Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Cost Based Multi-Way Equi-Join Optimization in MapReduce

[1]Dr K. Venkata Ramana, [2]T. Venkata Sundeep
[1]Assistant Professor, [2]MTech Student
[1, 2]Dept of CS&SE, Andhra University College of Engineering (A),
Visakhapatnam, A.P., India

Abstract: ***MapReduce is a prominent programming model above shared nothing architecture for processing big data with a parallel, distributed algorithm on a cluster. Join is an important operation is very inefficient in MapReduce. In this work, a time cost based evolution model is proposed for multi-way join by considering the time cost calculation. A multi-way join consists of start pattern joins and chain joins as sub joins. Based on the cost model, optimization techniques are used and a join operation with lower time cost elected for performing join operation. The optimization techniques include: the star pattern sub join are processed first them after, a chain pattern sub join with minimum number of tuples in the intermediate results is considered and processed; at last, chain pattern sub join is processed by using several MapReduce jobs or by using single MapReduce job which is obtained by the dynamic programming based algorithm. We conducted rigorous experiments, to prove that our method gives efficient results when compared to the other optimization methods.***

Keywords: MapReduce, multi-way join, sub join, equi join, start pattern, chain pattern, dynamic programming.

I. INTRODUCTION

Huge amount of data are collected in many areas, such as medical, finance, communication and governments. In present scenario there are urgent needs to analyze big data in these applications. However, solutions are strictly based on conventional distributed or parallel databases are very difficult to meet the needs of big data analysis. MapReduce is a prominent programming model above shared nothing architecture for processing big data with a parallel, distributed algorithm on a cluster [3]. In present Scenario, thousands of software applications for processing big data have been implemented by this approach including large-scale image processing, machine learning as well as many other areas. In big data analytics queries, equi-join is an important operation. However, it is not efficient operation to perform an equi-join operation in MapReduce when multiple datasets are involved in join operation. Although several methods are used to perform an equi-join, these methods have many advantages for the equi-joins on some special and specific datasets.

The methods for equi-joins are divided into two groups; Map-side join and Reduce-side join [9]. The Broadcast-Join and its improved approaches Semi-Join and Per-Split Semi-Join are all Map-side joins [5]. However, the performances of these methods for equi-joins decrease seriously when the scales of datasets involved in equi-joins increase because the efficiencies of these methods depend on the hardware of the clusters. There are also several approaches for Reduce-side joins on multiple datasets, such as the equi-join method with multiple MapReduce jobs (MRJs) [7,8] , multiple-dimensional Reducer matrix based multi-join (MDMJ) [2] and modifying original MapReduce frameworks [6]. These methods have different benefits or advantages when specific numbers of datasets are joined. Among these methods, an equi-join is processed by a series of MRJs or single MRJ. However, it is difficult to determine whether an equi-join should be processed by single MRJ or by multiple MRJs. If we are able to find out the time cost for disk I/O, communication and calculation of an equi-join[10], we could select a plan with time cost as low as possible from different schemes of the equi-join, then the efficiency of the equi-join can be increased.The contributions of this paper are listed in the following.

1. The time cost of calculation for an equi-join is extended based on the time cost model in literature. Therefore, time cost for an equi-join consists of three parts: disk I/O, communication and calculation.
2. Based on the time cost model, optimization methods are used and an equi-join plan with lower time cost is selected. Then the performance of the equi-join is improved. The optimization methods includes
   a) The star pattern sub-joins performs on only one attribute are first processed.
   b) Next, a chain pattern sub-join operation with optimal scale of intermediate results is processed.
   c) At last, an optimal plan for the chain pattern sub-join is obtained by dynamic programming.
3. We conduct large-scale experiments to verify the efficiency of our method.

II. RELATED WORK

2.1 Perform Equi-Join Operation on Single Attribute:

Performing Equi-join operation on single attribute is an equi-join based on multiple datasets and one attribute. For example, $R_1(a, b_1) \bowtie R_2(a, b_2) \bowtie \cdots \bowtie R_L(a, b_L)$ is a typical equi-join on one attribute *a*. During Map phase task, the Map () function produces <*key*, *value*> pairs based on a value of the equi-join attribute *a* (as key) and

name of a dataset and values of other attributes [*Ri*, (*a*+*bi*)]. During Reduce stage; *reduce ()* function produces a <*key*, *valuelist*> pair every time and the results or values in *valuelist* are classified according to the dataset and the values are joined where the *valuelist* is the list of *value*s with the same *key* (in <*key*, *value*> pair). Finally, the results are obtained and collected.

2.2 Equi-Join Operation on Multiple Attributes:

The equi-join on multiple attributes is a join based on multiple datasets and several attributes. Existing approaches for equi-joins focus on Reduce-side join and we briefly summarize them in the following.

