A Mapping Based Approach for Multidimensional Data Indexing

Rui Zhang

Department of Computing and Information Systems
University of Melbourne, Australia

Email: rui@csse.unimelb.edu.au

Abstract

The most common approach to improve performance for databases is through indexing. Mapping based approach is an easy to implement paradigm for indexing multidimensional data. It does not need complicated structures or algorithms, but some transformations (mapping functions) to convert multidimensional data to one dimensional data. Then the converted data can be indexed using a robust and efficient one-dimensional index such as the B^+-tree. This paper talks about a number of mapping based indexes for typical types of multidimensional queries.

Keywords: Multidimensional indexing, mapping based indexing

1 Introduction

The most common approach to improve performance for databases is through indexing. Mapping based approach is an easy to implement paradigm for indexing multidimensional data. It does not need complicated structures or algorithms, but some transformations (mapping functions) to convert multidimensional data to one dimensional data. Then the converted data can be indexed using a robust and efficient one-dimensional index such as the B^+-tree. This paper talks about a number of mapping based indexes for typical types of multidimensional queries.

This paper starts with an introduction of multidimensional data and queries, and then describes the key techniques in mapping based multidimensional data indexing as well as window query and *k*NN query processing. The paper concludes with a summary and a discussion on future work.

2 Multidimensional Data and Queries

Common types of multidimensional data involve: (i) spatial data, where the data dimensionality is usually low (e.g., 2 or 3); (ii) records with multiple attributes, which are commonly seen in data tables and the dimensionality is medium; (iii) multimedia data, where the dimensionality can get very high (e.g., hundreds or even thousands).

Various types of queries have been studied on multidimensional data. Among them, the three most basic and popular ones are: (i) the point query, which returns the objects at a query point Q; (ii) the window query (range query), which returns all the objects enclosed or intersected by a query hyper-rectangle W; (iii) the k-nearest neighbor (kNN) query, which returns k objects whose distances to the query object Q are no larger than any other object' distance to Q.

3 Mapping Based Multidimensional Indexing

Indexing is a commonly used technique to boost query processing efficiency. For multidimensional data, mapping based indexing is a technique that has attracted ongoing study and demonstrated competitive performance.

This type of indexing involves three steps in general:

- Data mapping and indexing, where multidimensional data are mapped to a 1-dimensional space (e.g., using a space-filling curve like Z-curve) and indexed by a B^+-tree.
- Query mapping and data retrieval, where the query object (or window) is mapped to a 1-dimensional value ranges and perform a range query on the B^+-tree to retrieve candidate query answers.
- Filtering out false positives, where candidate query answers are compared with the query object (window) in the original multidimensional space and false positives are filtered.

Next, we describe two specific examples of mapping based indexes for processing window queries and kNN queries.

3.1 The P^+-tree

The P^+-tree [2] is a mapping based index designed for processing window queries and kNN queries on multidimensional data.

Window queries are common in CAD, medical image and GIS systems. For example, the Forest CoverType data set has 54 attributes, out of which 10 are quantitative. To analyze the data, (partial) window queries that specify some ranges of elevation, aspect, slope, cover type, etc are typical. KNN queries are common in content-based retrieval systems. While a large number of indexing techniques have been proposed to improve performance, most of these techniques are not sufficiently robust for a wide range of queries. For example, the Pyramid technique [1] is efficient for window queries, but perform less satisfactorily for kNN queries. On the other hand, metric-based schemes such as the iDistance [3] are usually superior for kNN queries, but may not be usable for window queries. Moreover, these schemes typically perform well for certain workloads (data set size, dimensionality, data distribution, etc) and become inferior to sequential scan in other cases. In comparison, the P^+-tree supports both window and kNN queries under different workloads (different data set size and dimensionality, different data

distribution, queries with different selectivity and different shapes) efficiently. The index is based on the Pyramid technique, which is primarily designed for hypercube-shaped window queries (where the query rectangle has equal sides in all dimensions). While the Pyramid technique is shown to be much better than the Hilbert R-tree and the X-tree for data sets of dimensionality larger than 8, it has three deficiencies: its performance is sensitive to the positions of the query hyper-cube; it is less effective for clustered data sets; and it is inferior for partial window queries. In these cases, a sequential scan over the data set may be more effective.

