Efficient Processing of Multi-way Joins Using MapReduce

Linlin Ding, Siping Liu, Yu Liu, Aili Liu, and Baoyan Song*

School of Information, Liaoning University, Shenyang Liaoning, P.R. China
{dinglinlin,bysong}@lnu.edu.cn,
{spliu,dearboll23,aili07liu}@gmail.com

Abstract. Multi-way join is critical for many big data applications such as data mining and knowledge discovery. Even though lots of research have been devoted to processing multi-way joins using MapReduce, there are still several problems in general to be further improved, such as transferring numerous unpromising intermediate data and lacking of better coordination mechanisms. This work proposes an efficient multi-way joins processing model using MapReduce, named Sharing-Coordination-MapReduce (SC-MapReduce), which has the functions of sharing and coordination. Our SC-MapReduce model can filter the unpromising intermediate data largely by using the sharing mechanism and optimize the multiple tasks coordination of multi-way joins. Extensive experiments show that the proposed model is efficient, robust and scalable.

Keywords: MapReduce, multi-way joins, sharing and coordination.

1 Introduction

Multi-way join is an important and frequently used operation for many big data applications including data mining and knowledge discovery. Since join processing is expensive, especially for large data sets, multi-way join is a costly operation. When processing multi-way joins of big data, a natural solution to ensure the reasonable response time is parallel processing. As a parallel programming model, MapReduce [1] becomes the popular big data programming model for its simplicity, flexibility, fault-tolerance and scalability.

MapReduce is designed to process a single input data set, so multi-way join is not directly supported by MapReduce framework. Although lots of research have been devoted to processing join using MapReduce[2–6], the existing works still have several problems to be further researched. For instance, there are numerous intermediate data to be transferred from Map phase to Reduce phase. When the final output is much smaller than the original input, the numerous unpromising intermediate data would waste the bandwith and I/O. In addition, when processing multi-way joins using several passes MapReduce, the next MapReduce computation cannot start until the previous computation is over.

* Corresponding author.

H. Wang et al. (Eds.): ICYCSEE 2015, CCIS 503, pp. 73–80, 2015.

In this work, we investigate the problems of multi-way join query processing in MapReduce and analyze the performance and bottlenecks of the existing solutions. An efficient multi-way join query processing model using MapReduce, named Sharing-Coordination-MapReduce (SC-MapReduce) is designed. By using the mechanisms of sharing and coordination, SC-MapReduce can enhance the parallelism and reduce the network cost. In SC-MapReduce, first, a sharing mechanism is proposed to filter the unpromising intermediate data and reduce the network and I/O cost. Then, We design a multi-tasks coordination mechanism for processing multi-way joins. Using this coordination mechanism, the next task do not need to wait for the completion of the previous to start its processing, which will save the waiting time and enhance the parallelism..

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents our SC-MapReduce model. The sharing and coordination mechanisms of SC-MapReduce model are given in Section 4. Section 5 reports the experimental results. Section 6 concludes the paper.

2 Related Work

2.1 Problem Statement

Definition 1. *(**Two-way Join**) Given two data sets $R_1(A_1, S_1)$ and $R_2(A_1, S_2)$ on a common attribute A_1. S_1 and S_2 can be the single attribute or the array of multiple attributes, as shown $R_1(A_1, S_1) \bowtie R_2(A_1, S_2) = (A_1, S_1, S_2)$.*

Definition 2. *(**Multi-way Same Attribute Join**) Given n (n>2) data sets $R_i(A_1, S_i)$(i=1,2,...,n) on a common attribute A_1. S_i can be the single attribute or the array of multiple attributes, as shown $R_1(A_1, S_1) \bowtie R_2(A_1, S_2) \bowtie, ..., \bowtie R_n(A_1, S_n) = (A_1, S_1, S_2, ..., S_n)$*

