Singaporean Journal of Scientific Research(SJSR)
International Journal of Technology and Engineering System(IJTES)
Vol.8.No.3 2016 Pp.176-189
available at :www.iaaet.org/sjsr
Paper Received : 08-03-2016
Paper Accepted: 19-04-2016
Paper Reviewed by: 1.Prof. Cheng Yu 2. Dr.M. Akshay Kumar
Editor : Dr. Chu Lio

AN EFFECTIVE CLUSTERING METHOD WITH SEARCH STRUCTURE FOR LARGE MULTIDIMENSIONAL DYNAMIC INDEXES

[1] Sunil Sunny Chalakkal MCA, M.Phil, [2] Dr. M. Rajalakshmi , [3] Dr. R. Vijayakumar

ABSTRACT: One of the most elemental data analysis techniques is clustering, which is extensively employed in numerous analytic applications. On the other hand, recognization and evaluation of multidimensional clustering results, in particular the cluster values and semantics, had become a complicated task. In case of huge and multifaceted data, to evaluate the cluster quality, tremendous level of statistical information about the clusters is frequently required. Simultaneously in order to recognize the significance of the clusters , comprehensive display of multidimensional attributes of the data are mandatorily required. The paper describes the designed the Distributed Weighted Possibilistic C-Means (DWPCM) algorithm based on MapReduce, which intends at enhancing the cluster speed. Also, the Cluster Tree++ indexing scheme is formulated for the purpose of distributed multi dimensional point data. The proposed indexing structure can be observed as integrated features of the Cluster tree+ with Simple prefix B+ tree in order to diminish the memory utilization during the process of indexing. All the incoming original data item along with the time information is appended to the Cluster Tree. This information is used during the process of data updating for acquiring the new cluster structure. Therefore the cluster tree is always in the updated position .Hence this scheme ensures that it tremendously supports the adaptation to any type of clusters. The efficiency of data insertion, query and update process is improved very efficiently. The experimentation is done using real-world datasets like CAR, HYD and TLK, which clearly demonstrate that the proposed scheme can produce effective results with better scalability and selectivity with very low memory usage.

KEY WORDS: Indexing, cluster representation, nearest-neighbor search, distributed Multi-dimensional data sets, Cluster Tree++ indexing approach.

1. INTRODUCTION

During the process of index structure, it arranges the complete dataset to support well-organized inquiry. In recent times, several applications need effective access and handling of large-scale multi-dimensional datasets. For instance, several features obtained from the image datasets are high-dimensional vectors [1]. Moreover, in the area of bioinformatics, the huge-scale multi-dimensional datasets are formed by the data of gene expression attained from the microarray images of DNA.

Major challenging disputes during indexing of the datasets for well-organized querying are high dimensionality and massive size of these datasets. The design of indices to maintain high-dimensional data search turns out to be an active research area. Several schemes have been formulated for the purpose of indexing multi-dimensional datasets. These schemes can inventively encourage the search for nearest neighbor in comparatively low dimensional datasets [2]. Currently, the majority of studies in index design [3] concentrate on high-dimensional datasets. In spite of the dynamic data point introduction by the indexing strategies, their functionality might be misrepresented. The trouble with a dynamic index structure is that the recently-popped in data points possibly will source for the structure no longer competently handles the entire dataset. It can significantly increase the quantity of data retrieved for a query.

At the same time as the dimensionality rises and the dataset are very huge, the effectiveness for queries becomes a key issue. For creating a proficient index for the successful functioning in large dataset with higher dimensionality, on the whole data distributions or patterns must be taken into account to diminish the influences of subjective inserting. In [4, 5], to optimize an obtainable dynamic index, an efficient "packing" schemes are invented by considering the distribution of data. Clustering is a type of scrutiny method for the purpose of determining the concerned patterns and distributions of data and in the dataset. Provided a collection of $\boldsymbol{n}$ data points in a $\boldsymbol{d}$-dimensional space, a clustering scheme allocates the data points to $\boldsymbol{k}$ groups ($\boldsymbol{k}<<\boldsymbol{n}$) in accordance with the computation of the level of analogous among data points in order that the data points inside a set are extremely comparable to one another than the data points in other sets known as cluster. Supervised clustering schemes usually need a k number of sets, as a priori, however it is inapplicable for unsupervised clustering schemes. The discovery of the cluster organizations is extremely important to construct an index structure for implementation in high-dimensional datasets to assist efficient inquiries [6]. Numerous clustering schemes have been discussed in the review survey [7].

