International Research Journal of Computer Science (IRJCS)
Issue 12, Volume 4 (December 2017)
ISSN: 2393-9842
www.irjcs.com

EFFICIENT INDEX FOR A VERY LARGE DATASETS WITH HIGHER DIMENSION

Dr.S.THABASU KANNAN,
Principal, Pannai College of Engineering & Tech,
Sivagangai – 630 561, Tamilnadu, India
thabasukannan@gmail.com,

N.MANGALAM
Research Scholar & Lecturer, Department of Computer Science,
Raja Doraisingam Govt Arts College, Sivagangai - 630 561
mangalamkumarphd@gmail.com

Manuscript History
Number: **IRJCS/RS/Vol.04/Issue12/DCIS10080**
DOI: 10.26562/IRJCS.2017.DC1S10080
Received: 10, November 2017
Final Correction: 20, November 2017
Final Accepted: 02, December 2017
Published: December 2017
Citation: KANNAN, D. & N.MANGALAM (2017). EFFICIENT INDEX FOR A VERY LARGE DATASETS WITH HIGHER DIMENSION. IRJCS:: International Research Journal of Computer Science, Volume IV, 01-06. doi: 10.26562/IRJCS.2017.DCIS10080
Editor: Dr.A.Arul L.S, Chief Editor, IRJCS, AM Publications, India

Abstract-- The main aim of this paper is to develop a new dynamic indexing structure to support very large datasets and high dimensionality. This new structure is tree based used to facilitate efficient access. It is highly adaptable to any type of applications. The newly developed structure is based on nearest neighbors' method with exception of linearly scan the very large datasets. The NewTree surely minimizes adverse effect of the curse of dimensionality. It means that the most existing indexing techniques degrade rapidly when dimensionality goes higher. The major drawback here is the retrieval of subsets from the huge storage system. The NewTree structure can handle very efficiently and effectively during adding new data. When the new data are added and the shape of the structure does not change. The performance of the newly developed structure can be evaluated with SR Tree, existing indexing structure. The results clearly show that the efficiency of the newly developed structure is superior in both time complexity and memory complexity than SR Tree.

Keywords: cluster; high dimension; very large datasets; index; curse of dimensionality;

I. INTRODUCTION

Current clustering and indexing algorithms are not designed to deal with the very large datasets in high dimensional spaces. When the dimensionality increases, the time and space complexity increases exponentially with the dimension, and there are no easily imagined geometric shapes. This is called *curse of dimensionality*. The performance degrades rapidly as the dimensionality increases. Overlapping becomes more serious when dimensionality increases. The problem with high-dimensional index structure is often tackled by requiring the user to specify the subspace for indexing. An index structure organizes the whole dataset to support efficient queries. Recently many applications require efficient access and manipulation of very large data sets with multi-dimensions. In bioinformatics, gene expression data extracted from the DNA microarray images form very large datasets with more dimensions. To build an efficient index for a very large datasets with higher dimensionality, the overall data distributions or patterns should be considered to reduce the affects of arbitrary inserting.

The detection of the cluster structures can be very useful to build an index structure for very large datasets with high dimensions to facilitate efficient accessing. But most of our existing algorithms can only represent the very large datasets statically. These approaches can't handle new data to be added efficiently. When the new data are added and the shape of the cluster changes, the same approach must be applied again on the whole dataset without discriminating the new data and the already existing data. This inflexibility greatly limits the applications which have new data to be added frequently.

II. TREE INDEX STRUCTURES

The R-Tree: It supports the nearest neighbor search for low-dimensional datasets. It is a height-balanced tree with index records in its nodes. Here the internal nodes contain pointers to their children and the leaf nodes contain pointers to data objects. These are very useful in storing very large amounts of data on disk. It provides a convenient way of minimizing the number of disk accesses made. The main disadvantage is that the bounding boxes (rectangles) associated with different nodes may overlap. During retrieval, instead of following one path, we might follow multiple paths down the tree, although some branches do not contain the relevant result.

