A New Join Algorithm

Dong Keun Shin and Arnold Charles Meltzer

Samsung Electronics Co., Ltd.
Communication Systems R&D Center
SongPa P.O.Box 117, Seoul, Korea
email: dkshin@telecom.samsung.co.kr

Department of Electrical Engineering and Computer Science
The School of Engineering and Applied Science
The George Washington University
email: meltzer@seas.gwu.edu

Abstract

This paper introduces a new efficient join algorithm to increase the speed of the join relational operation. Using a divide and conquer strategy, stack oriented filter technique in the new join algorithm filters unwanted tuples as early as possible while none of the currently existing join algorithms takes advantage of any filtering concept. Other join algorithms may carry the unnecessary tuples up to the last moment of join attribute comparisons.

Four join algorithms are described and discussed in this paper: the nested-loop join algorithm, the sort-merge join algorithm, the hash join algorithm, and the new join algorithm.

1 INTRODUCTION

Since The problem of complexity of the time-consuming join operation is the major bottleneck for relational database management systems, many researchers have endeavored to increase the speed of the join. However, this issue remains less than fully explored.

After analyzing the nested-loop and the sort-merge join algorithms, Blasgen and Eswaran [4] concluded that the sort-merge join algorithm would be the choice when no suitable index existed and a nested-loop join algorithm performed acceptably when a suitable index existed. Both nested-loop and sort-merge join algorithms will be described and discussed later in Sections 2 and 4 respectively.

Since the cost of the main memory has been substantially reduced, it is now a well-known fact that the join algorithm based on hashing is more advantageous than nested-loop or sort-merge join methods: a fact that has been noted by DeWitt and Gerber [8].

In Section 2, the previous known three major join methods are briefly summarized. As a new solution for the join, Stack oriented filter technique used in Shin's join algorithm will be presented and explained in Section 3, while time complexities are discussed in Section 4. Finally, in Section 5, conclusions are given.

2 THE CURRENTLY EXISTING MAJOR JOIN ALGORITHMS

In this section, a brief overview of the major algorithms published until now for the join operation is given. The approaches for the join are described in three separate subsections.

2.1 The Nested-Loop Join Method

The nested-loop join method is the simplest among the three major algorithms. The two relations involved in the join operation are called the outer relation (or source relation) S and the inner relation (or target relation) T, respectively. Each tuple of the outer relation S is compared with tuples of the inner relation T over one or more join attributes. If the join condition is satisfied, a tuple of S is concatenated with a tuple of T to produce a tuple for the resulting relation R.

2.2 The Sort-Merge Join Method

Each of the source (S) and target (T) relation is retrieved, and their tuples are sorted over one or more join attributes in subsequent phases using one of many sorting algorithms (e.g., n-way merge). After the completion of the sorting operation, the two sorted streams of tuples are merged together. During the merge operation, if a tuple of the source relation S and a tuple of T satisfy the join condition, they are concatenated to form a tuple of the resulting relation R.

2.3 The Hash Join Method

In the straightforward hash join algorithm, the source and target relations, which one may call S and T, respectively, are read. The join attribute values of the source relation are first hashed by a hash function. The hashed values are used to address entries of a hash table called buckets. If the same hash function used for the join attribute value of the target relation is hashed to a non-empty bucket of the hash table, and one of the join attribute values stored in that bucket matches with the value, the equi-join condition is satisfied. The corresponding tuples of the source and target relations are concatenated to form a tuple of the resulting relation. The process continues until all the tuples of the target relation have been processed.

3 SHIN'S ALGORITHM FOR THE JOIN

In Shin's join algorithm, the source and target relations are repeatedly divided (or rehashed) by a maximum of five statistically independent hash functions until a group of source tuples and target tuples are found to have an identical join attribute. Many statistically independent hash functions can be derived from Shin's mapping hash function [14] by choosing a unique set of prime numbers for each mapping hash function. The source and target tuples in the group are merged after the final screening in order to produce resulting tuples.

