International Journal of Database Theory and Application
Vol. 6, No. 4, August, 2013

Optimizing Theta-Joins in a MapReduce Environment †

Changchun Zhang, Jing Li*, and Lei Wu

School of Computer Science and Technology, University of Science and Technology of China, Hefei, 230026, China
zccc@mail.ustc.edu.cn, lj@ustc.edu.cn, xcwulei@mail.ustc.edu.cn

Abstract

Data analyzing and processing are important tasks in cloud computing. In this field, the MapReduce framework has become a more and more popular tool to analyze large-scale data over large clusters. Compared with the parallel relational database, it has the advantages of excellent scalability and good fault tolerance. However, the performance of join operation using MapReduce is not as good as that of parallel relational database. Thus, how to optimize theta-join operations using MapReduce is an attractive point to which researchers have been paying attention. In this paper, a randomized algorithm named Strict-Even-Join(SEJ) is designed to solve the multi-way theta-joins in a single MapReduce job. Moreover, a dynamic programming algorithm is elaborated to optimize the multi-way theta-joins by calling the SEJ algorithm. The results of experiments show that our approach is feasible and effective.

Keywords*: MapReduce, Multiway Theta-Joins, Query Optimization, Skew, Cost Model*

1 Introduction

Cloud computing has been gaining more and more attention from the industry and academia. In this area, there are many important issues worthy of in-deep research, one of which is large-scale data processing. Data processing task on a share-nothing cluster can be performed on two kinds of systems: parallel relational database and MapReduce-based system. Parallel relational database technology, which has been used in mainstream data management over the past three decades, can adeptly solve data processing issues. However, the scalability of parallel relational database has encountered unprecedented obstacles. This approach is not qualified for the requirement of large-scale data analysis. According to the CAP [1], consistency, availability and tolerance to network partitions cannot be simultaneously satisfied in distributed systems. Parallel relational database, aiming at the pursuit of a higher level of consistency and fault tolerance, cannot reach an excellent scalability.

† The work is supported by the National Science and Technology Supported Program "National Financial Data Analysis and the Key Technologies and Applications of Information Service Based on Cloud Computing (2012BAH17B03)" and USTC-Lenovo Cloud Computing Laboratory.

* to whom correspondence should be addressed.

As a parallel computing model for data analyzing and processing, MapReduce [3] has triggered widespread concern since it was introduced by Google in 2004. In the early design, MapReduce was committed to the dealing with large-scale data analysis on low-cost server clusters. Its scalability and availability were placed in a priority position. Recently, a large number of data analysis applications have been expressed by MapReduce, including database query, data mining and graph processing. The related applications of MapReduce are discussed in book [9] in detail.

The optimization of the join operation in parallel relational database has been studied in many years, such as the optimization of equi-joins and theta-joins. Theta-join is a cartesian product filtered by an arbitrary condition which compares values from both datasets. Equi-join is a special case of theta-join. Join operation for large datasets using MapReduce is a hot issue in recent years. Some high-level languages have been proposed for MapReduce as Hive [5] and Pig [6], which automatically transform a SQL query into a set of MapReduce jobs. People can leverage them to process their data analysis without having to write a set of map and reduce functions. However, the optimization of the multi-way theta-joins has not been solved very well in Hive or Pig.

In our paper, the problem of the multi-way theta-joins is solved using MapReduce. Without modifying the original MapReduce environment, we can achieve the expected final results only by overwriting map and reduce functions. In particular, we make the following major contributions:

1. We propose a randomized algorithm named Strict-Even-Join (SEJ) for computing multi-way theta-joins in a single MapReduce job. It uses Lagrangian method to compute the approximate fragments of each relation and minimizes the communication cost between map and reduce phases. It can also guarantee that the data is balanced across reducers when input datasets are skew.
2. We describe the cost models of multi-way theta-joins and equi-joins respectively. Based on the algorithm SEJ and cost models, we design a dynamic programming algorithm to generate the best MapReduce implementation for multi-way theta-joins.
3. We validate the cost models and the algorithms' efficiency. The result shows that our algorithms are feasible and effective.

This paper is a revised and expanded version of the paper[1]. Here, we design a dynamic programming algorithm, describe the cost models of multi-way joins and experimental evidence for the benefit of our methods. This material includes Sections 4, 5, 6. The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 discusses the implementation details for Strict-Even-Join. Section 4 presents the query optimizer and the dynamic programming algorithm. Section 5 describes the cost models of theta-joins and equi-joins. Section 6 reports the results of the experiments. Section 7 concludes our paper.

2 Related Work

There is a rich history of studying join algorithms in the relation database[15, 19, 18]. [16, 17] are surveys on these algorithms. In particularly, the partition method of

[1]SEJ: An Even Approach to Multiway Theta-Joins using MapReduce, presented at the second international Conference on Cloud and Green Computing, (2012) November 1-3, Xiangtan, China

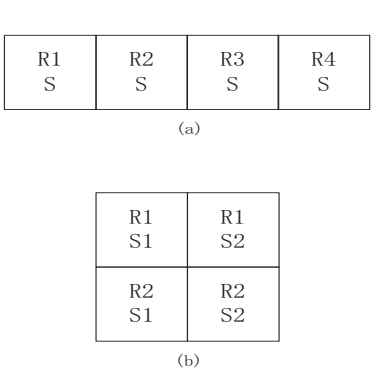
our work based on [18], we have proposed a random algorithm using MapReduce to solve multi-way theta-joins and provided a concrete method for constructing partition functions.

