Optimizing Query Performance using Hash and Sort Merge Join

Siddhi Joshi¹ Jignesh Vania²

¹²Department of Computer Science and Engineering

LJIET

Abstract—This paper introduced a method for producing Hash Merge and Sort Merge Join with extending performance. Hash merge join is non-blocking join algorithm that deals with data items from remote sources via unpredictable, slow, and bursty network traffic. The HMJ algorithm designed with two goals: 1) Minimize the time to produce the first few results, and 2) Produce join results even if the two sources of the join operator occasionally get blocked. Sort Merge Join is conventional join algorithm in the database. It is used when large database is required for query optimizing. I implement hybrid version of both algorithm to increase performance of join. And compare this hybrid version with already implemented algorithm to check performance based on execution time. I conclude that hybrid gives better execution time than hash and sort merge join algorithm after completion of experiment study.

Key words: Hash merge join, Sort merge join, Relational database system

I. INTRODUCTION

Database management system contains information about a particular enterprise. Provide an environment that is efficient and convenient to use. Before few years database application was built in file system. But file system is not efficient. So various optimization techniques are used like hash join and sort merge join.

In fact, these join techniques use options that can be found on the SQL Properties pane by setting queries. However, selecting a join algorithm is important enough to merit a dedicated topic. In this paper, we review the two traditional join algorithms, i.e., Hash join algorithm and sort merge join algorithm respectively and then hybrid both of them, by all kinds of experiments, and then use the Debug property on the SQL Join Properties pane [1] to run these algorithms.

In the rest of the paper, we review related work in section 2, followed by introduction of the hash join algorithm and sort merge join algorithm in section 3. In section 4, we give all kinds of experiments to evaluate the pros and cons of these two algorithms with new hybrid algorithm by design new optimizer. Finally, we conclude the paper and highlight our future work in section 5.

II. RELATED WORK

In relational database systems (RDBMS), join is one of the most fundamental operations, which efficiently retrieves information from two different tables based on a Cartesian product of the two tables [14,7,5,8,6,19,18,3,4,9]. Meanwhile, join is also one of the most difficult operations to implement efficiently in RDBMS, because in most cases there is no predefined association between tables that can be utilized to facilitate the join processing. Depending on different math operators used in the join condition, there are various types of joins in database systems, including equi-join, natural join, semijoin, outerjoin, and self-join. To process the different joins, there are mainly three sorts of algorithms proposed, namely, nested loop join, sort merge join, and hash join, etc [14,7,5,8,6,9,15,21,19,18,3,2,11,4]. These join algorithms are categorized based on how they partition the tuples from different tables.

Nested loop join is the most straightforward method to process joins [14,6,4]. Specifically, one of the tables being joined is designated as inner relation, and the other is the outer relation. Then for each tuple of the outer relation, all tuples of the inner relation are fetched from disk and compared with the tuple from the outer relation. Whenever the predefined join condition is satisfied, the two tuples are concatenated and output as a result. Given two tables R of size |R| and S of size |S|, the time complexity of nested loop join is in the order of O(|R|+|S|), which is inefficient when R and S are large. The block-oriented implementation of nested loop join, i.e., block nested join, tries to optimize I/O cost by choosing the table with larger cardinality to be the inner relation and the table with smaller cardinality to be the outer relation.

Compared to nested loop join, sort merge join is a more efficient technique which takes advantage of sorted tuples in the tables [7,5]. Specifically, sort merge join consists of two stages. In the first stage, both tables to be joined are sorted on the join attributes. Then, both tables are scanned in the order of the join attributes, and tuples meeting the join condition are concatenated to form a result tuple [14,7]. Sort merge join is superior to nested loop join, in that in sort merge join each table is scanned through only once, because the tuples in the tables are sorted and whenever a tuple in Block Nested Join and Sort Merge Join Algorithms: An Empirical Evaluation 707 the inner relation does not satisfy the join condition, we need not to examine the rest tuples in the inner relation any more. In terms of time complexity, sort merge join incurs O(nlogn) time, due to the fact that its running time mainly depends on the sorting time. If the two tables are presorted, or the join attributes are indexed, then sort merge join will incur much less computation time. The main idea of hash join method is that we use some predefined hash function to map all the tuples of one of the tables into a collection of buckets [14,6,15,3]. And tuples mapped to a same bucket have the same hash value on their join attributes. We then scan through the tuples in the other table, using the same hash function to find the bucket that the tuple is mapping to. If the bucket is not empty, we concatenate the tuple with each of the tuples in the bucket, and output the result. Otherwise, we discard the tuple and continue probing the next tuple.

