An End-Around Approach for Efficient Join Query Processing

Raksha Chauhan
Dept. of Computer Science & Engineering
Gujarat Technological University, Gujarat

Pratik A Patel
Dept. of Computer Science & Engineering
Gujarat Technological University, Gujarat

ABSTRACT

This paper introduced a method for producing immediate and result in multi-join query, in homogeneous and heterogeneous environment. In recent years Adaptive or Non Blocking join algorithms have attracted a lot of attention in streaming applications, where data is provided from autonomous data sources in heterogeneous network environments. This algorithms are better as compared to traditional algorithms is that they can generates join results as early as the first input tuples are on hand hence it improves pipelining, smooth out join result production and also masking source or network delays. As response time of the queries places a very important role in adaptive join, the join algorithm like Hash Join, Sort Merge Join are become unacceptable for this environment because they require preprocessing before generating the join result. Hence, in adaptive join technique only possible algorithm is Nested loop join. In Nested Loop Join, every single record of the outer relation is compared with every single record of the inner relation. The no. of comparisons done by the nested loop join can be reduced by making improvement in Block Nested loop Join. In proposed End-Around Block Nested loop join outer and inner table's comparison is done in parallel and whenever a row in first location didn't find a match then row from first location removed and placed at rear end as like in a queue, the matched row removed from inner relation and added to result set. Whenever, New tuple arrive is then pushed into rear end and process is continuing with new incoming tuples in streaming environments.

Keywords

Query processing, Streams, Joins, Block nested loop join

1. INTRODUCTION

Non Blocking join algorithms were explored to overcome the limitations of traditional join algorithms in streaming environments. As Non-Blocking algorithms are able to produce results whenever input tuples are on hand, hence, overcome situations like initial delay, slow data delivery, or bursty arrival, which can affect the efficiency of the join. Real time systems rarely stored all data in one large table and if to store data in large table it would require to maintain several duplicate copies of the same values and could destroy the integrity of the data. Instead, IT department all time and everywhere divide their data into several different tables for efficient access. Because of this, a method is required which simultaneously access two or more table using join operation.

Here, main focus is to show how join operators work in databases.

2. 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 given here.

2.1 Nested loop based join algorithm

In Nested loop join, every single row of one table (i.e. outer table) compare with every single row of the other table (i.e. inner table) based on join predicate. Inner join and outer join are the logical operations. The cost of Nested loop join algorithm is proportional to the size of outer table multiplied by size of the inner table.

Algorithm:

Let the two tables R and S, algorithm is follows:

For every record of table R,

Read record from table R,

For every record of table S,

Read record from table S,

Compare the join attributes

If matched

Then Store the records

2.2 Hash based join algorithm

Hash join algorithms [2] are used mostly for joining large tables. The algorithm of hash join is divided in two parts:

1. Build: - hash table is created using smaller table in a memory
2. Probe: - using hash table scan larger table with hash value

For Example:-

Consider schema of two tables Employee_Master and Employee_Info

Create Table Employee_Master (Id int, Name varchar (20), Designation varchar (20), Dept varchar (20))

Another table is: Create Table Employee_Info (Id int, Dt_of_Joining Date Time)

Query is written as:-

Select Id int, Name varchar (20), Designation varchar (20),

Dept varchar (20)

From Employee_Master inner join Employee_Info

On Employee_Master.Id = Employee_Info. Id

Order by Employee_Info.Dt_of_Joining desc

2.3 Sort based join algorithm

Sort merge join [3] algorithm is used to join two autonomous data sources. When the volume of data is big in a table, short merge join algorithm perform better than the nested loop join algorithm but less then hash join algorithm. Its performance is better than hash join algorithm when the joining data is already sorted on join condition 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.

2.4 Double Pipelined Hash Join (DPHJ)

The double Pipelined Hash Join (DPHJ) [4] 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 not scale well for large data sizes.

2.5 XJoin

It is a non-blocking join algorithm XJoin[5], it is optimized to produce first few results quickly and hide delays in data arrival by reactively scheduling background processing. We show that Xjoin is an efficient alternative for providing fast query responses to user even in the presence of slow and bursty network sources.

