Indian Journal of Science and Technology, *Vol 11(18), DOI: 10.17485/ijst/2018/v11i18/119112, May 2018* *ISSN (Print) : 0974-6846* *ISSN (Online) : 0974-5645*

Hash Semi Join MapReduce to Join Billion Records in a Reasonable Time

Marwa Hussien Mohamed[1]*, Mohamed Helmy Khafagy[2] and Mohamed Hasan Ibrahim[3]

[1]Department of Information Systems, Arab Academy for Science, Technology and Maritime Transport, Cairo, Egypt; eng_maroo1@yahoo.com
[2]Department of Computer Science, Fayoum University, Cairo, Egypt;Mhk00@fayoum.edu.eg
[3]Department of Information Systems, Fayoum University, Cairo, Egypt; mhi11@fayoum.edu.eg

Abstract

Objective: MapReduce is a programming model used to support massive data sets. Big data are the most important issue today to analyze these data. **Methods/Statistical Analysis**: MapReduce is used to discover hidden patterns and relations in data to get more helpful information by using two simple functions map and reduce written by the programmer, it includes load balancing, fault tolerance and high scalability. The most important operation in data analysis are join, but MapReduce is not directly support join. **Findings:** This paper explains two-way MapReduce join algorithm, semi-join and per split semi-join and proposes new algorithm hash semi-join that used hash table to increase performance by eliminating unused records as early as possible and apply join using hash table rather than using map function to match join key with other data table in the second phase but using hash tables isn't affecting on memory size because we only save matched records from the second table only. Our experimental result shows that using a hash table with hash semi-join algorithm has higher performance than two other algorithms while increasing the data size from 100 million records to 50 billion. **Application/Improvements:** Running time is increased according to the size of joined records between two tables using 30 machines to run our data but our algorithm has the better running time than other algorithms.

Keywords: Hadoop, Hash Semi Join, MapReduce, Two-Way Join

1. Introduction

The most important issue in researches today is analyzing and process large data sets[1]. MapReduce-based-system is designed to process and analyze these data sets to gain more knowledge and helpful information to support industry and academia researches[2].

MapReduce are a programming model arises from 2004 by Google[2]. It's used to analyze and support heterogeneous datasets. It's becoming more popular according to the simplicity interface, handling fault tolerance, load balancing and high scalability.

MapReduce[3] utilizes the simplest programming according to using two main functions map function and Reduce function these two important functions are written by the programmer to gain his task and everything like handling fault tolerance and load balancing are done by the framework by default. Every record of data assigned to map task are generated as a key value pair then this output are sent to the reduce task to do the main operation assigned by the programmer to group these values with the same key to generating final output[4].

Apache Hadoop[3] is an open source framework was developed by Google. It's used to handle heterogeneous data, scales large number of nodes and automatically handling node failures[5]; it's used to distribute data processing and it's written in java.

Hadoop Distributed File System (HDFS)[6,7] was a file system used by Hadoop to store file system Metadata and application data separately. It has two separate servers to store Metadata at name node and application data at data node. It uses replication of data to protect data rather than using RAID[8]. All file data are stored in more than one data node.

According to the rapid increase in data size[9], we need to perform join operation to find hidden pattern and

*Author for correspondence

valuable information, but to multiple data set MapReduce has some limitations to perform join operations because MapReduce used Network connection to sent entire datasets among nodes in the cluster That may cause performance bottlenecks.

MapReduce isn't designed to match or combine information from two data sources. So that it has some limitation for join[10]. Most researches studies like equi-join it's used data flow management for key equality MapReduce, MapReduce merge applies some changes in MapReduce to get join prediction result by adding merge phase and programmer must write the code for distribution of records.

Researchers over 30 past years are using semi-join and hash tables in the database area, to do join operations on massive data sets[9]. However, MapReduce is designed to process single large dataset as input so that isn't have any data structure and database design like indexes or filters or query execution plan as in the database[11]. Join can be two ways for joining two tables or Multiway[12] for joining more than two tables.