On the other hand, the majority of the clustering schemes can merely group the datasets statically, indicating that the creation of clusters can just be off-line alone. These schemes cannot competently manage created data to get attached. In the scenario when the created data is inserted and the outline of the cluster transforms, the clustering method will be applied again on the complete dataset devoid of discerning the created data and the previously btainable data. This inflexibility significantly controls the applications that have original data to be added recurrently. Out of all those schemes, the ClusterTree [8] is the initial work to construct proficient index organization for clustering for high-dimensional datasets. It is a new dynamic indexing scheme offers a solid cluster representation to level the progress of efficient querying. ClusterTree is a tree of clusters and subclusters which integrates the cluster arrangement into the index construction to accomplish successful and well-organized attainment. The cluster is extremely adaptive to any category of clusters and can discover the new development of the data distribution. The spatially nearer data points are logically clustered collectively in the ClusterTree.

The linearly search of the high-dimensional dataset is not necessary for the ClusterTree for efficiently maintaining the retrieval of the adjacent neighbors. Till now, the ClusterTree is the initial work towards constructing well-organized index structure from clustering for high-dimensional datasets.

Currently, a huge amount of the dataset is time interrelated, and dataset processing may get humiliated dangerously due to the continuation of the outdated data in the dataset. To get rid of the outdated data and maintain the dataset constantly in the most updated status for the ease and efficient processing of dataset like query and insertion and updating. A few schemes are formulated to execute the maintenance of the complete set of data. Here a newly proposed ClusterTree++ indexing structure which has novel characteristics based on time point of view. Every incoming original data item along with the time information is appended to the Cluster Tree. This information is used during the process of data updating for acquiring the new cluster structure. This scheme ensures that, Cluster Tree++ is constantly updated and tremendously supports the adaptation to any type of clusters. The efficiency of data insertion, query and update process is improved very efficiently.The process described in the paper is structured as follows. The related work on index structure designs including the Cluster Tree and Clustering techniques summarized in Section 2. The new clustering method applicable to distributed multidimensional datasets is described in Section 3. The processing of Cluster Tree is described in section 4. The conclusion is given in Section 5.

2. RELATED WORK

In the area of data mining clustering is a key method and an important research area for researchers. In clustering a set of objects is divided into clusters in order that the patterns of the objects in that group are extremely similar to each other than the objects in the other clusters. For larger databases different clustering schemes are available such as KMEANS[9]. CLARANS[10],BIRCH[11],CURE[12],DBSCAN [13],OPTICS[14],STING[15] and CLIQUE, The above mentioned schemes can be partitioned into different schemes. The most renowned methods are partitioning, hierarchical and density based. Every method tries to challenge the clustering problems for data in large databases. Whereas most of them are not very effective for large databases. For density based clustering schemes has the intention to find clusters of arbitrary shape databases with noise, the cluster is defined as a high density area segregated with low –density regions in data space. DBSCAN-Density Based Spatial Clustering is a scheme on density based clustering.

In terms of size multidimensional databases are normally very large. Large volumes of data requires an effective well patterned access techniques to facilitate it ,due to unproductive access method the working on complicated data representation and reasoning might be lost. For different applications, the accumulation of multidimensional data is maintained at a minimum level since it needs better care to retain its efficiency. The intensity of maintenance can be complicated to fulfill and difficult to maintain which might be resulting to poor reaction times [16], where the performance reduces drastically when the count on dimension rises, finally they do not grow into higher dimensions [17].With the increase in dimensions the complications on the multidimensional data increases. Once the count of dimensions increases to three or four additional trouble rises which reduces the access techniques efficiency.

For more dimensionality such as 10 and more, the indexing methods available do not work out in an optimal way, which states that a sequential scan of the table happens to be quicker which means a lesser amount of time/less block access for the index to respond most queries [18].

Data becomes more sparse and the distance metrics lose their meaning in case of higher dimensional space.In case of more dimensions count such as 10-15 dimensions, when its not divided or segmented can develop into large since there is no sufficient data to initiate splitting of all dimensions. Space is wasted on unwanted information on the unsegregated dimensions.

During the time of higher dimensional space, data turns out to be extremely sparse and distance metrics lose their meaning. In case of more than 10-15 dimensions, the amount of dimensions that are not partitioned can develop into large as there are simply not adequate data to necessitate all dimensions to be split. This makes nodes to waste space on unnecessary information on these unsegment dimensions. The property of selectivity is not supported and the internal nodes can provide a small selectivity over the index tree. Handling large number of dimensions can be done by different dimensionality reduction schemes where the original space is reduced to a lower dimensional subspace{19],Additional resources and the original data are needed for the transformations of data or queries. Therefore just dimension reduction alone is not a feasible solution in different application areas and there raises a requirement for a better access technique to deal with medium to high dimensional vector. Different types of methods have been developed with the objective of effective management of multidimensional data. A key scheme Space Filling Curve (SFC) schemes are introduced.