The stack oriented filter technique (SOFT) is the main idea used in the new join algorithm. The fundamental data structure used in the SOFT is a stack. Each stack item in the SOFT represents a pair of hash tables: one hash table for source tuples and the other hash table for target tuples. In the process of the new join algorithm, a maximum of five pairs of hash tables in the stack may be created. A source hash table in a stack item includes 256 (Any suitable number can be used instead of 256.) bucket pointers for the linked lists of source tuples, and a target hash table in a stack item also includes 256 bucket pointers for the linked lists of target tuples. The number of bucket pointers to the power of the number of pairs of hash tables (e.g., 256**5 in this example) should be sufficiently large. If both input relations are too large to fit into the main memory, they are divided into a maximum of 256 subset files for each relation by the first hash coder, as is shown in step 1 in Figure 1. After the source and target tuples are hashed by the first hash coder, the tuples in the source subset file (Si) can possibly match with only the tuples in the target subset file (Ti). If an empty subset file exists, all tuples in the pair of the subset files would be eliminated since they have no potential to be included in the resulting relation.

As is shown in step 2 in Figure 1, the join attribute values of the source tuples are hashed by the second hash function; as a result, the source tuples are stored in addressed buckets in the source hash table. Using the same hash function, the target tuples are hashed and stored in the target hash table. While the tuples are being divided into a maximum of 256 groups, the first produced hash address is compared with (or subtracted from) subsequently produced hash addresses to detect if the produced hash addresses are the same. If so, the source tuples and target tuples are merged with final screening. After the hashing process, three kinds of pairs of buckets (ij) might be created. The pairs appear in the following combinations:

(1) The source and target buckets (Sij and Tij) are not empty.
(2) The source bucket (Sij) is not empty, but the target bucket (Tij) is empty.
(3) The target bucket (Tij) is not empty, but the source bucket (Sij) is empty.