Semi join between two tables returns all possible rows matches found in the first table and in the second table to do join as in the database[13]. Hash semi-join in the database is used to return all matched records from two tables only once as if matched records are repeated it gets only the first one matched. It has two types hash left semi-join it's used when the IN table is smaller than the FROM table and hash right semi-join it used when the IN table is larger than the FROM table[13].

In this paper, to improve join performance in Hadoop we using hash tables with semi-join MapReduce. We apply hash semi-join techniques only for two data sets. We discuss and compare semi join MapReduce and per split semi join with our new algorithm[14,15]. Semi join MapReduce solved the problem for deleting usefulness tuples as early as possible compared to others MapReduce join techniques[14]. Our new techniques are to invert the second phase at semi-join that's used map function and it's used in it () and close () functions with every records in search that takes more time to save the output from the first phase that contain all join key in hash-table and distribute the second table via distributed cache to go throw and find matched records to write it in hash table to get new output file has all joined records data from the second table matched with the join key then using broadcast join to get final join result. So that we can do join between two large tables and deleting all unused records to join operation to increase performance and reduce communication overhead.

The rest of this paper is organized as follow: Section 2 discuss semi-join and per-split semi-join algorithm related to our work. Section 3 discuss MapReduce join problem definition and how can our new algorithm solve this problem, new proposed hash semi join architecture, implementation, cost model for this algorithm and pseudo code for hash semi join algorithm. Section 4 we present our experimental results and performance analysis. Finally, we conclude and discuss the conclusion and future work.

2. Previous Work

Semi join MapReduce and per-split semi-join is a two-way join algorithm was firstly proposed in[14] these two algorithms are used when the size of two tables are extremely large.

2.1 Semi Joins

Semi Join MapReduce algorithm[14,15] often, used when the size of one data sets is extremely larger than the other. Multiple records will not be used for join so that deleting these usefulness records will affect the network workload and size of datasets to join. Semi join does join rapidly than map side join and reduce side join algorithms with huge data size. We use order table has a foreign key to doing join O_CUSTKEY and use customer table has a primary key to doing join C_CUSTKEY.

Phase 1:

Using the map and reduce function. First map phase used to generate key-value pair from the large table whose join key is foreign key not unique key at table structure (order table) whose size of table data are 400 million records. So we use the map function to generate join key and use reduce function to eliminate redundant records from join operation. Map function used to generate key-value pair from every record in the table but it generate only key from join key with no value and save this value in HDFS then we can sort this data that's be too small compared to all table records. Reduce the function used to group all records equally together and then eliminate redundant records then output file 1 that contain all join key from the first table this file is small enough to fit in memory. Using a hash table to save join key in memory isn't any cost and no overhead on network load and memory size. Output 1 Table after eliminating redundant records contain all customer ID do all orders table records from 200 million records in the customer table originally.

Phase 2:

The output file from phase 1 size it's small enough to fit in memory and customer table contain 200 million customers broadcast all records in this file using HDFS.

We using map function to get all key-value pair from customer table and store result in HDFS and partition this table according to number of splits and if we found key from first table equally to key from customer table we write in new HDFS output file all customer data records this file with all records has join key with customer data applying order table. Every time we check join key table we use in it () and close () function for every record that's got more time and we do this with every time iterate throw second table splits.

Phase 3:

Using broadcast join as map only phase. We load first table from HDFS with data about 400 million records and use map function to generate key value pair and using hash table to load the output file 2 from second phase and join records from two tables and generate final output files has 400 million joined records. Last phase cost by using a hash table and using the map function to generate key value pair.

This algorithm problems are extra scan for the large table orders and every time needed to use map function we do key value pair and using init () and close () function and for the second phase we go throw every record in second table to find matching key to doing join with join key output file from the first phase[16].

2.2 Per Split Semi-Join

Per-split semi-join algorithm[14,15] used to join tables one of these two tables are extremely large than the other and it enhance the problem for that may be one of unique join key from the first phase in semi-join not joined by a record in the second table also it has three phases to do join but second phase are highly cost.

Also we use order table has a foreign key to doing join O_CUSTKEY and use customer table has a primary key to doing join C_CUSTKEY.