Recently, join algorithm tend to be an attractive point which cannot deal very well when MapReduce is used. S. Blanas et al. described crucial implementation details of four two-way joins strategies in MapReduce [4]. Afrati et al. solved a problem on how to optimize a multi-way joins in a single MapReduce job [7]. They suggested a method based on lagrangian multipliers to properly select how many buckets of the share variable for minimizing the sum of the communication costs. However, this algorithm is not compatible with nonequi-join and its efficiency would reduce visibly when input datasets are skew. Our algorithm can solve these two problems. S. Wu et al. developed a query optimization schema for MapReduce-based processing systems [14]. They also mainly considered the optimization of equi-join algorithm. A. Okcan et al. proposed how to efficiently perform two-way theta-joins in a single MapReduce job only [8]. This work cannot be naturally used to processing multi-way theta-joins. During the preparation of the paper, we notice that the work presented in a newly accepted paper [20] is similar to ours. Our work is an independent work. Compared to [20], we present a Lagrangian method to partition the relations, which is easily be used in pratice, and also provide a concrete method for constructing partition functions. Moreover, the MapReduce implementation of the equi-width histogram is proposed and the cost model of the equi-join is considered in our query optimizer, which can efficiently deal with multi-way joins. In [11], authors transform a batch of queries into a new batch that will be executed more efficiently by merging jobs into groups and evaluating each group as a single query. Their method can be integrated into our work to improve efficiency for multi-way theta-joins.

D. Jiang et al. presented Map-Join-Reduce [12], a system that extends and improves the MapReduce system to efficiently process data analytical tasks. Users could use three functions: map(), join(), and reduce() to join multiple data sets. H. Yang et al. proposed a new function merge() for simplifying join processing [13]. They added to the MapReduce system a merge phase that could efficiently merge data already partitioned and sorted by map and reduce modules. However, they changed the internal implementation of Hadoop. But our work is based on the default version of Hadoop, and is much easier to be accepted and used by other people.

3 Designing and Implementation of Strict-Even-Join

Let us consider three relations R $\bowtie$ S $\bowtie$ T. An obvious method is to implement this situation based on cascades of two two-way joins. At first, we do R $\bowtie$ S and then join the intermediate data and T to generate the final results. This means that there are two rounds of MapReduce processes. The intermediate data, generated by the first MapReduce process, needs to be written back to and read from DFS by the second MapReduce process, which is not an efficient implementation. As we all know, MapReduce is not suitable for the iterative process [10] because the initialization of the map phase and the access to DFS during the iterative process waste much time. Therefore, an alternative algorithm is expected to respond to requirement that all three relations are joined at once in a single MapReduce job. For a given join operation and the inputs, our aim is to minimize the completion time comprised of every phase of a MapReduce job. The network overhead between mappers and reducers during



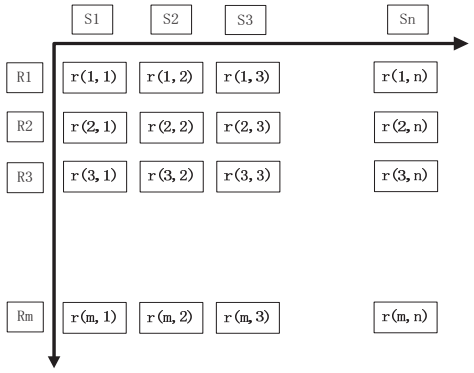


Figure 1. Exmaple for implementing RFigure 2. Replication pattern for $R \bowtie S \bowtie S$ using MapReduce

the shuffle phase will consume a lot of time, and the unbalanced data handled by reducers will also delay the execution time of the whole process. Nevertheless, the latter situation will not happen because our algorithm guarantees even distribution of the data across reducers. So we can mainly focus on the network overhead during the shuffle phase in the MapReduce process. Our goal is to minimize this network overhead.

3.1 Multi-way Theta-joins in a Single MapReduce Job

First, a two-way theta-joins $R \bowtie S$ is considered. Let us assume that we have four reducers. R is fragmented into four disjoint fragments $(R1, R2, R3, R4)$ and the following relationship is guaranteed: $R1 \cup R2 \cup R3 \cup R4 = R$. To process the join operation in parallel, the map function sends 4 fragments of R to 4 reducers, and sends the entire S to all reducers. In each reducer, the reduce function produces the final results to DFS. Fig. 1(a) shows this situation. To obtain the correct results, each reducer has to receive one of the four fragments R and the entire relation S, because we don't know which values of R and S satisfy the theta-join condition if we don't sample from R and S beforehand. The communication cost between mappers and reducers in the shuffle phase is $r + 4s$, where r and s represent the size of relation R and S respectively. Fig. 1(b) is another segmentation pattern. Relations R and S are fragmented into $(R1, R2)$ and $(S1, S2)$ respectively. At the same time, each fragment must comply to the following relationships: $R1 \cup R2 = R$; $S1 \cup S2 = S$. We use a two-dimensional cell to represent four reducers. In the MapReduce environment, the map function sends $R1$ to reduce (1,1) & reduce (1,2), sends $R2$ to reduce (2,1) & reduce (2,2), sends $S1$ to reduce (1,1) & reduce (2,1), and sends $S2$ to reduce (1,2) & reduce (2,2). In the reduce phase, the reduce function computes the final results. It's easy to prove that the correct results will be thusly produced in this way, without losing any possible results. The communication cost between mappers and reducers in the shuffle phase is $2r+2s$, which may reduce communication cost by a factor of $(2s-r)$ compared with the former situation. So what's the best segmentation pattern? To answer this question, relations R and S are fragmented into disjoint fragments $(R_1, R_2, \ldots, R_m)$ and $(S_1, S_2, \ldots, S_n)$. From Fig. 2, in order to get the correct results, the map function must send R_i to reduce (i,*) ('*' stands for matching anything) in the horizontal direction and send S_j to reduce (*, j) in the vertical direction. The communication

cost between mappers and reducers is: $cost = r \times n + s \times m$, and constraints are followed: $n \times m = k$, where k represents the number of reducers and m and n are positive integers. The minimum cost is: $cost = rn + sm \geq 2\sqrt{rn \times sm}$. The condition of the minimum cost is: $m = \sqrt{\frac{rk}{s}}$ and $n = \sqrt{\frac{sk}{r}}$.

