2009

A Dynamic Insertion Approach for Multi-Dimensional Data Using Index Structures

Yong Shi
Kennesaw State University, yshi5@kennesaw.edu

A Dynamic Insertion Approach for Multi-Dimensional Data Using Index Structures

Yong Shi
Building 11, Room 3060
Department of Computer Science and Information Systems
Kennesaw State University
Kennesaw, GA 30144
yshi5@kennesaw.edu

ABSTRACT

Nowadays large volumes of data with high dimensionality are being generated in many fields. Most existing indexing techniques degrade rapidly when dimensionality goes higher. A large amount of data sets are time related, and the existence of the obsolete data in the data sets may seriously degrade the data processing. In our previous work[7], we proposed ClusterTree$^+$, a new indexing approach representing clusters generated by any existing clustering approach. It is a hierarchy of clusters and subclusters which incorporates the cluster representation into the index structure to achieve effective and efficient retrieval. It also has features from the time perspective. Each new data item is added to the ClusterTree$^+$ with the time information which can be used later in the data update process for the acquisition of the new cluster structure. To improve the performance of this index structure, we propose a dynamic insertion approach for time-related multi-dimensional data based on a modified ClusterTree$^+$, keeping the index structure always in the most updated status which can further promote the efficiency and effectiveness of data query, data update, etc. This approach is highly adaptive to any kind of clusters.

1. INTRODUCTION

Recently large volumes of data with high dimensionality are being generated in many fields. Many approaches have been proposed to index multi-dimensional data sets for efficient querying. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data sets efficiently since it may greatly increase the amount of data accessed for a query. We also observe that a large proportion of the data sets are time related, and the existence of the obsolete data in the data sets may seriously degrade the data processing. However, few approaches are proposed to implement the maintenance of the whole data sets to get rid of the obsolete data and keep the data sets always in the most updated status for the convenience and effectiveness of data processing such as query and insertion.

In our previous work[7], we proposed ClusterTree$^+$, an efficient index structure from clustering for multi-dimensional data sets. Clustering is an analysis technique for discovering interesting data distributions and patterns in the underlying data sets. Given a set of n data points in a d-dimensional space, a clustering approach assigns the data points to k groups ($k << n$) based on the calculation of the degree of similarity between data points such that the data points within a group are more similar to each other than the data points in different groups. Each group is a cluster. The ClusterTree$^+$ approach builds an index structure on the cluster structures to facilitate efficient queries.

In this paper, we propose a dynamic insertion approach for time-related multi-dimensional data based on a modified ClusterTree$^+$. The ClusterTree$^+$ index structure has features from the time perspective. Each new data item is added to the ClusterTree$^+$ with the time information which can be used later in the data update process for the acquisition of the new cluster structure. This approach guarantees that the ClusterTree$^+$ is always in the most updated status which can further promote the efficiency and effectiveness of data query and update. Our approach is highly adaptive to any kind of clusters.

The rest of the paper is organized as follows. Section 2 summaries the related work on index structure design. Section 3 presents the dynamic insertion approach of ClusterTree$^+$. Section 4 gives the conclusion.

2. RELATED WORK

Existing multi-dimensional tree-like indexing approaches can be further classified into two categories: data partitioning and space partitioning.

A data partitioning approach partitions a data set and builds a hierarchy consisting of bounding regions. Popularly used index structures include R-Tree, R*-tree, SS-Tree, SS$^+$-Tree and SR-Tree. An R-tree [4] is a height-balanced tree with index records in its nodes. The R*-tree [2] is an R-tree variant which incorporates a combined optimization of area, margin and overlap of each enclosing rectangle in the tree nodes. In contrast to the R-Tree, SS-Tree [1] uses hyperspheres as region units. SR-Tree [5] is an index structure

ACMSE '09 March 19-21, 2009, Clemson, SC, USA.

which combines the bounding spheres and rectangles for the shapes of node regions to reduce the blank area.

