4Q-5

Implementation Technique of Join Operation on KD-Tree Indexed Relations

Lilian HARADA, Masaru KITSUREGAWA, MikioTAKAGI
University of Tokyo

1. Introduction

In [1] we have introduced join strategies for KD-tree indexed relations. We proposed five basic strategies with page fetch and page unload policies to reduce the I/O cost of the join of very large relations indexed by KD-trees. Analytical analysis and simulation results showed that efficient page fetching can be done using the KD-tree index information. Besides, in order to maximize the memory efficiency and to minimize the I/O cost we proposed four extended strategies with garbage collection not at the level of pages, but at the level of chunks of tuples. Introducing this garbage collection mechanism, analytical expressions and simulation results showed that the I/O cost can be minimized to one scan of each relation.

In order to verify and analyze the total cost of the proposed strategies with garbage collection, they are under implementation now. Here, after informally recalling the basic ideas of the join strategies, we present the adopted storage allocation and garbage collection mechanism and show some preliminary results of our implementation.

2. Join Operations on KD-Tree Indexed Relations

Consider the join of two very large relations R and S indexed by KD-trees. The basic idea of our join strategies is to fetch and load a set of pages of relation R (called wave) and based on these pages, determine a range of the join attribute (called join range) in which to perform the join operation. Then, in the remaining space of the memory, fetch all the pages of relation S whose join attribute value are in the join range, in a nested loop way.

Fig. 1(a) illustrates a simple example of the relations R and S clustered by a 2-dimensional KD-tree, the wave of relation R, the join ranges, and the corresponding pages of relation S to be loaded into the main memory and processed in each of the three join steps. Fig. 1(b) shows the corresponding example when the garbage collection at the level of chunks of tuples is introduced. Fig. 2 presents the simulation results of the number of page accesses required to join these very large relations using two of the proposed strategies in [1] (algorithm 3 with garbage collection at page level, and algorithm 3m, at tuple chunck level).

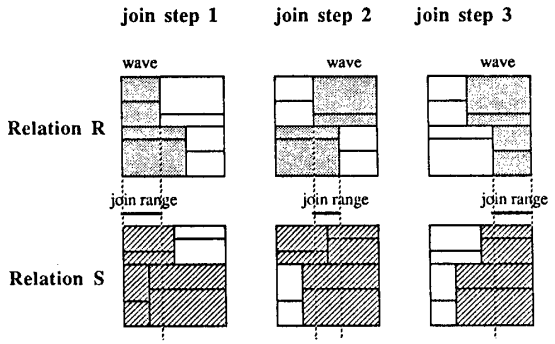

Fig.1(a) Basic Join Strategy
- Garbage Collection at page level

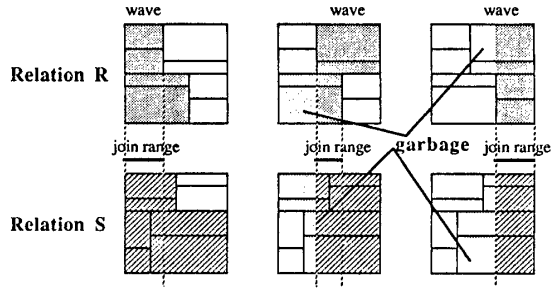

Fig.1(b) Extended Join Strategy
- Garbage Collection at tuple chunk level

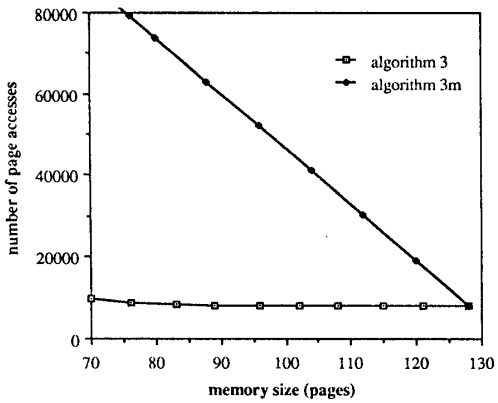

Fig.2 Performance of the Join Strategies

3. Storage Allocation and Garbage Collection on the Extended Join Strategies

3.1 Storage Allocation

As described in section 2, the proposed join strategy consists in fetching and loading the pages of the wave of relation R into the memory, and then fetching the pages of relation S in the remaining space of the memory. The

storage allocation in our implementation has two peculiarities :

(1) the memory is divided into two areas, one for each relation. This is done to avoid segmentation caused by different tuple length of the relations. According to the requirements of each join step, however, the two areas may shrink and grow dynamically.