Similarly, for a three-way theta-joins R $\bowtie$ S $\bowtie$ T, all reducers should be represented by a three-dimensional cell. The map function sends R_i to reducer (c_i,*,*), sends S_j to reducer (*,c_j,*) and sends T_k to reducer (*,*,c_k). To extend this method for the multi-way theta-joins, we use the following notation to simplify our discussion:

- k = the number of reducers
- n = the number of relations involved in the theta-join
- R_i = the i^{th} relation for i = 1,2,...,n
- r_i = the size of R_i relation
- cell (L_1,L_2,...,L_n) = an n-dimensional cell, with L_i being the fragments of i^{th} relation in the cell and $L_1 \times L_2 \times \ldots \times L_n = k$
- reduce (c_1,c_2,...,c_n) = an n-dimensional reducer, with $1 \leq c_i \leq L_i$.

When doing multi-way theta-joins, the map function must send R_1 to reduce (c_1,$\overbrace{*\ldots*}^{n-1}$), and send R_i to reduce ($\overbrace{*\ldots*}^{i-1}$,$c_i$,$\overbrace{*\ldots*}^{n-i}$). When sending relation R_i, the map function makes k/L_i new copies of R_i and send each copy to a different reducer. The communication cost of sending R_i is $r_i \times k/L_i$. Therefore, the total communication cost of the multi-way theta-joins is:

$$cost = \sum_{1}^{n} r_i \times \frac{k}{L_i}. \tag{1}$$

The constraints are following : $\prod_{i=1}^{n} L_i = k$, where L_i is a positive integer. The lagrangian multipliers is used to solve this optimization problem. The function F is formed:

$$F = \sum_{1}^{n} r_i \times \frac{k}{L_i} + \lambda(\prod_{i=1}^{n} L_i - k). \tag{2}$$

F is used to partial derivative of L_i:

$$\frac{\partial F}{\partial L_i} = -r_i \times \frac{k}{L_i^2} + \lambda \times \frac{k}{L_i} = 0. \tag{3}$$

L_i is solved:

$$L_i = \frac{r_i \times k^{\frac{1}{n}}}{(\prod_{i=1}^{n} r_i)^{\frac{1}{n}}}. \tag{4}$$

The minimal cost is:

$$\min cost = n \times (\prod_{i=1}^{n} r_i)^{\frac{1}{n}} \times k^{1-\frac{1}{n}}. \tag{5}$$

Note that the value of L_i is not necessarily integer. However, the values tell us approximately how many fragments each relation should be divided into. They also

tell us the desired ratios of the relations, for example, $L_i/L_j = r_i/r_j$, the number of fragments for a relation is proportional to its size. We can pick good integer approximation to L_i, as well as the value of k that is in the approximation range.

We take an example of two-way theta-joins to understand this result. Assume we have 36 reducers to join two relations R and S; moreover, both R and S have the same size. Without any optimization, the map function will send 36 fragments of R to the corresponding reducer and send the entire S to every reducer. Therefore, R is copied one time and S is copied 36 times. The communication cost is $r+r\times36 = 37r$. After optimization, according to formula (4), $L_1 = 6$; $L_2 = 6$. So R is copied 6 times and S is also copied 6 times. The communication cost is $r\times6 + s\times6 = 12r$. This can reduce communication cost by a factor of $37r/12r = 3.08$.

Let us consider the general condition. Without any optimization, the communication cost of the multi-way theta-joins is $1+r_2\times k+r_3\times k+\ldots+r_n\times k \approx O(nrk)$. After optimization according to formula (5), the communication cost is reduced to $O(nrk^{1-\frac{1}{n}})$.

3.2 Data Partitioning

In the last section, we get an n-dimensional cell to partition the input datasets. When the map task is launched, every tuple of one input dataset will be sent to the corresponding reducer according to the partition signature which is an n-dimensional vector. For instance, the tuple of input datasets belongs to $r(2, 1, 2)$ that should be sent to the sixth reducer. However, the default version of Hadoop can only shuffle an intermediate pair based on a single partition signature value.

To solve this problem, we must convert this n-dimensional partition signature into a single value which represents the number of the reducer. Given the n-dimensional partition signature $s = reduce(c_1, c_2, \ldots, c_n)$ and the n-dimensional partition cell (L_1,L_2,...,L_n), the single signature value s is calculated by formula (6).

$$s = \sum_{i=1}^{n-1} (c_i - 1) \times (\prod_{j=i+1}^{n} L_j) + c_n - 1. \tag{6}$$

For example, we have a three-dimensional partition cell $(2, 2, 2)$. The three-dimensional vector can be converted to a single value according to formula (6). The result is shown in Figure 3.

