Regular Paper

P-file: Grid File for Correlated Data Set

KYOJI KAWAGOE*

This paper proposes a new kind of grid file, called a Parallelogram File (P-file), for maintaining multi-dimensional data sets, especially for correlated data sets. In a grid file, which is organized essentially as a set of grid partitions and grid directories, the size of the directory grows exponentially as the number of correlated or non-uniformly distributed data increases. To solve the problem, the paper focuses on the grid shape, which is box-shaped in the original grid file, changing it to a parallelogram in multi-dimensional space, using an affine transformation for a set of grids. The transformation results in a decrease in the grid directory size and the number of bucket accesses required for a range search.

1. Introduction

Multi-dimensional file structures have attracted much interest. Many new database application areas, such as CAD and cartography, involve large amounts of multi-dimensional data. Several multi-dimensional file structures have been proposed that avoid the demerits of one-dimensional indexing for each attribute, so that all keys are combined into a single multi-dimensional index structure.

Research on this area can be categorized into four types:

- 1) Multi-dimensional trees [ROBI81], [BENT70]: Generalizations of single-attribute tree structures, such as B-trees, to N dimensions
- 2) Multi-dimensional extendible hashing [OTOO84], [OTOO86], [WHAN85]: Generaliz of an extendible hashing technique to N dimensions
- 3) Multi-dimensional linear hashing: [OTOO85a], [OTOO85b], [OUSK85]: Generalization of a linear hashing or dynamic hashing technique to N dimensions
- 4) Grid files: [NIEV84], [FREE87], [FALO87]: Representation of data as points in a partitioned N-dimensional space

Of these structures, grid files are the most widely used because of their simple address calculation method and two-disk access strategy. A grid file partitions a K-dimensional data space into a set of grids. Each grid is defined by a K-dimensional array, called a scale, each of whose elements represents a boundary in a dimension. Therefore, in the grid file there are K scales and a K-dimensional array called the grid directory that contains information on mapping between each grid and the corresponding data bucket. The two-disk access strategy here means that either a successful or an unsuccessful single-point search can be handled in exactly two disk accesses, one for the grid directory and one for data buckets, where as other methods based on pointer chains or overflow buckets require considerably more than two disk accesses for unsuccessful searches.

*Software Engineering Development Laboratory, NEC Corporation, 2-11-5, Shibaura, Mita, Tokyo 108, Japan.

As described before, access to a grid file consists of access to the grid directory to find the address of the desired data bucket, and access to a data bucket to find the desired data using the address. The first grid directory access is simply to find the desired scales for each dimension from the search condition. The second data bucket access is simply a direct access to the disk by the bucket number obtained from the grid directory access. Therefore, the access in the grid file is crearly very simple. Even in the range query, the grids containing data points to be searched for can be determined with the help of the grid directory.

However, the grid file technique is far from mature and many improvements need to be made. The P-files presented in this paper are designed to solve some of the problems of grid files, which stem from the large size of their directory entries in the case of correlated data sets. To decrease the grid directory size, the grid shape was changed from a hyper-rectangle to a hyper-parallelogram, which is the basic concept behind the proposed P-files.

The next section describes the problems involved in the original grid files. Section 3 explains P-files. Section 4 shows some simulation results for P-files. Section 5 discusses the pros and cons of the these files.

2. Problems of Grid Files

Although grid files are designed to handle a large amount of multi-dimensional data efficiently, they still involve some problems.

The first is their poor ability to handle sets of correlated data. There are many such data in the real world, for instance in the following types of databases:

