A Survey on Modern Join operators:: MINER and DINER

[1]KAMPELLI NARESH, [2]V.NAVATHA, [3]B.VEERA PRATHAP

1. M.Tech-(SE) Pursuing, 2. M.Tech-(SE) Pursuing ,

3. HOD, Dept of CSE, MOTHER THERESSA COLLEGE OF ENGINEERING & TECHNOLOGY

Abstract

Adaptive join algorithms were replaces traditional join operators in data processing environment. These modern join operators perform the join processing streaming data continuously. The intent of these algorithms are to start producing the first output tuples as soon as possible next produce the remaining results at a fast rate. One of the early adaptive join algorithm Multiple Index Nested-loop Reactive join (MINER) is a multi-way join operator used for joining an arbitrary number of input sources. Next the Double Index NEsted-loops Reactive join (DINER), a new adaptive two-way join algorithm for result rate maximization It will improve producing result tuples at a significantly higher rate, while making better use of the available memory.

Key Terms: Query processing, join, DINER, MINER, streams.

Introduction

Join operation is considered as one of the fundamental operations of relational databases and it is also difficult operation to efficiently implement. Joins are one of the basic constructions of SQL and databases such as, they combine records from two or more database tables into one row source, one set of rows with the same columns and these columns can originate from either of the joined tables as well as be formed using an expressions and built-in or user-defined functions.

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 project, we 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 reentrant 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

We 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 multiway join operator that inherits its principles from DINER. Our experiments using real and synthetic data sets demonstrate that DINER outperforms previous adaptive join algorithms in producing result tuples at a significantly higher rate, while making better use of the available memory. Our experiments also shows that in the presence of multiple inputs, MINER manages to produce a high percentage of early results

Existing System

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 important desideratum 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. Even if the overall strategy has been proposed for a multiway join, most existing algorithms are limited to a two-way join. Devising an effective multiway adaptive join operator is a challenge in which little progress has been mad.

Existing Join Techniques: The main three categories of join algorithms are

a) Nested-loop join algorithm

b) Sort-merge join algorithm

c) Hash-based join algorithm

Proposed System

we propose 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 (Double Index NEsted-loops Reactive) 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 nonoverlapping 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

MINER

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. Weare able to generalize the reactive phase of DINER for multiple inputs using a novel scheme that dynamically changes the roles between relations

Execution of Join Statements:- To choose an execution plan for a join statement, the optimizer must make these interrelated decisions:

a. Access Paths: - As for simple statements, the optimizer must choose an access path to retrieve data from each table in the join statement.

b. Join Method: - To join each pair of row sources, any database must perform a join operation. Join methods include nested loop, sort merge and hash joins.

c. Join Order: - To execute a statement that joins more than two tables, SQL joins two of the tables and then joins the resulting row source to the next table. This process is continued until all tables are joined into the result.

Chooses Execution Plans for Joins:- The query optimizer considers the following when choosing an execution plan:

a. The optimizer first determines whether joining two or more tables definitely results in a row source containing at most one row. The optimizer recognizes such situations based on Unique and Primary Key constraints on the tables. If such a situation exists, then the optimizer places these tables first in the join order. The optimizer then optimizes the join of the remaining set of tables.

b. For join statements with outer join conditions, the table with the outer join operator must come after the other table in the condition in the join order. The optimizer does not consider join orders that violate this rule. With the help of query optimizer, the optimizer generates a set of execution plans, according to possible join orders, join methods and available access paths. The optimizer then estimates the cost of each plan and chooses the one with the lowest cost.

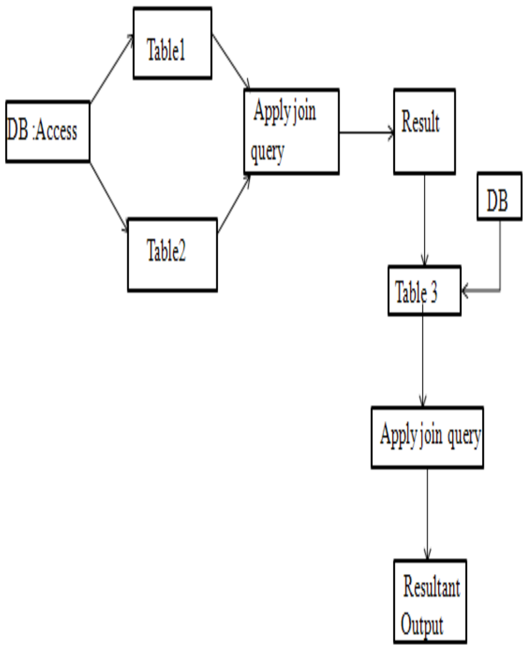

Figure 1. The General Flow Diagram of MJoin

a. Get the three tables from single database

b. Apply join query on A and B

c. If (A join with B)