Definition 3. *(**Multi-way Different Attributes Join**) Given n (n>2) data sets $R_i(A_{i-1}, S_i, A_{i+1})$(i=1,2,...,n) on common attributes A_i. S_i can be the single attribute or the array of multiple attributes, as shown $R_1(A_0, S_1, A_1) \bowtie R_2(A_1, S_2, A_2) \bowtie, ..., \bowtie R_n(A_{n-1}, S_n, A_n) = (A_0, ..., A_n, S_1, ..., S_n)$*

2.2 Join of MapReduce

There are many existing works of processing joins using MapReduce [4–12]. MapReduce online [5] proposes a pipelined job interaction mechanism to avoid intermediate data materialization. Map-Join-Reduce [9] improves MapReduce runtime to process complex data analysis tasks on large clusters. Paper [7] studies and optimizes multi-way equi-join in MapReduce by selecting a query plan with the lowest input replication cost. Paper [6] is the first one to study all theta-joins and explores optimality properties for them in MapReduce-based systems. Paper [4] studies the problem of processing multi-way theta-joins using MapReduce from a cost-effective perspective. Vernica et al. [8] present an in-depth study of a special type of similarity join in MapReduce.

2.3 Multi-way Joins of MapReduce

The multi-way join contains two types, multi-way same attribute join and multi-way different attributes join. The two-way join is the special case of multi-way join. Processing multi-way join using MapReduce can be implemented by one pass MapReduce computation and multiple MapReduce computations in sequential. Paper [2] analyzes the efficiency of multi-way join from the view of network communication cost. Foto N.Afrati [3] discusses how to identify one pass MapReduce or multiple MapReduce computations.

For one pass MapReduce processing multi-way same attribute or two-way join, Map phase is mainly responsible for labeling the join tuples and identifying the relation of the tuples. The real join operations are implemented in Reduce phase. However, in Shuffle phase, it can transfer the corresponding unpromising intermediate data many times which would increase the I/O and communication cost. For the multiple MapReduce computations processing multi-way different attributes join, each MapReduce computation completes two-way join until all the data sets are processed in sequential. The next MapReduce computation waits for the completion of the previous MapReduce computation, and then starts its computation.

3 SC-MapReduce Framework

3.1 SC-MapReduce Overview

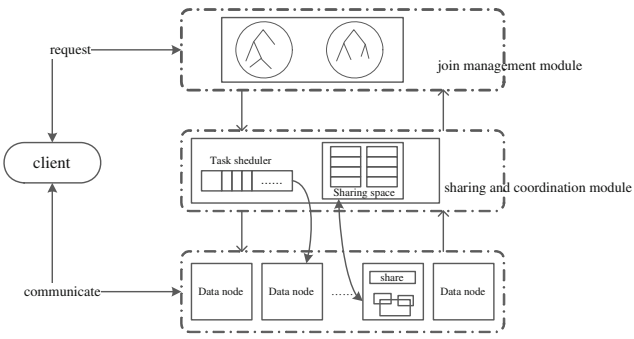

Fig. 1. Architecture of SC-MapReduce Framework

In order to enhance the performance of multi-way joins, in this paper, we propose SC-MapReduce, a Sharing-Coordination-MapReduce framework. Figure 1 shows the whole architecture of SC-MapReduce which briefly inherits the original MapReduce framework, containing the Master node and the Slave nodes. The Master node is mainly responsible for managing the whole system, scheduling and distributing the computation jobs. Furthermore, the join management

module and the sharing and coordination module are added in the Master node. The sharing and coordination module contains the sharing mechanism and the coordination mechanism. The real computation is done by the Slave nodes.

Join Management Module. The join management module is in charge of receiving the connecting request from clients and designing the join order of the multi-way join, then generates the execution queue of multi-way join.

Sharing Mechanism. The sharing mechanism mainly processes the communications in one computation course. Each Slave node has a share space to save the sharing information to filter the unpromising intermediate data.

Coordination Mechanism. The coordination mechanism designs for the parallelism of multiple MapReduce computations of multi-way join. When processing multi-way joins, the SC-MapReduce framework can first complete processing a part of join data and wait for the completion of the remaining part from the previous computations, so as to enhance the parallelism of SC-MapReduce.

