G

Grid File (and Family)

Apostolos N. Papadopoulos[1], Yannis Manolopoulos[1], Yannis Theodoridis[2], and Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Riverside, Riverside, CA, USA

Definition

The Grid File is a *multidimensional indexing* scheme capable to efficiently index database records in a symmetrical manner, i.e., by avoiding the distinction between primary and secondary keys. The structure is dynamic and adapts gracefully to its contents under insertions and deletions. A single record retrieval costs two disk accesses at most (upper bound), whereas range queries and partial-match queries are also executed efficiently. The Grid File can be thought of as a generalization of dynamic hashing (e.g., *extendible hashing*) in multiple dimensions.

Historical Background

Until the 1980s, many file structures for the processing of single-attribute queries have been proposed, i.e., queries on the primary key or any secondary key for which a corresponding index has been built. Multi-attribute queries are the ones where the user seeks objects that satisfy constraints (such as equality or range) on several attributes. Such queries can be executed by accessing all the corresponding indices (if they exist) and combining the partial results or resorting to sequential scanning.

To speed up the processing of multiple attribute queries, a better solution is to create an index that leads the search directly to the objects of interest. Such an index can be designed if a data record with k attributes is envisioned as a point in a k-dimensional space. A multi-attribute range query would then be a hyper-rectangle in this k-dimensional space, and the answer to it would be all points inside this rectangle. Access methods that can handle multidimensional points are called *point access methods* (PAMs). In 1975, Bentley proposed such a basic PAM, which is called the k-dimensional tree or k-d tree [2]. The Grid File is yet another structure designed to handle similar cases, proposed by Nievergelt et al. in 1984 [10]. Since then, several variations have been proposed in the literature in an effort to optimize its space and time performance behavior.

Foundations

TheGrid File can be viewed as an access method comprising of two separate parts: (i) the *directory* and (ii) the *linear scales*. To illustrate

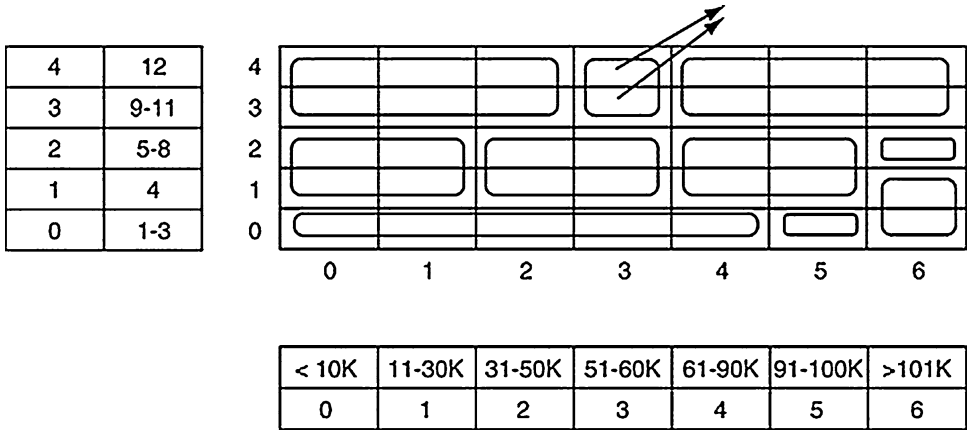
L. Liu, M.T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_187-2

this, assume that an index is to be constructed on the *employee* file using two attributes, say *salary* and *dept* (extension to more dimensions is straightforward). The Grid File imposes a grid on the two-dimensional attribute space. Each cell in this grid corresponds to one data page. The data points that "fall" inside a given cell are stored in the cell's corresponding page. Each cell must thus store a pointer to its corresponding page. This information is stored in the Grid File's *directory*. However, cells that are empty do not use a page. Rather, two or more cells can share a page (i.e., point to the same page). The grid adapts to the data density by introducing more divisions in areas where there are more points.

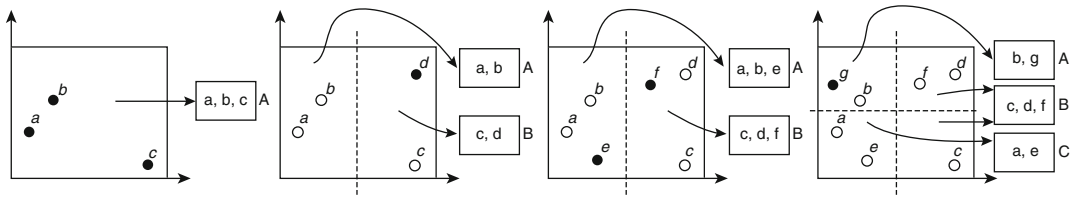
The information of how each dimension is divided (and thus how data values are assigned to cells) is kept through *linear scales*. There is one linear scale per dimension (indexed attribute). Each linear scale is a one-dimensional array that divides the values on a particular dimension in such a way that records (points) are uniformly distributed across cells. An example of a Grid File on the "dept" and "salary" attributes appears in Fig. 1.