1, 1, 1	0
1, 1, 2	1
1, 2, 1	2
1, 2, 2	3

2, 1, 1	4
2, 1, 2	5
2, 2, 1	6
2, 2, 2	7

Figure 3. Example of converting 3-dimensional cell to a single value

3.3 Algorithm

We shall now describe the algorithm SEJ that yields the minimum cost optimization of multi-way theta-joins in a single MapReduce job.

Step 1. Write the cost expression; Construct the Lagrangian equations for the join; Find the optimal solution $(L_1, L_2, \ldots L_n)$ which is stored in $divide[1,\ldots,n]$.

Step 2. Using the input datasets $R_1, \ldots, R_n$ and the $divide[1,\ldots,n]$, we can write the map and reduce function to deal with multi-way theta-joins. The map and reduce functions are shown in Algorithm 1. The function $findNum_i$ in line 5 in the map function will not be described here, as it can be implemented according to formula (6).

Algorithm 1: Strict-Even-Join (multi-way theta-joins in a single MapReduce job)

```
1  MapInput : x ∈ R1 ⋃ R2 ... ⋃ Rn, divide[1 ... n]
2  switch x do
3      case x ∈ Ri
4          key = random(1, Li)
5          for number in findNumi(key, divide) do
6              output(number, (x,"Ri"))
7          end
8      endsw
9  endsw
10 ReduceInput : (number, [(x1, tagi), (x2, tag2), ..., (xk, tagk)])
   tupleList1 = ∅, ..., tupleListn = ∅
11 for xj, tagj in inputList do
12     if tagj = Ri then
13         tupleListi = tupleListi ⋃ {xj}
14     end
15 end
16 joinResult = thetaJoinAlg(tupleList1, ..., tupleListn)
17 output(joinResult)
```

3.4 Analysis of Skew Data

SEJ can implement any multi-way theta-joins in the reduce process. It can also balance the data across reducers. Considering a theta-join with selectivity σ, the algorithm produces $\sigma|R_1||R_2|\ldots|R_n|$ output tuples. Each reducer should be responsible for $\sigma|R_1||R_2|\ldots|R_n|/k$. What would happen when input datasets are skew. For example, some reducers might have much more values that satisfy the join condition, while other reducers have nearly non of that. Fortunately, this is very unlikely to take place because of the randomization of assigning tuples from R_i $(i = 1\ldots n)$. Our experiments in Section 6 will show that the join output is generally distributed over reducers evenly although we don't have an analytical proof. The larger join output size reducers produce, the smaller the sample variance will be. Sample variance in output is only likely when the size of output is very small. However, in this case, the total join output is so small that output imbalance has a small effect on the execution time.

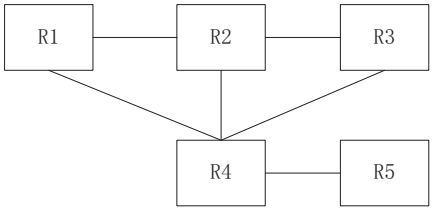


Figure 4. Example of a join graph

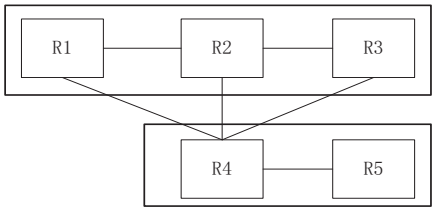


Figure 5. Example of a partition for a join graph

4 Optimizing Multi-Way Theta-Joins using MapReduce

In the last section, the communication cost in the shuffle phase of SEJ is $O(nrk^{1-\frac{1}{n}})$. We can draw a conclusion that the more relations are joined in one step, the more replications of tuples that are necessary, the more cost of the shuffle phase that is. Although SEJ can save the MapReduce rounds, it will increase the network overhead. Thus, it is a good idea to partition relations $R_1 \bowtie R_2 \ldots \bowtie R_n$ into join groups. Each join group executes two-way or multi-way joins, which is computed by a single MapReduce job. Then we search for the best plan to generate the final results by combining these join groups.

4.1 Generating Optimal Query Plan

To simplify the discussion, we first define a join graph for queries $R_1 \bowtie R_2 \ldots \bowtie R_n$.

Definition 1 *Join Graph*
Given a query Q, the join graph of Q is defined as $G_Q = (V, E)$*, where*

- *if relation* R_i *is involved in Q ,* R_i *is a note in V*
- *if relations* $R_i \bowtie_{i.k,j.k} R_j$ *is a join condition in Q, an undirectred edge* $e = (R_i, R_j)$ *exists in E.*

An example of a joining graph for $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$ is shown in Fig. 4.

Definition 2 *A partition p of joining graph G, satisfying:*

- $\forall\ G_i \in p$, G_i *is a subgraph of G. Given a join graph* G_Q*=(V,E),* $G_i = (V_i, E_i)$ *is a subgraph of G, satisfying:* $\forall\ v \in V_i$, $v \in V$; $\forall\ e \in E_i$, $e \in E$.
- $\forall\ G_i \in p$, $G.V = \cup_{G_i \in p} G_i.V$.
- $\forall\ G_i, G_j \in p$, $G_i.V \cap G_j.V = \phi$.

Based on the above definition, all the relations are included in a division p of G_Q. Each subgraph of a partition p corresponds a join strategy. An example of one partition p containing two parts is shown in Fig. 5. For a subgraph G_i in p, the join strategies are as follows:

- if $|G_i.V| <= 1$, no join is defined.
- if $|G_i.V| = 2$, if join operation is "=", improved repartition join [4] is used; else SEJ is used.