Phase 1:

Using map function only to generate key/value pair from order table that's have 400 million records and generate number of files contain all join key with no value and without elimination of redundant records output files and we save this file into HDFS.

Phase 2:

Using a full MapReduce job it's used to load customer table with 200 million record into main memory and using map to generate key value pair and using hash table to iterate throw output file from phase1 which saved in HDFS we partition customer table into number of splits. We iterate with every part of output file 1 with every split of second table if a join key is matched we output this record and adding a tag to table name, we use in it () and close () function that takes long time for every record and then using reduce function to group all records joined from output phase 1 with customer table. Output file 2 size are larger than output 2 from semi-join phase 2 because we aren't eliminating redundant records like semi-join.

Phase 3:

Using map only function we load orders table from HDFS and do key value pair for all records and do direct join according to the table tag number added to the output file from the second phase. Output files with 400 million joined records. Per split semi-join third phase are low cost than third phase in semi-join algorithm[17] and second phase are used to join every part of first table with the same part of the first table that's makes direct join in third phase are more easily and second phase are highly cost and has more time in small data size.

3. Hash Semi-Join MapReduce

Hash semi-join algorithm used to solve the problem of joining two tables one of them has more records usefulness in join and we need to delete this records as early as possible to increase running performance and reduce network load and bottleneck when we need to buffer million and billion of records at reduce phase .

3.1 Problem Definition

In this work, we focus on two-way join MapReduce only. We need to solve the problem of semi-join MapReduce and presplit semi-join MapReduce[14], this type of join algorithms joining only two large tables it depends on deleting unused records as early as possible to reduce bottlenecks, but this algorithms suffer from scanning for the large table more time and may be filtered records will not join with another table. We use hash table in the first phase after filtering join key by using first map and reduce function applied to the first table, we save output from this step directly to hash tables and load the second table via distributed cache to solve the problem of semi-join and presplit semi-join to use map function the needs to init and close function for every records while matching the join key. Also by using hash tables we get high performance[18].

3.2 Hash Semi-Join Architecture

Hash semi-join reduce in cost for the second phase in semi-join and per split semi-join by using hash table based aggregation and no need to use map function to join records in the second phase by sending output result from first MapReduce phase to be saved directly to hash table in memory using read and write function to file only. Figure 1 shows hash-Semi-Join architecture. Hash semi-join has two phases the first phase are a MapReduce job and the second phase are map only job.

Phase 1:

We use order table and customer table to do join by using customer key. We load order table in HDFS and do map function to generate key value pair but this map function only get the key without any value and then using reduce function to eliminate redundant records from output files. order table originally size 28,192,375,890 byte 400 million records and after using map function HDFS save output records 1,288,970 byte directly into hash table and we load into HDFS second table customer and loop throw hash table and customer table if join key matched we write records data to output file 2 this file has all customer data were joined with orders table.

Phase 2:

We load firstly order table and by using map function to generate key value and by using broadcast join to load output file from first phase and do direct join to get joined records with size 45,387,838,750 bytes has 400 million joined records. Hash semi-join algorithm depends on the first phase on main memory size to save output file which contain joined records but using main memory isn't any effect on join result because output joined records file are small to fit in memory of memory .semi join and presplit semi-join algorithms highly cost in second phase. Hash semi-join increase in performance and reduce cost.