- In multiple MRJs approach, the equi-join is processed in a series of MRJs. At first, two datasets are joined and the moderate outcomes are written into HDFS(Hadoop Distributed File System) as an input dataset of next MRJ. In the next MRJ, the results from the previous MRJ are read from HDFS and a new dataset is selected to perform join operation. Thus, for an equi-join with n datasets, n-1 MRJs are needed.
- In MDMJ method, Reduce nodes are converted into a multi-dimensional Reducer matrix. When an equi-join is processed, tuples in a dataset are copied from a Map node to Reduce nodes repeatedly. In Reduce stage, moderate outcomes are stored in buffer and then the time cost for disk I/O is reduced. The datasets are joined in the order specified in an original SQL statement. By this way, the equi-join operation can be implemented in one MRJ and the performance is increased. When the number of attributes, the number of Reduce nodes and the scales of the datasets increase, the time cost for communication would be exponentially increases and the performance of the equi-join decreases seriously.
- In the approach for joining datasets with bloom filters, a bloom filter is constructed for an input dataset, and the redundant tuples are filtered out in another input dataset involved in the equi-join in Map phase. Thus, the number of tuples involved in the equi-join is reduced and the performance of the equi-join is improved.
- In Network-aware multi-way join for MapReduce (NAMM) [6] tuples are redistributed directly between Reduce nodes with an intelligent network aware algorithm so that the workload is redistributed amongst Reduce nodes. By considering network distance and workloads of Reduce nodes, datasets are considered to perform join operation, and then the workload of each Reduce node is alleviated and the performance of an equi-join is improved.
- Yang et al proposed a Map-Reduce-Merge join [1]. Merge phase is added to MapReduce so that the partitioned and sorted data could be merged and then the final results can be obtained. The model could express relational algebra operators and several equi-join algorithms are performed for the model. However, they did not determined experimental results of their method.

The equi-join algorithms improve the performance of an equi-join in single MRJ. They only optimize single MRJ to improve performance and the improvement is limit, especially when the scales of datasets increase.

III. THE EXTENDED TIME COST MODEL FOR EQUI-JOIN OPERATION

3.1 The Time Cost Model for Single MRJ

In Map stage, datasets are read from HDFS, and data blocks with <*key*, *value*> pairs are produced and stored into local disk. In Reduce phase, the data copied from Map tasks through network are aggregated, sorted and calculated then the final equi-join results are written on HDFS. Therefore, the total time cost of a MRJ consists of the disk I/O in Map phase, copying data in the network and calculation in Reduce phase.

The Time Cost in Map Phase:

In Map phase, assume there are m number of Map tasks and c_m is the time cost for each Map task, and CO_M is the time cost in Map phase task. If we define the total scales of input datasets of an equi-join for a MRJ is I_S , the total scales of input datasets in each Map task for read is $\frac{I_S}{m}$. The time cost for a Map task is $c_m = t_{in} + t_{out}$, where t_{in} and t_{out} are the time for reading and writing respectively. For the Reduce-side join methods, we have $t_{in} = K_1 \times \frac{I_S}{m}$, where $K1$ is a constant factor of disk I/O capability for reading the data, $t_{out} = (p + K_2) \times \tau \times \frac{I_S}{m}$, where τ denotes the output ratio of a Map task, which is confined to query and can be computed by using estimation of selectivity, K_2 is a constant factor of write capability for disk I/O and p is a random variable which denotes partition ,sorting and compressing of data in the buffer. The time cost for a Map Task is:

$$c_m = \left(K_1 \times \frac{I_S}{m}\right) + \left(K_2 \times \tau \times \frac{I_S}{m}\right) + \left(p \times \tau \times \frac{I_S}{m}\right)$$

The cost for entire Map phase based on time is given as $CO_M = c_m * \frac{m}{m^p}$ where m^p is the number of map tasks running in parallel.

Time Cost for Copying Data in Network:

The time cost for copying data from Map tasks to Reduce tasks the number of tuples (or) scale of data produced by the Map tasks. It also consists of copying data over network as well as serving the network layer protocols. Suppose

there are n Reduce nodes in the system, and c_{CP} denotes cost for copying the output data from a single Map task to n Reduce nodes. Time taken by map task is given as, $c_{CP} = \left(K_3 \times \tau \times \frac{I_S}{m} \times \frac{1}{n}\right) + (q \times n)$, where K_3 denotes a constant factor of copying data over network , *q* is a random variable represents the cost of serving a Map task of *n* connections to *n* Reduce tasks. After the data copying ends, Reduce tasks begin to process the data received. Suppose that CO_{CP} denotes the time of copying output data from Map tasks to *n* Reduce nodes, then we have $CO_{CP} = c_{CP} \times \frac{m}{m^p}$. Assume the time when Reduce tasks begin is CO_{RS}, CO_{RS} is determined by the time when the last Map task ends data copying. If $c_m \geq c_{CP}$, then time taken is givens as the sum of the last Map task ends copying output data and time taken for copying map task to *n* reduce nodes , then $CO_{RS} = CO_M + c_{CP}$. If $c_m \leq c_{CP}$, and when the last Map task ends, some of the Reduce nodes are still copying output data, then $CO_{RS} = c_m + CO_{CP}$.