Space partitioning approaches recursively divide a data space into disjoint subspaces. K-D-B Tree and Pyramid-Tree are this type. The K-D-B Tree [6] partitions a d-dimensional data space into disjoint subspaces by $(d-1)$-dimensional hyper-planes which are alternately perpendicular to one of the dimension axes. The Pyramid-Tree [3]'s main idea is to divide the data space first into $2d$ pyramids, each sharing the center point as its peak (the tip point of a pyramid).

ClusterTree$^+$ is an efficient index structure built from clustering for multi-dimensional time-related data sets. The ClusterTree$^+$ has the advantage of supporting efficient data query for multi-dimensional data sets. It has two separate structures, one is ClusterTree*, the other is a modified B$^+$ tree – B^t tree.

The ClusterTree* is a hierarchical representation of the clusters of a data set. It has two kinds of nodes: internal and leaf nodes. The internal nodes are defined as:

$$Node : [Node\_id, \gamma, ot, nt, (Entry_1, Entry_2, \cdots, Entry_\gamma) \\ (m_{node} \leq \gamma \leq M_{node})],$$
$$Entry_i : (\mathcal{SC}_i, BS_i, SN_i),$$

where $Node\_id$ is the node identifier, γ is the number of the entries in the node, ot is the oldest time when data were inserted into that node or its descendants, nt is the newest time when data were inserted into that node or its descendants, and m_{node} and M_{node} define the minimum and maximum numbers of entries in the node. An entry is created for each subcluster of the cluster for which the current non-leaf node represents. In entry $Entry_i$, $\mathcal{SC}_i$ is a pointer to the i-th subcluster, BS_i is the bounding sphere for the subcluster and SN_i is the number of data points in the i-th subcluster.

The leaf nodes are defined as:

$$Leaf : [Leaf\_id, \gamma, ot, nt, (Entry_1, Entry_2, \cdots, Entry_\gamma) \\ (m_{leaf} \leq \gamma \leq M_{leaf})],$$
$$Entry_i : (ADR_i, T_i, L_i),$$

where γ is the number of data points contained in the leaf node, and m_{leaf} and M_{leaf} are the minimum and maximum numbers of entries. $Entry_i$ contains the address of the data point residing at the secondary storage (ADR$_i$), the time information when the data point is inserted into the structure (T$_i$), and the link to the time data point in the B^t tree (L$_i$).

The B^t tree indexes on the time data which corresponds to the times the data were inserted into the structure. It originates from the B$^+$ tree with some modifications as follows:

- There is no minimum number requirement of entries in internal and leaf nodes so that there will be no cases of underflow. This corresponds to the nature of the B^t tree that the time data in it will be deleted in batches for the user-specified deletion requirement.
- In the leaf nodes, each entry has an extra field which is a link to the data point it is associated with in the ClusterTree*, thus we can traverse from the B^t tree back to the ClusterTree* efficiently.

An example of the ClusterTree$^+$ structure is shown in Figure 1. Due to the space limitation, only part of the structure is shown. From Figure 1 we can see that currently there are 9 data which are inserted at time 1, 2, 3, 4, 5, 6, 7, 9 and 10. At the leaf level, the first leaf node has three data entries

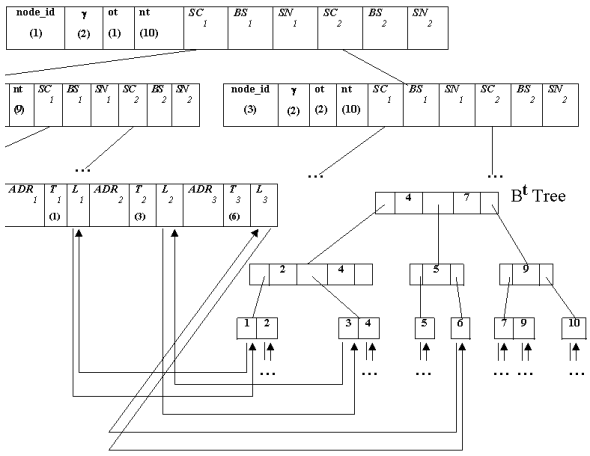

Figure 1: An example of ClusterTree$^+$ including ClusterTree* and B^t tree.

which are inserted at time 1, 3 and 6. The L$_i$ fields of these entries of the leaf node are connected with the leaf nodes of the B^t tree which is on the bottom-right part of the Figure 1.