It is the main concept in the SOFT that

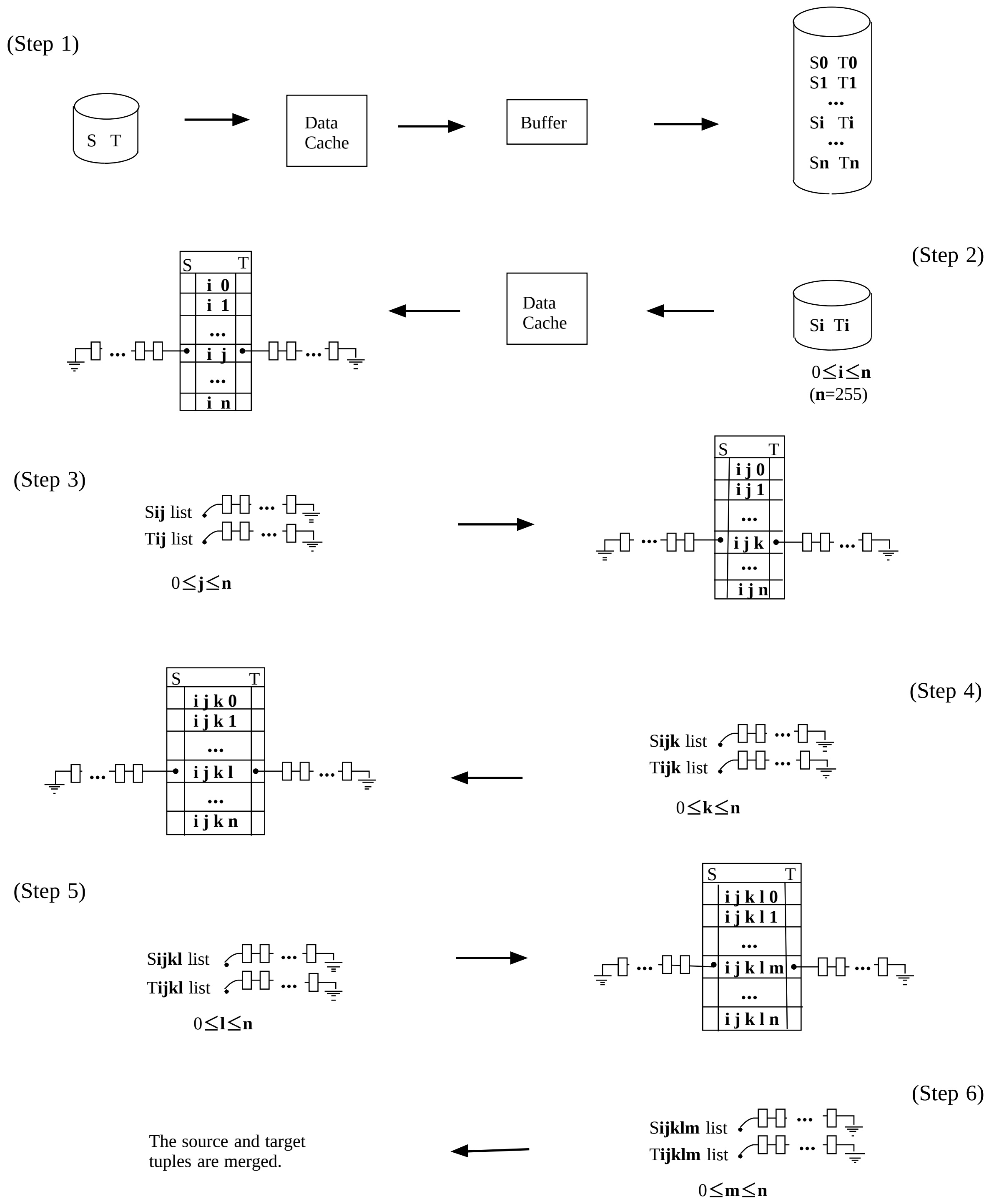

Figure 1. The SOFT in the Shin's Join Algorithm

when one of the two buckets is empty, the tuples in the corresponding bucket are unnecessary and they are filtered out.

The algorithm proceeds from the pair of buckets of address 0 to the pair of buckets of address 255 checking if both source and target buckets are not empty. When both buckets are not empty, the next bucket address is saved and the tuples in the source bucket and the corresponding target bucket will be rehashed (or divided) by the next statistically independent hash function.

During the rehashing process, the algorithm compares the first produced hash address with the others. If all the produced hash addresses are identical (the termination condition for a division process), the tuples are merged with final screening; otherwise, the tuples are divided further by another functionally different hash coder.

Steps 3, 4, and 5 in Figure 1 can be explained similarly. In step 6, no available hash function is left and unnecessary data have been filtered; therefore, the source and target tuples are merged without being rehashed. As far as data structure is concerned, the linked list data structure is better than the array data structure for the buckets in steps 2 through 5 because in these steps most of the buckets may be empty. Therefore, by using the linked lists for the buckets, memory space can be conserved.

As the new join algorithm is shown in Figure 2, push and pop are the names of the procedures operating in the stack of the SOFT: push inserts a pair of source and target hash tables onto the top of the stack, and pop deletes a pair of source and target hash tables from the top. In the SOFT, the stack pointer always points to the current pair of hash tables--the item at the top of the stack--by incrementing its value when pop is called. By referring to the value in the stack pointer, the function Bottom_Of_Stack can tell whether the stack pointer points to the first or lowest pair of hash tables as the current item of the stack.