The Time Cost in Reduce Phase:

For suppose D_R^i is the dataset that i^{th} Reduce node received , then the time cost at reduce node consists of time taken for reading, calculation and the writing of reduce phase output which is denoted as

$$C_{R_i} = \left((p + K_1) \times Scale\left(D_R^i\right)\right) + \left(E_q\left(D_R^i\right) \times K_4\right) + (K_2 \times \gamma \times Scale(D_R^i))$$

Where $Scale\left(D_R^i\right)$ is the scale function which returns scale of dataset D_R^i . $(p + K_1) \times Scale\left(D_R^i\right)$ is the preprocessing of reduce task such as sorting , merging and shuffling and also sending the data to reduce() function. $E_q\left(D_R^i\right) \times K_4$ is the time taken for calculation of equi-join on the datasets as this is also taken in to consideration because it depends on the time taken by the equi-join algorithms and scale of the datasets where K_4 is constant factor for calculation is. $K_2 \times \gamma \times Scale(D_R^i)$ Denotes the time cost taken by the Reduce task to write the output in to HDFS. The time cost model is same as proposed in Theta-join processing [10].

Suppose C_R stands for the total time taken by the entire reduce task. The entire time taken by the reduce task is decided by the last reduce task which has more scale of D_R^x data and writes its output to HDFS. So the total cost for Reduce task is dominated by the Reducer node which has highest scale of data. So the cost for Reduce phase is given as

$$CO_R = \left((p + K_1) \times Scale\left(D_R^i\right)\right) + \left(E_q\left(D_R^i\right) \times K_4\right) + (K_2 \times \gamma \times Scale(D_R^i))$$

The time cost model for entire MapReduce job is given as

$$C = \begin{cases} CO_M + c_{CP} + CO_R \text{ , if } c_m \geq c\_CP \\ c_m + C_{CP} + C_R, \quad if\ c_m < c_{CP} \end{cases}$$

Although the above formula is same as the time cost in [10] CO_R is different from the T_R in. When we consider time cost of calculate for a MapReduce job, we have to determine the values of τ,γ, D_R^x and $E_q\left(D_R^x\right)$. In the following section we will discuss the single MapReduce job for equi-join on single attribute and equi-join on multiple attributes.

3.2 The Time Cost of an Equi-join for Single MRJ on Single Attribute:

Suppose we have to join datasets $D_1, D_2, \ldots \ldots D_n$ which are sent to Map task. In map task each tuple is read from the dataset and sent to map () function which in turn produces (Key, Value) pairs. The sum of scales of input datasets is equal to the output of Map task so $\tau = 1$. In Reduce task the tuples are sent to reduce () function based on the dataset which the tuples are came from and then Cartesian product is performed when no common attributes so the total cost of calculation part of Reduce task is given as

$$E_q\left(D_R^i\right) = \sum_{i=1}^{n} Scale\left(TD_i^x\right) + Scale\left(TD_1^x \propto TD_2^x \propto TD_3^x \ldots \ldots TD_n^x\right)$$

The output ratio γ is given as

$$\gamma = \frac{Scale\left(TD_1^x \propto TD_2^x \propto TD_3^x \ldots \ldots TD_n^x\right)}{\sum_{i=1}^{n} Scale\left(TD_i^x\right)}$$

Where TD_i^x stands for the number of tuples in the datasets D_i which is received by a reduce node x. For suppose assume that data is distributed equally such that each reduce task receives almost equal. Then

$$Scale\left(TD_i^x\right) = \sum Scale\left(TD_i^x\right)$$

and

$$Scale\left(TD_i^x\right) = \frac{Scale(D_i)}{n}$$

Where n is the number of reduce nodes.

