ISSN: 2321-7782 (Online)
Volume 2, Issue 1, January 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

# A Review on Adaptive Join Algorithms for Efficient Query Processing On Heterogeneous Data Sets

**Warish Patel**[1]
Computer Science and Engineering
Parul Institute of Technology
Vadodara - India

**Dinesh Vaghela**[2]
Asst.Prof.
Computer Science and Engineering
Parul Institute of Technology
Vadodara - India

***Abstract:*** ***Adaptive algorithms have recently attracted lot of interest for the researchers. This work is focused on how adaptive join operators works in Distributed and heterogeneous environment. It also focuses on quality improvement on join operation. Joins are debatably the most important relational operators. This is so because efficient join processing is important to the overall efficiency of a query processor. Enhancing the performance of larger database relational systems depends greatly on the cost of performing join operations Adaptive join algorithms have recently attracted a lot of consideration in emerging applications where data is provided by autonomous data sources through homogenous or heterogeneous network environments. In traditional join techniques, they can start producing join results as soon as the first input tuples are available, thus improving pipelining by smoothing join result production and by masking source or network delays.***

***Keywords: Query processing, Joins, Stream, DINER, MINER, Distributed database.***

## I. INTRODUCTION

In Real systems hardly ever stored all their data in one large table. To do so would necessitate maintaining several duplicate copies of the same values and could threaten the integrity of the data Instead, IT department everywhere almost always divide their data among several different tables. Because of this, a method is needed to concurrently access two or more tables by using join operation. Here, key focus is to show how join operators work in databases.

Joins are one of the basic constructions of SQL and Databases as they combine records from two or more databases relations into one row source, one set of rows with the same columns and these columns can originate from either of joined tables as well as be formed using expression or built in or user defined functions.

Adaptive query processing has emerged as an answer to the problems that occur because of the fluidity and unpredictability of data arrivals in such environments [1]. Joins are used for joining records or fields from two or more tables in a database by using a value common to both the tables and the result set can be stored or saved in table [1]

Previously, there are four different types of joins and they are defined by ANSI (American National Standard Institute) and they are INNER, OUTER, LEFT, and RIGHT.

## II. INTRODUCTION OF ADAPTIVE ALGORITHM

A number of additional challenges in adaptive joins compared to traditional joins [3] are: The input relations are provided by autonomous network sources. The implication is that one has little or no control over the order or rate of arrival of tuples.[4]

 

Data is transported through unreliable network environment. It is often inappropriate or in-efficient because most traditional join algorithms cannot produce results until at least one of the relations is completely available, the complete data might be available after a long time.

Sometimes these algorithms are unusable, if data is completely available but they produce fractional results. The availability of fractional join results is important for wide range of applications. Their main advantage over traditional join techniques is that, they can begin producing join results as soon as the first input tuples are available, thus improving pipelining by smoothing join result production and by masking source or network delay and there are two Objectives:

a) Experimental study of DINER (Double Index Nested Loop Reactive Join)

b) Calculate the performance comparison of the DINER, MINER and MJoin.

## III. VARIOUS METHODS OF JOIN TECHNIQUES

### *A. Nested-loop join algorithm[4]*

Nested-loop join is considered as a one of the simplest algorithm of join where, for each record of the first table the entire records of the second table has to be scanned. This process is repeated for each and every record of the first table that is for all the first table records. The loop is of two levels and they are outer loop and the inner loop. First table loop is called as outer loop and the second table loop are called as inner loop. As this, Nested loop join algorithm has a repeated input/output scans of one of the table. They are considered as inefficient.

### *B. Sort-Merge Join Algorithm [10]*

Sort merge algorithm is considered as proficient join algorithm when compared to Nested loop join algorithm. Sort merge join algorithm has two operations and they are sorting and merging. In sorting operation the two tables to be joined are sorted in ascending order. In merging operation the two sorted tables are merged.

