A NEW TRACE BACKING ALGORITHM AND CIRCULAR LIST JOIN FOR MAXIMIZING STREAMING DATA JOIN

1MS. G. HEMALATHA, 2DR. K. THANUSHKODI

[1]Assistant Professor (SG), CSE Dept, Karunya University, Coimbatore, India, 0422-2614020
hema_latha207@yahoo.com
[2]Principal, Akshaya College of Engineering, Coimbatore, India,
thanush12@gmail.com

ABSTRACT

An increasing number of database queries are executed by interactive users and applications. Since the user is waiting for the database to respond with an answer, the initial response time of producing the first results is very important. The user can process the first results while the database system efficiently completes the entire query. The state-of-art join algorithms are not ideal for this setting. 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. The main advantage of adaptive 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. Since the response time of the queries places a vital role in adaptive join, the join techniques like Hash Join, Sort Merge Join cannot be used because they require some prework before producing the join result. The only possible join technique that can be used in adaptive join is Nested Loop Join. In Nested Loop Join each row of the outer relation is compared with each row of the inner relation. The no. of comparisons done by the nested loop join can be reduced by using a technique called trace backing. In trace backing technique whenever a miss match occurs, the next tuple of the outer relation is compared with the mismatched inner relation tuple, instead of looping all the tuples of the inner relation. Finally a new circular linked list join method is discussed which may be a better option to perform streaming data Join.

Keywords:

Adaptive *Join, Trace Back technique, Nested Loop Join*

1. INTRODUCTION

Modern information processing is moving into a realm where we often need to process data that is pushed or pulled from autonomous data sources through heterogeneous networks. Adaptive query processing has emerged as an answer to the problems that arise because of the fluidity and unpredictability of data arrivals in such environments [1]. An important line of research in adaptive query processing has been towards developing join algorithms that can produce tuples “online”, from streaming, partially available input relations, or while waiting for one or more inputs [4], [7], [9], [12], [14], [15]. Such non-blocking join behavior can improve 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, on-line aggregation and approximate query answering systems. Compared to traditional join algorithms (sort-, hash or nested loops-based [13]), adaptive joins are designed to deal with some additional challenges: The input relations they use are provided by external network sources. The implication is that one has little or no control over the order or rate of arrival of tuples. Since the data source reply speed, streaming rate and streaming order, as well as network traffic and congestion levels, are unpredictable, traditional join algorithms are often unsuitable or inefficient. For example, most traditional join algorithms cannot produce results until at least one of the relations is completely available.

Waiting for one relation to arrive completely to produce results is often unacceptable. Moreover, and more importantly, in emerging data integration or online aggregation environments, a key performance metric is rapid availability of first results and a continuous rate of tuple production. 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. All existing algorithms work in three stages. During the Arriving phase, a newly arrived tuple is stored in memory and it is matched against memory-resident tuples belonging to the other relations participating in the join.

Since the allocated memory for the join operation is limited and often much smaller than the volume of the incoming data, this results in tuple migration to disk. The decision on what to flush to disk influences significantly the number of results produced during the Arriving phase. The Arriving phase is suspended when all data sources are temporarily blocked and a Reactive phase kicks in and starts joining part of the tuples that have been flushed to disk. An importance of this phase is the prompt handover to the Arriving phase as soon as any of the data sources restarts sending tuples. Each algorithm has a handover delay which depends on the minimum unit of work that needs to be completed before switching phases.

This delay has not received attention in the past, but we show that it can easily lead to input buffer overflow, lost tuples and hence incorrect results. When all sources complete the data transmission, a Cleanup phase is activated and the tuples that were not joined in the previous phases (due to flushing of tuples to disk) are brought from disk and joined. There are two new adaptive join algorithms for output rate maximization in data processing over autonomous distributed sources. The first algorithm, Double Index NEsted-loop Reactive join (DINER) is applicable for two inputs, while Multiple Index NEsted-loop Reactive join (MINER) can be used for joining an arbitrary number of input sources.