As shown in Figure 1, when a client submits a multi-way join to the Master node, it first designs the join order of the multi-way join by the join management module. Then, the sharing and coordination module of Master node can obtain the sharing information of the multi-way join and design the implementation of multiple MapReduce computations. Then, the Map and Reduce tasks can use the sharing information to filter the unpromising intermediate data. The multiple computations can also start early to process parts of the inputs. The details of sharing and coordination mechanisms are illustrated in Section 4.

3.2 Availability and Scalability

Availability. The SC-MapReduce only adds some new functions in MapReduce, so it only analyzes whether the new modules influence the whole availability of SC-MapReduce. For the fault-tolerance of sharing information, the sharing information of each Map task is sent to the JobTracker by heartbeat mechanism, so it is simultaneous to the implementation of Map tasks. The fault-tolerance of sharing information is the same to the Map tasks in MapReduce. For the fault-tolerance of coordination mechanism, if the previous MapReduce computation fails, the next computation will wait endlessly. We set a threshold t, when the waiting time exceeds the threshold t, then the next computation will stop and the failed one will restart.

Scalability. The SC-MapReduce follows the scalability of the MapReduce. First, the new sharing and coordination mechanisms do not modify HDFS, so the users can add new Slave nodes naturally. Second, although the new modules are added to the Master node, the sharing mechanism can be realized by only following the implementations of Map tasks. The coordination mechanism only occupies few resource, so the performance of the Master node is not influenced.

4 Sharing and Coordination Module

The sharing and coordination module mainly contains the sharing mechanism and the coordination mechanism. The sharing mechanism is mainly charge of

identifying the sharing information to filter the unpromising intermediate data during one MapReduce computation course. The coordination mechanism can generate the parallel implementation of multiple MapReduce computations.

4.1 Sharing Mechanism

In the traditional MapReduce, the implementations of Mappers and Reducers are independent without communication among Mappers, neither among Reducers. Each Mapper has no processing information of the other ones, the same as the Reducers. When the amount of multi-way joins final results is much smaller than the original data, processing the unpromising data would waste the running time and increase the network and I/O cost. Therefore, if the Master node can share some useful sharing information among the Map and Reduce tasks, the unpromising data can be filtered by the sharing information so as to enhance the performance of multi-way joins.

The main course of sharing mechanism is as follows. First, for the input join data sets, the Map tasks process the first few tables according to the join queue obtained by the join management module. Second, each Map task gains and saves its local join attributes, and then sends them to the Master node. Then, after receiving the local sharing information of the Map tasks, the Master node generates the global sharing information and sends it to the other tables. After the Map tasks receiving the global sharing information, they can use it to filter the unpromising intermediate data when processing the other join tables.

However, MapReduce is suit for the single file input. When processing multiple files, the JobTracker will split all the files, then generates the default job queue of Map tasks and distributes the Map tasks to the free nodes. Therefore, we modify some functions of the original MapReduce. SC-MapReduce modifies the scheduling and constructing mechanisms of Map tasks so as to scan the tables according to the join queue. After scanning one join table, the sharing information can be sent, received and distributed among Map tasks and the Master node. The Master node and the Slave nodes all have sharing space to send and receive the sharing information. The JobTracker is in charge of starting and stopping the scheduling queue according to the join queue order.

4.2 Coordination Module

As we know, there are multiple MapReduce computations to complete the multi-way join. The previous MapReduce computation may be the input of the next MapReduce computation, so each MapReduce computation should wait for the completion of the previous computation. However, for the next computation, there are parts of input data which can be processed first without affecting the following data. For example, multi-way join $R_1 \bowtie R_2 \bowtie R_3$, the first MapReduce computation completes the join of $R_1 \bowtie R_2$. The next job realizes join of R_3 and the output of R_1 and R_2. We can beforehand process the Map tasks of R_3 and wait for the completion of $R_1 \bowtie R_2$, and then gain the final results. Therefore, the parallelism of system can be enhanced by optimal coordination mechanisms.