In the new join algorithm, there are several other frequently used procedures such as Assign_Source_And_Target, No_More_Next_ Bucket_Addr and Save_Next_Bucket_Addr. The module Assign_Source_And_Target uses the header pointers of both source and target linked lists based on the saved next bucket address of the current pair of hash tables in order that the tuples in the linked lists are processed through the filter again. Each next bucket address is incremented and saved to keep track of the subsequent bucket address. Whenever the procedure Assign_Source_And_Target is called, another next bucket address, which has non-empty buckets for both source and target hash tables, is found and saved by the procedure Save_Next_Bucket_Addr. As a result, the procedure push saves the contents of the current pair of hash tables and increments the stack pointer in order that the next upper pair of hash tables becomes the current one or the top of the

```
begin
  Initialization;
  finish := false;
  repeat
    Hash_Source_And_Target_Relations;
    If Only_One_Hash_Address_Produced   then
    begin
      Merge_Tuples_And_Output;
      If No_More_Next_Bucket_Addr   then
      begin
        If Bottom_Of_Stack  then
          finish := true
        else
        begin
          pop;
          if No_More_Next_Bucket_Addr   then
          begin
            if Bottom_Of_Stack  then
              finish := true
            else
            begin
              pop;
              if No_More_Next_Bucket_Addr   then
              begin
                if Bottom_Of_Stack  then
                  finish := true
                else
                begin
                  pop;
                  if No_More_Next_Bucket_Addr   then
                  begin
                    if Bottom_Of_Stack  then
                      finish := true
                    else
                    begin
                      pop;
                      if No_More_Next_Bucket_Addr   then
                      begin
                        if Bottom_Of_Stack  then
                          finish := true
                        else
                        begin
                          Assign_Source_And_Target;
                          Save_Next_Bucket_Addr;
                          push;
                        end;
                      end;
                    end;
                  end;
                  else
                  begin
                    Assign_Source_And_Target;
                    Save_Next_Bucket_Addr;
                    push;
                  end;
                end;
              end
              else
              begin
                Assign_Source_And_Target;
                Save_Next_Bucket_Addr;
                push;
              end;
            end;
          end
          else
          begin
            Assign_Source_And_Target;
            Save_Next_Bucket_Addr;
            push;
          end;
        end;
      end
      else
      begin
        Assign_Source_And_Target;
        Save_Next_Bucket_Addr;
        push;
      end;
    end
    else
    begin
      Assign_Source_And_Target;
      Save_Next_Bucket_Addr;
      push;
    end
  until finish;
end.
```

Figure 2. The Shin’s Join Algorithm

stack.

When pop is called, the stack pointer is decremented in order that the pair of hash tables directly under the current one becomes the current pair of hash tables. After the pop, the boolean function No_More_Next_Bucket_Addr should be called in order to see if there is any saved next bucket address for the current pair of hash tables. If there is none, the current pair of hash tables is checked to see if it is the first or lowest one. If so, the join process is terminated by breaking the repeat loop.

The algorithm shown in Figure 2 is an explanatory version of the main module of a simulation program. One may consider the flow of the new join algorithm as preorder tree (in this example, a 256-ary tree of depth five) traversal.

4 DISCUSSION

When the number of tuples in the source relation (the smaller relation) is S, the number of tuples in the target relation (the larger relation) is T, and the number of tuples in the resulting relation is R, then the time complexity of the nested-loop join algorithm is O(S*T). Considering the upper bound (N) of S and T, we can simplify O(S*T) as O(N*N). the time complexity of the sort-merge algorithm is O((S+T) log (S+T)). Since S+T is the total number of the input tuples (N), the time complexity can be represented as O(N log N). The time complexity of hash-based join algorithms is O(S+T+R). Assuming that R is relatively smaller than S+T, the time complexity becomes O(S+T), which is O(N). It is hard to rely solely on asymptotic time complexity analysis to understand the actual performance due to the number of accesses to the secondary storage and other I/O and communication factors which are not heavily considered in this paper. However, the time complexity analysis will be more reliable in the future when sufficient main memory, effective I/O and communication channels, and necessary hardware will be provided.

Shin’s join algorithm requires a maximum of five hashings for each join attribute to determine whether the associated tuple is necessary or not. Therefore, when the input (S+T) is greater than the output (R) as usual, the time complexity of the algorithm is represented as O(5*(S+T)), which is O(5*N). Then it can be simplified as O(N). Comparing the new join algorithm with others, one can see that none of the currently existing join algorithms effectively takes advantage of any filtering scheme while the new join algorithm filters unwanted data efficiently.

Usually, the performance of hash join method is largely dependent on the ratio of the selected hash table size to the number of input tuples. It is an overhead to calculate a hash table size based on the number of input tuples prior to each hash join operation. The hash table size is fixed in the new join algorithm, so one does not need to calculate a suitable hash table size for each join operation.

5 CONCLUSIONS

Three major join algorithms and a new join algorithm are illustrated and discussed in this paper: the nested-loop algorithm, the sort-merge algorithm, the hash algorithm, and the Shin’s algorithm. The Shin’s join algorithm repeats the division and filtering process many times in a recursive way; therefore, nearly 100 percent of unnecessary tuples are filtered.