In previous work [6] of Xjoin, we identified three classes of delays that can affect the responsiveness of query processing:

1) Initial delay:-longer than expected wait until the first tuple arrives from a remote source

2) Slow delivery:- data arrive at a fairly constant but slower than expected rate

3) Bursty arrival:- data arrive in a fluctuating manner.

2.6 Hash Merge Join

HMJ [2] main idea is to minimize time to produce first few result and produce join result if two sources of operator already blocked. HMJ was implemented taking benefits of XJoin and Progressive Merge Join with two phases: The hashing phase and the merging phase. The hashing phase, in a memory hash-based join algorithm produces join results as soon as data arrives. In merging phase, join results produced if the two sources are blocked.

2.7 MJoin

The basic idea of the MJoin[8] algorithm is simple: generalize the symmetric binary hash join and the XJoin algorithms to work for more than two inputs. Main objective of this study is to maximize the output rate during the memory-to-memory phase of the MJoin. Like XJoin, in MJoin, the disk to memory phase generates outputs while its inputs are blocked, while the disk to-disk phase generates final answers after the inputs have terminated.

2.8 Progressive Merge Join

PMJ [3,10] is non-blocking version of the sort merge join algorithm. It splits the memory into two partitions, as tuples arrives; they are inserted in their memory partition. When the memory exhaust, the partitions are sorted on the join attribute and are joined using any memory join algorithm. Thus, resultant tuples are obtained each time the memory gets exhausted. 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. Merging phase is switch when two sources are blocked and hence produce join results.

2.9 Rate based Progressive Join

RPJ [9] 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 exhausted, it tries to find which tuples have the smallest chance to participate in joins. In this Rate-based Progressive Join dynamically changes its execution according to its data properties i.e distribution, arrival pattern, etc. RPJ introduced a novel optimal flushing algorithm which is based on the same data statistics i.e distribution, arrival pattern, 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 is selected for execution which is the one expected to produce the higher output rate.

3. PROPOSED SYSTEM

In Block Nested Loop Join[7], To join relations ‘r’ and relation ‘s’, the outer loop reads the blocks of relation ‘r’ and inner loop reads the blocks of relation ‘s’. If there is enough memory to fit relations ‘r’ and‘s’ than the join operation is done more efficiently. The number of disk accesses is done in Block Nested loop Join – is to read the blocks of relation ‘r’ and to access the disk for reading the blocks of relation ‘s’.

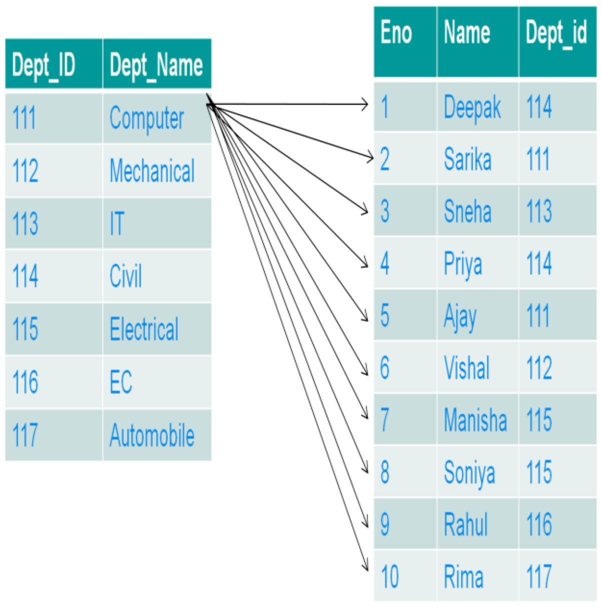
```
Algorithm: Block Nested Loop Join
for each block bR of ‘r’ do
{blocks of relation ‘r’ are scanned one by one.
for each block bS of s do
{blocks of relation ‘r’ are scanned one by one.
Compute bR *bS in memory
}}
```

The BNLJ algorithm is improved version of the NLJ algorithm, it is used for transferring blocks of participating relation efficiently rather than transferring the tuples in the join operation. BNLJ works by reading a block of tuples from the outer and inner relation. In one block at a time-BNLJ, each relation chunk’s is transferred for join operations in a memory from hard disk.