The basic structure of the P^+-tree is essentially a B^+-tree that indexes subspaces of points under a new transformation method. The data space is first partitioned into subspaces based on clustering. Next, points in each subspace are mapped onto a single dimensional space using the Pyramid technique, and stored in the B^+-tree. To discriminate the points within each cluster, they are transformed into non-overlapping regions in the single dimensional space. The crux of the scheme lies in the choice of the transformation that has two crucial properties. First, it maps each subspace into a hypercube so that the Pyramid scheme can be applied. Second, it shifts the cluster center to the top of the pyramid, which is the case that the Pyramid technique works very efficiently.

3.2 iDistance

The iDistance [3] technique is a mapping based index designed specifically for processing *k*NN queries on high-dimensional data.

Many emerging database applications such as image, time series, and scientific databases, manipulate high-dimensional data. In these applications, one of the most frequently used and yet expensive operations is to find objects in the high- dimensional database that are similar to a given query object. Nearest neighbor search is a central requirement in such cases. There is a long stream of research on solving the nearest neighbor search problem, and a large number of multidimensional indexes have been developed for this purpose. Existing multidimensional indexes such as R-trees have been shown to be inefficient even for supporting range queries in high-dimensional databases; however, they form the basis for indexes designed for high-dimensional databases. To reduce the effect of high dimensionality, use of larger fanouts, dimensionality reduction techniques, and filter-and-refine methods have been proposed. Indexes have also been specifically designed to facilitate metric based query processing. However, linear scan remains an efficient search strategy for similarity search. This is because there is a tendency for data points to be nearly equidistant to query points in a high-dimensional space. While linear scan is effective in terms of sequential read, every point incurs expensive distance computation, when used for the nearest neighbor problem. For quick response to queries, with some tolerance for errors (i.e., answers may not necessarily be the nearest neighbors), approximate nearest neighbor (NN) search indexes such as the P-Sphere tree have been proposed. The P-Sphere tree works well on static databases and provides answers with assigned accuracy. It achieves its efficiency by duplicating data points in data clusters based on a sample query set. Generally, most of these structures are not adaptive to data distributions. Consequently, they tend to perform well for some datasets and poorly for others.

On the contrary, iDistance, a new technique for KNN search, can be adapted to different data distributions. This technique first partitions the data and define a reference point for each partition. Then it indexes the distance of each data point to the reference point of its partition. Since this distance is a simple scalar, with a small mapping effort to keep partitions distinct, a classical B^+-tree can be used to index this distance. As such, it is easy to graft iDistance on top of an existing commercial relational database. This is important as most commercial DBMSs today do not support indexes beyond the B^+-tree and the R-tree (or one of its variants). The effectiveness of iDistance depends on how the data are partitioned, and how reference points are selected.

For a KNN query centered at q, a range query with radius r is issued. The iDistance KNN search algorithm searches the index from the query point outwards, and for each partition that intersects with the query sphere, a range query is resulted. If the algorithm finds K elements that are closer than r from q at the end of the search, the algorithm terminates. Otherwise, it extends the search radius by r, and the search continues to examine the unexplored region in the partitions that intersects with the query sphere. The process is repeated till the stopping condition is satisfied. To facilitate efficient KNN search, partitioning techniques and reference point selection strategies as well as a cost model to estimate the page access cost of iDistance KNN searching are proposed.

3.3 Partitioning and Reference Point Selection

To support distance-based similarity search, we need to split the data space into partitions and for each partition, we need a reference point. Two partitioning techniques are studied.

