AN ENHANCED JOIN ALGORITHM FOR OUTPUT RATE IMPROVEMENT

*Mr Pratik A. Patel**
PG-Student- Department of Computer Science,
Parul Institute of Technology
Vadodara (Gujarat), India.

Mr Arpit M. Rana
Assistant Professor- Department of Computer Science,
Parul Institute of Technology
Vadodara (Gujarat), India.

***Abstract*— This work is focused on how join operator works in a single, homogeneous and heterogeneous environment. Adaptive join algorithms have recently attracted a lot of attention in emerging applications where data is provided by autonomous data sources through heterogeneous network environments. Their main advantage over traditional join techniques is that 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. In this paper, the first propose Double Index NEsted-loops Reactive join (DINER), a new adaptive two-way join algorithm for result rate maximization. DINER combines two key elements: an intuitive flushing policy that aims to increase the productivity of in-memory tuples in producing results during the online phase of the join, and a novel re-entrant join technique that allows the algorithm to rapidly switch between processing in memory and disk-resident tuples, thus, better exploiting temporary delays when new data are not available. Then extend the applicability of the proposed technique for a more challenging setup: handling more than two inputs. Multiple Index NEsted-loop Reactive join (MINER) is a multi-way join operator that inherits its principles from DINER.**

Keywords-Query processing, Streams, Joins, MINER AND DINER

I. INTRODUCTION

Adaptive join algorithms were created in order to lift the limitations of traditional join algorithms in such environments. By being able to produce results whenever input tuples become available, adaptive join algorithms overcome situations like initial delay, slow data delivery, or bursty arrival, which can affect the efficiency of the join. Real systems rarely stored all their data in one large table. To do so would require 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 simultaneously access two or more tables by using join operation. Here, main focus is to show how join operators work in databases

II. EXISTING WORK

There are three basic join algorithms: Hash based join, Sort Merge based join, Nested loop based join algorithm. There are some other algorithms are also available. The all existing work is give here.

A. .Nested loop based join algorithm:

In this, Nested loop join compares each row from one table (i.e. outer table) to each row from the other table (i.e. inner table) looking for rows that satisfy the join predicate. Inner join and outer join are the logical operations. The cost of this algorithm is proportional to the size of outer table multiplied by size of the inner table. As this Nested loop join algorithm has a repeated input/output scans of one of the table they are considered as inefficient

Pseudo code for algorithm:-
Let the two tables be A and B,
then the algorithm of Nested-loop algorithm are as
For each record of table A
Read record from table A
For each record of table B
Read record from table B
Compare the join attributes
If matched
Then Store the records

B. Hash based join algorithm:

Hash joins [3] are used when the joining large tables. The optimizer uses smaller of the two tables to build a hash table in memory and the scans the large tables and compares the hash value (of rows from large table) with this hash table to find the joined rows.

The algorithm of hash join is divided in two parts:

1. Build: - In-memory hash table on smaller of the two tables.

2. Probe: - This hash table with hash value for each row second table

For Example:-

Consider schema of two tables Emp_Master and Emp_Info

Create Table Emp_Master (Id int, Name varchar (10), Designation varchar (10), Dept varchar (10))

Another table is,

Create Table Emp_Info (Id int, Dt_of_Joining Date Time)

Query is written as:-

Select Id int, Name varchar (10), Designation varchar (10), Dept varchar (10) From Emp_Master inner join Emp_Info on Emp_Master.Id = EmP_Info. Id Order by Emp_Info. Dt_of_Joining desc

C. Sort based join algorithm:

It is also known as sort merge join [4] algorithm. Sort merge join is used to join two independent data sources. They perform better than nested loop when the volume of data is big in tables but not as good as hash joins in general. They perform better than hash join when the join condition columns are already sorted or there is no sorting required.
Existing work on adaptive join algorithms can be classified in two groups:-hash based join and sort based join. Examples of hash based algorithms are XJoin, MJoin, Hash Merge join, and Progressive Merge join.

D. Double Pipelined Hash Join (DPHJ):

The double Pipelined Hash Join (DPHJ) [5] is another extension of the symmetric hash join algorithm. DPHJ has two stages. The first stage is similar to the in-memory join in the symmetric hash join and XJoin. In the second stage, pairs that are not joined together in the first phase are marked and are joined in disk. DPHJ is suitable for moderate size data, but does not scale well for large data sizes.

E. XJoin:

It is a non-blocking join operator, called XJoin[6] which has a small memory footprint, allowing many such operators to be active in parallel. XJoin is optimized to produce initial results quickly and can hide intermittent delays in data arrival by reactively scheduling background processing. We show that Xjoin is an effective solution for providing fast query responses to user even in the presence of slow and bursty remote sources.
In previous work [7] of Xjoin, we identified three classes of delays that can affect the responsiveness of query processing: 1) initial delay, in which there is a longer than expected wait until the first tuple arrives from a remote source 2) slow delivery, in which data arrive at a fairly constant but slower than expected rate and 3) bursty arrival, in which data arrive in a fluctuating manner.

