Asian Journal of Information Technology 13 (2): 53-57, 2014
ISSN: 1682-3915

A Comparative Study of Online Joining Approaches Based on Memory Flushing Patterns

Raksha Chauhan and Pratik A. Patel
Department of Computer Science,
Parul Institute of Technology, Vadodara (Gujarat), India

Abstract: Traditional query processors generate full, accurate query results, either in batch or in pipelined fashion. Researchers reviewed that this strict model is too rigid for queries over distributed data sources. For executing such query, space is required in main memory to accommodate the data for query processing. When main memory gets full then some data must be moved to disk for further processing as part of flushing. An optimal strategy should be used to remove the victim data from memory to the disk. Victim data must be selected on the basis of its contribution in future query processing. In this study various join algorithms and flushing techniques used in online environment is reviewed.

Key words: Processors, data, main memory, optimal strategy, flushing

INTRODUCTION

An important line of research in adaptive query processing has been toward developing join algorithms that can produce tuples "online" from streaming, partially available input relations or while waiting for one or more inputs (Dittrich *et al*., 2002; Levandoski *et al*., 2011; Ives *et al*., 1999; Wilschut and Apers, 1993). Such non-blocking join behaviour canimprove pipelining by smoothing or "masking" varying data arrival rates and can generate join results with high rates thus, improving performance in a variety of query processing scenarios in data-integration, online aggregation and approximate query answering system. Traditional join algorithms are designed with the implicit assumption that all input data are available earlier and then to process. To process these data streams there are some issues: First issue is that there is requirement of non-blocking join algorithms to process these streams to get first result as early as possible in spite of traditional blocking join algorithms which requires all the input data before hand for processing. Second issue is there is requirement of optimal flushing policy which flushes in memory data to disk when main memory becomes full and also makes room for new incoming data to get processed. Various algorithms are designed to resolve these issues. There are three basic join algorithms: hash based join, sort merge based join, nested loop based join algorithm. Here, researchers have carried out the study for X-Join, Hash Merge Join (HMJ), Rate based Progressive Join (RPJ) and Adaptive Global Flush algorithm (AGF) which are used in internet environment.

EXISTING RESEARCH

Researchers have studied some of the related research for the online joining environment on different factors. Existing research on adaptive join techniques can be classified in two groups: hash based (Dittrich *et al*., 2002; Mokbel *et al*., 2004) and sort based (Urhan and Franklin, 2000).

X-Join: Examples of hash-based algorithms include DPHJ and X-Join, the first of a new generation of adaptive non-blocking join algorithms to be proposed. X-Join was inspired by Symmetric Hash Join (SHJ) (Mokbel *et al*., 2004) which represented the first step toward avoiding the blocking behavior of the traditional hash-based algorithms. It is a non-blocking join operator, called X-Join (Mokbel *et al*., 2004) which has a small memory footprint, allowing many such operators to be active in parallel. X-Join is optimized to produce initial results quickly and can hide intermittent delays in data arrival by reactively scheduling background processing. Researchers show that X-Join is an effective solution for providing fast query responses to user even in the presence of slow and bursty remote sources.

X-Join (Urhan and Franklin, 1999) has 2 fundamental principles. It produce first result as soon as possible. It also produce results when one of two input sources are blocked, i.e., not producing inputs. X-Join is based on symmetric hash join but with some variations. It produces hash table for both the sources so whenever a tuple arrives at one of the two sources it is inserted in its hash table and then its join attribute is used to probe the hash

Corresponding Author: Raksha Chauhan, Department of Computer Science, Parul Institute of Technology, Vadodara (Gujarat), India

table of another source. Variation of this with symmetric hash join is that it moves part of hash table from memory to disk when ever required. As with traditional hash based algorithms, X-Join organizes each input relation in an equal number of memory and disk partitions or buckets, based on a hash function applied on the join attribute. Each partition has two portions one reside in memory and other on disk.

