Join Strategies on Grid-Files

7 J − 6

Lilian Harada, Masaru Kitsuregawa, Mikio Takagi
University of Tokyo

1. Introduction

Recently, many research on multi-attribute clustered relations have been done . The multi-attribute clustering technique is the one which divides the relation into several pages according to the value of not only the primary-key attribute. It treates all attributes symmetrically. In a previous publication [1], we have introduced join strategies on KD-tree indexed relations. In this paper we present join strategies on Grid-files (Grid-join algorithms) showing that the I/O cost is reduced to the minimum.

2. Grid-Files

Let R be a relation having k attributes $A_1, A_2, ..., A_k$ composed of tuples $t = (a_1, a_2, ..., a_k)$.

D is the base space of relation R and denotes the Cartesian product of the domains of attributes referred to by the relation, i.e.,

$$D = \prod_{i=1}^{k} dom(A_i) = \prod_{i=1}^{k} [MIN_i, MAX_i).$$

The relation is partitioned in pages, which are the unit of access of the relation. The pages are disjoint sets of actual tuples of the relation. Let relation R be partitioned in |R| pages P_j, j = 1, ..., |R|. The space p_j of each page P_j is given by $p_j = \prod_{i=1}^{k} [\alpha_{ij}, \beta_{ij})$ $(\alpha_{ij} < \beta_{ij},\ \alpha_{ij},\ \beta_{ij}\ \varepsilon\ [MIN_i, MAX_i)$ for *all i*). In the Grid-file, the insertion of tuples causes the partitioning of the base space D at prefixed values, generating grid subspaces. The assignment of grid subspaces of the base space of the relation to the data pages P_j is the task of the *grid directory*. A grid directory consists of two parts : first, a dynamic k-dimensional array called *grid array*; its elements (pointers to data pages P_j) are in one-to-one correspondence with the grid subspaces of the partition; and second, k one-dimensional arrays called *linear scales*; each scale defines a partition of a domain of the relation.

3. Cluster, Wave and Join Range

Consider a range $[\chi_i, \delta_i)$, where $\chi_i < \delta_i$ and $\chi_i, \delta_i\ \varepsilon\ [MIN_i, MAX_i)$. We define *cluster of* $[\chi_i, \delta_i)$ ($C_{[\chi_i, \delta_i)}$) as the space given by :

$[MIN_1, MAX_1) \times \cdots \times [\chi_i, \delta_i) \times \cdots \times [MIN_k, MAX_k)$.

Thus the cluster of a range $[\chi_i, \delta_i)$ of an attribute A_i represents the subspace of the relation whose tuples have the attribute value a_i in this range. Fig. 1(a) shows an example for a 2-attribute Grid-file where $dom(A_1)$ = [0,8) and $dom(A_2)$ = [0,8), illustrating a cluster of [2,4) for attribute A_1.

Now we define *wave of* $[\chi_i, \delta_i)$ ($W_{[\chi_i, \delta_i)}$) as the set of pages which contains the cluster $C_{[\chi_i, \delta_i)}$ i.e.,

$\{P_j \mid p_j \cap C_{[\chi_i, \delta_i)} \neq \phi$, for j = 1,...,|R|$\}$.

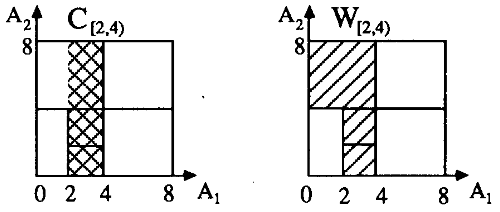

Fig. 1 (a) Cluster (b) Wave

Thus the wave of a range $[\chi_i, \delta_i)$ of an attribute A_i represents the set of pages which can actually contain tuples with the value of attribute A_i in this range. Fig. 1(b) shows the example of wave of [2,4) for attribute A_1.

The wave represents the physical pages that are accessed. The cluster represents a logical space of these pages, which contains the tuples to be actually processed. Both concepts of cluster and wave are based on a given range of the attribute A_i. We call this range as *join range*.

4. Join Processing on Grid-files

In the join of relations R and S, for a given join range, the waves on this range are loaded from the secondary storage to the main memory, and the processing of the clusters on this join range are executed. Fig. 2 shows an example of the join ranges, clusters and waves for relations R and S with attributes A_1 and A_2. Let suppose the join on attribute A_1. After loading the first wave of relations R and S on the memory, the join is applied to the tuples of cluster on [0,2). Then, the next waves of both relations are loaded in the memory and the computing of the cluster on the join range [2,4) is executed. This procedure is repeated until the computing on all the domain of A_1 is finished. The join range determines the pages of the relations to be loaded in the memory and to be processed in each step. Thus, the determination of the join range is fundamental for this join strategy on Grid-files.

