Providing a Method for Effective Implementation of Hash Join in Big Data with Hadoop

RouhAllah Bayramzadeh[1] , Kourosh Nemati [2,*]

[1]Department of Software Engineering, Faculty of Technical and Engineering, Islamic Azad University, Noor, Iran.
[2]Department of Software Engineering, Faculty of Technical and Engineering, Islamic Azad University, Noor, Iran.
*Corresponding author

Summary
Big data are data with special structure that are used to store a large amount of information. To process these data, distributed methods such as MapReduce are required. The structure of MapReduce in software such as Hadoop is consisted of a distributed system with two phases of map and reduce. Effective implementation of orders in this system has decreased running time and consumption of nodes in distributed system. The purpose of this study was to understand how we can use MapReduce in an efficient way to implement hash join in this programming model. As we know, SQL is one of the high-level declarative languages to process queries. Therefore, this study aimed to implement hash join algorithm as one of the important orders of SQL in Hadoop and its query algorithms and investigate their efficiency to provide an algorithm with minimum running time for processing a large volume of data compared to SQL and improve hash join and query speed.

Keywords:
hadoop; hash join; big data

1. Introduction

Ref [1] discusses, big data have an important role in information technology. These data, due to the emergence of big networks and internet, have been generalized to different aspects of life, because more data are produced and data will be transferred with higher speed. With the emergence of web2.0, the global web network experienced a revolution .Web applications became more interactive and give the freedom to users to interact and collaborate with each other with methods that were not available in the past. Ref [2] discusses, Today, users are not only watching webpages, but provide the content. Websites are full of contents produced by users from weblogs, videos, social-media networks, and other varied technologies of Web2.0 .In this regard, query algorithms are very important. However, when data grow exponentially, other query algorithms cannot be consistent with this rapid acceleration and data will be processed slowly that is not acceptable in today's world where everything is rapid.

A huge volume of input data necessitate data distribution on several computers that each of them creates a node in a cluster that all are related to each other in a distributed system. Moreover, all machines perform a task on data that are locally stored on their disks that leads to distributed process and becomes parallel to big data sets. MapReduce has been developed to use web indexing technology in Google. Over time, experts use this framework for more complicated and interesting computations such as query processing on raw data. This led to the development of in web and applications that were not available at that time. MapReduce manages infrastructures such as parallelism, data distribution, and load balancing and the user has to only concentrate on those computations that are performed in each machine. These computations are divided into map and reduce computations. Also, in big data systems, SQL queries have key role in the relationship between user and system. MapReduce framework requires users to run their applications by coding map and reduce functions.

Although this low level coding in programming applications creates flexibility, it increases debugging of the application. High level declarative languages can simplify the development of applications in MapReduce. Recently, several declarative applications such as SQL and their translators are developed and integrated with MapReduce to support these languages. Examples of these systems can be Pig Latin/Pig and HiveQl/Hive. In practice, these languages have more effective roles compared to handwritten applications. For example, more than 95% of Facebook Hadoope applications cannot be coded by hand, but are produced by Hive. The purpose of this study is to test the success of MapReduce to join relationships in processing queries of databases. As in [3] on Google programming model at MapReduce is reviewed and has concentrated on investigating the above model and provides recommendations for improvements. A study on optimal framework for MapReduce queries [4] was conducted . In this study, a framework was presented to optimize MapReduce queries of SQL. A study entitled "Ysmart: Yet another SQL to MapReduce Translator" presents a new translator that has better performance compared to [5,6]. The most important advantage of Ysmart is that it can translate complex queries that have inter-queries correlations.

Manuscript received May 5, 2026
Manuscript revised May 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.5.18

2. THE PROPOSED METHOD

Ref [7,8] shows the purpose of this study is to investigate hash join in relational query languages and to run it in MapReduce system more effectively. A hash join is an example of join algorithms and is used in running a management system of relational database. Hash join requires a join proposition that means comparing the values of two tables with operator and equals .Also, an algorithm includes several subsidiary methods. In order to improve SQL queries and hash join algorithm on MapReduce system, a solution is needed that reduces the number of references to disk memory.