Block nested loop join algorithms creates blocks of outer relation that fit into input buffer pages of memory and then scanning is performed over the inner relation ‘s’ for every single block of the outer relation. Key on outer relation, for every single block of relation ‘r’ is scanned, the *bS* blocks of relation‘s’ are scanned and for *bR* blocks of relation ‘r’, the (*bS* **bR*) times the blocks of relation‘s’ are scanned. The total block transfer is ((*bR* * *bS) + bR*) [7].

Improvement is here over nested loop join is disk input and output rate is improved. But no of comparison is same as in case of nested loop join. So here as parts of extension of proposed block nested loop join algorithm is to make it more suitable for online environment by reducing no. of comparison. Technique is discussed below.

In the Block Nested loop join every single row of the outer relation is compared with every single row of the inner relation same as like Nested loop join, thus results in a Cartesian product as shown in below fig 1.Here two tables Department table and Student table are subject to Join Operation



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil
115	Electrical
116	EC
117	Automobile

Eno	Name	Dept_id
1	Deepak	114
2	Sarika	111
3	Sneha	113
4	Priya	114
5	Ajay	111
6	Vishal	112
7	Manisha	115
8	Soniya	115
9	Rahul	116
10	Rima	117

Eno	Dept_id	Name	Dept_Name
1	111	Deepak	Computer
2	111	Sarika	Computer
3	112	Sneha	Mechanical
4	113	Priya	IT
5	114	Ajay	Civil
6	114	Vishal	Civil
7	115	Manisha	Electrical
8	116	Soniya	EC
9	117	Rahul	Automobile
10	117	Rima	Automobile

Fig 1: Actual Processing of Block Nested loop Join

For every single record of the inner relation, outer relation’s all records are scanned, if there is any match found then those matched results are taken out in a new table. Execution time is calculated.

Here suppose the outer table has 'n' records and inner table has 'm' records, than the no of comparisons is 'n*m' has been made for results. By using End-Around approach in comparison to be used instead of conventional approaches in an existing block nested loop joins to maximize the no. of resultant tuples.

In the proposed End-Around approach, outer relation(or outer table) is a relation containing less number of records and other table containing a higher record then outer relation is inner relation(or inner table).

First step, outer relation’s all rows are compared with inner relation’s rows in parallel at a time.

Second Step, while comparing the matching rows is removed from the relation “inner” and is added to the result set.

Third Step, when a row did not find a match in the first location, it is again added in the rear part of the relation (or table) as shown in below fig 2.

Fourth Step, from the outer relation when the matching rows are removed, empty space is created where other rows are inserted in a way takes place in the queue.

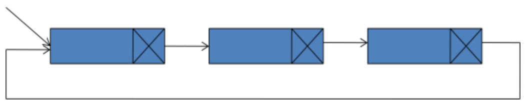
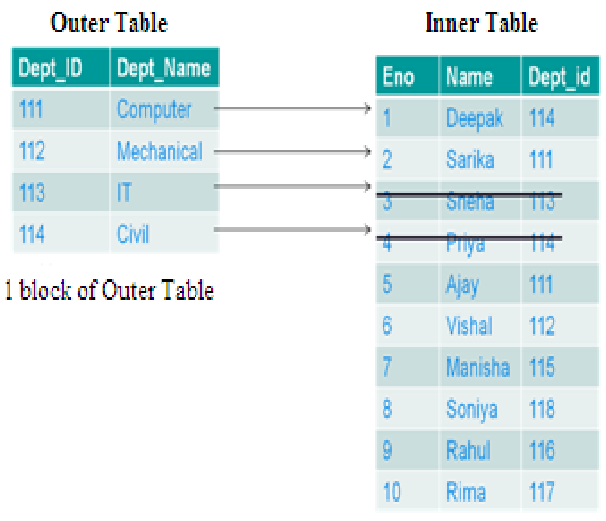


Fig 2: Records pushed to the table in the form of a queue

In steaming join where the tuples are continues arriving at different rate, this type of join can be used. As long as tuples arrives in a system added to the rear end of the tables and unmatched rows which is moved upward in a table are again added into the rear end to find a match with outer relation. The following fig 3-13 will explain this Join technique.