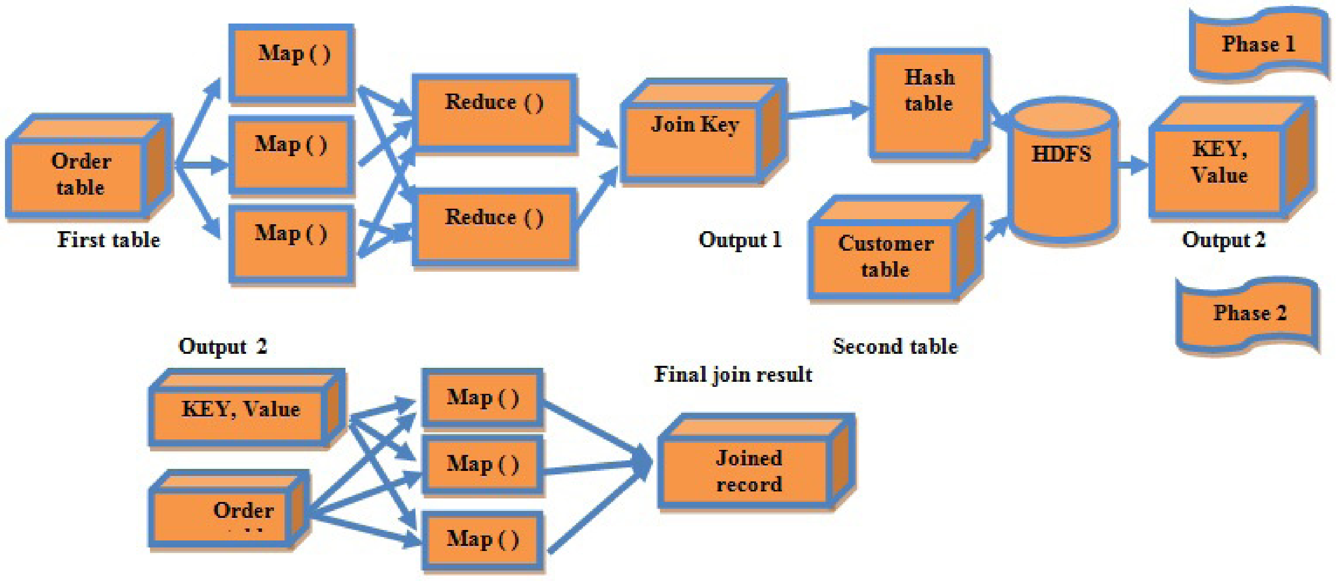

Figure 1. Hussein: Hash-semi-join architecture.

3.3 Hash Semi-Join Example

Hash semi-join example is shown in Figure 2. If we have order table with data OID, CID and type and customer table CID and name firstly we load two tables into HDFS Hadoop Distributed File System we get order table and apply map function to get key join in this table we get all CID customer whose do orders then apply reduce function to eliminate redundant customer ID then we save all data to hash table secondly we load customer table via distributed cache and iterate by using hash table throw customer if we find CID we write all customer data in this table then save this file in HDFS and apply broadcast join to load order table and do map function to make join between new table and orders data by adding all customer data finally we get joined output records.

3.4 Hash Semi-Join Cost Model

Hash semi-join cost for join two tables (customer table and order table). We copy our tables from local disk using copy from local command to copy data to HDFS this step on master node. When we start running algorithm name node sends parts of data to all data nodes to start first job mappers to apply map function on the first table to read data locally. We have to types of I/O: local read it's reading data from local mode and streaming I/O it's reading data from different nodes and from TPC/IP inner process communication. Every join key records are customer ID numbers sizes are small to get a load in memory. Size of million up to billion records IDs it's may be megabytes or more. So that size of hash semi-join first part is the size of the first table (orders) join keys without repeated IDs.

Output 1 size equally the size of all CID at orders table without repeated customers. Hash semi-join phase 1 cost: Local read: size (order) + size (customer: local I/O read), Data transfer: size (orders) + size (customers), Local write: size (join keys without repeated)

Then we save output files into hash table and reading customer table for I/O local cost to loop for join key with hash table if we find matched we using hash table to write all customer data this are saved in memory and saved output results to HDFS. We have all customer data needed for join records. Output 2 size equally the size of all CID at orders table without repeated customers with full data. Phase 1 part 2 costs: Local read: size (output 1: streaming I/O read) + size (customer: Local I/O read), Data transfer:

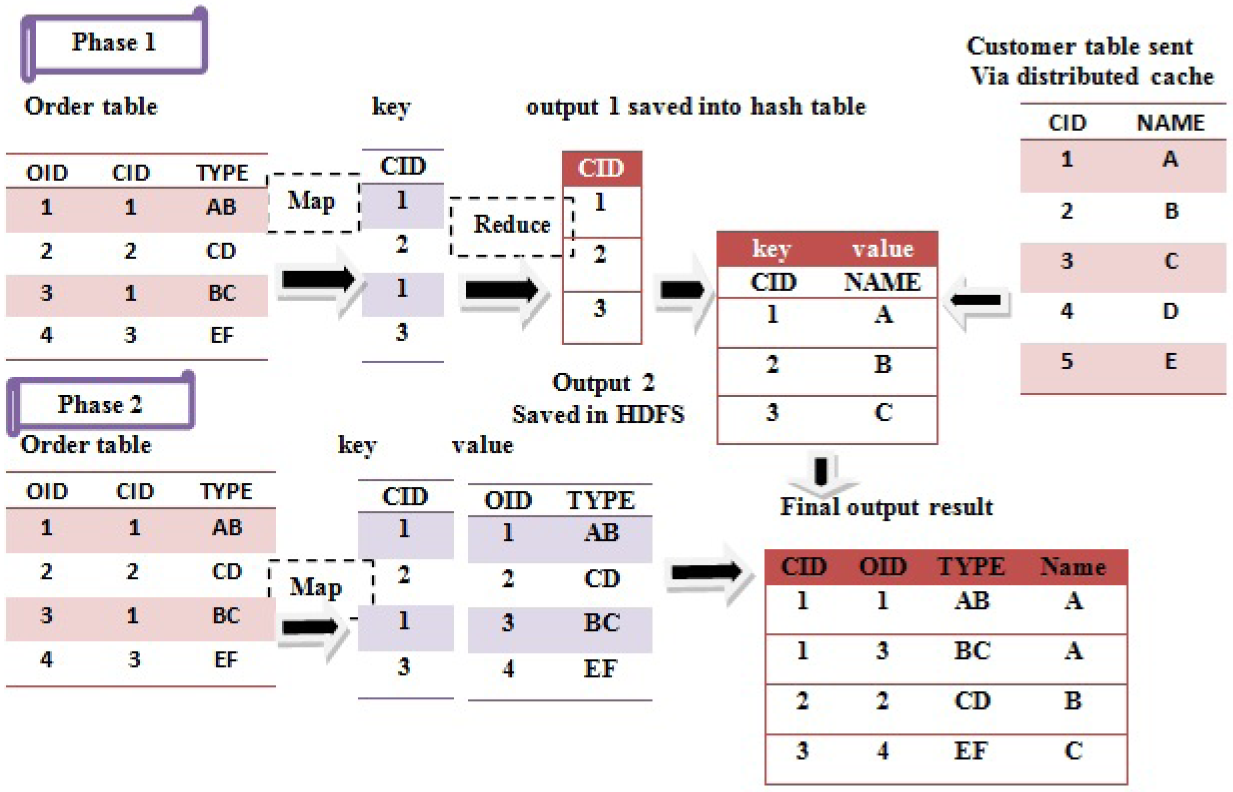

Figure 2. Hussein: Hash-semi-join example.

size (customers) + hash table, Local write: size (join keys without repeated with full data) using a hash table.

In this part we use a hash table to save customer join key data in memory firstly and then save all results to HDFS. Using hash tables reduce in time and give high performance than semi–join while we read and write directly and reduce network bottleneck when we need to write every output records to output file via HDFS but we only send final result only.

Hash semi-join phase 2 costs we only load order data from HDFS and using distributed cache with output 2 from the first phase and do join operation by adding all customer data to order the table. So that we get final join result. Phase 2 costs: Local read: size (output 2) + size (orders) for each map, Data transfer: size (output 2) + size (orders), Local writes: size (joined records at customer table) + size (orders table).

Every time we get new output result we must replicate this data to different nodes through the network to support fault tolerance and balancing this cost must be included in total join cost.

4. Experimental Results

We present experimental results of our implementation. We have 30 cluster machines one of them master node (name-node) and 29 cluster machines are slave node (data node). Cluster configuration consists of Intel Core I5 2.4 GHz processor, 4 GB memory for every node, 500 GB SATA disk and operating system Ubuntu 13.14 Linux with Apache Hadoop release 1.2.1.