3.3 The Time Cost of an Equi-join for Single Map Reduce Job on Multiple Attributes:

In this we calculate the time cost of equi-join for multi dimension MapReduce job (MDMJ). In this the reducer nodes are divided in to multi-dimensional matrix based on the scales of datasets involved in the equi-join. The tuples are distributed to many reducer nodes because a tuple may be useful for join the result of multiple tuples, so the input tuples are copied several times in map() function and partition() function decides assignment of keys to reducers, so that tuple is copied for each attributes that involved in equi-join. The total scale of input datasets of equi-join I_S is the sum of scales of all datasets and scale of output of Map task depends on the copy factor of the dataset D_i i.e. number of times the tuple is copied then we have

$$\tau \times I_S = \sum_{i=1}^{n} (Scale(D_i) \times CoT_i$$

The CoT_i can be calculated by using langrangian formula.

In Reduce task, after copying the dataset from various Map tasks the tuples are partitioned, sorted according to the attribute on which equi-join to be performed. In reduce() function the input tuples of first dataset are read in to the buffer, and when tuples of second datasets are available in the buffer then the equi-join is performed and the results are stored in the buffer and sorted according to the attribute on which join is performed with the next dataset. We will follow the above steps to process join. As the tuples are already stored in the buffer we will traverse the data only once. Therefore the join is performed in the specified order.

Suppose we have equi-join $D_1(A1) \bowtie D_2(A_1, A_2) \bowtie D_3(A_2, A_3) \ldots\ldots\ldots\ldots \bowtie D_n(A_{n-1})$ for each reduce node the tuples of dataset $\sum_{i=1}^{n} D_i$ are sorted. From the above description we will perform $D_1 \bowtie D_2$ first, then we sort their join attribute according to the next join attribute A_2 and the process continues. So the time cost for calculation is given as

$$E_q\left(D_R^i\right) = \sum_{i=2}^{n} Scale\left(IntD^x{}_{1,i-1}\right) + Scale(TD_i^x) + Scale\left(IntD^x{}_{1,i}\right) + \sum_{i=2}^{n} Scale\left(int D_{1,i}^x\right) \times log\left(Scale\left(intD_{1,i}^x\right)\right)$$

Where TD_i^x denotes the dataset i received by the reducer node; $IntD_{1,i}^x$ denotes the intermediate join results of the dataset $TD^x{}_1 and\ TD^x{}_i$. At last, the amount of writing of data in to HDFS and sorting this is performed on the intermediate tuples.

IV. OPTIMIZATION METHODS FOR EQUI-JOIN

The problem of an equi-join is in what order the datasets are to be joined such that total time cost is as low as possible. We choose an optimal plan for equi-join with cost as low as possible. In naïve method we have to enumerate and evaluate the time cost for each possible equi-join plan which is a NP-Hard problem. There are two types of sub-joins in equi-join:

1. Star sub-join
2. Chain Pattern Sub-join

In star sub-join, a dataset is joined with other datasets on a single common attribute. Suppose $D_s(A_{i-1}, A_i) \bowtie D_{s+1}(A_i, A_{i+1}) \bowtie \cdots D_t(A_{t-1}, A_t)$ is a sub-join of $D_1(A_1) \bowtie D_2(A_1, A_2) \ldots. \bowtie D_n(A_{n-1}, A_n)$ where $1 \leq s \leq t < n$. If a chain pattern sub-join is not a sub-join of any other chain sub-join then it is complete chain sub-join. A query optimization method for an equi-join has following steps: 1) A Star pattern sub-join must be processed first and it is replaced by the intermediate results in the original equi-join. 2) In a chain pattern subjoin the datasets which are having minimum tuples in the result when they are joined are processed. Then complete chain is replaced by set of moderate outcomes. 3) The time cost obtained in the step-2 when a chain pattern sub-join is evaluated by Dynamic Programming and the optimal equi-join scheme is chosen. As we know that joins obey associative property when they are executed in any order, so it does not affect join properties.

Dynamic Programming Based Optimization Method:

We will use dynamic programming for optimal plan for chain pattern sub-join. In dynamic programming all the possible plans are enumerated and a plan with low time cost is selected. Assume that we have chain pattern equi-join $D_1(A_1) \bowtie D_2(A_1, A_2) \bowtie \cdots D_r(A_{r-1})$ and $Cost(i, j)$ denotes the time cost for joining the relations D_i and D_j. For suppose the chain pattern sub-join is decomposed in to two sub-joins namely $D_i\ to\ D_k$ and $D_{k+1}\ to\ D_n$ such that intermediate results of both sub-joins are joined together to produce a complete equi-join of $D_i\ to\ D_n$. So the minimum cost of both can be written as below

$$Cost(1,n) = \min\ (\ Cost(1,k) + \ Cost(k+1,n) + \ joinCost(int(1,k), int(k+1,n))$$

for $1 \le k \le n-1$

Where joinCost (p, q) stands for join time cost of the relations p and q; int (x, y) denotes the intermediate results by joining the relations x and y. If we consider the chain pattern join is further decomposed in to three sub-joins then the above cost formula can be written as