- Statistical databases

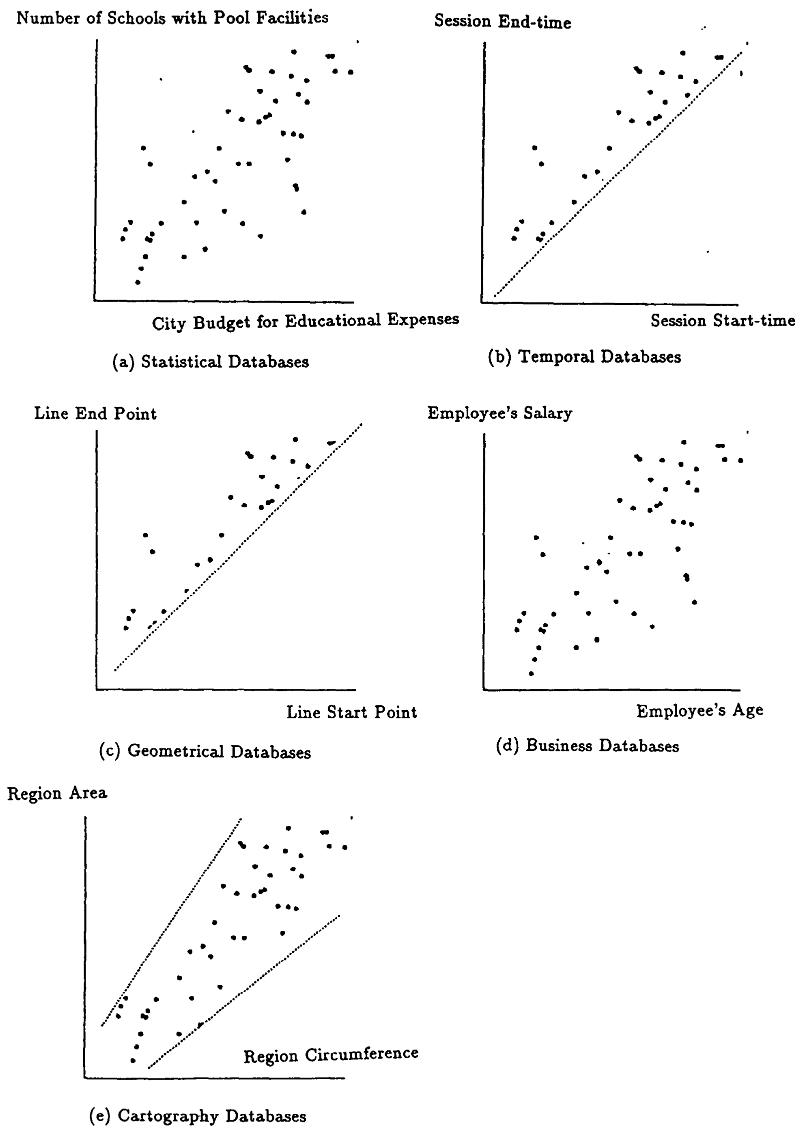


Fig. 1 Examples of Non-Uniform Data.

- Temporal databases
- Geometrical databases
- Business databases
- Cartography databases

Examples of non-uniform distributed data in the above applications are shown in Fig. 1.

Another interesting example of a correlated data set is the grid directory for an interpolation-based grid file [OUKS85], [FREE87]. It consists of a sequence of pairs (range number, level), which is searched from the beginning each time the block is accessed. As the level increases, the range number also increases. For example, for level 2, the range of corresponding range numbers is from 1 to 4, while for level 3 it is from 50 to 20 [FREE87]. In this method, the directory is also managed by the grid file structure. In the grid file for the grid directory, a set of grid directory data becomes non-uniform. For correlated data, the grid directory size grows exponentially in the original grid file.

Another problem with grid files is their implementation strategy. There are no dominant strategies for grid file implementation. The basic file structure is based on a scale partitioning technique presented by Nievergelt et al. [NIEV84]. In their paper, multi-dimensional space is divided into a set of grids by partitioning each axis into a set of intervals. This method increases the grid directory size, because if a grid is split along some coordinate axis, it becomes necessary to split many grids in K-1-dimensional space. Many redundant grid entries will then appear.

To avoid these problems, several modifications have been proposed. A tree-based directory structure was introduced [HINR85] to decrease the directory size and the number of directory accesses. In the structure, the directory for the grid directory is constructed in the same way as the first grid directory structure. The second directory requires only a small amount of memory space, and therefore needs no secondary storage. Another structure, to which the interpolation-based index structure is applied, is called an interpolation-based grid file [FREE87]. In the structure, the mapping from key values to the directory address is performed by calculating a certain function, called an interpolation function. However, these modifications weaken two significant advantages of the original grid files over other multi-dimensional file structures, namely, their two-disk access strategy and simple address calculation. In any modification, these advantages should be retained and maintained to allow performance improvements in relational databases.

