Advanced Hash Join Scheme With Dynamic Hashing

KangMok Kim
Information and Communcation Engineering
Yenugnam University
Gyeungsan, Gyeungbuk, South Korea
seorihwa32@ynu.ac.kr

Gyu Sang Choi
Information and Communication Engineering
Yeungnam University
Gyeungsan, Gyeungbuk, South Korea
castchoi@ynu.ac.kr

Abstract—**Grace Hash Join and Hybrid Hash Join are the methods that avoid full-table scanning through partitioning stage using a hash function. But these methods are spent for much time on merge stage because foreign keys are not sorted so these methods scan tables until last bucket. In this paper, we try to solve the given problem with the help of Dynamic Hashing method. Our suggested method is that we divide according to MSB (Most Significant Bit) when we configure hash directory of the table. If bucket overflow occurs or bucket is full we split the bucket.**

Keywords-component; Database, Join, Hybrid Hash Join, Grace Hash Join, Dynamic Hashing

I. INTRODUCTION

In use huge rather massive Database Management System, the join algorithms are used for efficient data processing. Join algorithms are important algorithms while processing massive data and have a feature that update data on memory repeatedly [1,2,3]. Among them, Hash Join is one that uploads hash table in memory based on one of the two given tables and extract matching data when we scan another table and compare uploaded hash table on memory and another table.

In this paper, we consider the problem of existing Hybrid Hash Join algorithm. And we also explain Dynamic Hashing algorithm expected to solve that problem.

We arrange our paper in the following manner. In Section 2 we explained the techniques Hybrid Hash Join Algorithm and proposed Dynamic Hashing Algorithm while Section 3 describes the conclusion of our work.

II. THE DYNAMIC HASHING SCHEME

A. Motivation

Join operation in Structured Query Language (SQL) produces a new table by combining two or more tables in relational database management systems. The hash-join scheme is one of the implementations of the join operations and it shows high performance while compared to other schemes. In addition, there are several variation of the scheme exits like grace hash-join and hybrid hash-join. In this paper, we will focus on the hybrid hash-join scheme since it will provide higher performance compared to other hash-join schemes.

Here in our work we measured the execution time of hybrid hash-join scheme with two tables, R and S, where the primary key in R table is the foreign key in S table. In this experiment, we generated records for R and S tables by a normal distribution, and the numbers of records in R and S tables are 100K and 1M respectively. It means that the number of records in S table is 10 times more than in R. In addition, we set the record size to 16 bytes and the bucket size to 4 Kbytes.

Figure 1 shows the execution time of hybrid hash-join operation. Whereas the elapsed time of hash is similar to disk I/O time in the first phase, the probing time is the dominant component of total execution time in the second phase compared to building and disk I/O times. This probing operation first selects a specific record in S table and then keeps scanning the hash table of R table by comparing the primary key in R table and the foreign key in S table until finding the corresponding record in the hash table. It means that the probing operation needs to scan the hash table of R table multiple times. Thus, it is very important to minimize the probing time, in order to improve the performance of the hybrid hash-join operation.

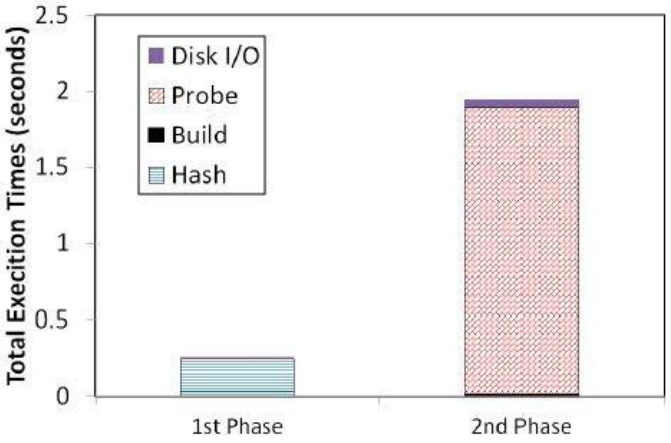

Figure 1. The breakdown execution time of hybrid hash-join operation

B. The Proposed Scheme

Hybrid Hash Join Algorithm also spent to much time on probe stage because a foreign key is not sorted so that method are scanned until table's last bucket. So we suggest Dynamic Hashing Algorithm that can improve performance to solve the

given problem by minimizing unnecessary comparison operations.

Dynamic Hashing method uses a number of buckets dynamically to solve overflow problem when records are inserted frequently. So that algorithm increase or decrease number of the bucket for saved record as occasion demands. So this algorithm is used binary tree structure that used hashing key as an index and we transpose to binary tree dynamically.

First of all, we provide each Identifier to Hash function and generate Hashing key h(k). At this time, Identifier is input parameter of the hash function. After generated h(k), We separate bucket using h(k). At this time, the separate way is that h(k)'s MSB is equal to 0 or 1. When data input to Bucket, If that bucket occurs overflow-Bucket is full- we split that bucket if h(k)'s next significant bit is 0 or 1.