Sort records of table A based on the join attribute Sort records of table B based on the join attribute

Let i = 1 and j =1

Repeat Read record A (i)

Read record B (j)

If join attribute A (i) < join attribute B(j)

Then i++

Else If join attribute A(i) > join attribute B(j)

Then j++

Else Put records A (i) and B (j) into the Qr

### *C. Hash Based join algorithm[11]*

In hash based join algorithm hashing and probing are the two processes. A hash table is produced by hashing all records of the first table using a particular hash function. Records from the second table are also hashed with the same hash function and probed [3] If any match is found, the two records are concatenated and placed in the query result. A decision must be made about which table is to be hashed and which table is to be probed. Since a hash table has to be produced, it would be better to choose the smaller table for hashing and the larger table for probing. The hash join algorithm is given as:

Let H be a hash function

 

For each record in table B

Read a record from table B

Hash the record based on join attribute value using

Hash function H into hash table

For each record in table A

Read a record from table A

Hash the record based on join attribute value using HProbe into the hash table

If an index entry is found Then Compare each record on this index entry with the record of table S If matched Then Put the pair into Qr

### *D. DINER (Double Index Nested-loopsReactive) Module*

MODERN information processing is moving into a realm where we often need to process data that are pushed or pulled from autonomous data sources through heterogeneous networks.

The key differences between DINER and existing algorithms are 1) an intuitive flushing policy for the Arriving phase that aims at maximizing the amount of overlap of the join attribute values between memory resident tuples of the two relations and 2) a lightweight Reactive phase that allows the algorithm to quickly move into processing tuples that were flushed to disk when both data sources block. The key idea of our flushing policy is to create and adaptively maintain three no overlapping value regions 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 produce many join results in a way that permits easy maintenance of the three-way partition of the values.

When tuples are flushed to disk they are organized into sorted blocks using an efficient index structure, maintained separately for each relation (thus, the part "Double Index" in DINER). This optimization results in faster processing of these tuples during the Reactive and Cleanup phases. The Reactive phase of DINER employs a symmetric nested loop join process, combined with novel bookkeeping that allows the algorithm to react to the unpredictability of the data sources. The fusion of the two techniques allows DINER to make much more efficient use of available main memory. We demonstrate in our experiments that DINER has a higher rate of join result production and is much more adaptive to changes in the environment, including changes in the value distributions of the streamed tuples and in their arrival rates.

### *E. MINER Module:*

MINER extends DINER to multiway joins and it maintains all the distinctive and efficiency generating properties of DINER. MINER maximizes the output rate by: 1) adopting an efficient probing sequence for new incoming tuples which aims to reduce the processing overhead by interrupting index lookups early for those tuples that do not participate in the overall result; 2) applying an effective flushing policy that keeps in memory the tuples that produce results, in a manner similar to DINER; and 3) activating a Reactive phase when all inputs are blocked, which joins on-disk tuples while keeping the result correct and being able to promptly hand over in the presence of new input. Compared to DINER, MINER faces additional challenges namely: 1) updating and synchronizing the statistics for each join attribute during the online phase, and 2) More complicated bookkeeping in order to be able to guarantee correctness and prompt handover during reactive phase. We are able to generalize the reactive phase of DINER for multiple inputs using a novel scheme that dynamically changes the roles between relations.

 

## IV. MEMORY ALLOCATED DINER AND MINER MODULE

To examine the impact that several parameters may have on the performance of the DINER algorithm, through a detailed sensitivity analysis. Moreover To evaluate the performance of MINER when vary the amount of memory allocated to the algorithm and the number of inputs. The main findings of this study include:

- A Faster Algorithm. DINER provides result tuples at a significantly higher rate, up to three times in some cases, than existing adaptive join algorithms during the online phase. This also leads to a faster computation of the overall join result when there are busty tuple arrivals.
- A Leaner Algorithm. The DINER algorithm further improves its relative performance to the compared algorithms in terms of produced tuples during the online phase in more constrained memory environments. This is mainly attributed to our novel flushing policy.
- A More Adaptive Algorithm. The DINER algorithm has an even larger performance advantage over existing algorithms, when the values of the join attribute are streamed according to a stationary process.
- Suitable for Range Queries. The DINER algorithm can also be applied to joins involving range conditions for the join attribute. PMJ also supports range queries but, it is a generally poor choice since its performance is limited by its blocking behavior.

An Efficient Multiway Join Operator. MINER retains the advantages of DINER when multiple inputs are considered. MINER provides tuples at a significantly higher rate compared to MJoin during the online phase. In the presence of four relations, which represents a challenging setup, the percentage of results obtained by MINER during the arriving phase varies from 55 percent (when the allocated memory is 5 percent of the total input size) to more than 80 percent (when the allocated memory size is equal to 20 percent of the total input size).