DINER follows the same overall pattern of execution discussed earlier but incorporates a series of novel techniques and ideas that make it faster, leaner (in memory use) and more adaptive than its predecessors. More specifically, 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 as show in Fig 1.

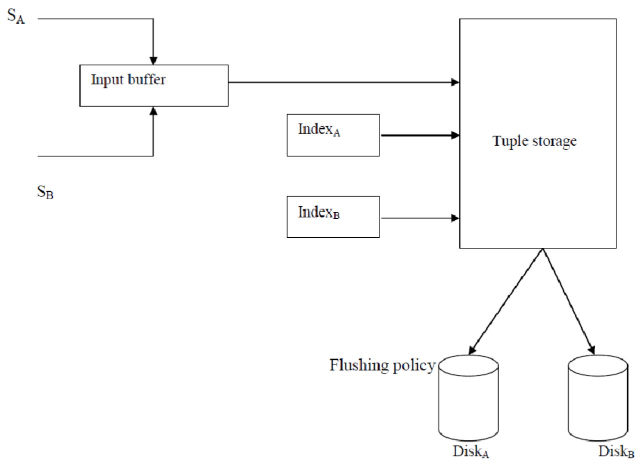


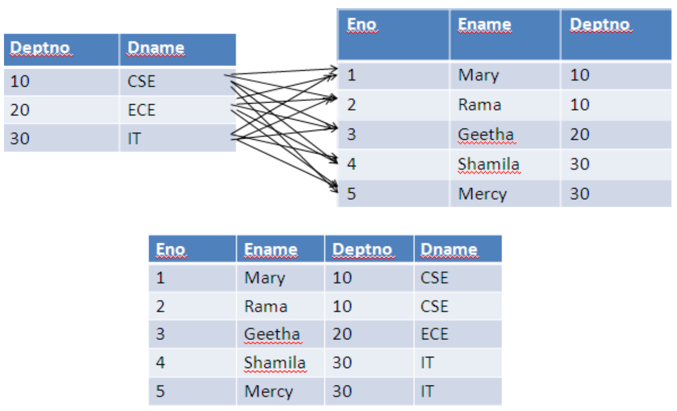
Fig 1.Arriving Phase

The 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. The efficiency of the DINER algorithm can still be improved by replacing the Nested loop algorithm with a better join technique. There are many Join techniques which out performs the Nested Loop Join like Hash Join, Sort Merge Join, XJoin, Double Hash Join etc., But all these join techniques require some pre-work to produce the resulting Join tuples. As we know in adaptive environment time is a vital factor and it cannot be spend to do the pre-work. So the only possible Join technique that can be used is Nested Loop Join.

2. Our Contribution

In Nested loop Join each row of the outer relation is compared with each row of the inner relation which in turns results in a Cartesian product as show in Fig 2. The other joins algorithms out performs the nested loop join but still cannot be used in the steaming join. Because in all the other join algorithms some amount of preprocessing has to be done, which not possible in adaptive environment as the arrive of input tuples cannot be predicted and as soon as it is arrived it has to be join before the other set of input tuples are ready. So the only possible join technique to be used in adaptive environment is Nested Loop join.

In this paper we proposed a trace backing algorithm to be used instead of nested loop join to maximize the no. of resultant tuples this in turn increased the resultant tuples produced during a time period. The proposed Trace backing algorithm [as show in the Fig 3] compares the outer row with the inner row, whenever there is a mismatch the outer row is incremented instead of inner row, the inner is incremented when there is match. This reduced the number of comparisons than it is required for Nested Loop Join.

