J. Parallel Distrib. Comput. 121 (2018) 42–52

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

An efficient theta-join query processing in distributed environment

Wenjie Liu [*], Zhanhuai Li

School of computer, Northwestern Polytechnical University, ShaanXi, Xi'an, 710072, China

HIGHLIGHTS

- Effective Max and Min values based filter strategy for theta-join computing in distributed environment.
- Divide and Merge method for theta-join which reduces network overheads greatly.
- Extensive experiments using real world and synthetic data sets.

ARTICLE INFO

Article history:
Received 12 March 2016
Received in revised form 3 April 2018
Accepted 5 July 2018

Keywords:
Parallel distributed framework
Theta-join algorithm
Query optimization
Large scale data processing

ABSTRACT

Theta-join query is very useful in many data analysis tasks, but it is not efficiently processed in distributed environment, especially in large scale data. Although there is much progress in dealing theta-join with MapReduce paradigm, the methods are either complex which require fundamental changes to MapReduce framework or only consider the overheads of load balance in the network, when data scale is large, they will make much computation cost and induce OOM (Out of Memory) errors. In this work, we propose a filter method for theta-join on the purpose of reducing the computation cost and achieving the minimum execution time in distributed environment. We consider not only the load balance in the cluster, but also the memory cost in parallel framework. We also propose a keys-based join solution for multi-way theta-join to reduce the data amount for cross product, then improve the performance of join efficiency. We implement our methods in a popular general-purpose data processing framework, Spark. The experimental results demonstrate that our methods can significantly improve the performance of theta-joins comparing with the state-of-art solutions.

1. Introduction

Large scale data processing related to analytic queries involves theta-join operations, which is becoming one of the most important challenges in recent years. Theta-join is defined as a binary function θ which belongs to $\{<, \leq, =, \geq, >, <>\}$. MapReduce is a prevalent framework to process large scale data in parallel, and can be used to process join operations. Due to the inherent limitations of key-equal feature, it can be easily used to support equal joins, but cannot be directly used for theta-joins.

Recently, there are a few works that focus on processing theta-join in MapReduce framework. 1-bucket-theta is a method proposed to evaluate one single theta-join in one MapReduce Job [11]. Its main idea is to balance the workload among reducers. It partitions the cross-product results of two input tables for theta-join with many rectangle regions of bounded size. The records in one region are distributed to one reducer, as each region includes almost same amounts of data, each reducer will receive the average workload, thus the parallelism of the system can be achieved. A multi-way theta-join is also processed by using 1-bucked-theta [18]. The method implements a chain-typed theta-join by using Hilbert curve. As Hilbert curve requires same scale for data sets, it cannot be used for join tables with different size. A randomized algorithm named Strict-Even-Join (SEJ) is designed to solve the multi-way theta-joins in a single MapReduce job [19]. It uses Lagrangian method to compute the approximate fragments of each relation and minimizes the communication cost between map and reduce phases. But the partition idea is as the same as 1-bucket-theta. To reduce the high I/O cost of intermediate results, a method which uses just two MRJs (MapReduce Jobs) to implement multi-way theta-joins in MapReduce has been proposed, multi-way theta-join is decomposed into a non-equal-join and a multi-way equal-join. But in each theta-join processing, it is also processed by 1-bucket-theta [15]. The latest work about theta-join is based on sorting, permutation-arrays and bit-arrays. It puts columns to be joined in sorted arrays and uses permutation arrays to encode positions of tuples in one sorted array w.r.t. the other sorted array [9]. This method needs fundamental changes of parallel frameworks, so it cannot be easily used in real applications.

[*] Corresponding author.
E-mail addresses: liuwenjie@nwpu.edu.cn (W. Liu), lizhh@nwpu.edu.cn (Z. Li).

https://doi.org/10.1016/j.jpdc.2018.07.007
0743-7315/© 2018 Elsevier Inc. All rights reserved.

From above works, we notice that 1-bucket-theta is the basis for most of theta-join and multi-way theta-join processing, it is widely used in many algorithms. Although it is useful and efficient, it is only considering the load balance among reducers, and not considering how to reduce the computation cost. When input tables are very large, it will gain low efficiency.