4.1 Dataset

We use the TPC-H benchmark[19-23] dataset to evaluate our implementation with original Hadoop. We use two table customers and orders to join according to join key where O_CUSTKEY = C_CUSTKEY where customer table has 50 million record and table orders has 100 million records as the start size for our experiment. We use SJ as abbreviation for semi-join, PSJ as abbreviation for per split semi-join and HSJ as abbreviation for hash semi-join. We apply three experimental results varying in data size to show algorithm performance.

4.2 Experiment 1

We present execution time performance for every join algorithm when the size of the customer table are fixed with 100 million records and increasing the size of order table by 100 million every time. Table 1 shows running time with every algorithm in seconds and Figure 3 shows performance for three algorithms where X-axis show time in seconds and Y-axis show size of two tables per split semi-join has worst time because of highly cost of second phase by partitioning order join key into number of files and keep redundant records for records apply orders. Semi join has more than hash semi-join according to using a second phase to find matched records from order table with join key customer table.

Hash semi-join algorithm has the best time performance at all in increasing the size of data and reduce in time by deleting the second phase and try to use hash table in memory to store joined customer data records apply orders data show that all running time are increased by 1% increasing in running time depend on tables size and number of customer who apply orders with order table and number of customers apply order are not fixed in all data tables.

4.3 Experiment 2

We show join performance with every algorithm when the size of data are increased in two tables. We apply this experiment to show the running time when increasing customer ID reference in order table this take time in last phase. Figure 4 shows that running 100 million and 200 million has more time than 150 million and 300 million

Table 1. Total run time for join algorithms when data size are increased in order table by 100 million and fixed in the customer table 100 million records

Comparison	100 and 100 million	100 and 200 million	100 and 300 million	100 and 400 million	100 and 500 million	100 and 600 million
Per-Split Semi Join	69	112	200	210	215	203
Semi Join	70	130	203	201	201	213
Hash Semi Join	67	102	182	181	175	176

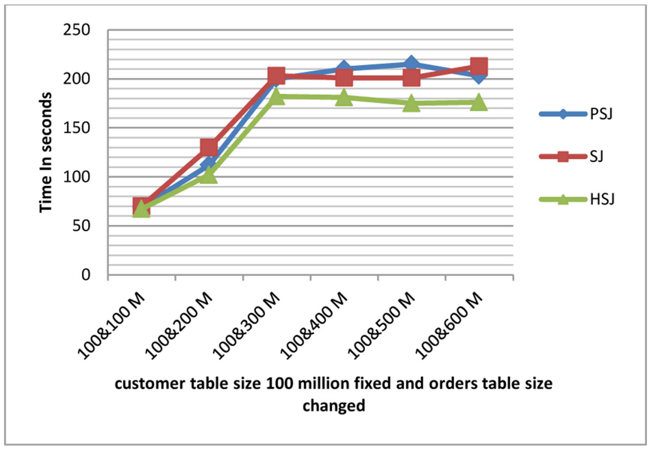

Figure 3. Hussein join algorithm run time by each phase.

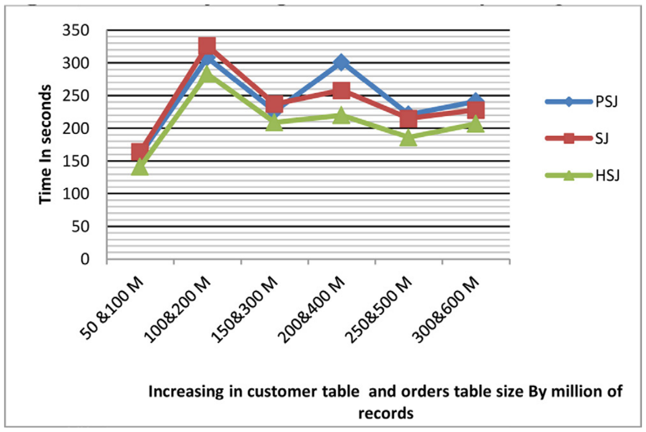

Figure 4. Hussein show algorithm performance when records are increased in two tables.

because last phase in all type of join algorithm get more time to add customer data to order table to get output joined records and size of customer apply orders are not equally in all tables it's increased in 100 million and 200 million experiment highly cost of second phase.