Searching for a record with given attribute values involves two operations: (i) the Grid File's directory is searched to locate the cell that the record is hosted in, (ii) the cell's pointer is followed to access the corresponding data page (say *A*), and (iii) the record is searched only in data page *A*. If the record is found in *A* then the search terminates successfully, otherwise the search for the specific record is unsuccessful (i.e., the record does not exist). The Grid File can also address multidimensional *range queries* by selecting the appropriate cells from each dimension's linear scale. For example, such a query may ask for all employee records with the *salary* attribute ranging between 10 and 40 K and the *dept* attribute ranging between 2 and 6. Again, the first step examines the directory and determines the cells that are intersected by the query range in both attributes, then the corresponding pointers to data pages are collected, and finally the data pages are examined for relevant records. The accessed cells may also contain some records outside the query range. These records are eliminated from further consideration, and they are not returned as part of the query answer.

Insertinga new record in this method is straightforward. First, the two linear scales are searched so as to map the record's *salary* and *dept* attribute values in each dimension. This mapping provides a cell in the directory. This cell is then accessed, and, using its pointer, the appropriate page, say *A*, for the new record is found. If this page has enough space to accommodate the new record, the insertion process is complete. Otherwise, a new page *B* is allocated. If page *A* was pointed to by more than one cell, the pointers of these cells are rearranged such that some will point to page *A* and some to page *B* (and the records of page *A* are redistributed accordingly



Grid File (and Family), Fig. 1 A Grid File



Grid File (and Family), Fig. 2 Insertions in the Grid File

between *A* and *B*). If page *A* was pointed by a single cell and overflows, a reorganization of the Grid File is needed. This reorganization will expand the directory and the scales by introducing a new column (or row) of cells.

In the sequel, the insertion process is illustrated by an example given in Fig. 2. White dots correspond to existing records, whereas black dots are used to indicate new records being inserted to the Grid File. Assume that each data page can host at most three records. Practically, this number is larger in real applications and depends on the size of the data page and the number of attributes. Assume that initially the Grid File is empty (does not contain any records). The first three records can be easily accommodated in the single data page *A* pointed by the single cell of the directory (corresponding to the whole data space), as illustrated in Fig. 2a. The next inserted record is *d*. However, the new record cannot be hosted by data page *A* because its capacity is exceeded. Therefore, another data page *B* is allocated, and records are distributed to the two data pages as it is shown in Fig. 2b. The next two insertions for records *e* and *f* do not cause any reorganization since the new records can be easily accommodated in the corresponding data pages pointed by the cells, as illustrated in Fig. 2c. Finally, the insertion of record *g* causes an overflow in data page *A*. The corresponding cell is split again using the other attribute, and one more data page is allocated, and records are distributed accordingly. The final shape of the Grid File is given in Fig. 2d.

Deletions are also supported, but they are handled differently. Initially, the deleted record is located using the directory, and the corresponding data page is determined. If the record is found, it is deleted from the data page. Instead of overflowing, data page deletions may cause the underutilization effect, which means that several data pages may contain too few records. Therefore, appropriate merging operations are required to maintain the storage utilization of the Grid File at an acceptable level. For a detailed description of the methods used for merging as well as for splitting, the reader is directed to [10].

The Grid File has a set of nice properties: (i) it is based on simple mechanisms for insertion, deletion, and search; (ii) it guarantees only two disk accesses for exact-match queries (one for the directory and one for the data page); and, (iii) it treats all indexed attributes symmetrically which leads to simple directory management policies. However, it has a set of serious disadvantages such as: (i) it introduces a space overhead for the directory, which can be large for high-dimensional spaces; (ii) it has an extra update overhead, since reorganization affects many cells and not only the cell with the overflowing page; and (iii) it suffers from performance degradation if the attributes are correlated, since the uniform scheme for performing splits is not adequate to guarantee performance efficiency.