$$Cost(1,n) = \min\ (\ Cost(1,k_1) + \ Cost(k_1+1,k_2) + Cost(k_2+1,n) + \ joinCost(int(1,k_1), int(k_1+1,k_2), int(k_2+1,n))$$

for $i \le k_1 \le k_2 \le n-1$

The same formulation is applied to four chain pattern sub-join, five chain pattern sub-joins and so on. So we propose one scheme for all possible ones

$$Cost(1,n) = min\begin{cases} Cost(1,k) + \ Cost(k+1,n) + \ joinCost(int(1,k), int(k+1,n) \\ where\ 1 \le k \le n-1 \\ Cost(1,k_1) + \ Cost(k_1+1,k_2) + Cost(k_2+1,n) + \\ joinCost(int(1,k_1), int(k_1+1,k_2), int(k_2+1,n) \\ where\ \ i \le k_1 \le k_2 \le n-1 \end{cases}$$

By using above method we can obtain lowest time cost. The time cost of joining datasets using single MapReduce job is described in 3.1. The time cost for star pattern sub-join can be calculated using the method described in 3.2. The time cost for chain pattern sub-join can be computed by using the method in 3.3.

V. EXPERIMENTATION RESULTS

5.1 The Environment in Experiments

In this experimentation results, we will check the efficiency of our approach and compare with other equi-join algorithms or methods. As NAMN and MapReduce Merge algorithms modified MapReduce framework, they are not comparable to our method, then we will compare the MDMJ and multiple MRJs algorithms with our method. In addition, Pig and Hive are two databases based on Hadoop and need to compare with our method. The experiments are based on Hadoop platform, the hardware in experiments consists of 1 Master node and 3 slave nodes, all of the nodes are blade servers.

Table 1: Hardware and software lists in experiments

Hardware	Software
CPU:Intel Core i3	Operating System
1.7GHz*4	Ubuntu-14.04 LTS
Memory: 8G	Type of OS: 64bit
Disk:500GB	Hadoop: Hadoop-1.2.1

Table 2: SQL statements in experiments

Query No	SQL Statements
Query 1	SELECT * FROM R1, R2, R3, R4, R5 WHERE R1.R1_COL1=R5.R5_COL1 AND R2.R2_COL1=R5.R5_COL2 AND R3.R3_COL1=R5.R5_COL3 AND R4.R4_COL1=R5.R5_COL3
Query 2	SELECT * FROM R1, R2, R3, R4, R5 WHERE R1.R1_COL1=R5.R5_COL1 AND R2.R2_COL1=R5.R5_COL2 AND R3.R3_COL1=R5.R5_COL3 AND R4.R4_COL1=R5.R5_COL4
Query 3	SELECT * FROM R1, R2, R3, R4, R5 WHERE R4.R4_COL1=R5.R5_COL4 AND R3.R3_COL1=R5.R5_COL3 AND R2.R2_COL1=R5.R5_COL2 AND R1.R1_COL1=R5.R5_COL1
Query 4	SELECT * FROM R1, R2, R3, R4, R5 WHERE R1.R1_COL1=R2.R2_COL1 AND R2.R2_COL2 =R3.R3_COL1 AND R3.R3_COL2=R4.R4_COL1 AND R4.R4_COL2=R5.R5_COL1
Query 5	SELECT * FROM R1, R2, R3, R4, R5 WHERE R4.R4_COL2=R5.R5_COL1 AND R3.R3_COL2=R4.R4_COL1 AND R2.R2_COL2 =R3.R3_COL1 AND R1.R1_COL1=R2.R2_COL1

The experiments consist of three types of equi-joins: hybrid join with star pattern sub-joins, hybrid join without star pattern sub-joins, and chain pattern equi-joins. The datasets are produced randomly by specifying the scope of the data and the different number of tuples for different join attributes. In the following discussion, our method is denoted as JoinStrategy and multiple MRJs as MutipleJob.

5.2 The Experiments for a Hybrid Equi-join with Star Pattern Sub-joins

In the experiments (EXP1 for short), consider 2 tests for the SQL statements Query 1 in Table 2. For the equi-join operation, the number of tuples in each dataset is shown in Table 3 and Table 4 respectively. Compared with Table 3, the scales of datasets in star-center (dataset R5) in Table 4 are changed. The results of experiments are shown in Fig.1 and Fig.2 in which the scales (1-5) in x-axis are shown in Table 3 and Table 4 (from No.1 to No.5) respectively. From the experimentation results, the joinStrategy is more efficient than that of other methods. Comparing Fig.1 with Fig.2, whether the scales of datasets in star-center or in star-angle are changed, JoinStrategy is the most efficient approach among these methods. The reason is that in one MRJ approach only star pattern sub-joins are processed so that large amount of redundant disk I/O is avoided. However, the methods for MutipleJob, Hive and Pig decompose star pattern sub-joins into several MRJs, the time costs of MutipleJob, Hive and pig are greater than that of JoinStrategy. In Fig.2 the time cost for MDMJ is almost equal to the time cost of JoinStrategy; the time cost for MDMJ is more than that of the time cost of JoinStrategy.