## V. PERFORMANCE ANALYSIS

To demonstrate DINER's superior performance over a variety of real and synthetic data sets in an environment without network congestion or unexpected source delays. To plot the cumulative number of tuples produced by the join algorithms over time, during the online phase for the CSCO stock and the Weather data sets. To observe that DINER has a much higher rate of tuples produced those all other competitors. For the stock data, while RPJ is not able to produce a lot of tuples initially, it manages to catch up with XJoin at the end. To compare DINER to RPJ and HMJ on the real data sets when to vary the amount of available memory as a percentage of the total input size. The y axis represents the tuples produced by RPJ and HMJ at the end of their online phase (i.e., until the two relations have arrived in full) as a percentage of the number of tuples produced by DINER over the same time. The DINER algorithm significantly outperforms RPJ and HMJ, producing up to 2.5 times more results than the competitive techniques. The benefits of DINER are more significant when the size of the available memory given to the join algorithms is reduced.

## VI. CONCLUSION

This paper gives detail study of the various new adaptive join algorithm for maximizing the output rate of tuples, when two relations are being streamed to and joined at a local site. The advantages of DINER stem from 1) its intuitive flushing policy that maximizes the overlap among the join attribute values between the two relations, while flushing to disk tuples that do not contribute to the result and 2) a novel re-entrant algorithm for joining disk resident tuples that were previously flushed to disk. Moreover, DINER can efficiently handle join predicates with range conditions, a feature unique to this technique. A significant extension to this framework in order to handle multiple inputs. The resulting algorithm, MINER addresses additional challenges, such as determining the proper order in which to probe the in-memory tuples of the relations, and a more complicated bookkeeping process during the Reactive phase of the join. Through this experimental evaluation, we have

 

demonstrated the advantages of both algorithms on a variety of real and synthetic data sets, their resilience in the presence of varied data and network characteristics and their robustness to parameter changes.

Table-1: Comparison Table for joining algorithms:

| Algorithm | Comparisons on basis of Time Case Complexity | Time Complexity in BIG-O Notation |
|---|---|---|
| Hash Merge Join (HMJ) | High | O(N + M) |
| Rate-based Progressive Join (RPJ) | Low compare to HMJ | O(L(\|S1\| + \|S2\|)) |
| Multiway Join (MJOIN) | Low compare to RPJ and HMJ | |
| Double Index Nested Loops Reactive Join (DINER) | Low compare to MJoin, HMJ and RPJ | O(N * M) |
| Multiple Index Nested-Loop Reactive join (MINER) | Low compared to all above | O(N * M) |

## References

1. J.Jayashree and C.Ranichandra "Join Algorithm forEfficient Query Processing For Large Datasets" AsianJournal of Computer Science and Information Technology2: 3 (2012) 31 –35
2. MihaelaA.Bornea,VasilisVassalos, YannisKotidis,AntoniosDeligiannakis: Adaptive Join Operators forResult Rate Optimization on Streaming Inputs. IEEETrans. Knowl. Data Eng. 22(8): 1110-1125 (2010)
3. J. D. Ullman, H. Garcia-Molina, and J. Widom.Database Systems: The Complete Book. Prentice Hall,2001
4. M. A. Bornea, V. Vassalos, Y. Kotidis, and A. Deligiannakis. DoubleIndex Nested-loop Reactive Join for Result Rate Optimization. In ICDEConf., 2009
5. David Taniar, Clement H.C. Leung, WennyRahayu, SushantGoel. (2008) "High-Performance Parallel Database Processing and Grid Databases" A John Wiley
6. Z. G. Ives, D. Florescu, and et al. An Adaptive Query Execution System for Data Integration.In SIGMOD, 1999.
7. W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS. In PDIS, 1991
8. T.Urhan and M.J.Franklin.Xjoin: A Relatively scheduled pipilined join operator.IEEE Data Eng.Bull, 23920,2000
9. S.D Viglas,J.F.Naughton and J.Burger.Maximizing the output rate of multiway join queries over streaming information sources.In VLDB 2003: proceeding of the 29th international
10. J. Dittrich, B. Seeger, and D. Taylor. Progressivemerge join: A generic and non-blocking sort-based joinalgorithm. In Proceedings of VLDB, 2002. Rate-based Optimization. InProceedings of ACM SIGMOD Conference, 2005
11. M. F. Mokbel, M. Lu, and W. G. Aref. Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results. In ICDE Conf., 2004.conference on very large databases 2003.
12. Y. Tao, M. L. Yiu, D. Papadias, M. Hadjieleftheriou,and N. Mamoulis. RPJ: Producing Fast Join Results on Streams Through Rate-based Optimization. In Proceedings of ACM SIGMOD Conference, 2005

 