The next section describes the proposed new grid files, called P-files, which retain the above-mentioned two advantages.

3. P-Files

3.1 Overview of P-files

P-files are an extension of grid files. A simple modification to a grid file is performed to make it more effective in reducing the directory size in a P-file. A conventional implementation approach for the original grid files is to use scale-based grid files [FREE87], which have a set of linear scales for each dimension to define the position of the grid region. In P-files, this approach is retained as far as possible. The only difference between P-files and grid files is in the grid shape.

Figure 2 shows this difference. The grid file has a set of hyper-rectangle shapes, while the P-file has a set of hyper-parallelogram shapes. A concrete example of these two grid shapes is shown in Fig. 3, presented in two-dimensional space.

The reasons for this modification are as follows. In the correlated data set, there are many empty grids in K-dimensional space [WHAN85] for the original grid file. However, by changing the shape of grids, as in P-files, a smaller number of non-empty grids can cover the same K-dimensional space. For instance, in two-dimensional space, where only one data item can be stored in a single grid, 16 rectangular grids are required in the worst case, whereas four parallelograms are sufficient in P-files, as shown in Fig. 4.

The details of the affine transformation effect for a correlated data set are as follows:

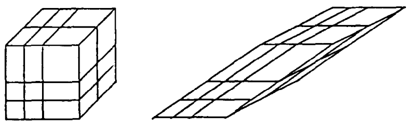
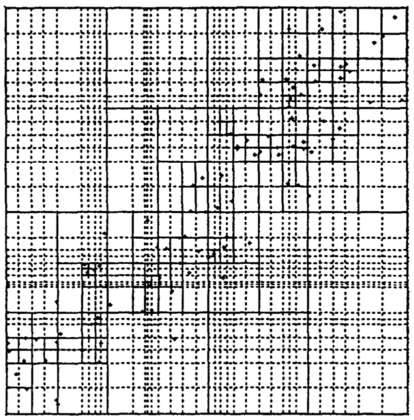


Fig. 2 Grid Shape.


(a) Grid File

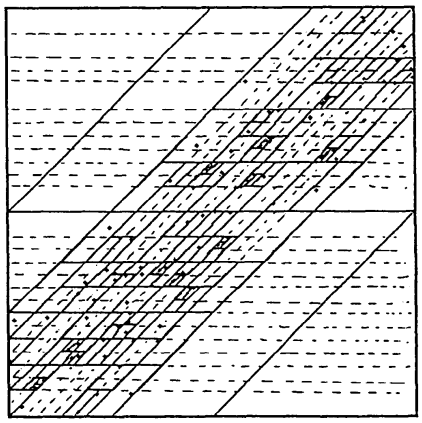

(b) P-file

Fig. 3 Grid Partition.

Suppose that there is a file with two attributes, X and Y, which have a functional dependence such as X=Y, and that only one record can be stored in a physical bucket of the file. This functional dependency is a special case of a correlated data set. In the worst case, the number of directory entries is N^2, where N is the number of records in the file. When an affine transformation of 45 degrees is applied, the functional dependency disappears and all the data are transformed into points on the new axis X-Y. Then, the required

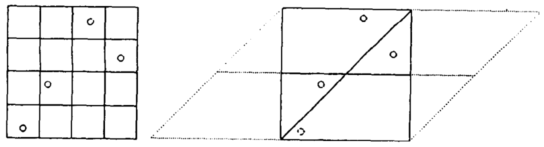


Fig. 4 Simple Example of Grid Shape.

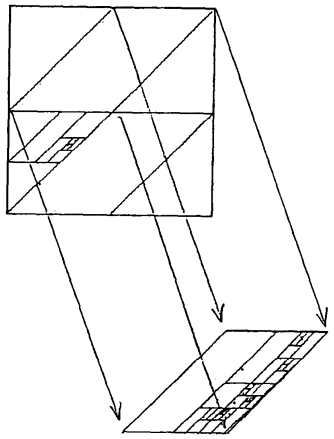


Fig. 5 Multi-level P-file (2-level Case).

number of logical buckets is only N, even in the worst case. Because we need to handle functional dependencies in many practical situations, the affine transformation can eliminate the need for a large directory in the grid file.