The SS-Tree: It is a similarity indexing strategy for high dimensional datasets. It uses hyper-spheres as region units. Each hyper-sphere is represented by a centroid point, which is the average value of the underlying data points, and a radius large enough to contain all of the underlying data points. Queries are very efficient because it only needs to calculate similarity between a region and the query point. Compared with the R_ -Tree, it has higher fan-out because the hyper-spheres for regions require half the storage space of the hyper-rectangles. It outperforms the R-Tree for dimensionality increases.

The SR-Tree: It is used to combine the bounding spheres and rectangles for the shapes of node regions to reduce the blank area. The region for a node in this tree is represented by the intersection of a bounding sphere and rectangle. So the overlapping area between two sibling nodes is reduced for high dimensionality. It takes the advantages of both rectangles and spheres, and enhances the query performance remarkably. However, the storage required is larger than the SS-Tree because the nodes need to store the bounding rectangles and bounding spheres. It requires more CPU time and more disk accesses than the SS-Tree for insertions.

III. CONSTRUCTION OF THE NEWTREE

Here we present a novel dynamic indexing approach to provide a compact cluster representation to facilitate querying based on clusters. The NewTree is a hierarchy of clusters and sub clusters, which integrates the cluster presentation into the index structure to achieve effective and efficient retrieval. It tries to reduce the adverse effect of the curse of dimensionality. Our cluster representation is highly adaptive to any kind of clusters and can detect new trends in the data distribution. NewTree can support the retrieval of the nearest neighbors effectively without having to linearly scan the high-dimensional dataset. It organizes the data based on their cluster information from coarse level to fine, providing an efficient index structure on the data according to clustering.

Algorithm for generate-sub-cluster

Input: k-medoid set M for a cluster C, |C|=n
Output: k semi-balanced sub-clusters $\{SS_1, SS_2, ..., SS_k\}$ for C

I. For each o ε C, 1<=j<=n
   a. Calculate its close medoid m_i
   b. Increase the number of points in SS_1, i.e $|SC_i| = |SC_i|+1$;
II. Get the maximum sub-cluster SC_i, where any l<=i<=k, $|SC_i| <= |SC_l|$
   If $|SC_i| <= n/2$ then return; else go to step III.
III. Decrease the number of data points in SCi as follows:
   a. d_{med} = SELECT(D(SCi), min($n/2$);
   b. for each data point o ε SC_i
      if $d(0,m_1) > d_{med}$
      x = choose-other-medoid (M,0)
      put 0 in SC_x, and $|SC_x| = |SC_x|+1$;

Here the average time complexity of SELECT is linear. Let *D(SC₁)* be the array of the distances between the data points in SC_1 and the medoid $\mathbf{m_l}$, the median distance d_{med} = SELECT (D(SCi), min(n/2)).The *Choose-Other-Mediod()* is used to pick a new subcluster for a data point. The selected subcluster should be as close as possible to the data point. Thus the above algorithm can guarantee to generate *k* subclusters whose sizes are all smaller than $n/2$

Algorithm to Build_NewTree

Input : β clusters {C1, C2,...C_β}

Output: Index structure for the β clusters

A. Generate a root to represent all clusters
   Create an entry E_i for each cluster C_i, add them into the root node;
   Push the root node into Stack
B. If Stack is not empty, current Node = pop(Stack) and Goto Step C;
   Else return;
C. For each entry $Entry_i$ in current Node, where 1 <= I <= γ
   If Entry.SN <= Page-size then
   a. Create a leaf node $child_i$ for $Entry_i$
   b. Let the pointer SC_i in $Entry_i$ point to $child_i$
   c. Save data points into disk pages.
   Else
   a. Create a non-leaf node $child_i$ for $Entry_i$
   b. Let the pointer SCi in $Entry_i$ point to $child_i$
   c. Generate k-medoids
   Call generate_sub_cluster to get k-subclusters for $child_i$
   d. Create an entry for each sub_cluster and add them to $child_i$
   e. Push $child_i$ into stack
D. Goto step B

