creating such algorithms, for example, Google's BigTable [8] or Facebook's Cassandra [1] which is now open-source and is maintained by the Apache Software Foundation. Many of the leading Information Technology companies have invested a lot into this research and have come up with a number of innovative ideas and products.

One of the most common operations in query evaluation is a Join. Joins combine records from two or more tables, usually based on some condition. They have been widely studied and there are various algorithms available to carry out joins. For example, the Nested-Loops Join, the Sort-Merge Join and the Hash Join are all examples of popular join algorithms. These algorithms (and more) are used for joining two as well as more datasets. But more often than not, when multiple datasets are involved, selectivity factor is exploited to structure the order in which the joins are made. Selectivity factor can be defined as the fraction of the datasets involved in the join that will be present in the output of the join. Join Algorithms have been studied extensively with many variants existing for each algorithm. For instance, Hash-Join itself has three different variations – Simple Hash Join, Grace Hash Join and Hybrid Hash Join [17].

The MapReduce framework is a widely used programming paradigm for distributed environments [2]. MapReduce provides an abstraction away from the details of parallelizing computation; the framework automatically divides a job into individual tasks, handles scheduling of individual tasks, distributes data and deals with machine failures. The basic MapReduce model expresses computations as a 'Map' and a 'Reduce' function. Hadoop is a framework for the execution of MapReduce jobs. Classic Hadoop has been widely used and studied since its release, but we focus on the more recently developed Hadoop YARN [3]. Both Classic Hadoop and Hadoop YARN use the idea of dividing resources into logical partitions (called 'slots' and 'containers' respectively) which are as-signed to executing tasks.

Map/Reduce framework hides management of data and job partitioning from the programmer and provides in-built fault-tolerance mechanisms. This lets the programmer concentrate on the actual problem at hand instead of worrying about the intricacies involved in a distributed system. It was designed for processing large amounts of raw data (like crawled documents and web-request logs) to produce various kinds of derived data (like inverted indices, web-page summaries, etc.). It is still a prominently used model at Google for many of its applications and computations [9]. Map/Reduce was not developed for Database Systems in the conventional sense. It was designed for computations that were conceptually quite straightforward, but involved huge amounts of input data. For example, finding the set of most frequent queries submitted to Google's search engine on any given day. It

is very different from the traditional database paradigm, in that it does not expect a predefined schema, does not have a declarative query language and any indices are the programmers prerogative. But at the same time, Map/Reduce hides the details of parallelization, has built-in fault tolerance and load balancing in a simple programming framework. The novel idea was so appealing that it led to an open-source version called Hadoop. Hadoop has become extremely popular in the past few years and boasts of big Web 2.0 companies like Facebook, Twitter and Yahoo! as part of its community.

For relational datasets, normalization is applied to minimize data redundancy so there is only one way to know a fact. The goal is relevant to big data analytics because factual confusion generated by disparate data variations can impair one's ability to arrive at accurate information. If the pathway to meaningful results from big data queries is clouded with these inaccuracies, enterprises begin to lose on their big data investments because the results they get from analytics are diluted by the inaccurate data the analytics are using.

To implement reasoning with uncertainty, it must be concerned with three things, they are:

1. To represent uncertain data
2. To select or add two or more different parts of data with uncertainty.
3. To draw inference using this uncertain data

II. RELATED WORK

Map/Reduce was primarily designed at Google for use with its web-indexing technologies. These included things like keeping track of the web-pages crawled so far, creating inverted-indices from them and summarizing the web-pages for search-result views. Over time, they started considering it for more interesting computations like query processing on the raw or derived data that they already had. This led to its wide-spread adoption and led to uses that were not envisioned at the time of designing. Companies have started using Map/Reduce to manage large amounts of data [11]. And when there are multiple datasets involved, there will always be a need for joining these datasets. The goal of this research is to test the viability of Map/Reduce framework for joining datasets as part of database query processing. There are three main contributions of this research: 1. Comparison of CPU time taken for data retrieving from the database before and using normal forms. 2. Elimination of data redundancy. 3. Less volume of data stores in Main Memory.

III. MODEL

Apache Hadoop is an open-source software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS (Hadoop Distributed File System) is highly fault tolerant and designed using low-cost hardware. The general structure of HDFS is described in the figure Fig.1.

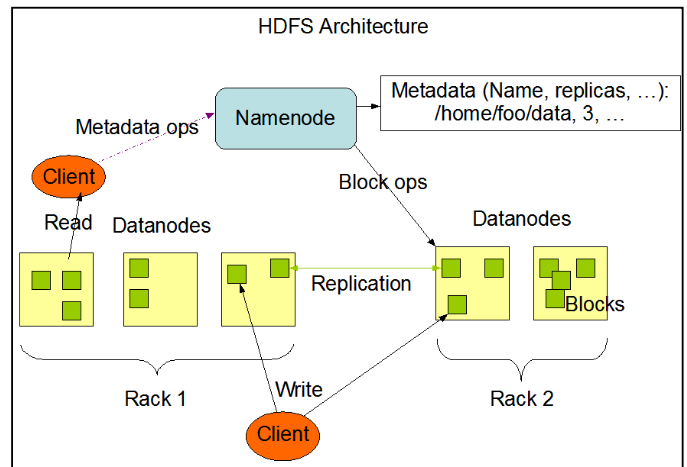

Fig. 1. General Structure of HDFS

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing. The following diagram describes a query processing in HDFS using normal forms to produce the output.