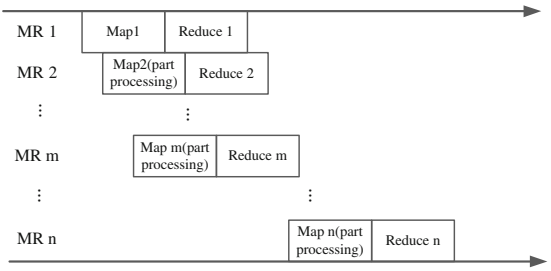

Fig. 2. Coordination Mechanism of SC-MapReduce

The coordination mechanism can coordinate the MapReduce computations. For the multi-way join with n MapReduce jobs, the concurrency is m when there are m MapReduce jobs running at the same time as shown in Figure 2. The processing course is as follows.

(1) Start m MapReduce computations in order and construct task queue for the part of the input data.
(2) For the i MapReduce computation, if its input is completed, then it implements normally. If it needs to wait for the output of the previous MapReduce, it designs the dormant state of task queue after completing the following Map tasks, and then communicates with the Master node.
(3) The Master node restarts the i MapReduce computation when the input of i MapReduce completes according to the implementation of the tasks, and then refreshes the task queue and implements the tasks.
(4) After some MapReduce computations complete, the concurrency is smaller than m. If there are tasks that have not start, the SC-MapReduce adds them into the task queue and starts the new tasks following step 1 to step 3.

5 Experimental Evaluation

5.1 Experimental Setup

The experimental setup is as follows. The experimental setup is a Hadoop cluster running on 9 nodes in a high speed Gigabit network, with one node as the Master node and the Coordinator node, the others as the Slave nodes. Each node has an Intel Quad Core 2.66GHZ CPU, 4GB memory and CentOS Linux 5.6. We use Hadoop 0.20.2 and compile the source codes under JDK 1.6.

We take three tables join as an example to evaluate the SC-MapReduce by comparing with MapReduce in two distributions, uniform and *zipf*. The data in our experiments are synthetic data of network logs. Each record has 100B-10000B and the join attribute is 1B-10B. The log files come from different parts of the network. The percentages of join attributes are 5%, 10%, 65% and 75%, with 10% as the default value. The size of the three tables join is 1G, 5G, 10G and 20G, with 10G as the default value.

5.2 Experimental Results

Figure 3(a) shows the performance of multi-way join running time with changing the join attributes percentage in uniform distribution. We can see that the performance of SC-MapReduce is better than MapReduce. The running time of MapReduce is relative stable for no filtering mechanism and uniform distribution. In SC-MapReduce, the join attributes of 5% and 10% are few, so the global sharing information is small and can filter more unpromising intermediate data. Therefore, the running time is relative short.

Figure 3(b) shows the performance of multi-way join running time with changing the join attributes percentage in *zipf* distribution. In *zipf* distribution, there will be one or few Reduce tasks to process much intermediate data, so the performance in *zipf* distribution is not as well as the uniform distribution. However, the sharing information of SC-MapReduce can also filter numerous unpromising intermediate data.

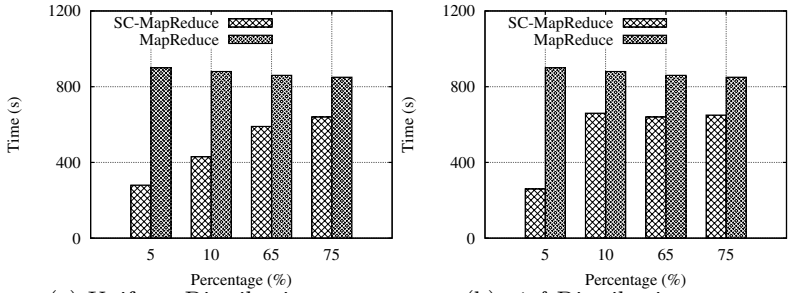

Fig. 3. Running Time with Changing Percentage

Figure 4(a) shows the performance of multi-way join running time with changing the data size in uniform distribution. With the increasing of data size, the running time of MapReduce and SC-MapReduce both increase, but SC-MapReduce is optimal to MapReduce. The global sharing information of SC-MapReduce can filter large scale of join data which are not the final results.