The hash join method is one of the most efficient algorithm for join processing, because the hash computation is fast and we only scan each of the tables once, i.e., with a time complexity of O(|R|+|S|). However, the main drawback of hash join algorithm is that it is suitable for equi-join processing, not for non-equi-join processing [14, 7, 5].
III. PROBLEM DEFINITION

As we seen in current scenario as far as application development is concerned is go smoothly. But in large application where data is large so in that application retrieving is the big issues. So database schema development and proper query optimization plans are required to overcome these issues.

A. Hybrid Join

Hybrid join combines the features of both algorithm hash join and sort merge join, doing both hashing and sorting. On the first pass, instead of using memory as a buffer, only as many blocks as are necessary to sorting R that fits in memory. The rest of memory is used for hash table that is processed at the same time that R and S are being sorting.

Let 1) There will be B + 1 steps in the hybrid join algorithm. First, choose a hash function h and a partition of its hash values which will partition R into Ro, , RB, such that a hash table for R. has 1 M I - B blocks, and R, , RB are of equal size. Then allocate B blocks in memory to B output buffers. Assign the other I M I - B blocks of memory to a hash table for Ro.

2) Assign the ith output buffer block to Ri for i = 1, , B. Scan R. Hash each tuple with h. If it belongs to R. it will be placed in memory in a hash table. Otherwise it belongs to R; for some i > 0, so move it to the ith output buffer block. When this step has finished, we have a hash table for R. in memory, and RL, , Rs are on disk.

3) The partition of R corresponds to a partition of S compatible with h, into sets S,,, , S,. Assign the ith output buffer block to Si for i = 1, , B. Scan S, hashing each tuple with h. If the tuple is in So, probe the hash table in memory for a match. If there is a match, output the result tuple, otherwise drop the tuple. If the tuple is not in S,, it belongs to Si for some i > 0, so move it to the ith output buffer block. Now RL, , RB and S,, , S, are on disk.

4) Read R; and build a hash table for it in memory.

5) Scan Si, hashing each tuple, and probing the hash table for Ri, which is in memory. If there is a match, output the result tuple, otherwise toss the S tuple.

IV. EXPERIMENT STUDY

We design different experiments to compare the Three algorithms implemented in this thesis, such as, Hash merge join(referred to as hash in this part), sort merge join algorithm (referred to as sort) and Hybrid join (referred to as hybrid), in term of the result of the join algorithm and execution time under the different circumstances.

Details of Datasets

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>No of Tuples</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>India_code</td>
<td>100000</td>
<td>3</td>
</tr>
<tr>
<td>Net_table</td>
<td>100000</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Dataset used in experiments

Query1: Select * from India_code i, Net_table n where i.cifnetid=n.cifnetid;

Query2: Select * from India_code i, Net_table n where i.cifnetid=n.cifnetid and rowno<1000;

Query3: Select * from India_code i, Net_table n where

V. CONCLUSION

The overall goal of query optimization process is to choose best plan of execution among many plans. This thesis introduces non-blocking and sort joins to increase throughput of optimization process. First I implement hash merge join and sort merge join algorithm of query optimization. Then I Implement hybrid join using both algorithm hash merge join and sort merge join. And compare execution time of hash merge join, sort merge join...
and hybrid join. After comparison I got efficient method for query optimization that is hybrid join.
Hybrid join is powerful than hash merge join and sort merge join in following things.
1) Execution time of hybrid join is much less than both algorithms.
2) I/O cost is also less than both algorithms.
3) We got optimal solution than both algorithms.
In future hybrid join will be implement using map/reduce framework to achieve better performance.

REFERENCES