In our paper, we propose a method which is also based on 1-bucket-theta, but do an efficient filter to prune much irrelevant records before cross product. We propose a key-based filter method which uses only the join attributes. The method is called max and min values based filter strategy (MMF method). We use the strategy to filter the input data records before we do cross product, which is widely used in processing theta-joins. As known to all, cross product is a time-consuming operation, especially for large data in a parallel framework, too many intermediate results may make OOM errors occur. So deleting useless records which do not contribute to the final results will reduce the memory cost and copied data transferred in network. We also apply our method into multi-way theta-join and design a key-based join strategy. We implement our methods and the baseline method of 1-bucket-theta in a prevalent distributed data processing framework-Spark, and conduct extensive experiments on large scale real and synthetic data sets. The results show our method outperforms 1-bucket-theta for theta-join queries.

The contributions of our work are summarized as follows:

- We proposed a max and min values based filter method for computing theta-join in a distributed framework. It uses the max and min values of the join attributes to filter useless data which are irrelevant to the final results from the input data sets, and then uses the idea of 1-bucket-theta to evenly partition the cross product results among reducers to achieve load balance from the parallel framework. It effectively reduces the memory cost and network overhead, and therefore improves the join efficiency.
- We propose a new join strategy for multi-way theta-join, which does the cross product only on the join attributes, then use equi-join to merge other output attributes, which greatly reduced the intermediate results of cross product and improved computation cost.
- We compare our methods with the baseline method 1-bucket-theta and SQL-based query engine in a distributed framework, the results show that our solution is more feasible and effective.

The remainder of the paper is organized as follows. In Section 2, we briefly review the MapReduce computing paradigm and the solution for binary theta-join. Section 3 introduces preliminary concepts and definition. Section 4 gives a presentation of the proposed approach, Section 5 analyzes the cost for binary and multi-way theta-join, then proposes the optimized versions of these two algorithms, experiments evaluation is described in Section 6. We discuss related work in Section 7 and conclude our work in Section 8.

2. Theta-join in MapReduce

In this section, we first present the MapReduce paradigm and how joins will be evaluated on it, then we briefly review the basic idea of 1-bucket-theta method and point out the limitations of it.

2.1. Joins in MapReduce

MapReduce [6] is a popular parallel computation framework, in which data are expressed as (key, value) pairs. It includes two main functions which are Map and Reduce. The map function transforms the input pair (k_1, v_1) to an output of (k_2, v_2). The output will be partitioned by hashing function to different reducers, then each reducer will take the input $(k_2, list(v_2))$ and perform user specified computation to reduce the values of k_2 then output the final results.

Joins in MapReduce includes equi-join and non-equi-join (called theta-join). Equi-join is easy to implement because MapReduce is a key–value based programming model, whose nature is key-equality and can join data sets on the keys with high performance. But as to theta-join, due to inherent limitations, there exists many problems such as load balance, data skew, and memory shortage. 1-bucket-theta is an effective way to solve binary theta-join in MapReduce framework, it uses a randomized algorithm to partition the cross product results to different reducers, which ensures that each reducer can process near same amount of data. Here we give a brief review of 1-bucket-theta.

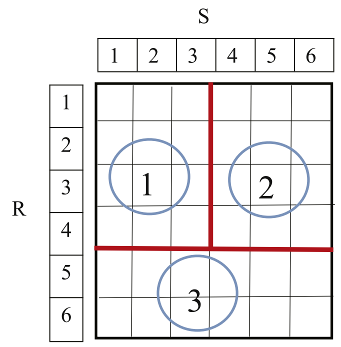

Fig. 1. The idea of 1-bucket-theta.

2.2. 1-bucket-theta

The method of 1-bucket-theta builds up a theta-join model between two data sets S and R with a join matrix M, which can represent and implement any theta-join queries. The data sets that will be joined have $|S|$ and $|R|$ records, and the cross product has $|R| \times |S|$ records. There are k reducers in the parallel framework, the method uses randomized algorithm to partition the cross product to k squares and each square is distributed to one reducer. The side length l of square is as follows:

$$l = \sqrt{|R| \times |S|/k} \tag{1}$$

Fig. 1 shows idea of 1-bucket-theta.

In Fig. 1, three reducers will receive 12 tuples, respectively, and each reducer joins these records and filters them according to the join condition, such as R.B < S.B. 1-bucket-theta requires minimal statistical information, which are the cardinalities of the input tables.

The idea of 1-bucket-theta is from the load balance point of view to achieve minimized maximum reducer input. But the tuples processed in each reducer may contain useless pairs that does not contribute to the final results. If we can filter part of the useless pairs in the input data sets before we perform the algorithm, we can achieve better efficiency.

3. Preliminary

In this section, we formally define the binary theta-join and multi-way theta-join query problems. To improve the multi-way theta-join query efficiency, we divide it into two kinds according to the output attributes.