- if $|G_i.V| >= 3$, if each of join operations is "=", multi-way equi-join [7] is used; else SEJ is used.

Algorithm 2: GenerationQueryPlan

Input: QueryGraph G
Output: OptimalPlan Op

```
1  for int i=2 to n do
2      for ∀ G_i in G do
3          min_cost = MAX
4          for ∀ partition p for G_i do
5              cost = estimatedCost(G_i^p)
6              if cost < min_cost then
7                  min_cost = cost
8                  G_i^p.plan = plan
9              end
10         end
11     end
12 end
13 return G.plan;
```

Algorithm 2 shows how to generate the best plan for multi-way theta-joins using MapReduce. We iterate all subgraph G_i containing i nodes of G_Q (line 2). For a specific G_i, all possible partitions for G_i are iterated (line 4). Then we select a minimum cost and a corresponding plan for G_i (line 5-8). The function *estimatedCost* will be discussed in Section 5. At last, this algorithm returns the best plan for the multi-way theta-joins using MapReduce.

4.2 Concurrent MapReduce Jobs

In Hadoop, concurrent MapReduce jobs may lead a worse performance, as jobs will compete for computing resources. Therefore, we perform a simple analysis to decide whether multiple MapReduce jobs should be simultaneously submitted. Given a job set $J = \{j_0, \ldots, j_n\}$, the jobs can be simultaneously submitted, satisfying:

- $\forall\ j_i, j_k \in J$, j_i and j_k are independent, which means they don't depend on each other's results.
- Let $m(j_i)$ and $r(j_i)$ denote the number of mappers and reducers of each job respectively. When $\sum_{i=0}^{i=n}(r(j_i) < MaxMappers$ and $\sum_{i=0}^{i=n}(m(j_i) < MaxReducers$, where $MaxMappers$ and $MaxReducers$ denote max numbers of mappers and reducers in the Hadoop System, respectively.

This strategy is similar to the strategy used in [14]. In this paper, we adopt the above simple strategy to improve the parallelism, which can avoid concurrent jobs to compete for the resources.

5 Cost Model of Multi-way Joins using MapReduce

To evaluate the cost of a specific plan, we should propose a cost model for the MapReduce framework. Generally, for MapReduce jobs, the heavy cost on a large-

scale sequential disk scan, as well as the frequent I/O of intermediate results, are to dominate the entire execution time. Therefore, the execution time model for a MapReduce job should be built based on the analysis of disk I/O cost and network I/O cost. Moreover, since entire input data may not be loaded into the system memory within one round [21], the model should also take account of this situation. We assume these map tasks are performed round by round, thus, the cost of the map phase and shuffle phase can be added by rounds. Before the discussion, parameters used in the analysis are listed in Table 1.

Table 1. Parameters

Definition	Parameter	Definition	Parameter
cost ratio of HDFS reads	ρ	join selectivity of R_1 and R_2	$g(R_1, R_2)$
cost ratio of HDFS writes	η	number of mappers	m
cost ratio of Network I/O	μ	number of reducers	k
number of tuples in R	$\|R\|$	accumulative selectivity of R	α_R
size of $R^{'}$ tuple	$f(R)$	projection selectivity of R	β_R

5.1 Building Equi-Width Histogram using MapReduce

Before evaluating the cost models of multi-way joins, we should build histograms for join attributes of relations to estimate the selectivity of predicates and joins. The similar work is shown in [14]. In this paper, we build the equi-width histogram for each join attributes of each relation, which is illustrated in algorithm 3. We parse each line of the relation into individual join attributes (line 1). For each join attribute i, suppose that its domain is [$low[i]$,$high[i]$], the jth bucket covers the range $[low[i] + \frac{j(attribute[i]-low[i])}{w}, low[i] + \frac{(j+1)(attribute[i]-low[i])}{w}]$, where w is the bucket width for attribute i. The Function $getID$ uses this rule to return a $bucketID$ for each join attribute (line 4). Then a key-value pair by composting the attribute ID and its corresponding $bucketID$ is generated to be sent to reducers. The reduce function computes the number of each $bucketID$ and stores the histograms to DFS. In current implementation, the equi-width histogram is simple and provide good enough estimations. The method of building more sophisticated histograms using MapReduce is orthogonal to our work, and we leave it in the future work.

Algorithm 3: Building Equi-width Histogram using MapReduce

```
1  MapInput: each record r of relation R
2  Object[] attributes = parse(r)
3  for int id=0 to attributes.length do
4      int bucketID = getID(id, attributes[i], low[i], high[i])
5      output(< id, bucketID >, 1)
6  end
7  ReduceInput : (< id, bucketID >, [1, ..., 1])
8  for value in inputList do
9      histogram[id][bucketID]+ = value
10 end
11 output(histogram)
```

5.2 Cost Model of Theta-Joins using MapReduce

We evaluate the cost model of multi-way theta-joins. For each map task, it receives the split of each relation R_i. The disk I/O cost $t_{Thetajoin-map}$ of each map task is:

$$t_{Thetajoin-map} = \rho \times \frac{S_I^{R_1,\ldots,R_n}}{m}. \tag{7}$$

where $S_I^{R_1,\ldots,R_n} = |R_1|f(R_1)+\ldots+|R_n|f(R_n)$. As a general assumption, each relation R_i is considered to be evenly partitioned among the m map task and $m^{'}$ is the current number of map tasks running in parallel in the system. Thus, the total cost of the map phase $T_{TwowayThetajoin-map}$ is:

$$T_{TwowayThetajoin-map} = t_{Thetajoin-map} \times \frac{m}{m^{'}}. \tag{8}$$

Let $t_{Thetajoin-shuffle}$ be the cost for copying the output of a single map task to k reduce tasks, including the data copying over network cost, as well as overhead of all serving network protocols.