In the above algorithm, *page-size* is determined by the disk block size (M) and the size of a data point (□), where *dimensionality X sizeof (element of a data point)*. It can be calculated by the formula: *pageSize = M/* Stack is used to store the nodes. When the stack is not empty, the node on top of the stack will be popped. The child nodes will be created for each of the entries belonging to the popped node. If some of the child nodes need to be further split, then they will be pushed into the stack. When the stack is empty then all of nodes are processed, the procedure of creating the NewTree is finished.

Height of NewTree: This is based on balanced partitioning algorithm. In step C-*c,* we can generate *k* roughly balanced subclusters whose sizes are no more than half of the parent clusters. After this algorithm is applied *O(*log*N)* times, where *N* is the size of the whole dataset, the number of the data points in each leaf node becomes equal to or less than the constant *page-size*. Thus, the maximum height of the NewTree is *O(* log*N).*

Time Complexity: During the growth of the NewTree, we need to scan the whole dataset *O(k)* times for each level of the NewTree. The height of the NewTree in the worst case is *O(*log*N)*_ . Thus, the time complexity to construct the NewTree is *O(k.N.*log*N)*. In the average case, the time complexity is *O(k.N.*$\log_k$ *N)*, since the height of the NewTree is *O(*$\log_k N$*)*.

IV. PERFORMANCE EVALUATION

Its main intention is to estimate the efficiency and effectiveness of the NewTree with the existing structure SRTree. The efficiency of the new algorithm will be evaluated by the parameter of time scaling with the dimensionality of data space and the size of the databases. We will measure the performance behaviors with varying number of data points. We will determine the influence of the data space dimension on the performance of query processing. We have conducted comprehensive experimental analysis to evaluate the performance of the NewTree. We measure the performance of constructing the NewTree and the SR-Tree under the same conditions. Figure 1 and Figure 2 show the scalability as the size of the datasets increases from 10,000 to 6,00,000 data points. The average time complexity of constructing a NewTree is *O(á k.N.*log*k N),* When *k = 10; and N* < 60,000, the time complexity degrades to be linear.

As expected in Figure1, the running time scales linearly with the size of the datasets. In Figure1 the speed-up factor of the NewTree over the SR-Tree is the CPU and disk I/O time its range between 2.5 to 8.1. We can also see the increasing scale of building NewTree is much smaller than that of SR-Tree. The reason is that the NewTree only needs to calculate the bounding sphere, while the SR-Tree has to calculate the minimum bounding hyper-rectangles and spheres.

Figure2 shows the scalability as the dimensionality of the datasets increases from 5 to 55. The datasets in Figure 2, each have 5,00,000 data points. The curves of the NewTree exhibit linear behavior with respect to dimensionality. For Figure 2 the speed-up factor is the CPU and disk I/O time its range between 4 and 18. The construction time of the SR-Tree is highly affected by dimensionality, while the NewTree is not impacted by dimensionality at all. The linear increment comes from the distance computation for the increasing dimensionality.