Also, suppose that we have a file with two-dimensional correlated data sets each of which has a uniform distribution. Its density function has a shape consisting of similar ovals with the same center, inclined to an axis [STAT89]. As a parallelogram can approximate an inclined oval much better than a rectangle, this class of data set can logically be divided into parallelograms whose number is smaller than that of rectangles. In other words, parallelograms can be adjusted to contain more data than rectangles.

3.2 Operations on a P-file

Consider that each key K consists of n values and is written $K=(k_1, k_2, \ldots, k_K)$. The K-dimensional space is partitioned into $m_1 * m_2 * \ldots * m_K$ grids in the grid file. Therefore, each key K is transformed into a K-tuple integer coordinate address, such as $(i_1, i_2, \ldots, i_K)$. Each coordinate i_j is independently obtained from each key value k_j. The address indicates the block region, in which data containing the key is obtained, by an address transformation any.

The scheme of a grid file is changed as follows in a P-file.

Each key is also transformed into a K-tuple integer coordinate address. However, the transformation is not independent of dimensions, because of the parallelogram shape of grids. For simplicity, the two-dimensional case is considered.

The key value $k=(k_1, k_2)$ is treated as a vector in two-dimensional space. Therefore, the vector k is transformed into the grid coordinate address vector $g=(g_1, g_2)$ by the transformation vector T, as follows:

$$g=Tk \qquad (1)$$

where

$$T=\frac{1}{\sin(B-A)}\begin{pmatrix} \sin(B) & -\cos(B) \\ -\sin(A) & \cos(A) \end{pmatrix}$$

Parameters A and B in Eq. (1) depend on the shape of the parallelogram, which defines the angles between two sides of the parallelogram and a horizontal line.

The generalization of the transformation matrix T is shown below:

$$g=Tk \qquad (2)$$

where $T=[t_1 t_1 \cdots t_K]^{-1}$, t_i is a coordinate vector, constructing an affine space and

$$\|t_i\|=1$$

For $n=2$, $t_1=[\cos(A)\sin(A)]'$ and $t_2=[\cos(B)\sin(B)]'$. Then, Eq. (1) can be obtained by calculating the inverse matrix of $[t_1 t_2]$.

By the above transformation matrix (2), the query procedure becomes as follows:

STEP 1 Transform k into g with the matrix T.
STEP 2 Calculate the grid directory address Dir.adr from g.
STEP 3 Obtain the bucket address Bl.adr from the directory indicated by the directory address Dir.adr
STEP 4 Access the Bl.adr bucket.

Except for STEP 1, the above query procedure is exactly the same as for a grid file [NIEV84]. The only difference is the existence of STEP 1. In STEP 1, the directory space is changed into affine space.

The affine transformation requires matrix calculation in which the number of multiplications is $O(K^2)$ for the transformation of each data item. However, this calculation cost is estimated to be lower than the disk I/O cost because of the lower CPU cost.

3.3 Determination of P-file Parameters

It is very important to decide how to determine coordinate vectors for the affine transformation in a P-file. There are many ways to determine a set of these vectors. Some of the methods are presented here.

The first is to solve a set of equations. Because the transformed data set should be uncorrelated, transformation matrix T has to be determined so as to satisfy the following equations:

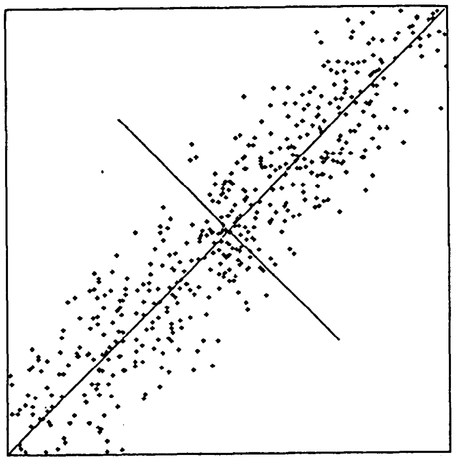


Fig. 6 Coordinate Axes (Use of Principal Component Analysis).