F. Hash Merge Join:

HMJ [8] is a hybrid query processing algorithm combining ideas from XJoin 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 in data arrives. The merging phase is responsible for producing join results if the two sources are blocked.

G. MJoin:

The basic idea of the MJoin[9] algorithm is simple: generalize the symmetric binary hash join and the XJoin algorithms to work for more than two inputs. Our primary goal is to maximize the output rate during the memory-to-memory phase of the MJoin. As with the binary XJoin, in MJoin, the disk to-memory phase is intended to allow the system to generate outputs while its inputs are blocked, while the disk to-disk phase is intended to generate any final answers after the inputs have terminated. Interestingly, for the MJoin, how we handle memory overflow determines the output rate of the memory-to-memory phase.

H. Progressive Merge Join:

PMJ [4] is the adaptive non-blocking version of the sort merge join algorithm. It splits the memory into two partitions. As tuples arrive, they are inserted in their memory partition. 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 the memory gets exhausted. Next, the partition pair (i.e., the bucket pairs that were simultaneously flushed each time the memory was full) is copied on disk. After the data from both sources completely arrives, the merging phase begins. 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 arrives. The merging phase is responsible for producing join results if the two sources are blocked.

I. Rate based Progressive Join:

RPJ [10] is the most recent and advanced adaptive join algorithm. It is the first algorithm that tries to understand and exploit the connection between the memory content and the algorithm output rate. During the online phase it performs as HMJ. When memory is full, it tries to estimate which tuples have the smallest chance to participate in joins.
In this work, we used RPJ (Rate-based Progressive Join), which continuously adapts its execution according to the data properties (e.g., their distribution, arrival pattern, etc.).

RPJ utilizes a novel flushing algorithm which is op-timal among all possible alternatives (based on the same statistics about data distributions, arrival patterns, etc.), and significantly enhances the efficiency of the memory stage. Furthermore, RPJ maximizes the output rate by invoking the memory-disk and disk-disk in a strategic order, i.e., the next stage selected for execution is the one expected to produce the highest output rate.

III. PROPOSED SYSTEM

MODULE DESCRIPTION:

A. DINER (Double Index Nested-loops Reactive) Module:

MODERN information processing is moving into a realm where 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 non-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 Clean-up 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. To demonstrate in the 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.

Its operation based on the following points:

Point 1: ReactiveNL initially selects one relation to behave as the outer relation of the nested loop algorithm, while the other relation initially behaves as the inner relation. Notice that the "inner relation" (and the "outer") for the purposes of ReactiveNL consists of the blocks of the corresponding relation that currently reside on disk, because they were flushed during the Arriving phase.

Point 2. ReactiveNL tries to join successive batches of OuterMem blocks of the outer relation with all of the inner relation, until the outer relation is exhausted. The value of OuterMem is determined based on the maximum number of blocks the algorithm can use (input parameter MaxOuterMem) and the size of the outer relation. However, as DINER enters and exits the Reactive phase, the size of that inner relation may change, as more blocks of that relation may be flushed to disk. To make it easier to keep track of joined blocks, we need to join each batch of OuterMem blocks of the outer relation with the same, fixed number of blocks of the inner relation – even if over time the total number of disk blocks of the inner relation increases. One of the key ideas of ReactiveNL is the following: at the first invocation of the algorithm, we record the number of blocks of the inner relation in JoinedInner. From then on, all successive batches of OuterMem blocks of the outer relation will only join with the first JoinedInner blocks of the inner relation, until all the available outer blocks are exhausted.

Point 3. When the outer relation is exhausted, there may be more than JoinedInner blocks of the inner relation on disk (those that arrived after the first round of the nested loop join, when DINER goes back to the Arriving phase). If that is the case, then these new blocks of the inner relation need to join with all the blocks of the outer relation. To achieve this with the minimum amount of book keeping, it is easier to simply switch roles between relations, so that the inner relation (that currently has new, unprocessed disk blocks on disk) becomes the outer relation and vice versa (all the counters change roles also, hence JoinedInner takes thevalue of JoinedOuter etc, while CurrInner is set to point to the first block of the new inner relation).

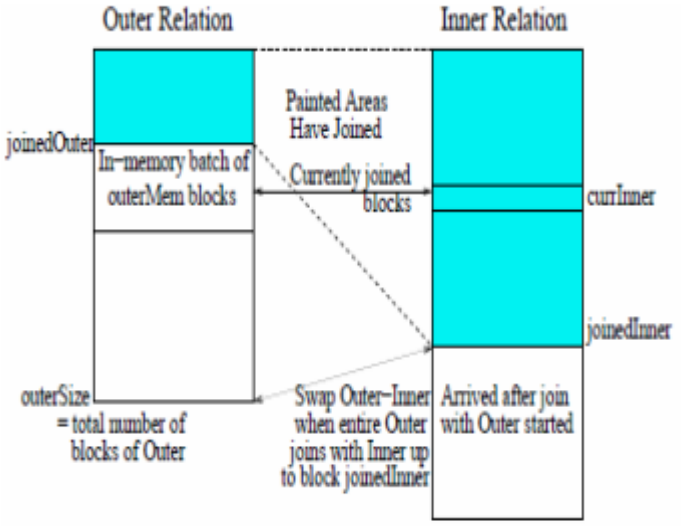

Figure 1: status of the algorithm during Reactive phase

Thus, an invariant of the algorithm is that the tuples in the first JoinedOuter blocks of the outer relation have joined with all the tuples in the first JoinedInner blocks of the inner relation.