Outer Table

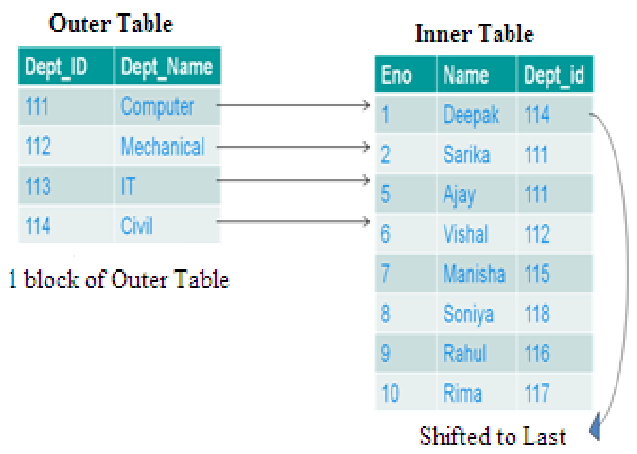
Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

1 block of Outer Table

Inner Table

Eno	Name	Dept_id
1	Deepak	114
2	Sarika	111
~~3~~	~~Sneha~~	~~113~~
~~4~~	~~Priya~~	~~114~~
5	Ajay	111
6	Vishal	112
7	Manisha	115
8	Soniya	118
9	Rahul	116
10	Rima	117

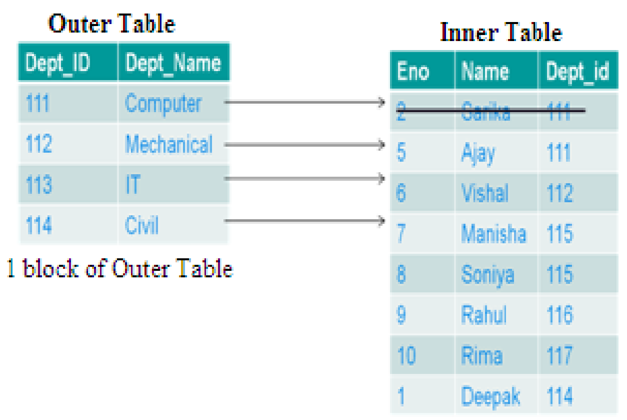
Fig 3: Matched found, two rows are moved into the result set



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
1	Deepak	114
2	Sarika	111
5	Ajay	111
6	Vishal	112
7	Manisha	115
8	Soniya	118
9	Rahul	116
10	Rima	117

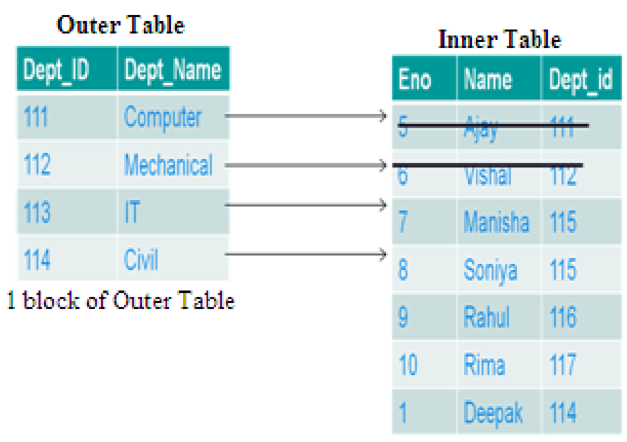
Fig 4: No matched found, inner table first row moved to the rear end



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
~~2~~	~~Sarika~~	~~111~~
5	Ajay	111
6	Vishal	112
7	Manisha	115
8	Soniya	115
9	Rahul	116
10	Rima	117
1	Deepak	114

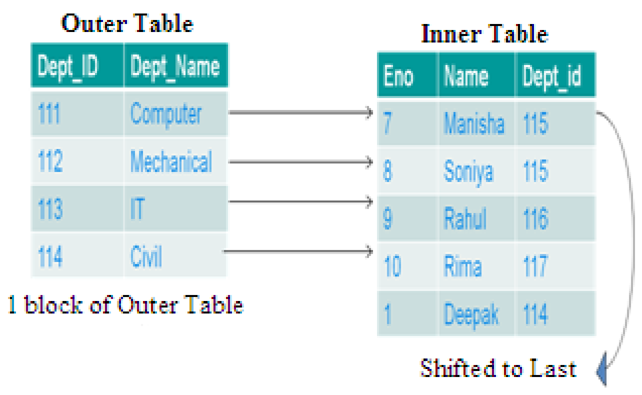
Fig 5: Matched found, one row is moved into the result set