X-Join operates in 3 phase

First phase: This is tuple arriving phase which runs for as long as either of the data sources sends tuples, the algorithm joins the tuples from the memory partitions. Each incoming tuple is stored in its corresponding bucket or partition and is joined with the matching tuples from the opposite relation. When memory gets exhausted, the partition with the greatest number of tuples is flushed to disk. The tuples belonging to the bucket or partition with the same designation in the opposite relation remain on disk. When both data sources are blocked, the first phase pauses and the second reactivephase begins.

Second phase: Reactive phase, activated whenever the first phase terminates. It first chooses a disk resident partition from one source according to some criterion and It then uses the tuples from the disk-resident portion of that partition to probe the memory-resident portion of the corresponding partition of the other source (Urhan and Franklin, 1999). Output is produced whenever a match is found. Final, output is produce after checking duplication of data (according to some duplication policy), i.e., whether, this data is send before or not. Once, this partition has processed then there is need to check whether first stage is started back if so then second phase is halted and first phase is started again otherwise another portion from disk is selected for continuing with the second phase.

Third phase: Clean up phase, starts when all tuples from both data sources have completely arrived. It joins the matching tuples that were missed during the earlier two phases.

In X-Join the reactive stage can run multiple times for the same partition. Thus, a duplicate avoidance strategy is necessary in order to detect already joined tuple pairs during subsequent executions. For this purpose two time stamps Arrival Time Stamp (ATS) and Departure Time Stamp (DTS) are attached to every tuple. ATS shows the time when the tuple is first received from its source and DTS is introduced when the tuple is sent to disk from memory to make space for new arrivals. Tuple with overlapping ATS and DTS are already produced result with first phase so these are not to be considered in second and third phases. Another timestamps are also used to collect the time the disk residing partition is used by second phase to produce output.

Progressive Merge Join (PMJ): Progressive Merge Join (PMJ) (Dittrich *et al*., 2002), one of the early adaptive algorithms also supports range conditions but its blocking behaviour makes it a poor solution given the requirements of current data integration scenarios which is online environment. It partitions the memory into two partitions. PMJ has 2 operating phase.

Sorting phase: Whenever tuples arrive, they are inserted in their memory partition and when the memory gets full, the partitions are sorted on the join attribute and are joined using any memory join algorithm. Thus, output tuples are obtained each time when the memory gets exhausted. Next, the partition pair (i.e., the bucket pairs that were simultaneously flushed each time when the memory was full) is copied on disk. After the data from both sources completely arrives, the merging phase begins.

Merging phase: The algorithm defines a parameter F, the maximal fan-in which represents the maximum number of disk partitions that can be merged in a single "turn". F/2 groups of sorted partition pairs are merged in the same fashion as in sort merge. In order to avoid duplicates in the merging phase, a tuple joins with the matching tuples of the opposite relation only if they belong to a different partition pair.

Hash merge join: HMJ (Mokbel *et al*., 2004) is a hybrid query processing algorithm combining ideas from X-Join and progressive merge join. This introduces the Hash-Merge Join algorithm (HMJ) for the join operator occasionally gets 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. HMJ has two main goals: it minimize the time to produce first few results. Produce join results if two sources of operator already blocked.

Hashing phase: Produce join result quickly using in memory hash based join and also performs in a same way as X-Join perform except that when memory gets full, a

flushing policy decides which pair of corresponding buckets from two relation flushed on disk. Once tuple r is received from source R or from S, if memory is already full (or exhausted) then HMJ selects partitioned to flushed to disk or else if there is enough memory to accommodate tuple r than compute that value h(r) and join all tuples (S h(r)). (R h(r)). Researchers are taking two sources R and S with N buckets with different sizes. If source R blocked (one of the source) is blocked than also hashing phase can produce results. In memory hash table of R join with S which can still be received. HMJ transfer control to merging phase when all data is processed or both sources are blocked. Flushing is done by selecting two partitions with same hash value, one from each source and flushed to disk.