Toward improving the behavior of the Grid File, several research efforts have been performed. Some of these efforts are highlighted in the following lines. One of the first variations, the BANG File, has been proposed by Freeston [4]. The BANG File is based on a self-balanced tree-based directory, which better reflects changes of the data distribution. To achieve better storage utilization, the Twin Grid File has been proposed by Hutflesz et al. in [5]. The new scheme is as efficient as the original Grid File during range query processing but shows significant improvements regarding storage utilization. Blanken et al. proposed the

Generalized Grid File [3], which offers fast access for single-attribute queries. The Multilevel Grid File [12] is another research effort to improve the performance of the original structure for exact-match, partial-match, and range queries. This new scheme uses multiple grid levels and succeeds in better directory management and more efficient query processing than the original structure.

In addition to the variations proposed in the literature, there are efforts to use the Grid File in a parallel environment, toward more efficient data management. In [13], the authors study the problem of partitioning a Grid File to multiple disk devices toward more efficient search. When a data page split is performed, the new data page is carefully allocated to a disk. Since disks can be accessed in parallel, several data pages can be read simultaneously during range query processing, offering significant performance improvements in comparison to a single-disk system. More complex queries on Grid Files, like *spatial joins*, have been also parallelized [6] toward reduced query response times. A different approach has been followed by [7]. The authors have proposed a method to load a Grid File in parallel. The data file is initially partitioned to the available processors using dynamic programming and sampling, and then each processor builds its own part of the Grid File.

Key Applications

Spatial Databases

In Spatial Databases, it is commonly required to join spatial data sets or perform nearest neighbor searches. Several algorithms have been proposed for such operations by adopting the Grid File as the underlying access method [1].

Data Mining

The Grid File can also be used for clustering data sets to identify correlation characteristics of the underlying value space. This stems from its ability to group patterns into blocks and cluster them with respect to the blocks by a topological neighbor search algorithm [11].

Data Warehouses

The Grid File can be used for efficient data cube storage in warehouses [9].

Future Directions

The Grid File has eventually emerged as a popular theoretical access method. However, although it has been widely honored in theory, in practice it has not been used by the database industry.

Experimental Results

A detailed performance evaluation of the Grid File can be found in [10], where the authors offer a detailed experimental section studying the properties of the structure regarding capacity of data pages, directory size, and evaluation of splitting and merging policies. Moreover, interesting experimental results can be found in [3, 5], which compare the original Grid File with the corresponding variation proposed in each work.

Cross-References

- Extendible Hashing
- K-D Trees
- Multidimensional Indexing
- Range Query
- Spatial Join

Recommended Reading

1. Becker L, Hinrichs K, Finke U. A new algorithm for computing joins with grid files. In Proceeding of 9th International Conference on Data Engineering, 1993, p. 190–97.
2. Bentley JL. Multidimensional binary search trees used for associative searching. Commun ACM. 1975;18(9):509–17.
3. Blanken HM, Ijbema A, Meek P, van den Akker B. The generalized grid file: description and perfor-

mance aspects. In Proceeding of 14th International Conference on Data Engineering, 1990, p. 380–88.
4. Freeston M. The BANG file: a new kind of grid file. In Proceeding of ACM SIGMOD International Conference on Management of Data, 1987, p. 260–69.
5. Hutflesz A, Six H -W, and Widmayer P. Twin grid files: space optimizing access schemes. In Proceeding of ACM SIGMOD International Conference on Management of Data, 1988, p. 183–90.
6. Kim J -D, Hong B -H. Parallel spatial join algorithms using grid files. In Proceeding of 6th International Conference on Database Systems for Advanced Applications, 1999, p. 226–34.
7. Li J, Rotem D, Srivastava J. Algorithms for loading parallel grid files. In Proceeding of ACM SIGMOD International Conference on Management of Data, 1993, p. 347–56.
8. Lim Y, Kim M. A Bitmap Index for multidimensional data cubes. In Proceeding of 15th International Conference Database and Expert System Applied, 2004, p. 349–58.
9. Luo C, Hou WC, Wang CF, Wang H, Yu X. Grid file for efficient data cube storage. Computers and their applications, conference paper CATA, 2006, p. 424–29.
10. Nievergelt J, Hinterberger H, Sevcik KK. The grid file: an adaptable, symmetric multikey file structure. ACM Trans Database Syst. 1984;9(1):38–71.
11. Schikuta E, Erhart M. The BANG-clustering system: grid-based data analysis. In Advanve in Intelligent Data Analysis, 2nd International Symposium, 1997, p. 513–24.
12. Whang K -Y, Krishnamurthy R. The multilevel grid file – a dynamic hierarchical multidimensional file structure. In Proceeding of 2nd International Conference on Database Systems for Advanced Applications, 1991, p. 449–59.
13. Zhou Y, Shekhar S, Coyle M. Disk allocation methods for parallelizing grid files. In Proceeding of 10th International Conference on Data Engineering, 1994, p. 243–52.