3. DYNAMIC INSERTION TO CLUSTERTREE$^+$

Dynamic insertion of high dimensional data becomes a critical issue in real high dimensional databases. Here we present a dynamic insertion approach for multi-dimensional data using a modified ClusterTree$^+$, keeping the structure always in optimal status.

A new coming data point in ClusterTree$^+$ can be classified into one of three categories:

- *Cluster points*: are either the duplicates of or very close to some data points in a cluster within a given threshold.
- *Close-by points*: are the data points which are neighbors to some points in the clusters within a given threshold.
- *Random points*: are the data points which are either far away from all of the clusters and can not be bounded by any bounding sphere of the ClusterTree$^+$, or might be included in the bounding spheres of the clusters at each level, but they do not have any neighboring cluster points within a given threshold.

Different insertion strategies are designed for each type of new data points. The neighbor information of the new data point is available by applying the query process of the new data point on the ClusterTree$^+$ so that the new data point can be classified into one of the three categories mentioned above. We can directly insert the *cluster points* into a certain leaf node of the structure without any modification of the tree structure. As for the *close-by points*, they will be inserted into a certain leaf node of the structure which contains its nearest neighbors, but the radius and centroid of the certain node should be slightly adjusted. If a new coming data point is a *random point*, there are two cases: if absolutely no nodes contain it, it will be collected for creating another ClusterTree$^+$ later; if it can be bounded by bounding spheres of some clusters in the ClusterTree$^+$, but they

don't have any neighboring cluster points in it, it should be treated differently.

For the latter case of the random points, we use *maximum inclusion depth* to represent the lowest level of the subclusters whose bounding sphere contains the random point. We here propose a *delayed insertion* approach which stores a single data point without changing the radius and centroid of any node in the ClusterTree$^+$, and optimizes the new inserted data points when the amount of random points reaches a certain threshold.

However, there are two major problems in the random points processing mentioned above:

- the threshold of the amount of random points to optimize the current cluster is hard to decide for the complicated cases of the clusters. In a cluster, if the new coming random points are all scattered sparsely in the bounding sphere of the cluster, the original subclusters still can represent the correct status of this cluster efficiently and effectively even though the amount of the new coming random points is very large. On the other hand, if the new coming random points are close to each other, the status of the current cluster should be updated even though the total amount of the new random points is not very large. Thus the threshold could not be simply defined as a certain percent of the whole data points in the current cluster.
- the cluster reorganization process is quite time-consuming when more nodes and data points are involved, and the reorganization could be recursively propagated upward to the top node of the whole tree structure. Thus we should avoid the reorganization cases as much as possible.

According to these concerns, we focus on the dynamic insertion of the second case of the *random points*. The random points mentioned below are of this kind.

There are two ways to implement the dynamic insertion issue in our structure:

- based on the partition of the bounding sphere of the current cluster. It is possible that we separate the "space" of the cluster into several parts which have different possibility of welcoming new coming random points. Of course, the "welcome possibility" of those areas occupied by some subclusters is 0 since those new data points in such areas are either cluster points or close-by points. Based on the possibility level of different areas and some presumptions, we can predict the positions of the new coming data points and further determine if current inserted new data point is an outlier or one which should belong to a cluster. The problem of this approach is that the bounding spaces of those clusters containing high dimensional data points are normally very sparse, so the efficiency and effectiveness can not be guaranteed.
- based on the new coming random points themselves to improve the performance of dynamic insertion. This is a more reasonable and practicable one. Our design is based on this solution.

Definition (pseudo-cluster): a group of the random points in a certain cluster which are close to each other. This term is helpful for the further analysis and processing of the dynamic insertion.