In semi-join and presplitsemi join algorithm but hash semi-join isn't affected because we reduce time by deleting second phase Figure 4 show running time for all algorithms total time. Table 2 shows time performance for every join algorithm hash semi-join has best in performance.

4.4 Experiment 3

We increase the size of two tables by 10 Billion records and size of customer table by 5 Billion records. Figure 5 shows that hash semi-join algorithm has the best time to apply join but per split semi-join better than semi-join when we increase the size of data by 10 Billion records this experiment shows that hash semi-join and presplit semi-join are better than semi-join in running time. Presplit semi-join is running time increased by 12%, semi-join algorithm time increased by 13% and hash semi-join time increased by12% and get high performance in running time for all experiments.

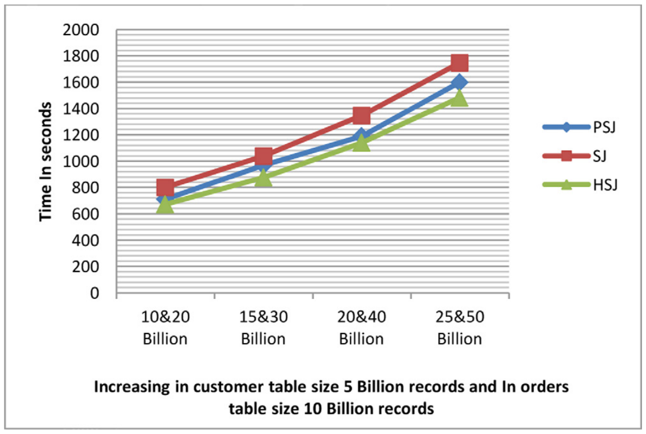

Figure 5. Hussein show algorithm performance when records are increased in two tables.

4.5 Performance Analysis

The main advantage of hash semi-join are reduced in sorting and shuffling costs between Mappers and reducers by reducing in records sizes and sending only records used for the join. Hash semi-join performance depends on some properties:

Table 2. Total run time for join algorithms when data size are increased in two tables firstly the size of the customer table and second size of the order table

Comparison	50 and 100 million	100 and 200 million	150 and 300 million	200 and 400 million	250 and 500 million	300 and 600 million
Per-Split Semi Join	160	308	226	301	221	241
Semi Join	164	326	237	258	215	228
Hash Semi Join	141	283	209	220	186	207

- Number of join key between two tables (when join keys increased performance also increases otherwise join key decreased performance decreased).
- Size of join key saved into hash table to match records with the second table.
- Datasets number of columns is important factor for performance.

5. Conclusion

This work shows new proposed join algorithm hash semi-join that used hash table to join records and eliminate unused records early to avoid shuffling and reduce network load to get high performance compared with previous MapReduce join algorithms semi-join and presplit semi-join. We run new join algorithms with various data size to see the performance while increasing in data size and increasing in matched records between two tables. Our experimental result shows that our algorithm used memory to get high performance and it's being the best one in running time than two others and memory size not affect in cost because we send only records we will use to apply join from customer table and remove other records.

In the future work we want to implement the join algorithms using several datasets benchmarks and increasing number of nodes to show performance. Implement hash semi-join to run on multi-way join and compare performance with others multi-way join algorithms map side join, reduce side cascade join and reduce side one shot join and using reusing output result using hive query language. We can using index to increase join performance with hash semi join.