$$t_{Thetajoin-shuffle} = \mu(\Sigma_{i=1}^{i=n}\frac{\alpha_{R_i}\beta_{R_i}|R|f(R_i)k}{mL_i} + q \times k). \tag{9}$$

q is a random variable which represents the cost of a map task serving k connections from k reduce tasks. Intuitively, there is a rapid growth of q as k gets larger. α denotes the output ratio of a map task, which is query specific and can be computed with the selectivity estimation. β denotes the projection selectivity (the tuple size is reduced to $\beta \times 100\%$ of its original size after ruling out the unnecessary columns). Since there are $m/m^{'}$ rounds in the map phase, thus the total cost of the shuffle phase $T_{Thetajoin-shuffle}$ can be computed as follows:

$$T_{Thetajoin-shuffle} = t_{Thetajoin-shuffle} \times \frac{m}{m^{'}}. \tag{10}$$

Each reduce task performs a cross-product and stores the results to HDFS. Thus the cost of the single reduce task $T_{TwowayThetajoin-reduce}$

$$T_{Thetajoin-reduce} = \eta\frac{J\Pi_{i=1}^{i=n}\alpha_{R_i}|R_i|}{k}. \tag{11}$$

where $J = g(R_1,\ldots,R_n)\Sigma_{i=1}^{i=n}(\beta_{R_i}f(R_i))$. Hence the total cost of theta-joins using MapReduce $T_{Thetajoin}$

$$T_{Thetajoin} = T_{Thetajoin-map} + T_{Thetajoin-shuffle} + T_{Thetajoin-reduce} \tag{12}$$

5.3 Cost Model of Equi-Joins using MapReduce

We evaluate the cost of two-way joins and multi-way joins respectively, since their shuffle cost models are different from each other. Let $T_{TwowayEquijoin-map}$, $T_{TwowayEquijoin-reduce}$, $T_{NwayEquijoin-map}$ and $T_{NwayEquijoin-reduce}$ denote the map and reduce of two-way and multi-way equi-joins, respectively. They are the same

as that of multi-way theta-joins. We no longer describe them. The shuffle cost of two-way equi-joins $T_{TwowayEquijoin-shuffle}$ is calculated by formula (13).

$$T_{TwowayEquijoin-shuffle} = \mu(\frac{\alpha_R\beta_R|R|f(R)}{mk} + \frac{\alpha_S\beta_S|S|f(S)}{mk} + qk) \times \frac{m}{m'} \quad (13)$$

The shuffle cost of multi-way equi-joins $T_{NwayEquijoin-shuffle}$ is different from that of two-way equi-joins. Suppose the relations are joined on a attribute set χ. We use c_x to denote the number of reducers for attribute $a_x(a_x \in \chi)$. According to [7], we have $k = \prod_{x=1}^{x=n} c_x$. To improve the performance, the number of required reducers is set to be proportional to the size of corresponding relation. Thus, c_x can be computed. For relation R_i, if it contains a join attribute set $\chi^{'}(\chi^{'} \subset \chi, \chi = \{A, B, C\})$, we need to replicate its data to c_{R_i} reducers, where $c_{R_i} = \prod_{\forall r_i \notin \chi' \bigwedge r_i \in \chi} c_x$. Therefor, the shuffle cost of multi-way equi-joins $T_{NwayEquijoin-shuffle}$

$$T_{NwayEquijoin-shuffle} = \mu(\Sigma_{i=1}^{i=n}\frac{I_{R_i}}{mk} + qk) \times \frac{m}{m'} \quad (14)$$

Where $I_{R_i} = c_{R_i}\alpha_{R_i}\beta_{R_i}|R_i|f(R_i)$ and $c_{R_i} = \prod_{\forall r_i \notin \chi' \bigwedge r_i \in \chi} c_x$. The interesting reader can further read more detais in [7].

6 Experiments

All experiments are executed on 10 blades running Hadoop 0.20.2 [2], each with 2.4GHz*12 core CPU, 20G RAM, 270G hard disk. All blades are directly connected to a Gigabit switch. Each blades runs at most 12 map tasks and 12 reduce tasks. The other major Hadoop parameters are listed in Table 2.

Table 2. Hadoop parameter confuguration

Parameter	value	Parameter	value
fs.blocksize	$64M$	io.sort.spill.percentage	0.8
io.sort.mb	$100M$	io.sort.factor.	100
io.sort.record.percentage	0.05	dfs.replication	3

6.1 Effect of SEJ

Firstly, we compare SEJ with cascades of two-way theta-joins. Suppose The three-way joins $R(A, B) \bowtie S(B, C) \bowtie T(C, A)$ needs to be computed. Each dataset $R(A, B)$, $S(B, C)$ and $T(C, A)$ contains 0.1 million records; each record has ten attributes and each attribute is drawn uniformly at random from its range. We take the following four cases as examples:

- Case 1:

SELECT r.A, s.B, t.C
FROM R as r, S as s, T as t
WHERE $r.B - s.B > 95000$ AND $s.C = t.C$
AND $t.A = r.A$.

- Case 2:

SELECT r.A, s.B, t.C
FROM R as r, S as s, T as t
WHERE $|r.B - s.B| < 3$ AND
$t.A - r.A > 98000$ AND $s.C = t.C$.