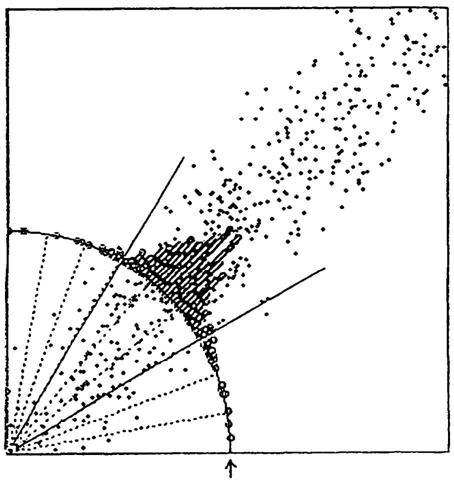


Fig. 7 Coordinate Axes (Use of Distribution after Projection).

- $\Sigma_{i,j}(g_l^i-g_l')*(g_m^j-g_m')-0$

  for any l, m where l not$=m$

  where

  g' shows the average value of g,

  g_l is the l-th-dimensional value of a data $g=T^*k$ and

  g^i is the i-th data item of T^*k.
- Determinant $(T)=1$
- $\|t_q\|=1$

  $q=1, \ldots, K$

  where $T=[t_1\cdots t_K]^{-1}$

  and

  $\|.\|$

  means a vector norm.

In the above equations, there are more variables than equations. Therefore, an optimization procedure is necessary to minimize the residue of the left-hard and right hand terms in each equation.

Calculation of the parameters is not simple in the above method. The following two methods are simpler and easier to use.

The first is to use a principal component analysis or factor analysis method, which finds a set of independent principal component vectors. In the method, eigen-vectors for a correlation matrix are calculated. This method can be used to select coordinate vectors in the proposed method. The result is hown in Fig. 6.

The second simple method is to calcuate the distribution of a data set on a unit hyper-circle with its origin at the cente.r As shown in Fig. 7, the distribution on the circle can be easily calculated by projecting the data set on the circle. Then, from the distribution, two angles can be determined, between which a very large set of data sets falls. As indicated in Fig. 7, 60 degrees and 30 degrees are considered appropriate as these angles.

3.4 Multi-level P-files

The above method of determining the parameters for affine transformation is not dynamic but static, and the characteristics of the data set should therefore be specified beforehand. In some instances, this assumption may be impractical. In order to make the determination dynamically, the following multi-level P-file will be needed.

The main merit of affine transformation in a P-file is that it expands dense regions into a broader regions and reduces the size of empty regions, without the need for complex procedures. When there are several clusters in the directory space, its format can be hierarchical. In other words, the directory space is first broken down into a set of parallelograms, where no two clusters are included in a single parallelogram. An appropriate affine transformation can then be performed in each cluster. This hierarchical structure is called a multi-level P-file. An example of a multi-level P-file is shown in Fig. 5.

The multi-level P-file is used as follows:

STEP 1: At first, a box-shaped grid structure is constructed in the usual manner.

STEP 2: When the number of grids exceeds a given threshold value, the directory and data are restructured in order to generate a P-file structure. The above-mentioned method is used to determine the P-file parameters.

STEP 3: The split and merge operations are executed as for grid files. When a number of grids within a parallelogram grid is generated in STEP 2, the grid area is divided into a set of parallelogram grids, whose parameters are determined again. The new grids are located in a lower level of the multi-level P-file directory tree.

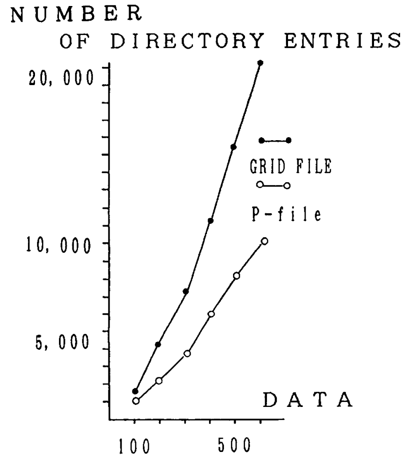


Fig. 8 Two-dimensional Non-uniform Data (page Size=1).