d. Then (result is stored in the First_ join table)

e. This First_Join table is stored in the database

f. After that this First_Join table join with the third table

g. The result will be display

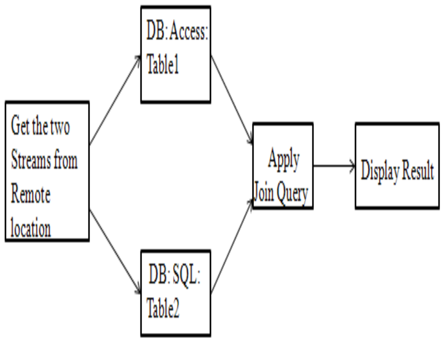

Figure 2. The General Flow Diagram of DINER

a. Get the two tables from two different databases say, A and B

b. If one relation is completely available in the buffer.

c. Then (A join with B)

d. Else

e. Wait for full relation

f. if (memory-resident tuples exist)

g. tries to join according to their matching

h. Comparing between two tables.

i. if not (flush to disk when memory becomes overflow)

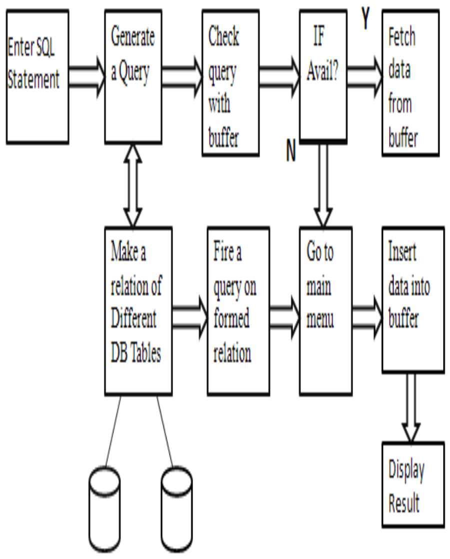

Figure 3. The General flow Diagram of MINER

a. Consider two tables A and B

b. Table A has two columns and table B has two columns

c. Table A and B contains two rows each.

d. If (query is available in buffer)

a. Then

e. Fetch index no. of that query which is already exist in the buffer

f. Display the result

a. Else

g. Go to Main menu

h. Put the (data/query) into Buffer

i. Display the Result and

j. Show the Resultant Time

CONCLUSION

In this work, we have successfully implemented three algorithms. The three algorithms are MJoin, DINER and MINER. In MJoin, uses single database. During first phase of MJoin, Each new tuple is coming from one location then calculate its start time and when tuple is stored in buffer, calculate its End time and calculated difference (start time- end times) for MJoin is 3.174 ms.

In DINER, uses heterogeneous database. This algorithm can efficiently handle join predicates with range conditions, a feature unique to this technique. Each new tuple is

coming from remote location. Calculate its start time, when execution started and this tuple is stored in memory then calculate its end time after that we will show the calculated
(output) i.e. difference between the start time and end time for DINER is 5.014 ms. In MINER, uses homogeneous databases. Each new tuple is coming and calculated its start time and end time. Calculates its data streaming rate is 4.375ms. But, these values for all three algorithms may be varying from time to time.

REFERENCES

[1]. J.Jayashree and C.Ranichandra "Join Algorithm for Efficient Query Processing For Large Datasets" Asian Journal of Computer Science and Information Technology 2: 3 (2012) 31 –35

[2]. 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)

[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, Wenny Rahayu, Sushant Goel. (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. Progressive merge join: A generic and non-blocking sort-based join algorithm. In Proceedings of VLDB,2002.

[11]. M. F. Mokbel, M. Lu, and W. G. Aref. Hash-Merge Join: A Non blocking Join Algorithm for Producing.