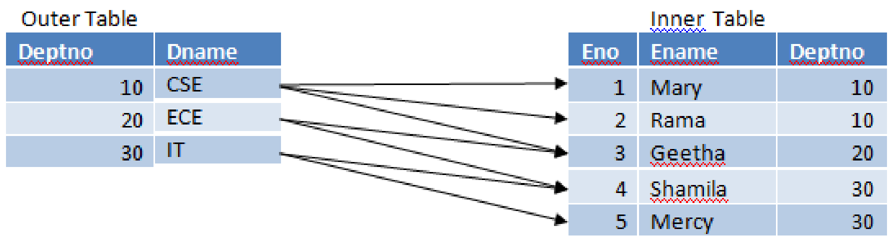


Deptno	Dname
10	CSE
20	ECE
30	IT

Eno	Ename	Deptno
1	Mary	10
2	Rama	10
3	Geetha	20
4	Shamila	30
5	Mercy	30

Eno	Ename	Deptno	Dname
1	Mary	10	CSE
2	Rama	10	CSE
3	Geetha	20	ECE
4	Shamila	30	IT
5	Mercy	30	IT

Fig 2 Draw Back of Nested Loop Join



Deptno	Dname
10	CSE
20	ECE
30	IT

Eno	Ename	Deptno
1	Mary	10
2	Rama	10
3	Geetha	20
4	Shamila	30
5	Mercy	30

Fig 3 Trace backing Technique

3. ALGORITHM FOR TRACE BACKING TECHNIQUE

Let RA and RB be the two relations to be joined, the smallest relation is taken as the outer relation and the other is taken as the inner relation. The first row of the outer relation is compared with first row of the inner relation, if there is a match increment the inner pointer and compare the first row of the outer relation and the second row of the inner relation else increment the outer pointer and compare the second row of the outer relation with the first row of the inner relation. This process is repeated until all the inner relation tuples are matched.

Input: Tuples from relations RA and RB
Output: Joined tuples, RA ⋈ RB

```
Set Ro ← smallest(RA,RB)
Set Ri ← largest(RA,RB)
PR=0;                              //      rightpointer
Pl=0;                              //      leftpointer
//n is the number of tuples in the relation
While PR <n(Ri)  do
While Pl < n(Ro) do
 // compare the tuples from two relations
 If (tRo[Pl]== tRi[PR]) then
    tRi[PR] ⋈ tRo[Pl]
    PR++;
 Else
    Pl++;
End if
End while
if (Pl == n(Ro)) then //check leftpointer status
leftpointer=0;
if(PR== n(Ri)) then //check rightpointer position
break;
End while
```

5. Performance Evaluation

The following graph shows the no. of comparisons required to perform the Nested Loop Join and Trace Back technique. The no. of comparison is greatly reduced in Trace Back technique. In the worst case scenario the Trace Back technique behaves similar to Nested Loop Join. In adaptive Join environment the worst case scenario occurs very rarely.

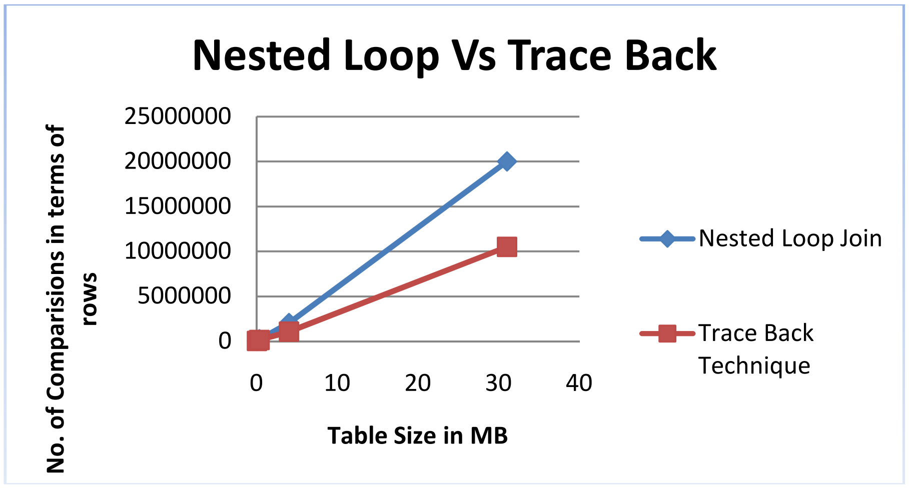


Fig 4. Comparison between Nested Loop and Trace Back Join Technique

The performance of this algorithm is evaluated on the basis of throughput [the number of joined tuples per unit time] and compared with DINER. By using the trace backing algorithm the number of joined tuples is increased. For example, in 15millisecond, DINER produced 72 joined tuples, where as trace backing technique produces 108 joined tuples.