Table 3**:** In experiment EXP1.a , the number of tuples in datasets and the final results

No.	T1 ($\times 10^6$)	T2($\times 10^6$)	T3 ($\times 10^6$)	T4 ($\times 10^6$)	T5 ($\times 10^6$)	T6 ($\times 10^6$)
1	6	6	2	2	2	2444916
2	6	6	2	4	2	2453824
3	6	6	2	6	2	2448719
4	6	6	2	8	2	2457335
5	6	6	2	10	2	2467211

Table 4: In experiment EXP1.b, the number of datasets in join and the final results

No.	T1 ($\times 10^6$)	T2($\times 10^6$)	T3 ($\times 10^6$)	T4 ($\times 10^6$)	T5 ($\times 10^6$)	T6 ($\times 10^6$)
1	2	2	2	2	2	2453578
2	2	2	2	2	4	4910520
3	2	2	2	2	6	7378282
4	2	2	2	2	8	9822087
5	2	2	2	2	10	12290426

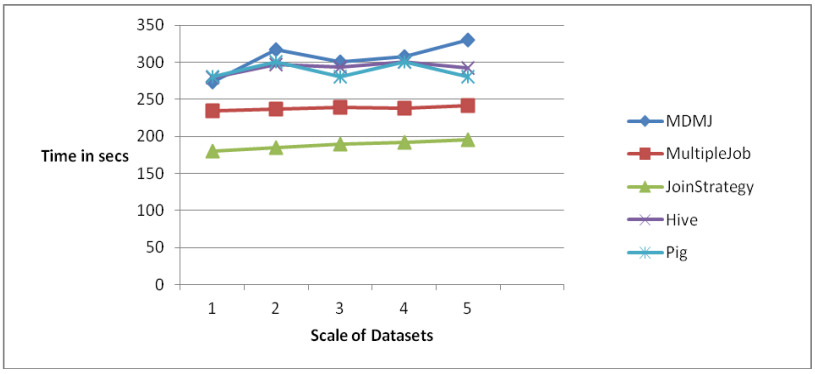

Fig 1: Time cost for datasets in Table 2

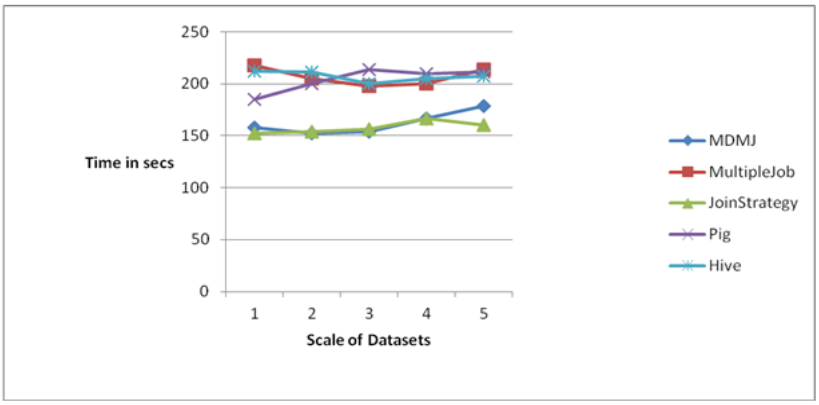

Fig 2: Time cost for datasets in Table 3

5.3 Experiments for Hybrid Equi-joins without Star Pattern Sub-joins

For the experiments in this section (EXP2 for short), we will test the efficiencies of queries Q2 and Q3 in Table 2. The difference of the two SQL statements is that the orders of datasets in the two equi-joins are different, and the order

of joining datasets in our approach would be rearranged according to the optimization method: the chain pattern sub-joins with smaller intermediate results are processed after star pattern sub-joins are processed. The experimental results are illustrated in Fig.3 and Fig.4 and the scales of datasets and results for the equi-joins are illustrated in Table 5.