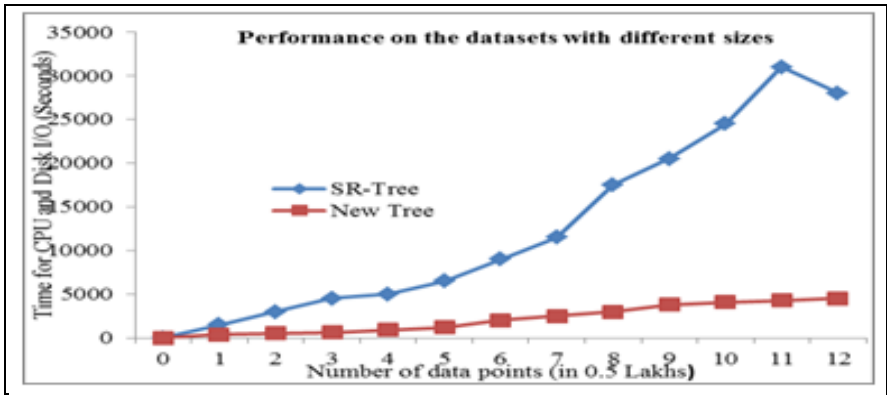

Figure 1: Performance of constructing NewTree and SR-Tree for 40-dimensional datasets

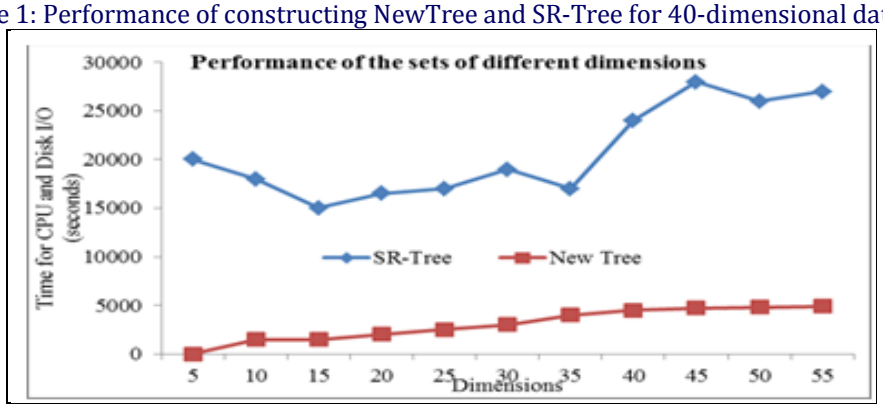

Fig2: Performance of NewTree and SR-Tree for 5,00,000 data points under different dimensions

Figure 3 gives the overall construction time of the NewTree under different dimensionality and dataset sizes. Each curve in Figure 3, represents CPU and disk I/O time of building NewTree for 13 datasets with different dataset sizes and the same dimensionality. Each curve in Figure 3 represents CPU and disk I/O time of building NewTree for 11 datasets with different dimensionality and the same dataset size, and it is labeled by the dataset size. They both show that the constructing time is linear in relation to dimensionality and dataset size.