Point 4. In earlier work, the flushing policy of DINER spills on disk full blocks with their tuples sorted on the join attribute. The ReactiveNL algorithm takes advantage of this data property and speeds up processing by performing an in-memory sort merge join between the blocks.

During this process, it is important that we do not generate duplicate join between tuples touter and tinner that have already joined during the Arriving phase. This is achieved by proper use of the ATS and DTS time stamps [4]. If the time intervals [touter.ATS, touter.DTS] and [tinner.ATS, tinner.DTS] overlap, this means that the two tuples coexisted in memory during the Arriving phase and their join is already obtained. Thus, such pairs of tuples are ignored by ReactiveNL algorithm.

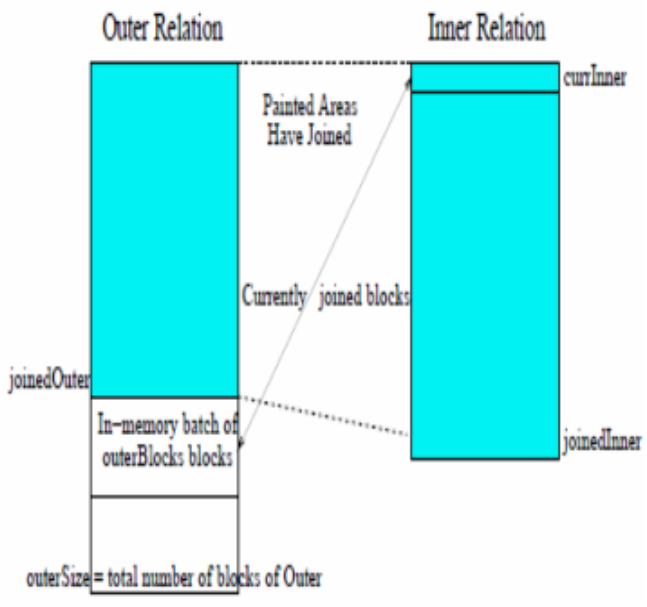

Figure 2: status of the algorithm after swapping the roles of two relations

B. MINER Module:

MINER extends DINER to multi way 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.

C. Memory Allocated DINER & MINER Module:

To investigate 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 bursty 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 non-stationary process. Moreover, it better adjusts its execution when there are unpredictable delays in tuple arrivals, to produce more result tuples during such delays.
- **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 behaviour.

An Efficient Multi way 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).

IV. 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 that 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. In the next set of experiments, to evaluate the performance of the algorithms when synthetic data are used. In all runs, each relation contains 100,000 tuples.

V.CONCLUSIONS

In this work, to introduce DINER, a 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. To also present 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.

REFERENCES

[1] Mihaela A.Bornea, Vasilis Vassalos, Yannis Kotidis, Antonios Deligiannakis: *Adaptive Join Operators for Result Rate Optimization on Streaming Inputs. IEEE Trans. Knowl. Data Eng. 22(8): 1110-1125 (2010)*

[2] M. A. Bornea, V. Vassalos, Y. Kotidis, and A. Deligiannakis. *DoubleIndex Nested-loop Reactive Join for Result Rate Optimization. In ICDEConf., 2009.*

[3] 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.nal conference on very large databases 2003.*

[4] J. Dittrich, B. Seeger, and D. Taylor. *Progressive merge join: A generic and non-blocking sort-based join algorithm. In Proceedings of VLDB, 2002.*

[5] Z. G. Ives, D. Florescu, and et al. *An Adaptive Query Execution System for Data Integration.In SIGMOD, 1999.*

[6] T.Urhan and M.J.Franklin.Xjoin*: A Relatively scheduled pipilined join operator.IEEE Data Eng.Bull,23920,2000*

[7] T. Urhan, M. J. .Franklin, and L. Amsaleg. *Cost Based Query Scrambling for Initial Delays. ACM SIGMOD Conf., Seattle, WA, 1998.*

[8] 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.nal conference on very large databases 2003.*

[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] 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.*

AUTHORS PROFILE

PRATIK PATEL has received B.E. in Computer Engineering from Gujarat University, India 2010 and is currently pursuing masters in Computer Science & Engineering in Parul Institute of Technology, Vadodara. Area of interest is Data mining. Have published two papers in International Journals.

ARPIT RANA has received M.E. in Computer Science & Engineering in Parul Institute of Technology, Vadodara. The author is currently working as Assistant Professor Senior in PIT and doing research work in Data Mining.