This algorithm should be runnable in distributed and parallel modes. To run $A \infty B$, two hash tables will be created in the main memory; one for A tuple and the other one for B tuple. These two hash tables are empty at first. In each moment, one of A and B tuples is processed. B hash table is searched to find B tuples that match A tuple. A and adapted tuples by B are immediately displayed as the output.

Then, A tuple will be added to A hash in order to match B tuples that are not processed yet. Algorithm will be ended when all A and B tuples are processed. If hash table cannot be placed in main memory, specific measures are required. To escape from this condition, hybrid hashing and partitioning schema are used. This technique creates the possibility to provide queries very fast. Moreover, this algorithm creates the opportunity to use parallelism; as a result, total queries response time in distribution systems such as MapReduced-based systems decrease.

3. Reduce-Side Cascade Join

In this algorithm, in order to join all databases in one run, two by two joins occur simultaneously. In other words, it is to join two databases at the same time. Considering n, tables $T_1, T_2, T_3, \dots, T_n$. The T_2 join T_1 ، result will join T_3 and all tables will joint in this fashion. This process, in relational algebra language, is shown as follow [9]:

$$\left(\left((T_1 \bowtie T_2) \bowtie T_3\right) \bowtie T_4\right) \dots \bowtie T_{n-1}) \bowtie T_n \qquad (1)$$

4. Optimizing Reduce-Side Cascade Join

It is obvious that joining two databases at the same time does not have efficiency. Usually, the volume of intermediate results is high and occupies very large space. If these results are deleted after the process, still system cascade joins experience burnout. Perhaps, there is a way to decrease the volume of intermediate results. Two methods are available to achieve this result:

2.1 Compressing the intermediate results

Joining the tables according to the ascending ability of their output, that is joining those tables that produce minimum join tuples; then, the result will join the next table that has minimum output.Compressing the intermediate results, not only provides space saving in HDFS, but in some cases is responsible to map next join that leads to decreased numbers of transferred bites on the network.

2.2 Optimization using the size of output

Determining the order of joining tables based on the size of output is difficult. What should be considered is that number of table's lines is a number that goes as follow:

$$T_1(K_1)*T_2(K_1)+T_2(K_2)*T_2(K_2)+\dots+T_1(K_n)*T_2(K_n)= \sum_{i=1}^{n} T_1(K_i)*T_2(K_i) \qquad (2)$$

Here, $T_m(K)_n$ shows the number of tuples of T_m that has K_n keys.

Therefore, if the keys of each table and corresponding tuples become observable in this table, it is possible to obtain output tuples in join. In other words, the output cardinal number is determined. This can be done as a preprocessing step. This preprocessing can be performed using numerators. For this purpose, output cardinal at preprocessing is characterized in pairwise manner for tables and is stored in a file. When the join has to be run, those tables that have minimum cardinal are selected and join immediately. Then, this intermediate result join to the table that has minimum cardinal output with one of previous tables. Next, the table with minimum join cardinal output will be selected by each of three existing joint tables to join the intermediate result.

5. Hash Join Algorithm

Hash join algorithm is composed of build phase and query phase. In simplest mode, smaller datasets are loaded in hash table within the memory in build phase. In query phase, larger datasets are scrolled and by query in hash table, join the related tuples. This algorithm is applicable for joining in equality mode.

Consider two databases of P and Q. A simple hash join algorithm is as follow:

```
Load p into in memory hash table H
End for
For all q ϵ Q do
    If H contains p matching with q then
       Add<p,q>to the result
    End if
   End for
```

5.1 Algorithm 1

This algorithm is faster than sort-merge join but to sort hash table, puts a relative large load on the memory. Distributed join algorithm is a distributed type of hash join algorithm. Smaller datasets are sent to each node on the distributed cluster. Then, they are loaded in a hash table on the memory. Each map function is activated by allocated pieces of input data and searches hash table to find equal tuples.