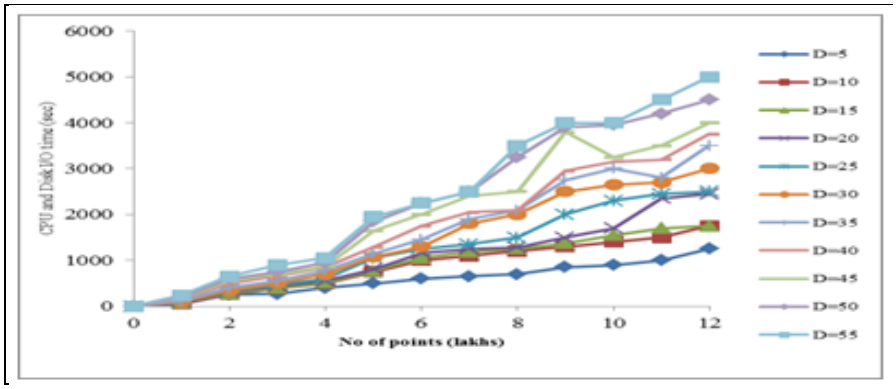

Figure 3: Performance of constructing NewTree for different data sizes

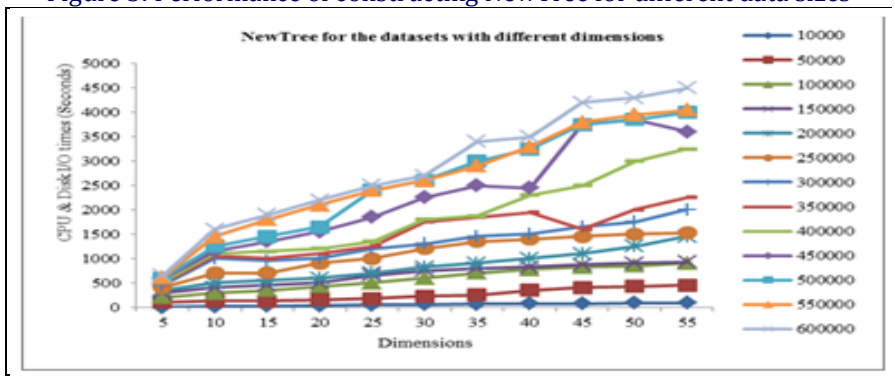

Figure 4: Performance of constructing NewTree

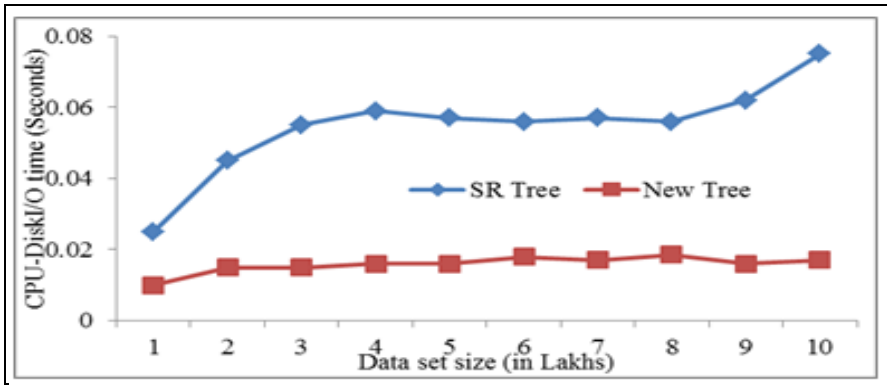

Figure 5: CPU-Disk I/O time of NewTrees and SR-Trees of Datasets with Different Sizes.

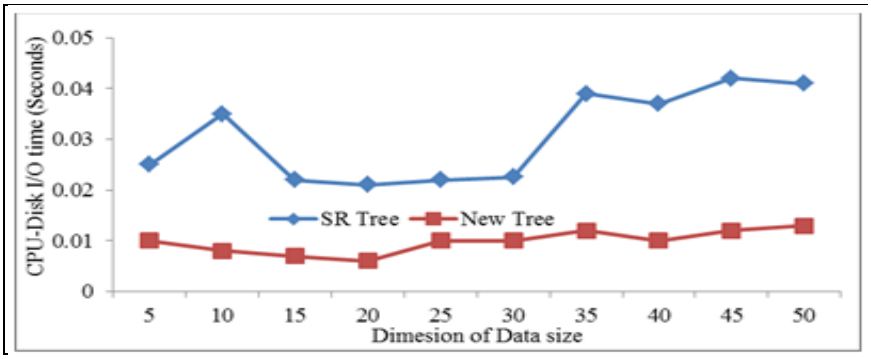

Figure 6: CPU-Disk I/O time of NewTrees and SR-Trees

V. CONCLUSION

Here we provided algorithms for efficient classification and retrieval in large high dimensional databases. To make the retrieval more efficient, we built a very efficient index structure, named *NewTree*. The NewTree can be very flexible. If the datasets are clustered and assigned semantic meanings for each cluster, an index structure can be built based on the available cluster information. Clustering provide powerful tools for understanding the natural pattern and distribution of large datasets. Identifying such underlying information in a dataset is extremely helpful in the management of all of its data objects.

