Available online at www.sciencedirect.com

ScienceDirect

CAAI Transactions on Intelligence Technology 1 (2016) 79–89
http://www.journals.elsevier.com/caai-transactions-on-intelligence-technology/

Original article

RPK-table based efficient algorithm for join-aggregate query on MapReduce

Zhan Li [a], Qi Feng [b], Wei Chen [c], Tengjiao Wang [a,*]

[a] Peking University, China
[b] Natural Science Foundation of China, China
[c] The Chinese University of Hong Kong, Hong Kong, China

Available online 28 May 2016

Abstract

Join-aggregate is an important and widely used operation in database system. However, it is time-consuming to process join-aggregate query in big data environment, especially on MapReduce framework. The main bottlenecks contain two aspects: lots of I/O caused by temporary data and heavy communication overhead between different data nodes during query processing. To overcome such disadvantages, we design a data structure called Reference Primary Key table (RPK-table) which stores the relationship of primary key and foreign key between tables. Based on this structure, we propose an improved algorithm on MapReduce framework for join-aggregate query. Experiments on TPC-H dataset demonstrate that our algorithm outperforms existing methods in terms of communication cost and query response time.

Keywords: Join-aggregate query; MapReduce; Query optimization; RPK-table; Communication cost

1. Introduction

Recently, big data attracts more and more attention. Join-aggregate query which returns aggregate information on the join of several tables is widely used in big data analysis. For instance, many TPC-H[1] queries contain joinaggregate operation for performance evaluation. However, it is time-consuming to run join-aggregate query in existing systems like Hive [1] and Pig [2].

Distributed system has been proven powerful for large-scale datasets analysis. MapReduce [3] is an important parallel computing framework in distributed system. It has been studied and applied widely in academia and industry. However, both the join and aggregate operation processed on MapReduce are time-consuming [4,5]. The main performance bottlenecks come from the following aspects: reading and writing lots of temporary data on disk and heavy communication overhead between different data nodes. The overall response time increases when the data size scales up.

Consider of the following query in TPC-H:

SELECT c.name, COUNT(*) FROM Customer c, Orders o WHERE c.custkey = o.custkey GROUP BY c.name

This SQL query uses TPC-H benchmark schema and computes the number of orders that every customer takes. As the join attribute and aggregate attribute are not the same, we can't execute join operation and aggregate operation in one single MapReduce job. Traditional approach to execute this query need two MapReduce jobs. First job loads two table, exchanges tuples between different data nodes, performs the join operation on the two datasets and then writes the joined result on disk. Then the second job loads the temporary joined result and computes the final aggregate results. The overall computation is time-consuming due to the heavy communication cost and I/O cost. When queries contain multiple joins

* Corresponding author. Internet Research and Engineering Center, School of Electronic and Computer Engineering, Peking University, China. Tel.: +86 755 26035225.
E-mail address: tjwang@pku.edu.cn (T. Wang).
Peer review under responsibility of Chongqing University of Technology.
[1] www.tpc.org/tpch.

http://dx.doi.org/10.1016/j.trit.2016.03.008

on different tables, the query performance will continue to decrease.

To overcome the above drawbacks, we design a data structure called Reference Primary Key table (RPK-table). In RPK-table we store the relationship of primary key and foreign key between tables. This structure is independent of remote data distribution and movement in environment. We also propose a new algorithm for join-aggregate query execution on MapReduce. We only need one single MapReduce job for join-aggregate query. The major performance disadvantages, communication cost and I/O cost, is reduced in our algorithm with the help of RPK-table. Experiments in Section 4 validate the effectiveness of our proposed algorithm in improving the query response time.

The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3, we describe our proposed structure RPK-table and explain our algorithm processed on MapReduce framework. Section 4 shows the performance of our algorithm. Finally we state the conclusion in Section 5.

2. Related work

Both aggregate query and join query have been studied in many recent research works. We analyze some related works which optimize query processing in different situations. These fall into the following broad categories:

- Optimization method for performance improvement in traditional database. Previous works such as [6,7] propose several methods to improve the query performance in traditional database. Their methods like concurrent execution of multiple queries, indexing techniques and physical database design are hard to be implemented and maintained in distributed environment. They also do not consider communication cost during query execution. Join Partition Method (JPM) and Aggregate Partition Method (APM) [8] are two parallel processing methods for join-aggregate query. The JPM method need to process aggregate and join operation separately. The APM method need to broadcast one whole table into all processors. These two methods are sub-optimal on MapReduce framework.
- Modifying or extending MapReduce framework to improve performance. Some researchers [11,12] extend map-reduce model to process join aggregate in one job. Some researchers [13–15] aim to improve the performance by building middleware or cache structure on top of MapReduce framework. These are optimizations specifically on join strategies and they need to modify the core framework of MapReduce. Our proposed algorithm, on the other hand, is more general and can improve the efficiency of join-aggregate operator.
- Pre-computing query results. Some analysis systems store huge amount of data for decision-making. The time interval of data updating in these systems is long enough. Thus some query results can be pre-computed ahead and stored on the disk [16,17]. However, as new data is generated more and more rapidly, data updating becomes more and more frequently. Re-computing query results when data updates becomes complicate and time-consuming. Also this approach cannot assure the efficiency of adhoc query processing for data-intensive application. We focus on improving join-aggregate computation without pre-computation techniques.
- Approximate query processing. Some researchers use approximation method to decrease query response time. These approaches contain sample method [21] and online computation [18–20]. They return an approximate result with a certain error bound or guarantee for each query. The response time will decrease a lot but they do not provide an exact result. Furthermore, its requirement for random data retrieval makes it difficult to be performed on distributed system. The goal of our work is to design an algorithm which improves the performance of join-aggregate queries without sacrificing accuracy.

3. Our approach

In distributed system the whole dataset is partitioned and located in different nodes depending on the availability of storage resources. During join-aggregate query execution, each worker accesses data splits from different nodes and then performs join operation on primary and foreign key attributes.

After that the required attribute values are filtered and the aggregate results are computed. When the data size is huge, the execution may become complex. Lots of temporal data need to be written and read in disk and the communication overhead between different nodes may increase heavily. This will result in increasing query processing time and decreasing system performance. To achieve better performance, we first design a new data structure which minimizes the storage cost. Then we propose an optimization join-aggregate query processing algorithm on MapReduce with the help of our structure.

3.1. Data structure RPK-table

In this sub-section, we describe the Reference Primary Key table. Then we explain the reason for designing it and the way to store it.

First we give our definitions here.

Definition 1. Table which contains the primary key in table relationship between primary and foreign key is defined as **target table**.

Definition 2. Table which contains the foreign key in table relationship between primary and foreign key is defined as reference table.

Since join-aggregate queries often perform equi-join on tables in their primary and foreign keys, we design a structure called RPK-table. In RPK-table, we store the