Merging phase: Deals with buckets which are previously flushed to disk in hashing phase due to exhausting memory. Merging phase is similar to Progressive Merge Join (PMJ). Difference is that HMJ transfer control between hashing and merging phase many times while in PMJ merging starts after data is finished and is processed in memory. Merging phase applies refinement of traditional Sort-Merge Join (SMJ) algorithm for each individual bucket. Two refinements are:

- Avoiding the steps of blocking behavior of separating the sorting and merging conclude that join result are produced during the merging
- Duplicates are avoided during the merging phase by ensuring that the matching tuples belonging to the same sorted pair do not join

Double index nested loop join: DINER (Bornea *et al*., 2010) is adaptive and unblocking join technique which supports range join condition. Goal of DINER is to produce join result correctly and avoids operations that may expose the correctness of the output because of memory overflow. DINER increases the number of join tuples as being highly adaptive to the (often changing) value distributions of the relations as well as to potential network delays. DINER has two index characteristics for its framework: small memory footprint. The ability to have sorted access based on the index keys. DINER has three working phase: arriving phase, reactive phase and clean up phase.

Arriving phase: In this phase tuples arrives from one or both relation and processed in memory. When new tuple available match with tuples of opposite relation which is reside in memory and generate result tuples as soon as input data are available. When the memory becomes full then some data from memory must be flushed to disk. Victim data is chosen on the basis of range of the values of the join attribute. For example, if the values of the join attribute in the incoming tuples from relation RA tend to be increasing andthese new tuples generate a lot of joins with relation RB then this is an indication that we should try to flush the tuples from relation RB that have the smallest values on the join attribute (of course, if their values are also smaller than those of the tuples of RA) and vice versa (Bornea *et al*., 2010). The key idea of flushing policy is tocreate and adaptively maintain three non-overlapping valueregions that partition the join attribute domain, measure the "join benefit" of each region at every flushing decision point and flush tuples from the region that doesn't producemany join results in a way that permits easy maintenance of the three-way partition of the values. When tuples are flushed to disk they areorganized into sorted blocks using an efficient index structure, maintained separately for each relation (thus, the part "Double Index" in DINER)

Reactive phase: This phase invoked whenever both relations are blocked. In this phase join perform between those tuples which are earlierly flushed from both relations to disk partition. Algorithm switches back to the Arriving phase when enough tuples arrives in memory.

Clean up phase: The clean-up phase starts when both relations received entirely.

ADAPTIVE GLOBAL FLUSH SCHEME

It is a new technique that aims to flush hash partition groups which are least contributive to overall result from in-memory to disk. In online or streaming environment, data are flushed to disk to make room for new incoming data, if memory is exhausted. Adaptive Global Flush (AGF) (Viglas *et al*., 2003) policy flushes partition groups simultaneously from both hash partition. To flush hash table data that contribute least to overall result is the key success to any flushing policy. The decision making by AGF depends on three considerations.

Data arrival pattern: Due to unreliable network connections, arrival rates and delay of input pattern can change overall through put. At a time of higher data arrival in a partition group that group will contribute more to overall result or at a time of lower data arrival implies less contribution.

Data properties: This directly affects the result production. If data is sorted in a partition group then that group may contribute a large amount of data in overall result in a time period T which depends on join attribute.

Global contribution: Refers the number of overall results which has been produced by each partition group in multi join query plan. This contribution is changing continuously due to the selection of join operators. Selection may change over time due to new data arriving and flushing.

The main idea of the adaptive global flush algorithm is to collect and observe statistics during a query runtime to help the algorithm choose the least useful partition groups to flush to disk (Levandoski *et al*., 2011). Which statistics are being collected can be shown from Table 1 in the document.

Here, researchers are taking single parameter N as input, representing the amount of memory to be flushed. Main reason behind this is to evaluate each partition group at every operator and score every group based on global contribution. After that algorithm flushes lowest score partition to disk until N amount of memory gets free. AGF algorithm uses four steps as:

Step 1 (Input estimation): It uses the input statistics in order to estimate the number of tuples arrives at each operator until then ext memory flushes. Goal is to predict where i.e., at which partition will data arrive and also provide an idea of how productive each group will be.