The NewTree is a data partitioning algorithm. The NewTree decomposes a cluster into subclusters based on a *k*-medoid partition algorithm. Then the decomposition can be performed recursively until the nodes can be fit into one disk page. We have also presented an approach to dynamically reconstruct the NewTree when new data points are added. By integrating the cluster representation with the index structure, the NewTree supports a multi-dimensional index for the nearest neighbor search, which can effectively exploit the structure of the dataset.

We also conducted the performance comparison with the SR-Tree, which is one of the newest index structures based on data partitioning. The overall results show that the NewTree outperforms the SR-Tree for the datasets with various sizes and dimensionalities. The experiments also demonstrated that the clustering and further subclustering approach could greatly reduce the fraction of data that needs to be searched to find the nearest neighbors. Disk I/O speed has been improved over last ten years, but the performance gap between processors and magnetic disks will continue to widen. Disk I/O time is always the bottleneck to response time.

The *curse of dimensionality* is the major challenge of the high-dimensional data mining. So the data mining results also suffer the adverse effects of the curse of dimensionality. There is no approach which can completely solve this problem. Any existing approach can only reduce the effect of the problem and can then be applied to certain areas and applications. Also we should be concerned with the practical applicability of high-dimensional indices and searching strategies for a given complex database residing in an infrastructure. Our final goal of the research is to integrate new indexing techniques into the existing large database systems, and provide the users a friendly, efficient interface.

REFERENCES

1. S. Berchtold, C Bohm, and H. Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pages 142–153, Seattle, Washington, 2010, 98. 185

2. Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The SR-tree : An index structure for high-dimensional data. In Proceedings of 22th International Conference on Very Large Data Bases, VLDB'12, pages 28–39, Bombay, India, 2012.
3. N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The SR-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 2011.
4. K. Chakrabarti and S. Mehrotra. The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. In Proceedings of the 16th International Conference on Data Engineering, pages 440–447, San Diego, CA, February 2012.
5. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In Proceedings of the ACM SIGMOD conference on Management of Data, pages 73–84, Seattle, WA, 2011.
6. R. Kurniawati, J. S. Jin, and J. A. Shepherd. The SS+-tree: An improved index structure for similarity searches in a high-dimensional feature space. In Proceedings of SPIE Storage and Retrieval for Image and Video Databases, pages 13–24, February 2012.
7. N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pages 369–380, Tucson, Arizona, 2013.
8. J.T. 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, April 2013.
9. D.A. White and R. Jain. Similarity Indexing with the SS-tree. In Proceedings of the 12th Intl. Conf. on Data Engineering, pages 516–523, New Orleans, Louisiana, February 2014.
10. D. Yu, S. Chatterjee, G. Sheikholeslami, and A. Zhang. Efficiently detecting arbitrary shaped clusters in very large datasets with high dimensions. Technical Report 98-8, State University of New York at Buffalo, Department of Computer Science and Engineering, November 2013.
11. Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pages 103–114, Montreal, Canada, 2012.

Prof. Dr. S. Thabasu Kannan has been working as Professor and Principal in Pannai College of Engineering and Technology, Sivagangai and rendered his valuable services for more than two decades in various executive positions. He has published more than 100 research level papers in various refereed International/National level journals/proceedings. He has authored for 11 text/reference books on IT domain. He has received 11 awards in appreciation of his excellence in the field of research/education. He has visited 6 countries to present his research papers/articles in various foreign universities. He has been acting as consultant for training activities for various organizations in and around Madurai. His area of interest is Big data applications for bioinformatics domain. Under his guidance 8 Ph.D scholars pursuing and more than 150 M.Phil scholars were awarded. His several research papers have been cited in various citations.

N.Mangalam has been working as a Lecturer in Department of Computer Science in Raja Doraisingam Govt Arts College, Sivagangai for last 12 years. Her area of interest is Big Data Analytics. She is pursuing her Ph.D degree in Computer Science under the guidance of **Dr.S.Thabasu Kannan.** She is handling various subjects related to Computer Science for both UG and PG.

