Throughput Optimization using Multi-Join Query Plans

Siddhi joshi, Jignesh Vania

Abstract **- This paper introduces an efficient framework for producing high and early result throughput in multi-join query plans. While most previous research focuses on optimizing for cases involving a single join operator, this work takes a radical step by addressing query plans with multiple join operators. The proposed framework consists of two main methods, a flush algorithm and operator state manager. The framework assumes a symmetric hash join, a common method for producing early results, when processing incoming data. In this way, our methods can be applied to a group of previous join operators (optimized for single-join queries) when taking part in multi-join query plans. Specifically, our framework can be applied by 1) employing a new flushing policy to write in-memory data to disk, once memory allotment is exhausted, in a way that helps increase the probability of producing early result throughput in multi-join queries, and 2) employing a state manager that adaptively switches operators in the plan between joining in-memory data and disk resident data in order to positively affect the early result throughput. Extensive experimental results show that the proposed methods outperform the state-of-the-art join operators optimized for both single and multi-join query plans.**

Index Terms **– Hash-merge join, RPJ, PMJ.**

I. INTRODUCTION

Design of traditional join algorithm is based on assumption that all input data is available before join operation. Those algorithms are very useful to produce overall query result, but they are not suitable for applications that require result immediately. Some new non-blocking join algorithm suitable for such application like; i) when input data arrives in online environment ii) user is interested in first few result, rather than waiting for complete result.

In this paper we propose a new efficient non-blocking join algorithm for producing immediate result in multi-join query plan (extension in Adaptive Global Flush Algorithm). This non-blocking join is differs from previous technique; it explores a new approach to flush less memory when data arrival rate is slow in order to maximize throughput in multi-join query. This non-blocking join selects the N amount of data to be flushed to disk based on expected contribution of data in previous result with least contribution.

This non-blocking Join distinguishes itself from all other non-blocking join algorithm ([3], [4], [7], [8] and [9]). in two new aspects: i) This Join employs new memory flushing technique, which flush N amount of data when memory is full (data arrival rate is normal) or flush N/2 amount of data when data arrival rate is slow. This Join maximize overall query throughput for a multi-join query plan predicting the throughput contribution of in-memory data. ii) This Join employs a new operator state manager that has ability to switch any operator between joining in-memory data and on-disk data by considering most beneficial state in order to query throughput. Such operator state manager does not exist in previous techniques ([3], [4] and [7]).

II. LITERATURE SURVEY

Various non-blocking join algorithm ([3], [4], [7], [8]) uses the symmetric hash join which is the most widely used non-blocking join algorithm for producing immediate result. These methods based on; flush certain percentage of hash buckets to disk either individually [3], [8] or in groups [4], [9] when memory become full. All of these algorithms used symmetric hash join in order to achieve non-blocking behavior. The main difference between them in the flushing algorithm used to free memory.

These techniques focus only on the case of single join operator while our proposed Join extends with multi-join query plan. The closest work to this Join is the state spilling method [7]; the only work related with multi-join query plan. The main idea of state spilling is to score each hash partition based on contribution in past query result and to flush hash partition group with the lowest score.

Hash-merge join (HMJ) algorithm is a non-blocking join algorithm that deals with data items from remote sources via unpredictable, slow, bursty network traffic. The HMJ algorithm designed with two goals: 1) Minimize the time to produce the first few results, and (2) produce join results even if the

two sources of the join operator occasionally get blocked.

The HMJ algorithm has two phases: The hashing phase and the merging phase. The hashing phase employs an in-memory hash-based join algorithm that produces join results as quickly as data arrives. The merging phase is responsible for producing join results if the two sources are blocked. Both phases of the HMJ algorithm are connected via a flushing policy that flushes in-memory parts into disk storage once the memory is exhausted. Experimental results show that HMJ combines the advantages of two state-of-the-art non-blocking join algorithms (XJoin and Progressive Merge Join) while avoiding their shortcomings.

Hash Merge Join (HMJ) [4] main idea is to minimize time to produce first few result and produce join result if two sources of operator already blocked. HMJ consider only single join operator.

Rate based Progressive Join (RPJ) maximizes the output rate by optimizing its execution according to the characteristics of the join relations. RPJ utilizes a novel flushing algorithm which is optimal among all possible alternatives (based on the same statistics about data distributions, arrival patterns, etc.). But RPJ is also considered as single join operator.

This non-blocking join is concluding the benefits of all three ([3], [4], [7]) and propose its new approach with avoiding drawbacks of these techniques. First, this non-blocking Join employ a new flushing technique that take both input and output characteristics at each join operator which helps to predict the contribution of data in hash buckets. Second, this non-blocking join employs an operator state manager that switches operators in between in-memory, on-disk resident data or in block state, on the basis of which state is most beneficial to produce massive result for efficient throughput.

III. CONCLUSION

This paper introduces an algorithm to produce massive and immediate result in multi-join query plan. Algorithm calculate contribution of data in overall result and flush least useful data to disk to make room for new incoming data in online environment. Operator state manager switches operator in different states to maximize the overall immediate throughput. Experimental result shows that our proposed method is more efficient and scalable in compare to previous non-locking join algorithms when producing immediate and massive result. Future work can be done on improving the scoring of partition group; which fulfil the aim to flush least useful partition. Also estimate optimal data arrival rate which will help in order to produce efficient and massive result in multi-join query.

REFERENCES

[1] V. E Ioannidis, "Query Optimization", ACM Computing Surveys (CSUR), pages 121-123, March 1996.

[2] J. P Dittrich, B. Seeger, D. S Taylor, P. Widmayer, "Progressive Merge Join: A Generic and Non-Blocking Sort-Based Join Algorithm", In Proceedings of the International Conference on Very Large Data Bases, VLDB, pages 299–310, Hong Kong, Aug 2002.

[3] Y. Tao, M. Lung, D. Papadias, M. Hadjieleftheriou, N. Mamoulis, "RPJ: Producing Fast Join Results on Streams through Rate-based Optimization", In Proceedings of the ACM SIGMOD international conference on Management of data, pages 371-392, 2005.

[4] M. F Mokbel, M. Lu, W. G Aref, "Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results", In Proceedings 20th International conference on IEEE, pages 251-262, 30 March-2 April 2004.

[5] X. Lin, "Query Optimization Strategies and Implementation Based on Distributed Database", Computer Science and Information Technology, ICCSIT, 2nd IEEE International Conference, Page(s) 480-484, 8-11 Aug 2009.

[6] P. Bansal, Dr. R. Rathi, Mr. V. Jain, "DAP Join: Produce Massive and Immediate Result in Multi Join Query using Flushing", In Proceedings of IEEE Conference on Information and Communication Technologies, pages 356-361,11-12 April 2013.

[7] B. Liu, Y. Zhu, and E. A. Rundensteiner, "Run-time operator state spilling for memory intensive long-running queries," in *SIGMOD*, 2006.

[8] T. Urhan and M. J. Franklin, "XJoin: A Reactively-Scheduled Pipelined Join Operator," IEEE Data Engineering Bulletin, vol. 23, no. 2, Page(s) 27–33, 2000.

[9] levandoski et al.: on producing high and early result throughput in multi join query plans, IEEE transaction of knowledge and data engineering, vol. 23, no. 12, december 2011.