3.3.1 Space-Based Partitioning

A straightforward approach to data space partitioning is to subdivide the space into equal partitions. In a d-dimensional space, we have $2d$ hyperplanes. The method adopted is to partition the space into $2d$ pyramids with the center of the unit cube space as their top, and each hyperplane forming the base of each pyramid. Three reference point selection and partition strategies are studied:

(i) *Center of Hyperplane, Closest Distance.* The center of each hyperplane can be used as a reference point, and the partition associated with the point contains all points that are nearest to it.

(ii) *Center of Hyperplane, Furthest Distance.* The center of each hyper-plane can be used as a reference point, and the partition associated with the point contains all points that are furthest from it.

(iii) *External Point.* Any point along the line formed by the center of a hyper- plane and the center of the corresponding data space can also be used as a reference point. Here, a external point refers to a reference point that falls outside the data space.

3.3.2 Data-Based Partitioning

Equi-partitioning may seem attractive for uniformly distributed data. However, data in real life are often clustered or correlated. Even when no correlation exists in all dimensions, there are usually subsets of data that are locally correlated. In these cases, a more appropriate partitioning strategy would be used to identify clusters from the data space. The K-means clustering algorithm is adopted. The number of clusters affects the search area and the number of traversals from the root to the leaf nodes. We expect the number of clusters to be a tuning parameter, which may vary for different applications and domains. Once the clusters are obtained, we need to select the reference points. Two possible options are considered when selecting reference points:

(i) *Center of cluster.* The center of a cluster is a natural candidate as a reference point.

(ii) *Edge of cluster.* When the cluster center is used, the sphere areas of both clusters have to be enlarged to include outlier points, leading to overlap in the data space. To minimize the overlap, we can select points on the edge of the partition as reference points, such as points on hyperplanes, data space corners, data points at one side of a cluster and away from other clusters, and so on.

3.4 Cost Model

iDistance is designed to handle KNN search efficiently. However, due to the complexity of very high-dimensionality or the very large K used in the query, iDistance is expected to be superior for certain (but not all) scenarios. Cost models are developed to estimate the page access cost of iDistance, which can be used in query optimization (for example, if the iDistance has the number of page accesses less than a certain percentage of that of sequential scan, we would use iDistance instead of sequential scan). The cost model developed is based on both the Power-method and a histogram of the key distribution. This histogram-based cost model applies to all partitioning strategies and any data distribution, and it predicts individual query processing cost in terms of page accesses instead of average cost. The basic idea of the Power-method is to precompute the local power law for a set of representative points and perform the estimation using the local power law of a point close to the query point. In the key distribution histogram, we divide the key values into buckets and maintain the number of points in each bucket.

4 Conclusions and Research Trends

In summary, we have presented two mapping based indexes for multidimensional data. Among them, the P^+-tree is an efficient technique for window queries, while iDistance for kNN queries. The key in query processing efficiency of the mapping based indexes is the mapping function. Zhang et al. [4] generalize the mapping based indexing technique and process the GiMP framework.

Recently, studies on multidimensional data have turned to location optimization [20] and temporal aspect of the data. Queries on moving objects [5-8, 10, 12, 14-19], version data [9, 13], and data streams [11] become the hot topic.

References

[1] Stefan Berchtold, Christian Bhm, Hans-Peter Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. ACM SIGMOD International Conference on Management of Data (SIGMOD) 1998.

[2] Rui Zhang, Beng Chin Ooi, Kian-Lee Tan. Making the Pyramid Technique Robust to Query Types and Workloads. IEEE International Conference on Data Engineering (ICDE) 2004.

[3] H.V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, Rui Zhang. iDistance: An Adaptive B+-tree Based Indexing Method for Nearest Neighbor Search. ACM Transactions on Data Base Systems (TODS), 30(2), 2005.