5.2 Results

Eclipse is a discrete event simulation software that is used to simulate the details of different programming. The final simulation software is written by Java while its ending part is written for functional purposes and relationship with SQL through MapReduce language. Connection for SQL and services of distributed database will be important. Also, this function can be effective and is used to connect to a vast area of MapReduce infrastructures to have relationship with SQL. Eclipse and Cygwin environments have provided suitable simulation potential for the distributed environment of hadoop. Also, the possibility to make connection with SQL server is provided that make simulation and its running possible.

5.2 Simulation Parameters

To show the application of the proposed framework, in order to evaluate energy returns, a study was conducted on alternative samples in coding format. Table I describes used parameters in simulation. This table shows that the mean of consumption costs for each groups in MapReduce indicates different coding records. As expected, it is observed that using object-oriented Java format, both transmission energy and consumption energy will be nine times more than energy consumption in custom coding approach.

Table 1: Simulation parameters

Row	*Parameter*	*Value*
1	Number of worker nodes	10-50
2	Reading from disk	66584576 (Bytes/second)
3	Writing on the disk	61027123 (Bytes/second)
4	Writing speed at HDFS	46137344 (Bytes/second)
5	Memory of each worker node	100 MB
6	Transmission speed in network	44040192 (Bytes/second)
7	Size of memory blocks	67108864 (Bytes)
8	Maximum mapping numbers	5-35
9	Maximum reducer numbers	5-35

On the other hand, the use of Java coding format consumes energy as the custom format size. Custom records have the highest energy consumption, but provide low flexibility and performance. However, text java shows the end of other spectrum. Java has many advantages, but verbosity increases energy during connection (almost nine times more than formats based on order 14). Nevertheless, the use of binary Java is a compromise between text Java and custom formats. The results help to prove simulation framework for the suggested method and show its efficiency.

6. Architecture of Hadoop Cluster

In Fig.1, a Hadoop cluster that is used in this study is presented.

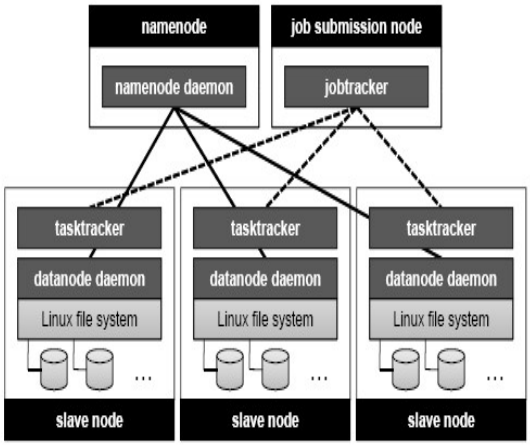

Fig. 1 Architecture of Hedoop cluster that is composed of three components

The structure of this architecture includes three separated parts:

- Master HDFS called name node
- Job submission node called job tracker

- Several worker nodes (in this figure we have three worker)

Each worker node launches a job tracker to implement MapReduce functions.

HDFS runs its daemon. Job submission runs job tracker that job tracker evaluates the progresses in running MapReduce and is responsible to run mapping and reducers. Usually, this service is run on two separate machines. Stack of a Hedoop cluster is constituted by worker nodes that run task tracker and daemon node to consume HDFS data. Worker nodes include three parts and have received MapReduce responsibilities from task tracker and at the same time, use HDFS file system to perform their responsibilities and store and recover data. However, although the used file system format was Linux, this can be solved and run by using Cygwin software that is a Linux virtual machine environment.