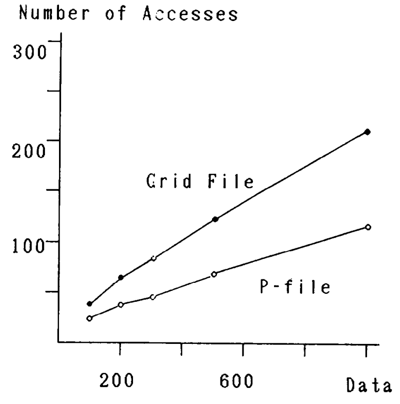


Fig. 10 Number of Bucket Accesses for a Range Search (The same data as in fig. 8 are used.).

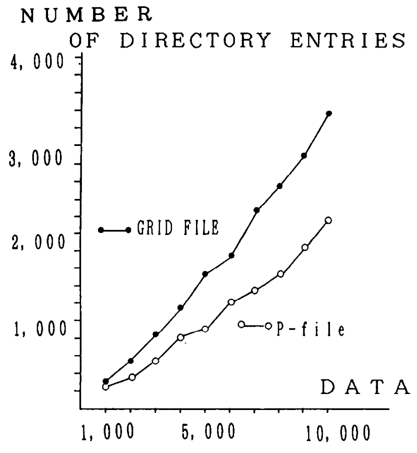


Fig. 9 Two-dimensional Non-uniform Data (Page Size=20).

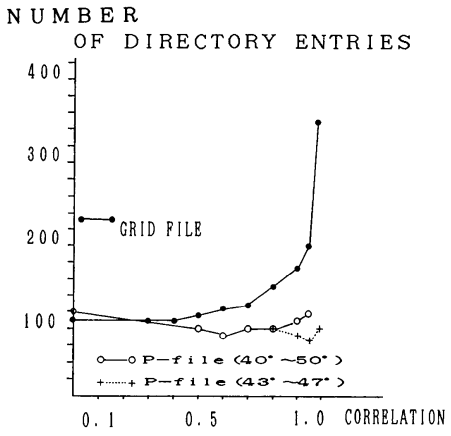


Fig. 11 Simulation Result with Correlation Factor Changed.

3.5 Intersection Procedure

The rest of this section describes the intersection procedure. The DELETE, SPLIT, and MERGE procedures are the same as those employed for grid files [NIEV84].

Checking to determine which grids can meet a user query requirement finds parallelogram grids that intersect with the box-shaped region corresponding to the query. One way of checking an intersection with two regions is as follows:

Assume that box region A is $[a_1, b_1]X[a_2, b_2] \ldots X[a_K, b_K]$ and that parallelogram P has edges L_i $(i=1, \ldots, 12)$.

Step 1: Preliminary check

Construct a box B that includes the parallelogram. The new box B is assumed to be $[c_1, d_1]X \ldots X[c_K, d_K]$. Perform the following simple check to determine whether box B intersects box A: "If, for any i, there always exists a value of j such that $[a_i, b_i]$ intersects $[c_j, d_j]$, then box A intersects box B."

STEP 2: When box B intersects box A, the following check is performed for box A and parallelogram P. The first check is to determine whether any edges L_i in parallelogram P intersect box A. If there are intersections, the parallelogram intersects the box. If there are no intersections, the box is either inside or completely outside the parallelogram, and the following chekc has to be performed in order to find which condition exists.

The second check is to determine whether or not a point inside the box is inside the parallelogram. This is a well-known procedure computer graphics for selecting or removing hidden surfaces. The number of time an infinite line appears in some direction from a point inside the box is counted. If the number of crossings is two, parallelogram P completely contains box A.

4. Simulation Results

To investigate the effect of a P-file on the directory size, the file is compared with the original grid file with a scale-based structure for a non-uniform data set. The data set is generated from a multi-variate normal distribution with the following normalized covariance matrix:

$$\begin{pmatrix} 1.0 & 0.8 \\ 0.8 & 1.0 \end{pmatrix}$$

Figures 8 and 9 show the relationship between the number of data and the directory size. In Fig. 8, which shows the case in which the block size is 1, we observe that the ratio of the directory size to the number of data in the P-file is a half to two-thirds less than in the grid file. On the other hand, in Fig. 9, which shows a case in which the block size, 20, is larger, the ratio in the P-file is still less than in the grid file.