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
~~5~~	~~Ajay~~	~~111~~
~~6~~	~~Vishal~~	~~112~~
7	Manisha	115
8	Soniya	115
9	Rahul	116
10	Rima	117
1	Deepak	114

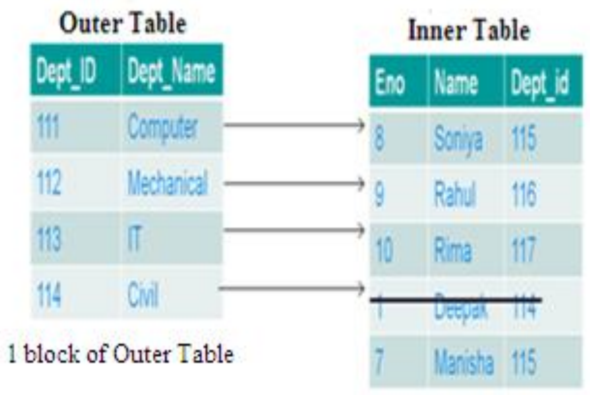
Fig 6: Matched found, two rows are moved into the result set



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
7	Manisha	115
8	Soniya	115
9	Rahul	116
10	Rima	117
1	Deepak	114

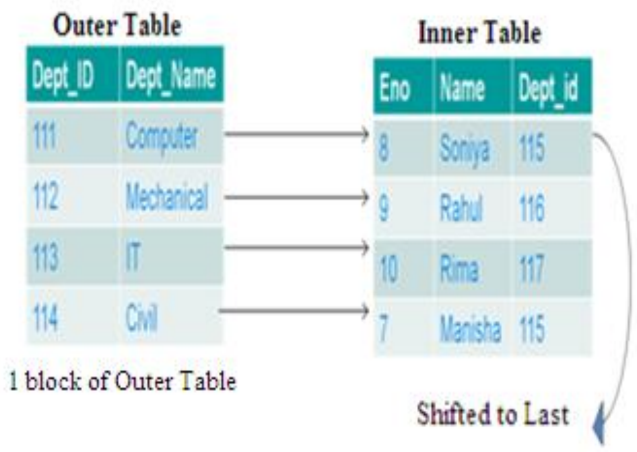
Fig 7: No matched Found, inner table first row moved to the rear end



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
8	Soniya	115
9	Rahul	116
10	Rima	117
~~1~~	~~Deepak~~	~~114~~
7	Manisha	115

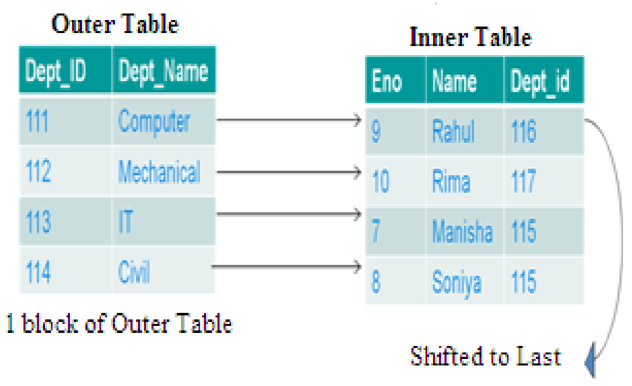
Fig 8: Matched found, one row is moved into the result set



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
8	Soniya	115
9	Rahul	116
10	Rima	117
7	Manisha	115

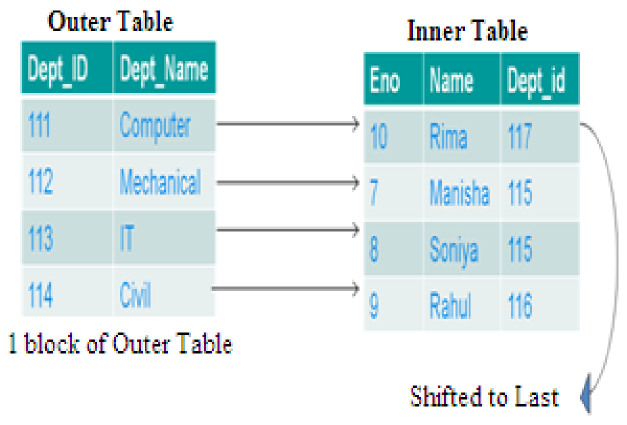
Fig 9: No matched found, inner table first row moved to the rear end



Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
9	Rahul	116
10	Rima	117
7	Manisha	115
8	Soniya	115

Fig 10: No matched found, inner table first row moved to the rear end

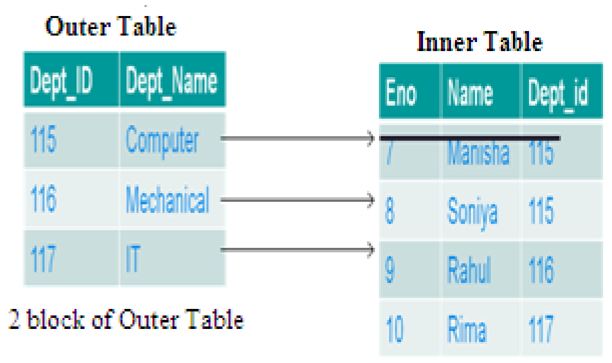


Dept_ID	Dept_Name
111	Computer
112	Mechanical
113	IT
114	Civil

Eno	Name	Dept_id
10	Rima	117
7	Manisha	115
8	Soniya	115
9	Rahul	116

Fig 11: No matched found, inner table first row moved to the rear end

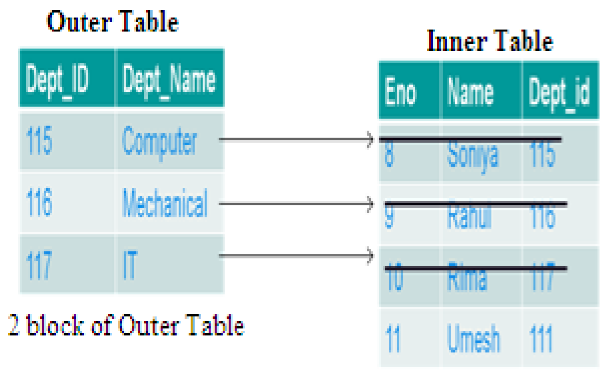
First block of outer relation is completely compared with inner relation this is shown in fig 3 to 11. Now, Second block of outer relation is compared with inner relation is shown below.



Dept_ID	Dept_Name
115	Computer
116	Mechanical
117	IT

Eno	Name	Dept_id
~~7~~	~~Manisha~~	~~115~~
8	Soniya	115
9	Rahul	116
10	Rima	117

Fig 12: Matched found, one row is moved into the result set



Dept_ID	Dept_Name
115	Computer
116	Mechanical
117	IT

Eno	Name	Dept_id
~~8~~	~~Soniya~~	~~115~~
~~9~~	~~Rahul~~	~~116~~
~~10~~	~~Rima~~	~~117~~
11	Umesh	111

Fig 13: Matched found, three rows are moved into the result set and new incoming tuple inserted to rear end

With our approach we have seen that all the rows of outer relation is get compared with inner relation in less time and if new tuples arrive that add at the rear end and comparison of inner and outer relation continue in parallel fashion as discussed above. Number of comparison is less as compared to Block Nested loop and Nested loop Join.

By using this technique no of comparison of Block Nested Loop Join is reduced, hence it reduces join time processing and increases output tuples with fast rate. In streaming Environment Fast early or partial results are vary essential for further processing.

Here, It is seen that with End-Around approach all the rows of outer relation has been compared with inner relation and also if new tuples arrive that push at the rear end. With new streaming tuples join process is continue and comparison is done in parallel fashion same as discussed above.

4. EXPERIMENT RESULTS

The given below graph shows the no. of comparisons required to perform the Nested Loop Join and Proposed End-Around Technique (EBNLJ for short). The no. of comparison is greatly reduced in proposed End-Around technique. End-Around BNLJ and Nested Loop Join behaves similar in worst case situation. In streaming environment worst case situation take place very rarely.