7. Used Orders

In this study, AdventureWorks database was used for Microsoft SQL Server 2008 R2 to investigate the efficiency of algorithm on virtual machine. In the following, two cases of SQL orders that can be used to run hash join can be observed.

```
SELECT p.Name, pr.ProductReviewID
FROM Production.Product AS p
LEFT OUTER HASH JOIN Production.ProductReview AS pr
ON p.ProductID = pr.ProductID
ORDER BY ProductReviewID DESC;
```

In order (1), it is specified that hash join in two tables of ProductReviewID and Product is run on common field of ProductID. Also, the order of results in the table is according to ProductReview filed and in descending order. Also, since the join is a left external join, all tuples from the left table (product) are shown in output table.

```
SELECT FirstBame, LastName
FROM Person.Person AS a
INNER JOIN Sales.PersonCreditCard AS b
ON (a.BusinessEntityID = b.BusinessEntityID)
OPTION (HASH JOIN);
```

The second order runs hash join on two tables of Person and PersonCreditCard with the common field of BusinessEntityID. Since this order is an inner join, only shows those tuples that their values exist on two tables. These orders are different in terms of recalling hash join, but their running modes are not different. Regarding their performance, if in construction phase, a join table was created from a smaller table, in query phase, larger tuples are searched in join table. This trend is observed in running hash join order. If both tables are equal in terms of number of tuples, fields, and size of tables, the priority for construction phase is for the left table.

In order to parallelize, a number of tuples from table A with all components of table B were processed by each worker node and hash table was temporarily constituted by table A that leads to the formation of a larger hash table from table A. When all tuples of table A in worker nodes become hash joint, sharing these cases creates table A as hash. Hash join formula is not similar for all worker nodes and therefore, tuples from one table that are processed in different nodes will not have similar hash. Hash join problem occurs in table B, too. In this study, in order to simplify the implementation and identification of pipelines parallelizing routine, by MapReduce system in Hadoop, half of nodes became responsible to join table A and at the same time, to produce output and the rest of worker nodes became responsible to join table B and at the same time, produce output. When two join tables are completed, all records of tables will be processed and end by the final output. Before that, by completing one of join tables, running ends. According to the discussion, algorithm should start by the division of tuples from tables between nodes where each of the nodes receives tuples from one of tables while it receives the next table as join input. Here, it is assumed that the number of tuples that a node receives from divided table is smaller than the other table that leads to consider received tuples as join table (construction phase) and other tables as query table (in query phase). Also, mapping phase is run on this stage between worker tuples.

8. The Results

In this section, the results of running the proposed method for SQL hash join in MapReduce system are investigated. For this section, several queries must be run. For the first one as Q1, the following order is presented.

```
SELECT a.FirstName, a.LastName, b.CreditCardID
FROM Person.Person AS a
INNER HASH JOIN Sales.PersonCreditCard AS b
ON a.BusinessEntityID = b.BusinessEntityID
ORDER BY a.LastName ASC;
```

Here, a hash join is run on tables and credit card of the person and full names of people that have credit card

with the number of credit card are shown. The common key between two tables is BusinessEntityID. In conducted experiences, number of records of person's table from person's scheme is 19972 and number of person's credit card from sale scheme is 19118. Since the numbers of records are very close, hash joints should be done by equal worker nodes to form hash join table for these tables. In the following, the running results are observed. The proposed model will be compared with other two modes. One mode is where hash join is run equally by all nodes and in other mode, hash join is not run and a common join is running using MapReduce system. By increased number of worker nodes, running time decreases. However, this will be done for a certain number of worker nodes and after that, it will have a small effect on output time. Also, it has inverse effect on some cases and the response time will be longer due to the complexity of communications (Fig. 2).