Another simulation was done to evaluate the number of directory accesses involved in a range query. The query condition used in the simulation for the range query is a partial match query where the first axis is specified and the range on the axis is from 0.2 to 0.8. The result is shown in Fig. 10. This lower directory access in the P-file is a result of the smaller number of directory grids. Because fewer buckets are accessed than directory grids, the performance improvement in P-file is expected to be significant.

A third simulation was done with several correlation factors for a block size of 20. As the data set becomes correlated, the grid file performance deteriorates, as shown in Fig. 11. However, the P-file performance tends to remain constant. in Fig. 11, two kinds of P-file angle are compared, where one is narrower than the other. The difference in the directory size between the two kinds of P-file angle selections is not significant. Therefore, it is shown that even if the data set has little correlatation or the correlation factor is overestimated, the resulting grid directory size is not larger than in the grid file.

5. Discussion

5.1 Performance of P-files

The following is a summary of the P-file performance.

- Number of Disk Accesses
  It has already been stated that the number of disk accesses for single-point access is always two in the case of grid files. P-files have the same performance as grid files. No other disk access is required in any circumstances, while a cache will decrease the number of disk accesses. The number of disk accesses for INSERTION or DELETION varies according to whether a SPLIT or MERGE operation has to be internally performed or not. If a SPLIT or MERGE operation is needed, the number of disk accesses is two GET and one PUT for data bucket access, and $O(V^{1-1/K})$ for directory access, where V is the total number of directory entries, which is the same as for grid files [KHOS85]. If neither the SPLIT nor the MERGE operation is needed, the number of disk accesses is exactly three: two GET operations and one PUT operation.
- Average Space Utilization
  For grid files, the average bucket utilization is reported to be around 70% [NIEV84]. P-files have the same average space utilization value. This is not affected by the correlation data set, where as the correlation affects the size of the grid directory, as seen in the previous simulation.

5.2 Comparison with Multi-dimensional Hashing

Besides P-files, there are other methods for efficient storage of non-uniform distributed data sets using the hashing method. They include Multidimensional Dynamic Quantite Hashing [KRIE87], the Symmetric Dynamic Index Maintenance Scheme [OTOO85a], and PLOP-Hashing [KRIE88].

Multi-dimensional quantile hashing is a method for non-uniform record distributions. However, it cannot well be applied when the record distributions are correlated, because it partitions points for each attribute independently.

The symmetric dynamic index maintenance method is a multi-dimensional indexing method using multi-dimensional linear hashing. Its inventor, E. Otoo, stresses that the method avoids exponential growth of the directory. However, the number of bucket accesses for non-uniform distributed data sets is more than two, and is five to ten times higher than the number of accesses for a uniform distribution, according to his simulation result [OTOO85a]. For P-files, the number of accesses is guaranteed to be two even for correlated data, as described previously.

PLOP-Hashing is a multi-dimensional hashing method with no directories. By expanding piecewise linear hashing functions in dense arease, the PLOP method can adapt to non-uniform distributions. However, like the above-mentioned quantile hashing, it is still unknown how efficiently the method organizes a correlated data distribution.

Finally, all the above hashing-based methods assume the use of an order-preserving hashing function before the calculation of multi-dimensional hashing functions. For these hashing methods to handle range queries, an order-preserving hashing function also needs to be provided. However, these assumptions are not practical, because these functions have rarely been used in practice. In contrast, P-files can be applied for both correlated data set and range query requirement with no hashing functions.

5.3 Rotated Grid Files.

P-files include another modification of the grid files. For example, a rotated grid file [FALO87] is presented in order to decrease the number of directories by rotating the directory space. In P-files, a rotated grid file is the case where the following relationship between angles A and B in Eq. (3) is satisfied:

$$B = A + \pi/2 \qquad (3)$$

(in radians)

5.4 Implications

It is foreseen that the P-file structure will be useful for decreasing the directory size. However, there are two considerations: (1) No significant change in the number of blocks is expected for P-files. The average load factor is likely to be ln 2, as in other multi-dimensional file structures. (2) The directory size does not increase linearly, but still is of a higher order. The reason for this is that the original grid directory structure [NIEV84] was used, in order to maintain the two disk-access strategy and simpler address computation.