(2) the tuples of a page have not to be placed contiguously in the memory, i.e., the page is divided into cells of various sizes which are linked. This implies that each page is fetched from the disk into an input buffer and then moved to the memory, divided in varisized cells.

3.2. Garbage Collection

The garbage collection task is to reclaim the unused storage space. In our join strategies, the pages which are fetched from the disk are partitioned in varisized cells and loaded into the memory. In each join step, the tuples which are once processed and will not be necessary any more generate holes in the memory. In each join step the cells shrink.

Thus, the garbage collection comprises two separate phases :

(a) identifying the storage space that may be reclaimed; and

(b) incorporating this reclaimable space into the available memory area.

4. Implementation

4.1. Environment

We are implementing the proposed join strategies on the SUN4-260. The OS utilized is the SUNOS4.0 and the total memory size is 8MB. The implementation is not being done under a real relational database environment and we are using the Unix files as relations. All development is being done using the C language.

4.2. Result

Fig. 3 shows the results of the join of two relations R and S using algorithm 3m [1]. The relations size is 4096 pages, the page size is 2KB and the tuple length is 512B. The effective memory used to load the data pages is varied from 70 pages to 128 pages.

The total time of this join operation is composed of the time to search the KD-tree indexes to determine the pages of the wave of relation R, the join range and the corresponding pages of relation S, the time to read the pages from the disk to an input buffer, the time to move the data from the input buffer to the holes in the main memory, and the time to join the data on memory.

Using algorithm 3m, each relation is almost scanned only once and so the total time is not I/O bound. We can observe that, on average, the join processing time of data on memory represents 48% of the total time, the disk read time 34%, the data move time to fill the holes 13 % and the KD-tree search time 5%.

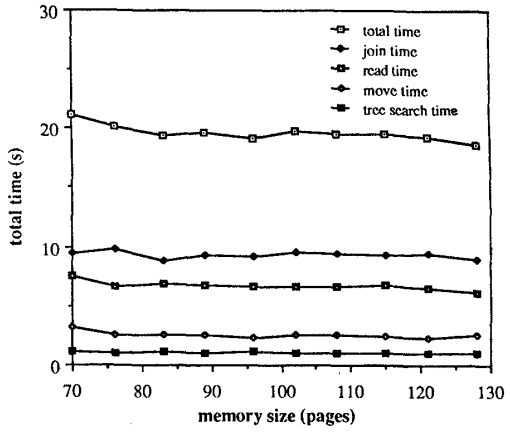

Fig.3 Time Components

4.3. Discussion

We can observe that the total time increases a little for smaller memory sizes. As shown in Fig. 3, the time to join the data on memory is constant and does not vary with the main memory size. However, for smaller memory size, the number of page accesses and consequently the I/O time increases. However, as shown in Fig. 2, the introduction of the garbage collection mechanism drastically reduces the I/O cost of algorithm 3m compared to the algorithm 3.

From Fig. 3 we can observe that the time to search the KD-tree indexes and the time to move the data to fill the holes are smaller than the time to scan the relations. Thus, it shows the effectiveness of the algorithm with garbage collection mechanism to reduce the I/O cost and consequently, the total cost.

5. Conclusion

We presented preliminary results of our implementation of join strategies using garbage collection to join relations with KD-tree indexes. The results showed that the I/O cost is reduced to the minimum and the overhead incurred is small compared to the I/O cost. Now we intend to extend this implementation to a multi-processor environment.

Reference

[1] Kitsuregawa, Harada, Takagi, "Join Strategies on KD-Tree Indexed Relations", IEEE Int. Conf. on Data Engineering, Feb.1989.