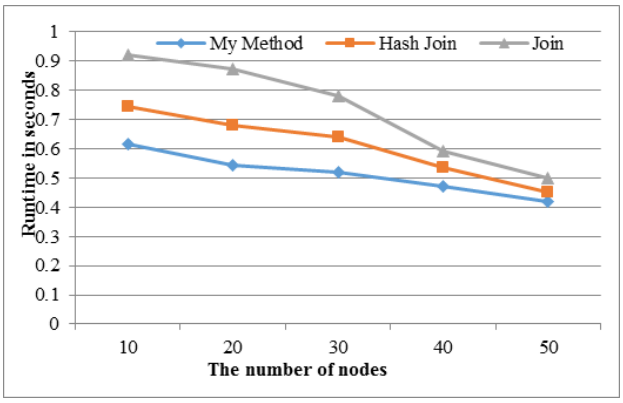

Fig. 2 Comparing running time of Q1

```
SELECT a.ProductID, a.OrderQty, a.UnitPrice, b.PersonID
FROM Sales. SalesOrderDetail AS a
Left Outer HASH JOIN Sales.Customer AS b
ON a.CustomerID = b.CustomerID
```

In Q2, a query is run to buy products by a customer where some of information include product's code, number of orders, cost of product, and ID of customer where customers who do not have PersonalID are shown and is done due to Left Outer order that is for hash join. These customers who do not have personal ID can be stores or people that are responsible to buy products and for these reasons, IDs can be empty. In this query, two tables of SalesOrderDetail and Customer are hash joint where they have 121317 and 19820 records, respectively. Hash join condition is the equality of CustomerID of two tables. In proposed hash join queries, number of tables' records are different from each other and one of them has a record about six times larger than the other one where to process hash join it is logical to have smaller numbers of nodes for worker nodes for consumer table. However, in this project it was stated that numbers of worker nodes for both tables are equal. The issue of changing worker nodes can be the subject of future studies. In Fig.3, the results of running Q2 and comparisons on MapReduce are shown.

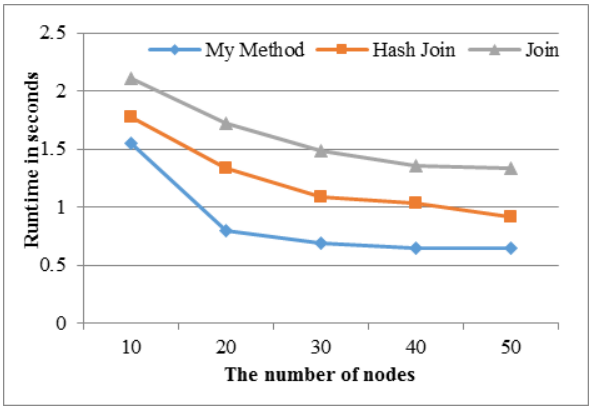

Fig. 3 Comparing running time of Q2

According to the running times in Figs.2 and 3, it is obvious that by increased worker nodes, running time decreases, but gradually, the effects become smaller. Also, from the comparison, it can be understood that due to increased number of processed records in Q2, processing time has increased, but not the same as the level that records are processed. This has been created due to using distributed MapReduce system and overlap of tables is one of the reasons for this issue.

First, the distance of nodes' performance is relatively high and decreases by increased number of nodes, but from a certain point, no significant change is observed in this context. The reason for this is that number of distributed nodes is low and more time is required for processing. As a result, the proposed algorithm has indicated its influence. Then, since number of user's nodes increases running time in distributed system, this difference is rarely noticed. In the diagram, it is observed that the suggested algorithm functions better which is followed by hash join and common join. Although this was predictable from the very beginning, the amount of improvement in proposed method and hash join is very important. As the diagram shows, the improvement level compared to the hash join of the proposed method shows improvement between 10 to 30%. In Fig.2, the trend is different that is due to increased number of records that have prolonged the process and here, the effect of the proposed method is characterized that is tangible in the diagram where a difference between 50 and 60% is

observable. On average, the improvement level can be considered as 30% that in first mode, shows 15 to 20%. Generally, the distance between methods is more observable and indeed, by higher record numbers, the effect of algorithm performance increases.