6. Conclusion

This paper presents a new kind of grid file, called a P-file. The file is constructed to transform the shape of each grid into a parallelogram in multi-dimensional space.

With this change, the distribution can be changed so as to decrease the directory size. The P-file has the same advantages as the grid file, a two disk-access strategy and simple calculation, whereas other modifications fail to retain these advantages by modifying the directory structures from array to either hierarchical trees or interpolation-based hashing.

The transformation method used in P-files can be applied to other kinds of grid structures, such as BANG files [FREE87] and multi-level grid files [WHAN85], as a pre-processor so as to change a correlated data set into a uniform data set. P-files can also be combined with other kinds of multidimensional structures, such as multi-dimensional B-trees and extendible hashing.

Acknowledgement

The author thanks Dr. Fujino, Mr. Ishiguro, Mr. Saya, and Mr. Kawakita for motivating him to write this paper. Thanks also to the referees for their useful comments and suggestions.

References

[BENT79] Bentley, J. L. Multidimensional Binary Search Trees in Database Applications, *IEEE Trans. on Softw.*, SE-5, 4 (July 1979).

[FALO87] Faloutsos, C. and Rego, W. A Grid Fuile for Spatial Objects, CSTR-1829, Univ. of Maryland, 1987.

[FREE87] Freeston, M. The BANG file: A New Kind of Grid File, *ACM SIGMOD* (1987), 260-269.

[HINR85] Hinrichs, K. H. Implementation of The Grid File: Design Concepts and Experience, *BIT* **25** (1985), 569-592.

[KHOS85] Khoshafian, S., Banerjee, J., Copeland, G. and Valduries, P. A Performance-directed Taxonomy for Single-key and Multi-key File Structures, Tech. Memo, MCC, 1985.

[KRIE84] Kriegel, H-P. Performance Comparison of Index Structures for Multi-Key Retrieval, *ACM SIGMOD* (1984), 186-196.

[KRIE87] Kriegel, H-P. and Seeger, B. Multidimensional Dynamic Quantile Hashing is Very Efficient for Non-uniform Record Distribution, *Proc. Int. Conf. on Data Engineering* (1987), 10-17.

[KRIE88] Kriegel, H-P. and Seeger, B. PLOP-Hashing: A Grid File without Directory, *Proc. of IEEE International Conference on Data Engineering* (1988), 369-376.

[NIEV84] Nievergelt, J., Hinterberger, H. and Sevcik, K. C. The Grid File: an Adaptive Symmetric Multikey File Structure, *ACM TODS*, **9** (March 1984), 38-71.

[OTOO84] Otoo, E. J. A Mapping Function for Directory of a Multidimensional Extendible Hashing, *Proc. of VLDB* (1984), 493-505.

[OTOO85a] Otto, E. J. Symmetric Dynamic Index Maintenance Scheme, *Proc. of Int. Conf. on Foundation of Data Organization* (1985), 283-296.

[OTOO85b] Otoo, E. J. A Multidimensional Digital Hashing Scheme for Files With Composite Keys, *Proc. of PODS* (1985), 214-229.

[OTOO86] Otoo, E. J. Balanced Multidimensional Extendible Hash Tree, *ACM SIGMOD* (1986), 100-113.

[OUKS85] Ouksel, M. Interpolation-Based Grid file, Principles of Database Systems (1985), 20-27.

[REGN85] Regnier, M. Analysis of Grid file Algorithms, *BIT*, **25** (1985), 335-357.

[ROBI81] Robinson, J. T. The K-D-B Tree: A Search Structure for large Multidimensional Dynamic Indexes, *ACM SIGMOD* (1981), 10-18.

[STAT89] Edited by Takeuchi, K. Statistics Dictionary, in Japanese, Toyo-Keizai-Shinpo-Sha, 1989.

[TAMM83] Tamminen, M. Performance Analysis of Cell Based Geometric File Organizations, *Comp. Vision, Graphics & Image Proc.*, **24** (1983), 160-181.

[WHAN85] Whang, K-Y. and Krishnamurthy, R. Multilevel Grid Files, *IBM Res. Rept., RC11516*, 1985.

(Received May 16, 1989; revised June 15, 1990)