[4] Rui Zhang, Panos Kalnis, Beng Chin Ooi, Kian-Lee Tan. Generalized Multi-dimensional Data Mapping and Query Processing. ACM Transactions on Data Base Systems (TODS), 30(3), 2005.

[5] Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, Rudolf Bayer. Integrating the UB-Tree into a Database System Kernel. International Conference on Very Large Data Bases (VLDB) 2000.

[6] Beng Chin Ooi, Kian-Lee Tan, Cui Yu, Stphane Bressan. Indexing the Edges - A Simple and Yet Efficient Approach to High-Dimensional Indexing. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS) 2000.

[7] Rui Zhang, Dan Lin, Kotagiri Ramamohanarao, Elisa Bertino. Continuous Intersection Joins over Moving Objects. Proceedings of the 24th International Conference on Data Engineering (ICDE), pp. 863-872, April 7-12, 2008.

[8] Mohammed Eunus Ali, Rui Zhang, Egemen Tanin, Lars Kulik. A Motion-Aware Approach to Continuous Retrieval of 3D Objects. Proceedings of the 24th International Conference on Data Engineering (ICDE), pp. 843-852, April 7-12, 2008.

[9] David Lomet, Mingsheng Hong, Rimma Nehme, Rui Zhang: Transaction Time Indexing with Version Compression. Proceedings of the VLDB Endowment (PVLDB), 1(1), 870-881, 2008.

[10] Sarana Nutanong, Rui Zhang, Egemen Tanin, Lars Kulik: The V*-Diagram: A Query Dependent Approach to Moving KNN Queries. Proceedings of the VLDB Endowment (PVLDB), 1(1), 1095-1106, 2008.

[11] Xiaoyan Liu, Xindong Wu, Huaiqing Wang, Rui Zhang, James Bailey, Kotagiri Ramamohanarao. Mining Distribution Change in Stock Order Streams. Proceedings of the 26th International Conference on Data Engineering (ICDE), pp. 105-108, 2010.

[12] Cui Yu, Rui Zhang, Yaochun Huang, Hui Xiong: High-dimensional kNN joins with incremental updates. GeoInformatica, 1 (14), 55-82, 2010.

[13] Rui Zhang, Martin Stradling. The HV-tree: a Memory Hierarchy Aware Version Index. Proceedings of the VLDB Endowment (PVLDB), 3(1), 397-408, 2010.

[14] Sarana Nutanong, Rui Zhang, Egemen Tanin, Lars Kulik. Analysis and Evaluation of V*-kNN: An Efficient Algorithm for Moving kNN Queries. VLDB Journal, 19(3): 307-332, 2010.

[15] Mohammed Eunus Ali, Egemen Tanin, Rui Zhang, Lars Kulik. A Motion-Aware Approach for Efficient Evaluation of Continuous Queries on 3D Object Databases. VLDB Journal, 19(5): 603-632, 2010.

[16] Sarana Nutanong, Egemen Tanin, Rui Zhang. Incremental Evaluation of Visible Nearest Neighbor Queries. IEEE Transactions on Knowledge & Data Engineering (TKDE), 22(5): 665-681, 2010.

[17] Rui Zhang, H. V. Jagadish, Bing Tian Dai, Kotagiri Ramamohanarao. Optimized Algorithms for Predictive Range and KNN Queries on Moving Objects. Information Systems, 35(8): 911-932, 2010.

[18] Rui Zhang, Jianzhong Qi, Dan Lin, Wei Wang, Raymond Chi-Wing Wong. A Highly Optimized Algorithm for Continuous Intersection Join Queries over Moving Objects. VLDB Journal, 21(4): 561-586, 2012.

[19] Tanzima Hashem, Lars Kulik, Rui Zhang. Countering Overlapping Rectangle Privacy Attack for Moving kNN Queries. Information Systems. 38(3): 430-453, 2013.

[20] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, Zhen He. Top-k Most Influential Locations Selection. Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM), Pages 2377-2380 , 2011.