9. Conclusion

In addition to utilizing storage and processing abilities of Hadoop framework and 30% increase in query potential and 30% decrease in processing time compared to Hadoop standard methods, this method functions faster compared to 30 SQL that proves the advantage of using Hadoop and the algorithm of interest. Also, this study introduced an accurate, efficient, and effective method for queries as well as fast running of information and their classification using advanced techniques of hash join algorithm. Indeed, the method that is proposed to run queries and communicate with SQL bank on clients, is faster than previous algorithms with high accuracy and involves less memory storage. Increased accuracy and speed and lees memory are due to this reason that to work with data, first, two hash tables are created in the memory and finally, bank tables lead to two hash tables. The results show that the proposed method provides accurate results and requires less time. The most important developed applications in this study refer to the concentration on discovering records of interest in banks that increase by breaking huge tables and converting them into hash and by allocating them to information processing clients, less memory is used and as a result, speed increases. Hash join algorithm runs query in banks accurately. This algorithm converts information into numbers by coding them in tables and therefore, arranges tables and improves the correspondence of tables and access to information of interest. Also, the difference between this algorithm and previous algorithms is that access to the parts of interest necessitates huge amounts of time and perhaps the tables are very large and cannot be fitted into the main memory. The proposed algorithm for distributed systems is suitable for small networks or limited number of systems. The advantage of this algorithm is that the query can be run with different numbers of systems and different amounts of memory. The tables of interest are divided into subparts considering memory of systems as well as their numbers. Also, hash join is an algorithm that is less expensive compared to some of previous methods. The results of simulation have shown that in different conditions, the proposed algorithm shows high efficiency.

Eclipse program provided the infrastructure for banks and broken tables to be distributed in network system and deliver the output through the network or introduced software to the server. This shares information between systems and facilitates the query that works with hash join algorithm and obtains its required information very fast.

10. Research Recommendations

During using Eclipse program and MadReduce and breaking tables and creating them in hash tables, some problems are likely to occur when the tables are created in the memory and are sent to clients, tables cannot be fitted into memory and systems cannot load and display all records of interest. To solve this problem, two algorithms can be used that can change tables and information for this purpose. These algorithms are hybrid hashing and partitioning that consider other SQL orders and different queries and improve them that leads to significant improvement in running the orders. Other methods for hash join on MapReduce are taken into consideration by orders of databases such as Oracle.

References

[1] R.C. Soudip, “Assisted Reuse of Pattern-Based Composition Knowledge for Mashup Development” PhD diss., University of Trento, 2013.

[2] P.B. Rich, “Understanding Terror, Terrorism, and Their Representations in Media and Culture” Studies in Conflict & Terrorism, Vol. 36, pp. 255-277, 2013.

[3] H. Christian, S. Lehnhoff, and M. Sonnenschein, “Paving the Royal Road for Complex Systems: On the Influence of Memory on Adaptivity”, In Selforganization in Complex Systems: The Past, Present, and Future of Synergetics, pp. 313-318. Springer International Publishing, 2016.

[4] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, “NVIDIA Tesla: A unified graphics and computing architecture”, Micro, IEEE, Vol. 28, pp. 39-55, 2008.

[5] P. J. De Moraes, J. Allison, J. A. Robinson, G. L. Baldo, F. Boeri, P. Borla, “Life cycle assessment (LCA) and environmental product declaration (EPD) of an immunological product for boar taint control in male pigs”, Journal of Environmental Assessment Policy and Management Vol.15, pp. 1-26, 2013.

[6] A. Swiffen, J. Nichols, The Ends of History: Questioning the Stakes of Historical Reason. Routledge, 2013.

[7] Z. Izsvák, Z. Ivics, R. H. Plasterk, “Sleeping Beauty, a wide host-range transposon vector for genetic transformation in vertebrates”, Journal of molecular biology, Vol. 302, pp. 93-102, 2000.

[8] B. Paul, Multinational Joint Ventures in Developing Countries (RLE International Business), Routledge, 2008.

[9] T. H. O. M. A. S. A. Cwik, R. Mittra, “The cascade connection of planar periodic surfaces and lossy dielectric layers to form an arbitrary periodic screen”, IEEE transactions on antennas and propagation, Vol. 35, pp. 1397-1405, 1987.