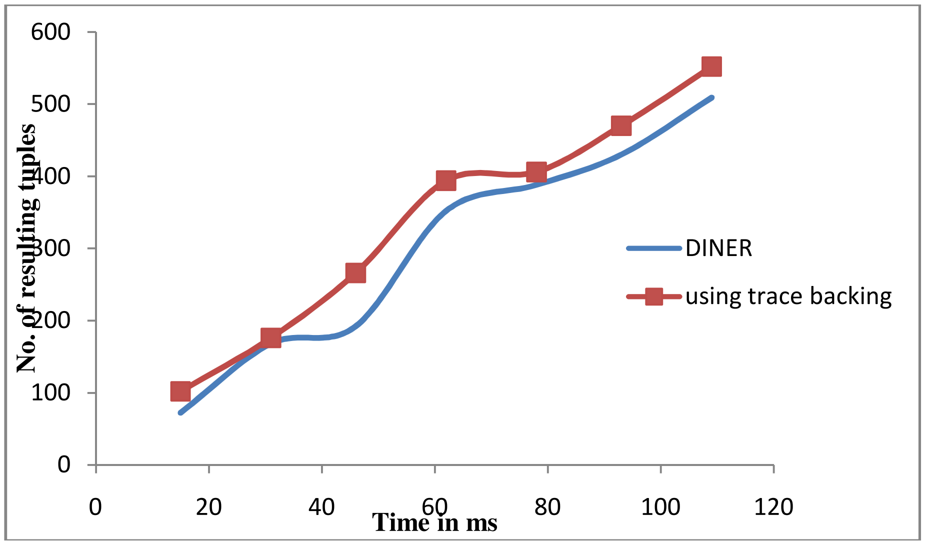


Fig 5 Performance Evaluations for Double Inputs

6. FURTHER SCOPE

In this Join type, table with the less number of rows are identified as the outer table and other table is identified as inner table. All the rows of the outer table is at a time compared with the rows of the inner table. This comparison is done in parallel. During this comparison the matching rows are removed from the inner table and are added to the result set. When the matching rows are removed from the outer table the other rows are pushed to the empty space as in the case of a queue. When a row in the first location did not find a match, it is again added into the rear part of the table as shown below.

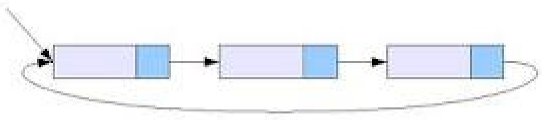
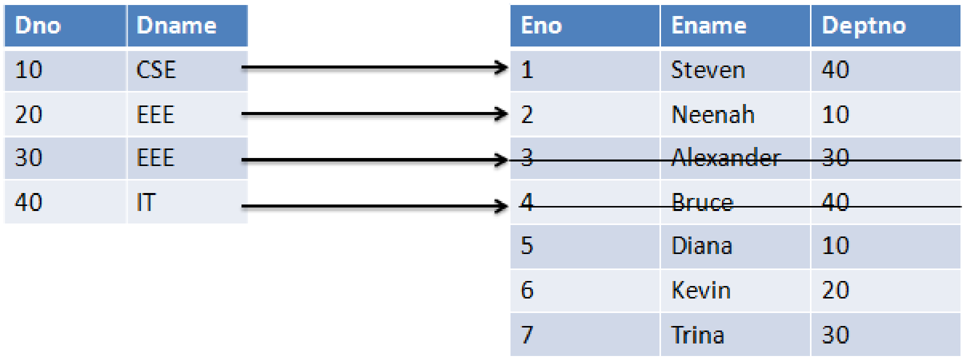


Fig 6 Records added to the table in the form of Queue

This type of join can be used in steaming join where the tuples are continues arriving at some rate. As the tuples arrives it is added to the rear end of the tables. The unmatched rows as it is moved to the front of the table are again inserted into the rear end of the table to find a match with the outer table. The following figure 7-10 explains this Join technique.