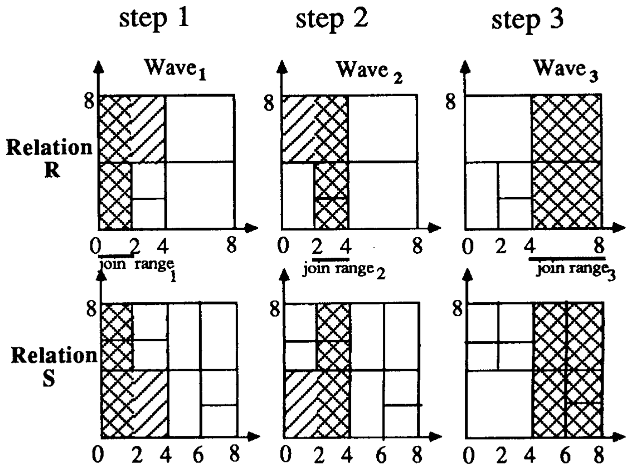

Fig. 2 Join Processing on Grid-Files

5. Basic Grid-Join Algorithm

Here we consider the join of two Grid-files R and S with |R| and |S| pages, respectively, such that |R| ≤ |S|. The available memory size is |M| pages such that |M| << |R|.

5.1. Description

In the basic Grid-join algorithm, the join range is taken from the smaller relation R and is determined by consecutive elements of the linear scale of the attribute A_i being joined. Thus, for example, if the linear scale of A_i of relation R is given by $X(0,1,...,nx)$, the join range of step t is given by:

$[X(t-1), X(t))_t$, such that $X(t-1)_t = X(t-1)_{(t-1)}$, for $t=1,...,nx$.

The idea of this basic Grid-join algorithm is to load the smallest wave of relation R in memory, and to reserve all the remaining memory space to load the correspondent wave of relation S. The example in Fig. 2 shows the join ranges when the linear scale of relation R is given by $X(0,2,4,8)$.

5.2. Performance Evaluation

Fig. 3 shows the number of page accesses for the basic Grid-join algorithm, varying the main memory size in pages. These simulation results are for Grid-files R and S which are clustered on 2 attributes, have 64 Ktuples and random data distribution. The page size is 16 tuples. We can observe that for large memory size, the number of page accesses is the minimum, that is, one scan for each relation. Decreasing the memory size, the curve increases linearly. For a given join range, if the wave of relation R can be entirely loaded in the memory, the number of I/O page accesses of relation S is proportional to the remaining memory size. When the memory size is more reduced, the curve abruptly increases. It is because even the wave of relation R can not entirely fit into the memory, and so, the waves of both relations are read from secondary storage and processed in a nested-loop way.

6. Extended Grid-Join Algorithm

6.1. Description

Here we introduce a garbage collection mechanism which discards the unnecessary tuples as soon as possible. We introduce the garbage collection mechanism in the basic Grid-join algorithm, increasing the effective memory space used for relation S. As exemplified in Fig. 4, with this dynamic garbage collection mechanism, after processing the join range in the first step, the effective memory space for the second join step is enlarged if the dotted portion is discarded as garbage and only the dashed portion is maintained in the memory.

6.2. Performance Evaluation

Fig. 3 also shows the number of page accesses for the extended Grid-join algorithm. As shown, the introduction of the garbage collection mechanism increases the effective memory space so that the number of I/O page accesses is reduced to one scan of each relation in almost all the range of variation of the memory size, which is the lowest I/O bound.

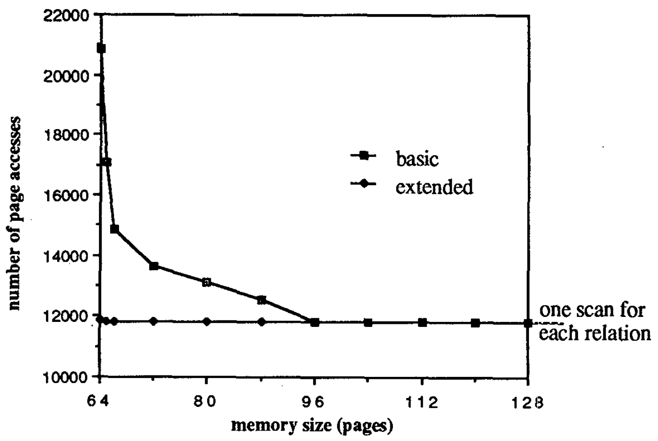

Fig. 3 Simulation Results

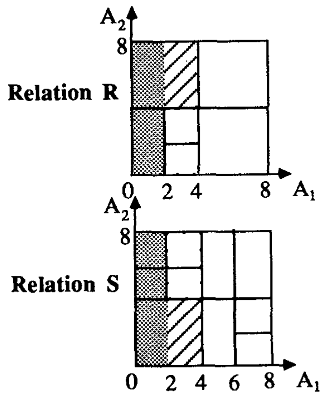

Fig. 4 Garbage Collection Mechanism

7. Conclusion

In a previous paper we have shown the efficiency of the join processing on KD-indexed relations, using our concepts of cluster, wave and join range. In this paper we showed that with the extended Grid-join algorithm, the space of the wave to be processed in each step was always on memory and thus the join could be processed with the lowest I/O bound of one scan per relation.

The selection of a multi-attribute clustering method by the user is application dependent and it is a database design problem. However, the concepts of cluster, wave and join range are general to allow the efficient processing of join queries for all the multiple attributes used for the relation clustering, whichever clustering method is chosen.

[References]

[1] M.Kitsuregawa, L.Harada, M.Takagi,"Join Strategies on KD-Tree Indexed Relations", Proc. of the 5th. Int. Conf. on Data Engineering, pp.85-93, 1989