No requirement for hash table size calculation can be considered as an important characteristic of the new join algorithm since other hash join algorithms may compute hash table size based on input size for each join operation. The new join algorithm uses not variable size hash tables but fixed size hash tables, so the algorithm requires no preprocessing for hash table size calculation.

This research has produced a new join algorithm to improve the performance of relational database management systems. This new join algorithm will accelerate the join operation, since the new join algorithm goes through frequent filtering processes to discard unwanted tuples efficiently. The more main memory space that is available, the more powerful the new join algorithm will be.

Acknowledgement

We would like to thank Arie Segev, Michael Stonebraker, Domenico Ferrari, Manuel Blum, Ward Maurer, Michael Feldman, Simon Berkovich, Michael Carey, Won Kim, David DeWitt, H. Jagadish, and

Arun Swami for a review of this article and their helpful suggestions.

References

[1] Abd-alla, A. M., and Meltzer, A. C. Principles of Digital Computer Design. Vol. I, Englewood Cliffs: Prentice Hall, 1976.

[2] Babb, E."Implementing a Relational Database by Means of Specialized Hardware." ACM Transactions on Database Systms, Vol. 4, No . 1, Mar. 1979: 1-29.

[3] Bitton, D., et al. "Parallel Algorithms for the Execution of Relational Database Operations. " ACM Transactions on Database Systems, Vol. 8, No. 3, Sep. 1983: 324-53.

[4] Blasgen, M. W., and Eswaran, K. P. "Storage and Access in Relational Data Bases." IBM System Journal, Vol. 16, No. 4, 1977: 363-77.

[5] Boral, H.,and DeWitt,D. J."Processor Allocation Strategies for Multiprocessor Database Machines." ACM Transactions on Database Systems, Vol. 6, No. 2, Jun. 1981: 227-54.

[6] Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." CACM, Vol. 13, No. 6 , Jun. 1970: 377-87.

[7] DeWitt, D. J. "DIRECT-A Multi-processor Organization for Supporting Relational Database Management System." IEEE Transactions on Computers, Vol. C-28, Jun. 1979: 395-406.

[8] DeWitt, D. J., and Gerber, R. "Multiprocessor Hash-Based Join Algorithms." Proceedings of the Eleventh International Conference on Very Large Data Bases, Stockholm, 1985: 151-64.

[9] Goodman, J. R., and Sequin, C. H. "Hypertree: A Multiprocessor Interconnection Topology." IEEE Transactions on Computers,Vol.C-30, No. 12, 1981: 923-33.

[10] Hsiao, D. K. Advanced Database Machine Architecture. Englewood Cliffs: Prentice Hall, 1983.

[11] Pang, H., Carey, M. J., and Miron, L. "Partially Preemptible Hash Joins. " Proceedings of the 1993 ACM SIGMOD, May 1993: 59-68.

[12] Schneider, D. A., and DeWitt, D. J. "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment." Proceedings of the 1989 ACM SIGMOD,Vol. 18, No. 2, Jun. 1989: 110-21.

[13] Shapiro, L. D. "Join Processing in Database Systems with Large Main Memories. "ACM Transactions on Database Systems, Vol. 11, No. 3, Sep. 1986: 239-64.

[14] Shin, D. K. A Comparative Study of Hash Functions for a New Hash-Based Relational Join Algorithm. Pub #91-23423, Ann Arbor: UMI Dissertation Information Service, 1991.

[15] Stonebraker, M. R., et al. "The Design and Implementation of INGRES." ACM Transactions on Database Systems, Vol. 1, No. 3, Sep. 1976: 189-222.

[16] Su, S. Y. W. Database Computers Principles, Architectures, and Techniques. New York: McGraw-Hill, 1988.

[17] Valduriez, P., and Gardarin , G. "Join and Semijoin Algorithms for a Multiprocessor Database Machine." ACM Transactions on Database Systems, Vol. 9, No. 1, Mar. 1984: 133-61.

[18] Hsiao, H. , Chen, M., and Yu, P. "On Parallel Execution of Multiple Pipelined Hash Joins." Proceedings of the 1994 ACM SIGMOD, Vol. 23, No. 2, Jun. 1994: 185-196.