Table 5: In experiment EXP2, the number of datasets in join and the final results

No.	T1 ($\times 10^6$)	T2 ($\times 10^6$)	T3 ($\times 10^6$)	T4 ($\times 10^6$)	T5 ($\times 10^6$)	T6 ($\times 10^6$)
1	2	2	2	2	6	89915
2	2	2	2	6	6	271685
3	2	2	2	10	6	271142
4	2	2	6	10	6	825062
5	2	2	10	10	6	820939
6	2	6	10	10	6	2458097
7	2	10	10	10	6	2453931
8	6	10	10	10	6	7386461

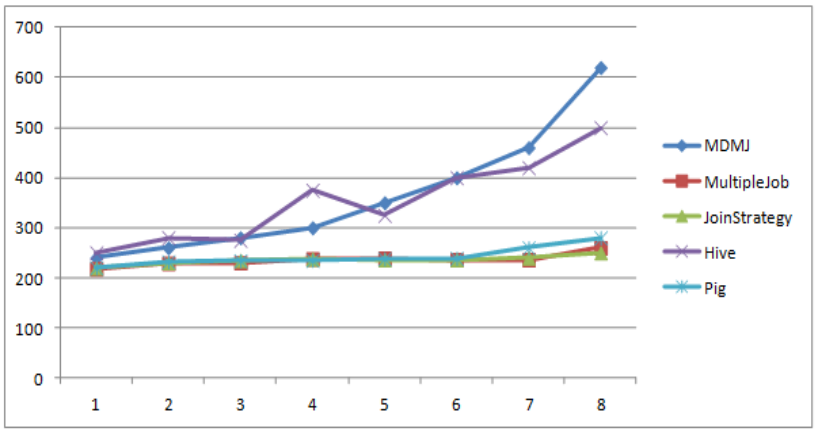

Fig 3: The time cost for the join Q2

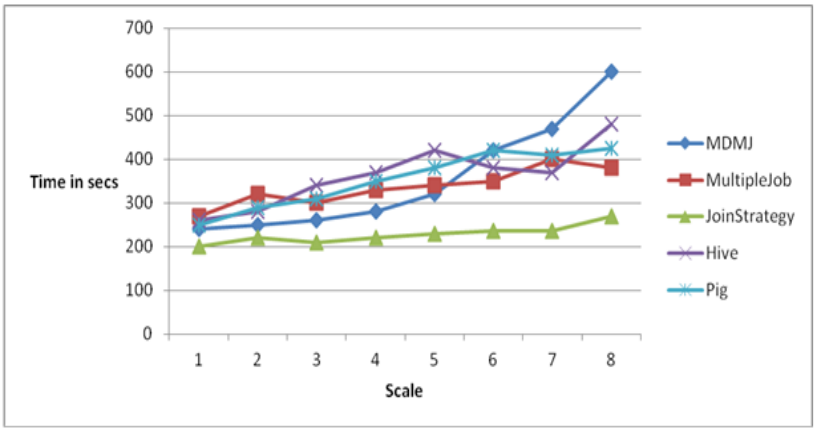

Fig 4: The time cost for the join Q3

Comparing the experimental results in Fig.3 and Fig.4, when we changed the order of datasets in an equi-join, the efficiency of MDMJ and JoinStrategy is unchanged but the performances of other methods decrease obviously. The reason is that MDMJ and JoinStrategy optimize the order of joining datasets while other methods only join datasets in the order specified in SQL statements. Once the order of joining datasets is unreasonable, the performance would decrease obviously. From the experimental results in Fig.4, the performance of an equi-join is improved when the chain pattern sub-joins with smaller intermediate results are processed first. The results indicate that *Property2* in Section 4.1 is suitable.

5.4 Experiments for Chain Pattern Joins:

We will verify that the plan produced by dynamic programming would improve the efficiency of an equi-join in the experimental (EXP3 for short). Like the experiments for hybrid equi-join without star pattern sub-joins, this experiment will test the joins with different order, the SQL statements Q4 and Q5 are illustrated in Table 2.

Table 6: In experiment EXP3 the number of datasets in join and the final results

No.	T1 ($\times 10^6$)	T2($\times 10^6$)	T3 ($\times 10^6$)	T4($\times 10^6$)	T5($\times 10^6$)	T6($\times 10^6$)
1	2	2	2	2	2	2459063
2	2	2	2	2	6	2447277
3	2	2	2	2	10	2455207
4	2	2	2	6	10	2458453
5	2	2	2	10	10	2458784
6	2	6	6	10	10	2453944
7	2	2	10	10	10	2455965
8	2	6	10	10	10	2462062
9	2	10	10	10	10	2461461