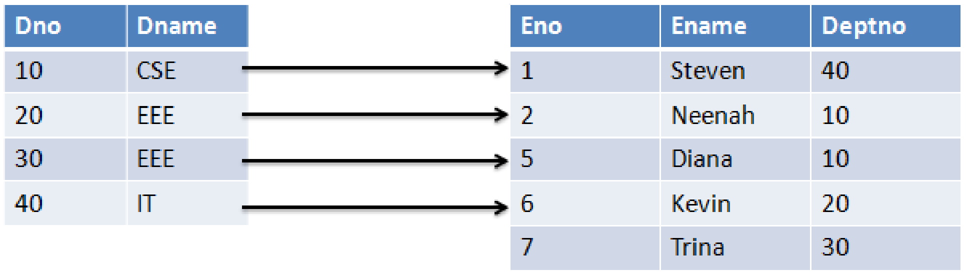
The comparison of outer table with the inner table is done in parallel, this helps to find out the matching row in less time. The number of comparison is less when compared to that of the nested loop join. Since the comparisons are done in parallel, the cost of the join may be little high but it can be compromised as the results are produced much faster than the traditional method.



Dno	Dname
10	CSE
20	EEE
30	EEE
40	IT

Eno	Ename	Deptno
1	Steven	40
2	Neenah	10
~~3~~	~~Alexander~~	~~30~~
~~4~~	~~Bruce~~	~~40~~
5	Diana	10
6	Kevin	20
7	Trina	30

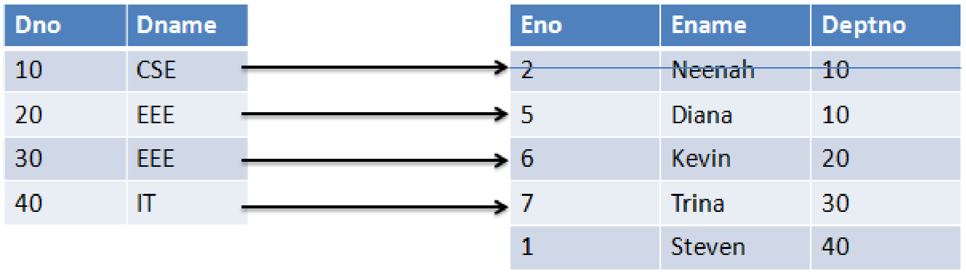
Fig 7. Only two matching rows are moved to the resulting set.



Dno	Dname
10	CSE
20	EEE
30	EEE
40	IT

Eno	Ename	Deptno
1	Steven	40
2	Neenah	10
5	Diana	10
6	Kevin	20
7	Trina	30

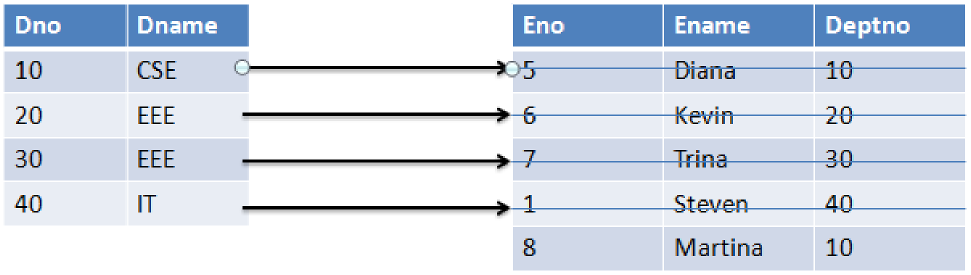
Fig 8. No rows are matched, the first row of the inner table moved to the rear end



Dno	Dname
10	CSE
20	EEE
30	EEE
40	IT

Eno	Ename	Deptno
~~2~~	~~Neenah~~	~~10~~
5	Diana	10
6	Kevin	20
7	Trina	30
1	Steven	40

Fig 9. One matching row is moved to the result set



Dno	Dname
10	CSE
20	EEE
30	EEE
40	IT

Eno	Ename	Deptno
~~5~~	~~Diana~~	~~10~~
~~6~~	~~Kevin~~	~~20~~
~~7~~	~~Trina~~	~~30~~
~~1~~	~~Steven~~	~~40~~
8	Martina	10

Fig 10. Four rows are moved to the result set and one new incoming tuple is moved to the rear end.