TABLE I. IDENTIFIERS AND BINARY REPRESENTATION

Identifiers	***Binary representation (Output of hash function)***
a0	100 000
a1	100 001
b0	101 000
b1	101 001
c0	110 000
c1	110 001
c2	110 010
c3	110 011

Table I shows an example of Identifiers and binary representation of Dynamic hashing method. The binary representation is the output of the hash function. When we input identifier 'a0' and 'a1' in Dynamic hashing system, the state of the hash structure is same to below Figure 2.

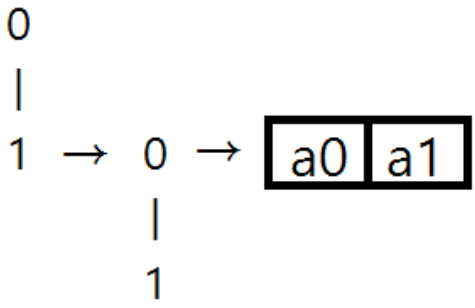

Figure 2. Hash structure when we input identifier 'a0' and 'a1'

Figure 3 shows the state of the hash structure when we input identifier 'b0' and 'b1' to Dynamic Hashing system. When we input identifier 'b0', bit sequence '10' bucket is full so we split this bucket. So we check next significant bit. After check process, bit sequence '101' bucket have 1 empty space.

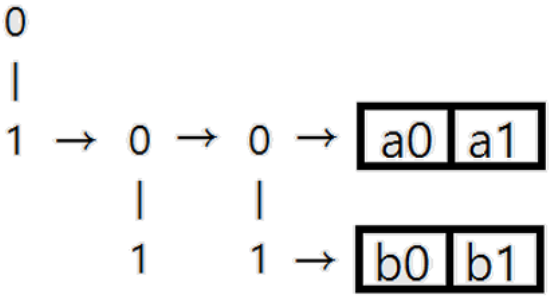

Figure 3. Hash structure when we input identifier 'b0' and 'b1'

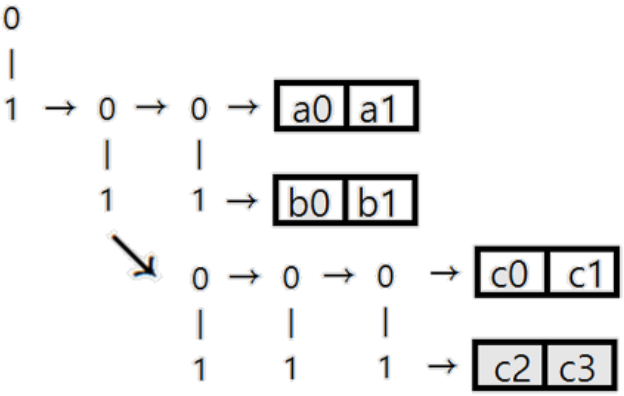

Figure 4. Hash structure when we input identifier 'c0', 'c1', 'c2' and 'c3'

Figure 4 shows the state of the hash structure when we input all of the identifier (a0, a1, b0, b1, c0, c1, c2, and c3) to Dynamic Hashing system. The Dynamic Hashing system has split three times.

We can get an advantage when we use Dynamic Hashing. First of all, we can maintain the efficiency of Data storage space because the hash structure is transform dynamically according to data. And Database performance does not decrease when the number of data is increased. And when their structure is split, they just split each Entry in bucket occurred overflow. The last one is that all of the data is split according to bit sequence so it seems to be sorted.

III. CONCLUSION AND FUTURE WORK

In this paper, we study the problem of Hybrid Hash Join and suggest how to solve that problem using Dynamic Hashing structure. In future work, we try to make Dynamic Hash structure and multi-bits split method. And we compare the performance of Hybrid Hash Join and performance of Dynamic Hash method. We expect Dynamic Hashing method show better performance than Hybrid Hash Join.

After evaluating Dynamic Hash method, we try to sort data on each bucket. If we sort data on the bucket, our algorithm decreases time of proving stage because scanning time is decreased.

REFERENCES

[1] Y. Choi, B. On, G. S. Choi, I. Lee, A Comparative Study of PRAM-based Join Algorithms, Journal of KIISE, Vol.42, No.3, pp.379-389

[2] D. Kang, D. Jung, J.Kang and J. Kim, μ-tree: An ordered index structure for NAND flash memory, Proc. of the 7th ACM/IEEE International Conference on Embedded Software, pp. 144-153, 2007.

[3] H. G. Shin, Y.S. Choi, B.Y. Won, I. G. Lee and G. S. Choi, The Hash Sort Scheme for Database Management Systems, Korea Computer Congress(KCC), Jeju University in Korea, 2015.