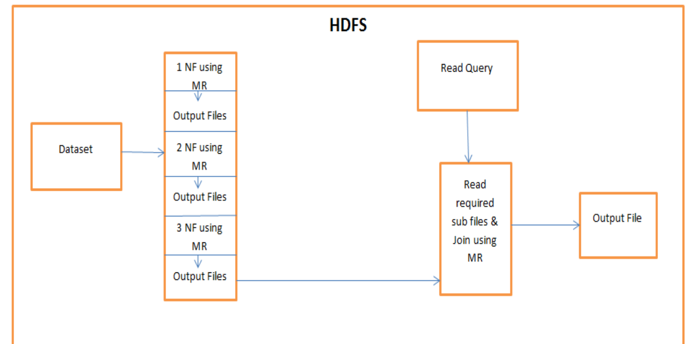

Fig. 2. Model of HDFS for using NFs

Terminology and notations used in the above Fig. 2 are MR: MapReduce NF: Normal Form

IV. TWO STAGE COMPUTATION IN HDFS FOR JOINING FILES

Avito [20] proposed an Anchor modeling for normalizing the data. For an Anchor model the hand-made execution plan aims to maximize merge join utilization. Hash join can be almost as fast as merge, but it requires a lot of RAM. If limited, the join must again be spilled during execution. Broadcast Join was studied by Blana [6]. If one of the datasets is very small, such that it can fit in memory, then there is an optimization that can be exploited to avoid the data transfer overhead involved in transferring values from Mappers to Reducers. This kind of scenario is often seen in real-world applications. For instance, a small users database may need to be joined with a large log. This small dataset can be simply replicated on all the machines. This can be achieved by simply using –files or -archive directive to send

the file to each machine while invoking the Hadoop job. Broadcast join is a Map-only algorithm.

Map Phase

The Mapper loads the small dataset into memory and calls the **map** function for each tuple from the bigger dataset. For each (key, value), the **map** function probes the in memory dataset and finds matches. This process can be further optimized by loading the small dataset into a Hashtable. It then writes out the joined tuples. The below shown configure function is part of the Mapper class and is called once for every Mapper. The below code reads the 'broadcasted' file and loads the same into an in memory hash table.

```
public void configure(JobConf conf) {
    //Read the broadcasted file
    T1 = new File(conf.get("broadcast.file"));
    //Hashtable to store the tuples
    ht = new HashMap <String , ArrayList <String >>();
    BufferedReader br = null;
    String line = null;
    try{
        br = new BufferedReader (new FileReader(T1));
        while(( line = br.readLine ())!=null)
        {
            String record [] = line.split("\t", 2);
            if(record.length == 2)
            {
                //Insert into Hashtable
                if(ht. containsKey(record [0]))
                {
                    ht.get(record [0]).add(record [1]);
                }
                else
                {
                    ArrayList <String > value = new ArrayList < String >();
                    value.add(record [1]);
                    ht.put(record [0], value);
                }
            }
        }
    }
    catch(Exception e) { e. printStackTrace (); } }
```

Listing 1: Loading the small dataset into a Hash table

The next piece of code is the actual **map** function that receives the records from the HDFS and probes the HashTable containing the tuples from the broadcasted file. Notice that it takes care of duplicate keys as well.

```
public void map( LongWritable lineNumber , Text value , OutputCollector <Text , Text > output , Reporter reporter)
throws IOException {
    String [] rightRecord = value.toString ().split("\t" ,2);
    if(rightRecord.length == 2)
    {
        for(String leftRecord : ht.get( rightRecord [0]))
        {
            output.collect(new Text( rightRecord [0]) , new Text (leftRecord + "\t"
            + rightRecord [1]));
        }
    }
}
```

Listing 2: Broadcast Join Map Function

Broadcast Join benefits from the fact that it uses a small 'local' storage on the individual nodes instead of the HDFS. This makes it possible to load the entire dataset into an in-memory hash table, access to which is very fast.

V. EXPERIMENTAL EVALUATIONS

We have considered the following type of sample data in the experimental evaluation of our proposed model.

Project Code	Project Name	Project Manager	Project Budget	Emp No	Emp Name	Dept No	Dept Name	Hourly Rate
PC010	Reservation System	Mr. Ajay	120500	S100	Mohan	D03	Data Base	21.00
PC010	Reservation System	Mr. Ajay	120500	S101	Vipul	D02	Testing	16.50
PC010	Reservation System	Mr. Ajay	120500	S102	Riyaz	D01	IT	22.00
PC011	HR System	Mr. Charu	500500	S103	Pavan	D03	Data Base	18.50
PC011	HR System	Mr. Charu	500500	S104	Jitendra	D02	Testing	17.00
PC011	HR System	Mr. Charu	500500	S315	Pooja	D01	IT	23.50
PC012	Attendance System	Mr. Rajesh	710700	S137	Rahul	D03	Data Base	21.50
PC012	Attendance System	Mr. Rajesh	710700	S218	Avneesh	D02	Testing	15.50
PC012	Attendance System	Mr. Rajesh	710700	S109	vikash	D01	IT	20.50

Table 1 Core Dataset

A series of experiments conducted increasing the data (number of tuples), noted CPU time, redundancy data and amount of memory used. While the data is increasing, runtime, redundancy and usage of memory is increasing.