CONCLUSION AND FUTURE SCOPE

The main advantage of adaptive join techniques is that they can start producing join results as soon as the first input tuples are available. The only Join technique that can produce the output without any preprocessing is Nested Loop Join. In Nested Loop Join each row of the outer relations are compared with the inner relation, which is same as Cartesian Join. The new technique trace backing reduced the number of comparisons when it is compared with nested loop join. By using this technique, the number of comparison are reduced which in turn increased the number of resultant tuples produced per unit of time. The trace backing technique produced 102 tuples and DINER produced 72 tuples within 15 millisecond.

In terms of future work, the flushing policy can be extended by considering more factors to specify which tuples have to be flushed. The flushing policy has an important role in improving the performance of the join operation.

REFERENCES

[1] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein ,UC Berkele "Online Aggregation and Continuous Query support in MapReduce", In Proceedings of VLDB, 2004

[2] Rahman, Nurazzah Abd Saad, Tareq Salahi. "Early Hash Join: A Configurable Algorithm for the Efficient and Early Production of Join Results", ITSim 2008.

[3] Ron Avnur and Joseph M. Hellerstein "Eddies: Continuously Adaptive Query Processing", Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, volume 29, pages 261-272. ACM, 2000.

[4] S. Chen, P. B. Gibbons, T. C. Mowry, and G. Valentin. "Fractal prefeching B+trees: Optimizing both cache and disk performance", In ACM SIGMOD International Conference on the Management of Data, June 2002.

[5] W. Hong and M. Stonebraker. "Optimization of Parallel Query Execution Plans in XPRS",Distributed and Parallel Databases, 1(1):9-32, 1993.

[6] Bornea A.Mihaela,Vassalos Vasilis, Kotidis Yannis and Deligiannakis Antonios;"Adaptive Join Operators for Result Rate Optimization on Streaming Inputs", IEEE transactions on knowledge and data engineering, volume 22,No.8, pp.1110-1125, August 2010

[7] T. Urhan and M. J. Franklin." XJoin: A Reactively-Scheduled Pipelined Join Operator", IEEE transactions on knowledge and data engineering, volume 23, No.2, April 2000.

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

[9] M. F. Mokbel, M. Lu, and W. G. Aref,"Hash-Merge Join: A Nonblocking Join Algorithm for Producing Fast and Early Join Results", ICDE, 2004.

[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.

[11] J. M. Hellerstein, P. J. Haas, and H. J. Wang. "Online Aggregation", In SIGMOD, pages 171–182, 1997.

[12] P. J. Haas and J. M. Hellerstein. "Symmetric Hash Joins", In Proceedings of ACM SIGMOD, 1999.

[13] S. D. Viglas, J. F. Naughton, and J. Burger," Maximizing the output rate of multi-way join queries over streaming information sources", VLDB '2003, pages 285–296.

[14] Shivnath Babu, Utkarsh Srivastava, and Jennifer Widom "Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries Over Data Streams " In Proceedings of the 2002 Annual ACM Symposium on Theory of Computing. ACM Press, NewYork, NY, pages 370–379.

[15] J. Gehrke. "Special issue on data stream processing." IEEE Computer Society Bulletin of the Technical Communication on Data Engineering, volume 26,No.1, Mar. 2003.

[16] A. Arasu, S. Babu, and J. Widom. "The cql continuous query language: Semantic foundations and query execution", Technical report, Stanford University Database Group, Oct. 2003.