Figure 4(b) shows the performance of running time with changing the data size in *zipf* distribution. The running time increases with the increasing of the data size in SC-MapReduce and MapReduce. The performance of SC-MapReduce is also better than MapReduce. In *zipf*, the sharing information of SC-MapReduce can be used to filter the unpromising intermediate data too.

6 Conclusions

In this paper, we research the problem of multi-way join query processing based on MapReduce framework. First, we propose an efficient multi-way join query processing model with the functions of sharing and coordination, SC-MapReduce.

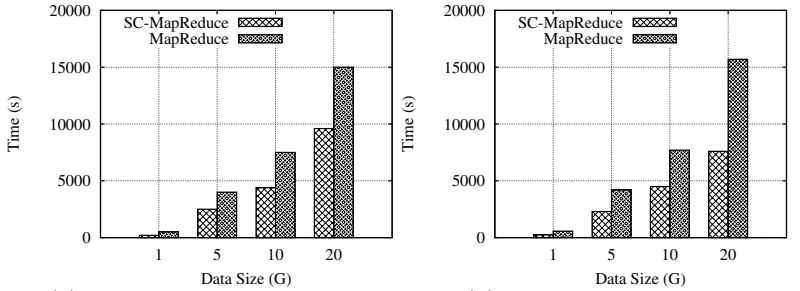

Fig. 4. Running Time with Changing Data Size

Then, the sharing and coordination module is illustrated in detail. Finally, extensive experiments show the efficiency, scalability of our proposed model.

Acknowledgement. This work was supported by the National Natural Science Foundation of China under Grant No.60873068,61472169; the Program for Excellent Talents in Liaoning Province under Grant No.LR201017.

References

1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM (CACM) 51(1), 107–113 (2008)
2. Okcan, A., Riedewald, M.: Processing theta-joins using MapReduce. In: SIGMOD, pp. 949–960 (2011)
3. Afrati, F.N., Ullman, J.D.: Optimizing Multiway Joins in a Map-Reduce Environment. IEEE Trans. Knowl. Data Eng (TKDE) 23(9), 1282–1298 (2011)
4. Zhang, X., Chen, L., Wang, M.: Efficient Multi-way Theta-Join Processing Using MapReduce. PVLDB 5(11), 1184–1195 (2012)
5. Pansare, N., Borkar, V.R., Jermaine, C., Condie, T.: Online Aggregation for Large MapReduce Jobs. PVLDB 4(11), 1135–1145 (2011)
6. Okcan, A., Riedewald, M.: Processing theta-joins using MapReduce. In: SIGMOD, pp. 949–960 (2011)
7. Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: EDBT, pp. 99–110 (2010)
8. Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: SIGMOD, pp. 495–506 (2010)
9. Jiang, D., Tung, A.K.H., Chen, G.: MAP-JOIN-REDUCE: Toward Scalable and Efficient Data Analysis on Large Clusters. IEEE Trans. Knowl. Data Eng (TKDE) 23(9), 1299–1311 (2011)
10. Fries, S., Boden, B., et al.: PHiDJ: Parallel similarity self-join for high-dimensional vector data with MapReduce. In: ICDE, pp. 796–807 (2014)
11. Ma, Y., Meng, X.: Set similarity join on massive probabilistic data using MapReduce. Distributed and Parallel Databases (DPD) 32(3), 447–464 (2014)
12. Lee, T., Bae, H.-C., et al.: Join processing with threshold-based filtering in MapReduce. The Journal of Supercomputing (TJS) 69(2), 793–813 (2014)