Experiment 1: Performance of query output using core dataset

The Normal Forms are not applied to the dataset. But MapReduce is used to get the require output for the given query.

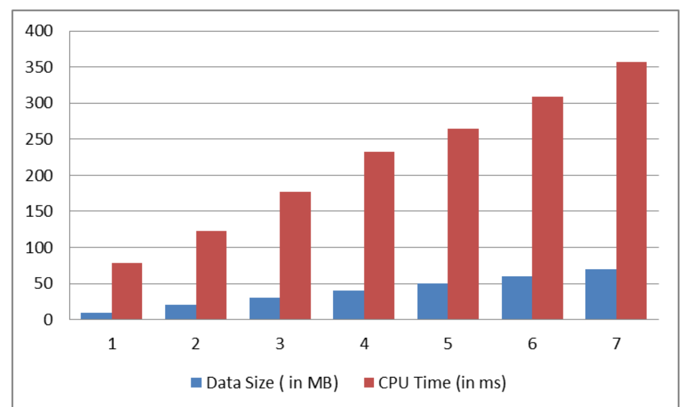

Fig. 3. Comparison between Data Size and CPU Time Without using Normal Forms

Experiment: 2 Performance of query output using Normalized Data

The Normal Forms are applied up to third using MapReduce and produced multiple files by removing redundancy. The same query is used on this data to produce the required output by joining the files using Broadcast Join.

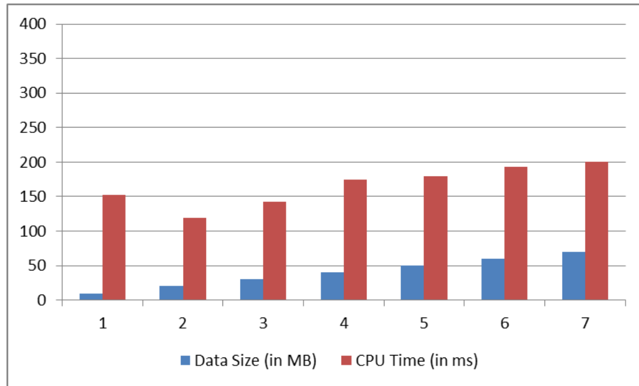

Fig. 4. Comparison between Data Size and CPU Time with using Normal Forms

VI. CONCLUSION

To produce the required result for the given query by joining files using Broadcast Join technique after applying normal forms on the core data takes less number of CPU cycles, removes redundancy and used less memory space. Experimental results show that Two-stage computation has better than Anchor modeling.

REFERENCES

[1] Apache cassandra.
[2] Apache hadoop. Website. http://hadoop.apache.org.
[3] Cloudera inc. http://www.cloudera.com.
Hadoop wiki - poweredby. http://wiki.apache.org/hadoop/PoweredBy.
[4] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. Proc. VLDB Endow., 2(1):922–933, 2009.
[5] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In SIGMOD '10: Proceedings of the 2010 international conference on Management of data, pages 975–986, New York, NY, USA, 2010. ACM.
[6] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. Proc. VLDB Endow., 1(2):1265– 1276, 2008.
[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.
[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.
[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. SIGOPS Oper. Syst. Rev., 37(5):29–43, 2003.
[10] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a notso-foreign language for data processing. In SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1099–1110, New York, NY, USA, 2008. ACM.
[11] K. Palla. A comparative analysis of join algorithms using the Hadoop map/reduce framework. Master's thesis, School of Informatics, University of Edinburgh, 2009.
[12] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data, pages 165–178, New York, NY, USA, 2009. ACM.
[13] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? Commun. ACM, 53(1):64–71, 2010.
[14] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P.Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow., 2(2):1626–1629, 2009.
[15] J. Venner. Pro Hadoop. Apress, 1 edition, June 2009.
[16] S. Viglas. Advanced databases. Taught Lecture, 2010. http://www.inf.ed.ac.uk/teaching/courses/adbs.
[17] T. White. Hadoop: The Definitive Guide. O'Reilly Media, 1 edition, June 2009.
[18] H.-C. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. Map-reduce-merge: simplified rrelational data processing on large clusters.Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1029–1040, 2007.
[19] Nikolay Golov1 and Lars Ronnback2. Big Data Normalization for Massively Parallel Processing Databases, International Conference on Conceptual Modeling, 2015.
[20] Jairam Chandar. Join Algorithms using Map/Reduce, Magisterarb. University of Edinburgh, 2010 - inf.ed.ac.uk