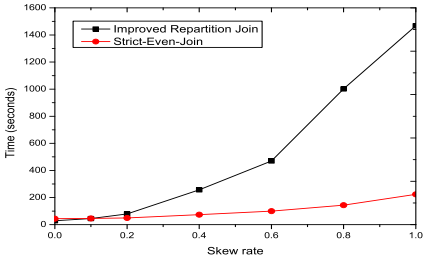


Figure 6. MapReduce time for two-way joins

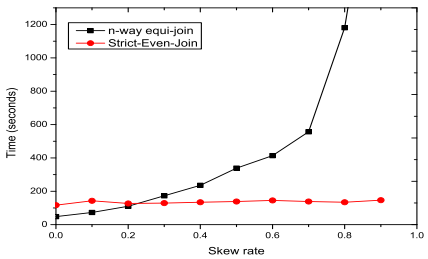


Figure 7. MapReduce time for multi-way joins

- Case 3:

  SELECT r.A, s.B, t.C
  FROM R as r, S as s, T as t
  WHERE $|r.B - s.B| < 3$ AND
  $|s.C - t.C| < 3$ AND $|t.A - r.A| < 3$.

- Case 4:

  SELECT r.A, s.B, t.C
  FROM R as r, S as s, T as t
  WHERE $r.B - s.B > 99000$ AND
  $s.C - t.C > 99000$ AND $t.A - r.A > 99000$.

Table 3 shows SEJ is generally better than cascades of joins in two MapReduce processes, because it wastes too much time for generating intermediate data, writing them to DFS by the first MapReduce job and reading them from DFS by the second MapReduce job. In particular, in the first case, the size of the intermediate result by $R \bowtie S$ is about 10^7 records that leads to the worse performance of cascades of joins than that of SEJ. In the second case, the performance of cascades of joins for computing $((S \bowtie T) \bowtie R)$ is nearly the same as the performance of SEJ, because the intermediate data is small enough that accessing to DFS wastes little time.

Table 3. MapReduce time (in seconds) for cascades of two two-way joins and Strict-Even-Join.

Case	R⋈S=temp temp⋈T	S⋈T=temp temp⋈R	R⋈T=temp temp⋈S	R⋈S⋈T
1	14400	1183	1180	983
2	1340	1160	3603	1140
3	1514	1516	1513	1020
4	2080	2083	2081	1422

Secondly, we compare SEJ with the two-way equi-join [4] and the multi-way equi-joins algorithm [7] when dealing with the skew data. The frequency of key dividing the total data size is defined as the skew rate. The results are shown in Fig. 6 and Fig. 7, respectively. When input datasets are uniform, their algorithms performances are better than ours, as the communication cost between map and reduce of SEJ is more than that of euqi-joins algorithms. However, with the increasing of skew rate, the efficiency of equi-joins algorithm declines because the skew input datasets make some reducers deal with too much data. This will increase the overall execution time of MapReduce process. On the other hand, SEJ has a stable performance when the skew rate increases, because it ensures the data is evenly distributed across reducers.

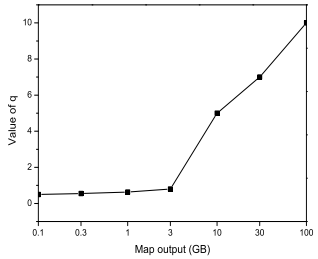


Figure 8. Value distribution of q

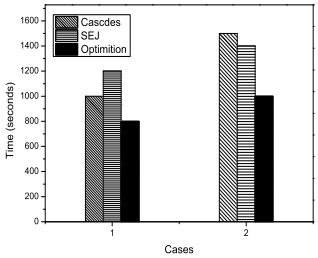


Figure 9. Optimization

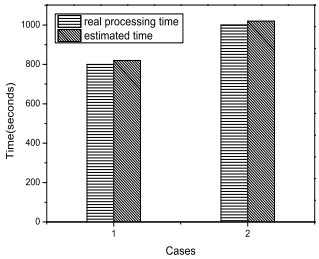


Figure 10. Cost Model Validation

6.2 Effect of Query Optimization

We use the TestDFSIO program to test the I/O performance of the system. In our cost model, the cost ratios of of Table 1 are set as follows: HDFS read (ρ)=1, HDFS write (η)=1.5, network I/O (μ)=1.5. Then we should compute the distribution of q which serves the estimation of MapReduce's running time. The number of reduce tasks is set to 108 and the value of q is computed by studying an output controllable program over a series of test data. The result is shown in Fig. 8. Then we add the number of join relations and use Algorithm 2 to select a best MapReduce implementation for four-way theta-joins: $R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(D,A)$. The size of each relation is 0.1 millions; each record has two attributes, each attribute is drawn uniformly at random from its range. The following two cases are taken as examples. Fig. 9 shows the results of the three kinds of methods for these two cases. We can observe that the join plan generated by our query optimizer can improve the efficiency of the multi-way theta-joins. In our implementation, we apply the I/O based model for its simplicity. As shown in Fig. 10, our estimation and real MapReduce execution time are very close.

- Case 1:

SELECT r.A, s.B, t.C, u.D
FROM R as r, S as s, T as t, U as u
WHERE $|r.B - s.B| < 2$
AND $|s.C - t.C| > 99900$
AND $t.D = u.D$ AND $u.A = r.A$.