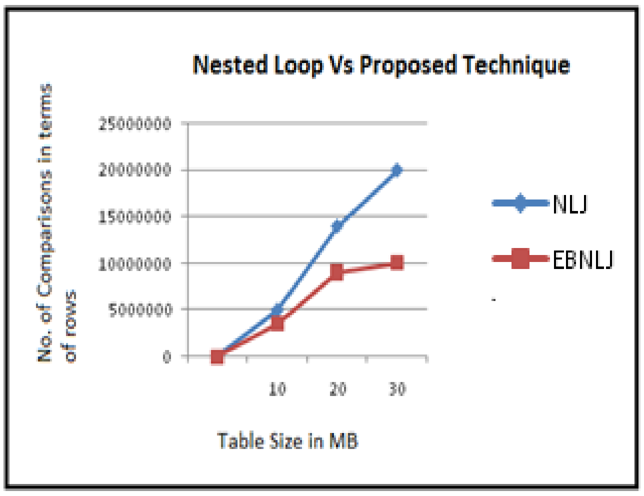


Fig 14: Shows Comparison of Nested Loop Join and End-Around Technique

The performance of proposed End-Around Block nested join and DINER algorithm is calculated on the basis of throughput i.e the number of joined tuples per unit time. By using proposed End-Around algorithm the number of joined tuples is increased. For example, DINER produced 76 joined tuples, whereas End-Around technique produces 121 joined tuples in 15millisecond.

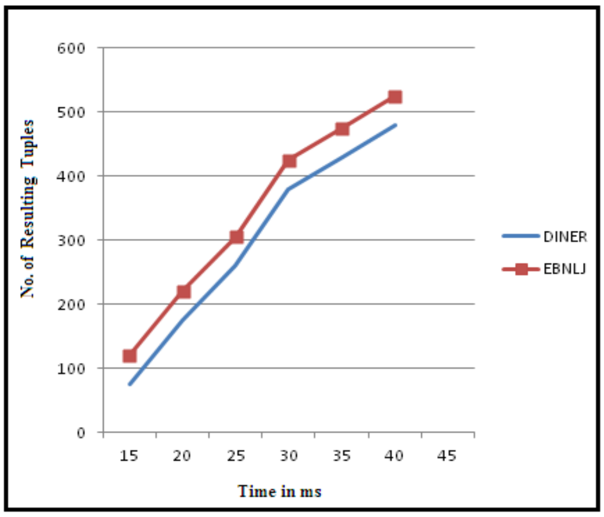


Fig 15- Performance comparison of proposed End-Around Technique and DINER

5. CONCLUSIONS

The major benefit of adaptive or Non Blocking join algorithm is that they can start generating joining results as early as the first input tuples are on hand. Nested loop join is the one join algorithm that can produce the output without any pre-work. In Nested Loop Join every single row of the outer relations are compared with the inner relation, which is same as Cartesian Join. Proposed End-Around technique reduced the number of comparisons compared to nested loop join and DINER. Using proposed End-Around approaches, the no. of comparison are reduced, hence, increased the no. of resultant tuples which is produced per unit of time.

Hence, it is proved that in 15millisecond, the End-Around BNLJ produced 121 tuples and DINER produced 76 tuples. For future work, the flushing policy can be enhanced by considering more factors to specify which tuples have to be flushed. The flushing policy has a significant role in improving the performance of the join operation.

6. REFERENCES

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

[2] 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.

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

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

[5] T.Urhan and M.J.Franklin, ”Xjoin: A Relatively scheduled pipelined join operator”, In IEEE Data Eng.Bull,23920,2000

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

[7] Deepak Shukla, Dr. Deepak Arora, Rakesh Kr.Pandey, K.K Agarwal, “An Efficient Approach of Block Nested Loop algorithm based on Rate of Block Transfer” In International Journal of Computer Application,Volume-21,No 3,May-2011

[8] S.D Viglas,J.F.Naughton and J.Burger, “Maximizing the output rate of multi-way join queries over streaming information sources”. In VLDB 2003: proceeding of the 29th international

[9] 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.

[10] J. Dittrich, B. Seeger, and D. Taylor. “Progressive merge join “On producing join results early”, In. ACM SIGMOD, 2003

[11] Mihaela A. Bornea, Vasilis Vassalos, Yannis Kotidis, and Antonios Deligiannakis. “Double index nested-loop reactive join for result rate optimization”. In Proc. Intl. Conf. onData Engineering (ICDE), pages 481–492, 2009.

[12] G. Abdulla, T. Critchlow, and W. Arrighi, “Simulation Data as Data Streams,” SIGMOD Record, vol. 33, no. 1, pp. 89–94, 2004.