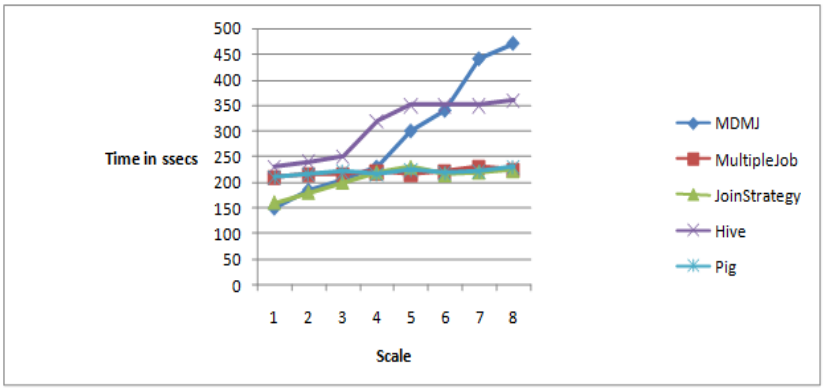

Fig 5: The time cost for join Q4

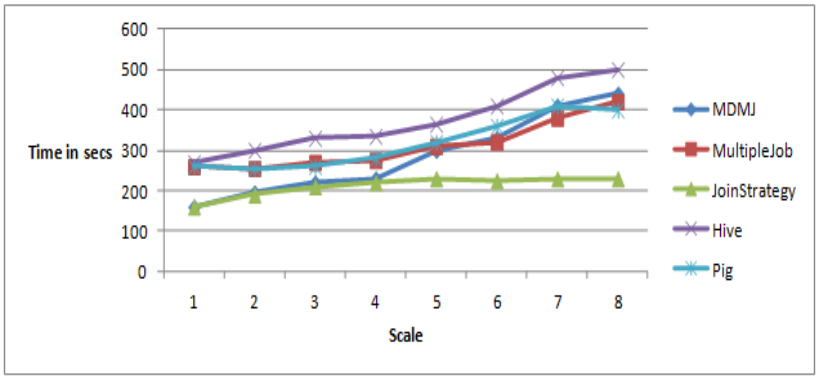

Fig 6: The time cost for join Q5

The experimental outcomes are portrayed in Fig.5 and Fig.6. The experimental outcomes are equal to the outcomes in Section 5.3. From the outcomes in Fig.5, the efficiency of the JoinStrategy is almost equal to Pig and Multiplejob approaches, the reason is that the order of datasets in SQL statement in Fig.5 is close to the order of joining datasets by our optimization methods. However, in Fig.6, the order of joining datasets is unreasonable in Multiplejob and Pig, and the efficiencies of Pig and Multiplejob decrease. Moreover, as the data scales are small before the datasets of No.4 in Table 6, in Fig.5 and Fig.6, MDMJ is more efficient than that of MultipleJob, so we choose MDMJ for multiple datasets in single MRJ by dynamic programming. Therefore, before the datasets No.4 in Table 6, JoinStrategy is more close to MDMJ. At this time, the plan for the equi-join tends to choose multiple MRJs. Therefore, from the datasets No.5 in Table 6, the optimal plan obtained from dynamic programming is the multiple MRJs. From the results, the performance of our method is more efficient.

VI. CONCLUSION

A new equi-join method is presented in this paper. A time cost evaluation model is extended to equi-join operations on multiple datasets and multiple attributes by considering the time cost of calculation. Then optimization methods are presented: star pattern sub-joins are first processed, then the intermediate result-set replaces the star pattern sub-joins and the original equi-join can be simplified; next, the scale of results for each chain pattern sub-join is estimated and the chain pattern sub-joins with minimal scale of results are processed; at last, for chain pattern sub-joins time costs are evaluated for each MRJ by dynamic programming and an optimal plan is chosen. We conducted extensive experiments and verified the efficiency of our approach. In the future, we will study on how to solve the problem of theta-join based on this model.

REFERENCES

[1] Yang, H.C., Dasdan, A., Hsiao, R.L., Parker, D.S.: Map-reduce-merge: Simplified relational data processing on large clusters. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 1029–1040. ACM (2007).

[2] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, pages 99–110, New York, NY, USA, (2010).

[3] ACM. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. SIGOPS Oper. Syst. Rev., 37(5):29–43, Oct. (2003).

[5] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in mapreduce. In Proceedings of the 2010 international conference on Management of data, SIGMOD '10, pages 975–986, New York, NY, USA, 2010. AC.

[6] Slagter, K., Hsu, C.-H., Chung, Y.-C., Park, J.H.: Network-aware multiway join for mapreduce. In: Park, J.J (J.H.), Arabnia, H.R., Kim, C., Shi, W., Gil, J.-M. (eds.) GPC 2013. LNCS, vol. 7861, pp. 73–80. Springer, Heidelberg (2013).

[7] Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: A warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2(2), 1626–1629 (2009).

[8] Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: A not so foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1099–1110. ACM (2008).

[9] Lee, K.H., Lee, Y.J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with mapreduce: A survey. ACM SIGMOD Record 40(4), 11–20 (2012).