- Case 2:

SELECT r.A, s.B, t.C, u.D
FROM R as r, S as s, T as t, U as u
WHERE $|r.B - s.B| < 2$
AND $|s.C - t.C| < 2$
AND $t.D - u.D > 99900$
AND $u.A - r.A > 99900$

7 Conclusion and Future Work

In this paper, we propose an random algorithm SEJ for implementation of multi-way theta-joins in a single MapReduce job. To minimize the communication cost between map and reduce phases, Lagrangian method is used to compute the approximate fragments of each relation. The result of the experiment shows that in most situations it is more efficient to join a multi-way joins in a single MapReduce job than cascades of two-way joins. When input datasets are skew, compared with other existing algorithms, SEJ is more stable and efficient due to its even distribution of datasets. Moreover, a dynamic programming algorithm is designed to partition

multi-way theta-joins into subgroups and select a best MapReduce implementation by combining these subgroups for this problem.

There is more work to do in the future. The concurrency strategies for the join operation in MapReduce should be considered and the efficient approach of building histograms using MapReduce will also developed in the upcoming work.

8 Acknowledgements

Thanks for the platform and the technical support provided by the Supercomputing Center of University of Science and Technology of China.

References

[1] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, **(2002)**, pp. 51-59.

[2] http://hadoop.apache.org/ 2012

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", In OSDI, **(2004)**, pp. 137-150.

[4] 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, **(2010)**, pp. 975-986.

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R. Murthy, "Hive - A Petabyte Scale Data Warehouse Using Hadoop", In ICDE, **(2010)**, pp. 996-1005.

[6] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing", In SIGMOD, **(2008)**, pp. 1099-1110.

[7] F. N. Afrati and J. D. Ullman, "Optimizing Multiway Joins in a Map-Reduce Environment", IEEE Transaction on Knowledge and Data Engineering, VOL. 23, NO. 9, **(2011)**, pp.1282-1297.

[8] A. Okcan and M. Riedewald, "Processing Theta-Joins using MapReduce", In SIGMOD, **(2011)**, pp. 949-960.

[9] J. Lin and C. Dyer, "Data-Intensive Text Processing with MapReduce", Synthesis Lectures on Human Language Technologies,**(2010)**.

[10] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu and G. Fox, "Twister: A Runtime for Iterative MapReduce", In HPDC, **(2010)**, pp. 810-818.

[11] T. Nykiel, M. Potamias, C. Mishra, G. Kollios and N. Koudas, "MRShare: Sharing Across Multiple Queries in MapReduce", In VLDB, **(2010)**, 3(1-2): 494-505.

[12] D. Jiang, A. K. H. Tung and G. Chen, "MAP-JOIN-REDUCE: Toward Scalable and Efficient Data Analysis on Large Clusters", IEEE Transaction on Knowledge and Data Engineering, VOL. 23, NO. 9, **(2011)**, pp. 1299-1311.

[13] H. Yang, A. Dasdan, R. Hsiao and D.S. Parker, "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters", In SIGMOD, **(2007)**, pp. 1029-1040.

[14] S. Wu, F. Li, S. Mehrotra and B. C. Ooi, "Query Optimization for Massively Parallel Data Processing", In SOCC, **(2011)**, pp. 12:1-12:13.

[15] D. A. Schneider and D. J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", In SIGMOD, **(1989)**, pp. 110-121.

[16] G. Graefe, "Query Evaluation Techniques for Large Databases", ACM Comput. Surv., VOL. 25, NO. 2, **(1993)**, pp. 73-169.

[17] P. Mishra and M. H. Eich, "Join Processing in Relational Databases", ACM Comput. Surv., VOL. 24, NO. 1, **(1992)**, pp. 63-113.

[18] J. W. Stamos and H. C. Young, "A Symmetric Fragment and Replicate Algorithm for Distributed Joins", IEEE Trans. on Par. and Dist. Sys. , VOL. 4, NO. 12, **(1993)**, pp. 1345-1354.

[19] P. A. Bernstein and N. Goodman, "Query Processing in a System for Distributed Databases (SDD-1)", ACM Transactions on Database Systems, VOL. 6, NO. 4, **(1981)**, pp. 602-625.

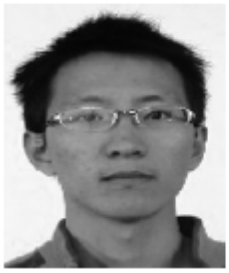
[20] X. Zhang, L. Chen and M. Wang, "Efficient Multi-way Theta-Join Processing Using MapReduce", Proceedings of the VLDB Endowment, **(2012)**, August 27-31; Istanbul, Turkey.

[21] P. Agrawal, D. Kifer and C. Olston, "Scheduling shared scans of large data files", Proceedings of the VLDB Endowment, **(2008)**, August 24-30; Auckland, New Zealand.

Authors

Changchun Zhang received the B.S. degree in computer science from the University of Science and Technology of China (USTC) in 2009. He is currently working toward the Ph.D. degree at the Department of Computer Science of USTC. His research interests include parallel algorithms, Cloud Computing and high performance computing.

Jing Li received his B.E. in Computer Science from University of Science and Technology of China (USTC) in 1987, and Ph.D. in Computer Science from USTC in 1993. Now he is a Professor in the School of Computer Science and Technology at USTC. His research interests Distributed Systems, Cloud Computing and Mobile Computing.



Lei Wu received the B.S. degree in computer science from the University of Science and Technology of China (USTC) in 2009. Now he is a M.E. candidate in Computer Science at USTC. His research interests parallel algorithms, Cloud Computing and high performance computing.