Step 2 (Local output estimation): It uses the size statistics andinput estimation (step 1) in order to estimate the contribution of each partition to the local output. To calculate local output, it uses. Combination of three calculations:

- Expected output from newly arrival at first partition with already residing in second partition in same group
- Expected output from newly arrival at second partition with already residing in first partition in same group
- Expected output from newly arrival at first partition with newly arrival at second partition in same group

Table 1: Statistics taken

Class	Statistics	Definition
Size	$prtSize_{Sj}$	Size of hash partition j for input S
	$grpSize_j$	Size of partition group j
	$tupSize_s$	Tuple size for input S
Input	$input_{tot}$	Total input count
	$unique_j$	No. unique values in part group j
	$prtInput_{sj}$	Input count at partition j of input S
Output	$obsLoc_j$	Local partition of group j
	$obsGlo_j$	Gobal partition of group j

Step 3 (Partition group scoring): It uses observed local output estimation (step 2) and output statistics to score each partition group. By using a ratio of obsLocj/obsGloj researchers can estimate that how many these local results become global results.

Step 4 (Partition group flushing): It uses score from partition group scoring (step 3) to flush partition to disk and it can be done by selecting least scoring partition group (iteratesuntil N memory get flushed).

CONCLUSION

In this study, researchers reviewed four flushing techniques to flush the data in memory to disk to make room for new incoming tuples. X-Join and progressive merge join is introduced first whose benefits used by HMJ and was implemented. By considering data arrival, global contribution and data property AGF introduced with effective results. Main difference between earlier mention techniques and AGF is that X-Join. PMJ, HMJ and DINER focus on query plans containing a single join operator where as AGF focus on query plans containing multiple join operators. Main focused of all these techniques on producing early and first results throughput in multi join query plan. And these algorithms also try to improve the flushing policy depending on various theoretical analysis and collected statics.

REFERENCES

Bornea, M.A., V. Vassalos, Y. Kotidis and A. Deligiannakis, 2010. Adoaptive join operator for result rate optimization on streaming inputs. IEEE Trans. Knowledge Data Engin., 22: 1110-1125.

Dittrich, J.P., B. Seeger, D.S. Taylor and P. Widmayer, 2002. Progressive merge join: A generic and non-blocking sort-based join algorithm. Proceedings of the 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China, pp: 299-310.

Ives, Z.G., D. Florescu, M. Friedman, A. Levy and D.S. Weld, 1999. An adaptive query execution system for data integration. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, June, 1999, ACM, New York, USA., pp: 299-310.

Levandoski, J.J., M.E. Khalefa and M.F. Mokbel, 2011. On producing high and early result throughput in multijoin query plans. IEEE Trans. Knowledge Data Engin., 23: 1888-1902.

Mokbel, M.F. and M. Lu and W.G. Aref, 2004. Hash-merge join: A non-blocking join algorithm for producing fast and early join results. Proceedings of the 20th International Conference on Data Engineering, March 30-April 2, 2004, Washington, DC. USA., pp: 251-263.

Urhan, T. and M.J. Franklin, 1999. XJoin: Getting fast answers from slow and burst networks. Technical Report CS-TR- 3994, UMIACS-TR-99-13, Computer Science Department, University of Maryland, February, 1999.

Urhan, T. and M.J.Franklin, 2000. Xjoin: A relatively scheduled pipilined join operator. IEEE Data Engin. Bull., 23: 27-33.

Viglas, S.D., J.F. Naughton and J. Burger, 2003. Maximizing the output rate of multi-way join queries over streaming information sources. Proceedings of the 29th International Conference on Very Large Data Bases-Volume 29, September 9-12, 2003, Berlin, Germany, pp: 285-296.

Wilschut, A.N. and P.M.G. Apers, 1993. Data flow query execution in a parallel main-memory environment. Proceedings of the 1st International Conference on Parallel and Distributed Information Systems, December 4-6, 1991, Miami Beach, FL., pp: 68-77.