6. References

1. Changchun Z, Lei W, Jing L. Efficient processing distributed joins with Bloom filter using MapReduce. International Journal of Grid and Distributed Computing. 2013 Jun; 6(3):43–58.
2. Ghemawat J, Sanjay G. MapReduce: Simplified data processing on large clusters. Communications of the ACM Association for Computing Machinery. 2004; 51(1):107–13.
3. Amresh K, Kiran M, Prathap BR. Verification and validation of MapReduce program model for parallel K-means algorithm on Hadoop cluster. International Journal of Computer Applications. 2013 May; 72(8):48–55.
4. Jeffrey D, Sanjay G. MapReduce: Simplified data processing on large clusters. Useni Associationos di communications of the ACM Association for Computing Machinery. 2008 Jan; 51(1):1–13.
5. Hesham HA, Mohamed HK, Ahmed MW. Comparative study load balance algorithms for MapReduce environment. International Journal of Computer Applications. 2014; 106(18):41–50.
6. Kyong H, Yoon J. Parallel data processing with MapReduce: A survey. SIGMOD Special Interest Group on Management of Data. 2011 Dec; 40(4):1–10.
7. Vikas J, Sunil AJ. Join algorithms using MapReduce: A survey. International Conference on Electrical Engineering and Computer Science; 2013 Apr. p. 40–4.
8. Ebada S, Atef G, Mohamed HK. Queue weighting load-balancing technique for database replication in dynamic content web sites. Proceedings of the 9th WSEAS International Conference on Applied Computer Science; Italy. 2009. p. 50–5.
9. Taewhi L, Kisung K, Hyoung-Joo K. Join processing using Bloom filter in MapReduce. Proceedings of the ACM Association for Computing Machinery Research in Applied; 2012 Oct. p. 100–5.
10. Xiaofei Z, Lei C, Min W. Efficient multi-way theta-join processing using MapReduce. PVLDB Proceeding Very Large Data Bases. 2012 Aug; 5(11):1184–95. PMCid: PMC4088276.
11. Andrew P, Erik P, Alexander R, Daniel JA, Samuel M, Michael S. A comparison of approaches to large-scale data analysis. SIGMOD Special Interest Group on Management of Data. International Conference on Management of Data; 2009 Jun. p. 165–78.
12. Foto NA, Jeffrey DU. Optimizing multiway joins in a MapReduce environment. IEEE Transactions on Knowledge and Data Engineering. 2011 Sep; 23(9):1282–98. Crossref.
13. Philip AB, Dah-Ming WC. Using semi-joins to solve relational queries. Journal Association for Computing Machinery. 1981 Jan; 28(1):25–40. Crossref.
14. Spyros B, Jignesh MP, Vuk E, Jun R, Euqene JS, Yuanyuan T. A comparison of join algorithms for log processing in MapReduce. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ACM Association for Computing Machinery; 2010 Jun. p. 975–86.
15. Duong VH, Sucha S, Phayung M. MapReduce join strategies for key-value storage. International Joint Conference on Computer Science and Software Engineering; 2014. p. 164–9.
17. Chandra J. Join algorithms using MapReduce. Publication: Magisterarb: University of Edinburgh; 2010. p. 1–16.
18. Dawei J, Beng CO, Lei S, Sai W. The performance of MapReduce: An in-depth study. PVLDB Proceeding Very Large Data Bases. 2010 Sep; 3(1):472–83.
19. Tournament Players Club. 2017. www.tpc.org

20. Ebada S, Atif G, Mohamed HK. Specification and implementation of dynamic web site benchmark in telecommunication area. Proceedings of the 12th WSEAS World Scientific and Engineering Academy and Society International Conference on Computers. 2008; 12:863–86.
21. Mina SS, Mohamed HK, Samah AS. JOMR: Multi-join optimizer technique to enhance MapReduce job. The 9th International Conference on INFO Romantics and Systems; 2014 Sep. p. 80–6.
22. Marwah NA, Mohamed HK, Fatma AO. HOME: HiveQL optimization in multi-session environment. Proceedings of the 5th European Conference of Computer Science (ECCS14); Geneva, Switzerland; 2014. p. 80–9. PMid: 24585649.
23. Hussien S, Mohamed HK, Fatma AO. JOUM: An indexing methodology for improving join in hive star schema. International Journal of Scientific and Engineering Research. 2015 Mar; 6(3):111–9.