In order to design our dynamic insertion approach, we modify the structure of ClusterTree*. It has two kinds of nodes: internal and leaf nodes. The internal nodes are defined as:

$$\begin{aligned} Node &: [Node\_id, \gamma, (Entry_1, Entry_2, \cdots, Entry_\gamma), \\ &\quad RPS, CM(m_{node} \leq \gamma \leq M_{node})], \\ Entry_i &: (\mathcal{SC}_i, BS_i, SN_i), \end{aligned}$$

where *Node_id* is the node identifier, γ is the number of the entries in the node, m_{node} and M_{node} define the minimum and maximum numbers of entries in the node. An entry is created for each subcluster of the cluster for which the current non-leaf node represents. In entry $Entry_i$, $\mathcal{SC}_i$ is a pointer to the i-th subcluster, BS_i is the bounding sphere for the subcluster and SN_i is the number of data points in the i-th subcluster.

The dynamic insertion in the internal nodes introduces RPS and CM. RPS is the structure containing the random points information:

$$\begin{aligned} RPS: &\quad [RP_1, RP_2, \cdots], \\ RP_i: &\quad (ADR_i, PC_i), \end{aligned}$$

where ADR_i is the address of the random point residing on the secondary storage, and PC_i is the pseudo-cluster number it is in. We will explain it later.

The CM is the correlation matrix of subclusters and pseudo-clusters of the current cluster. It stores the amount of common neighbor data points of each (subcluster, pseudo-cluster) pair and each (pseudo-cluster, pseudo-cluster) pair. Also it stores the amount of random points each pseudo-cluster contains and the amount of "original" data points each subcluster contains.

The leaf nodes have the same structure defined in Section 2. If a random point is in a leaf node, it can be inserted into it directly without delay since the leaf node does not contain subclusters which need to be readjusted.

In the dynamic insertion process, we should avoid reorganization as much as possible since it is very time-consuming. When a new random point comes to the suitable subcluster with its maximum inclusion depth, it is firstly inserted into this subcluster node's RPS structure with the pseudo-cluster number of 0. If it is close enough to a previously random points, a new pseudo-cluster is created including both of them; if it is close enough to several random points, it can be a common neighbor point of more than one pseudo-clusters. We inspect the correlations of the (subcluster, pseudo-cluster) pairs and those of the (pseudo-cluster, pseudo-cluster) pairs. If the amount of the common neighbors of two sides exceeds a certain threshold, we regard these two structures as being ready for mergence.

4. CONCLUSION

In this paper, we propose a dynamic insertion approach for multi-dimensional data using a modified ClusterTree$^+$ which can efficiently support the time-related queries and user-specified deletions. The ClusterTree$^+$ can keep the data sets always in the most updated status to promote the efficiency and effectiveness of data query and update. Further experiments will be conducted to demonstrate the efficiency and effectiveness of dynamic insertion using ClusterTree$^+$.

This approach can be helpful in the fields of data fusion where the data evolve dynamically and regular approaches

often fail to solve the problem of keeping a certain system always containing the most updated data. It can also dynamically supervise the data status of the system and efficiently get rid of obsolete data, and at the same time, reorganize the structure of the data sets when necessary.

5. REFERENCES

[1] White D.A. and Jain R. Similarity Indexing with the SS-tree. In *Proceedings of the 12th Intl. Conf. on Data Engineering*, pages 516–523, New Orleans, Louisiana, February 1996.

[2] Beckmann N. and Kriegel H.P. and Schneider R. and Seeger B. The R*-tree: an Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[3] B. C. Berchtold S. and K. H. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 142–153, Seattle, Washington, 1998.

[4] Guttman A. R-Trees: A Dynamic Index for Geometric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[5] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 369–380, Tucson, Arizona, 1997.

[6] J. Robinson. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 10–18, Ann Arbor, MI, Apr. 1981.

[7] Yong Shi and Aidong Zhang. Dynamic clustering and indexing of multi-dimensional datasets. In *4th International Conference on Information Fusion*, 2001.