CPU utilization and high level of overlapping among pages and the query interval are the major drawbacks of the SFC schemes.Space filling curve is incorporated by the UB-Tree[20],the B+Tree generates primary index for multidimensional data. Modifications to the kernel for the need of integration are shortfalls of UB-Tree, similar to other SFC the segments are not hyper cubic and will possibly correspond to disjoint space. The K-D-Tree is a most well known dimensional point data structures which also has alternates such as the HB-TREE [22], the BDTREE [23],the hybrid tree[24] and the quad –Tree.

The most similar deficiency to the entire K-D-Tree schemes is that for particular distributions, the hyperplane for partitioning the data objects equally is not found. For effective organization of temporal data some methods are there which facilitates the process of integrating with commercial database management systems [25].but the problem is that these methods cannot effectively support high dimensional queries.

3. CLUSTERING-BASED INDEXING STRUCTURES

The major concept of Clustering-based indexing structures is initially make use of the clustering algorithms with the intention of clustering the data points, and subsequently utilize approximation at search phase in such a manner that search can be made on the derived clusters which has the most chances of the closest neighbors of the query point .The common structural design of the clustering-based structures is given in Fig.1. Clustering-based indexing structures comprise two phases: clustering phase and search phase.

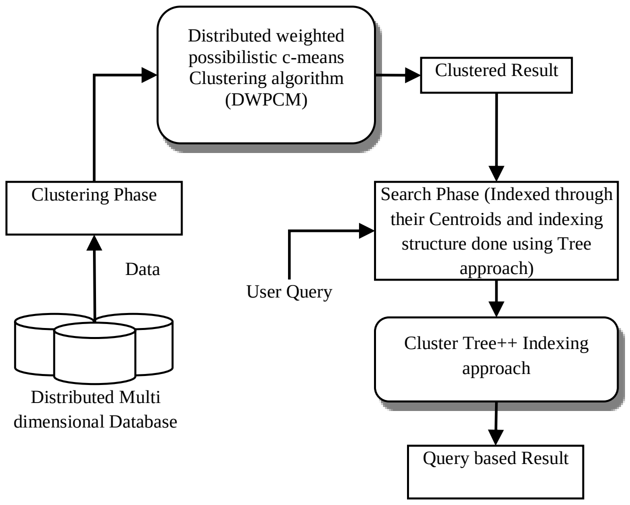
Distributed weighted possibilistic c-means Clustering algorithm (DWPCM)
Clustered Result
Clustering Phase
Search Phase (Indexed through their Centroids and indexing structure done using Tree approach)
Data
User Query
Cluster Tree++ Indexing approach
Distributed Multi dimensional Database
Query based Result

Fig.1. Proposed Work

3.1 Clustering phase

In this section, proposes a Distributed WPCM algorithm (DWPCM) in accordance with MapReduce. There are two major processes in this phase, they are calculating the degree of membership μ_{ij} and computing the clustering centers v_i. During the map phase, the Map function is used to compute the degree of membership μ_{ij}.

Table 1. Map function Algorithm

```
Algorithm 1. mapping (off_key, samp_value)

Problem: Given the global variable centers, the offset key and the sample value the map function
    algorithm calculates s_key' and s_value'

Algorithm

1.  Data Object is partitioned and sample instance is constructed from the sample value
2.  $\xi_i^{(t)}$ and $\lambda_i^{(t)}$ are calculated by Mapping function using equations 1 and 2
    2.1 Set min_Dist = Double.MAXIM_VAL;
    2.2 Set index_val = −1;
    2.3 Set $i = 0$
    2.4 Repeat steps 2.5 through 2.8 until $i <=$ to gcenters.length
    2.5 dist = CalculateDist(s_instance, gcenters[i]);
    2.6 Check whether dist < min_Dist, if no goto step 2.4
    2.7 Set min_Dist = dist;
    2.8 Set index_val = i; goto step 2.4
    2.9 Let index_val as s_key';
3.  Print c⟨s_key', s_value'⟩
4.  Perform combine (s_key, s_ Value)
5.  Do
6.  Set num_count = 0
7.  Repeat steps 7.1 through 7.3 while(V.hasNext())
    7.1 Compute the sample instance using V.next();
    7.2 Fill the array with those values
    7.3 Increment num_count by one;
8.  Let s_key as s_key';
9.  Compute s_value'
10. End Do
11. Print < s_key', s_list >
12. Call the procedure mapreduce(s_key', s_ list )
13. End
```

To revise cluster centers in parallel, two parameters, $\xi_i^{(t)}$ and $\lambda_i^{(t)}$, are introduced, where t indicates the serial number of data node. Following to the computation of the membership μ_{ij}, the Map function determines $\xi_i^{(t)}$ and $\lambda_i^{(t)}$ with the help of equation (1) and (2);

$$\xi_i^{(t)} = \sum_{k=1}^{n/p} w_k u_{ik}^m x_k \quad i = 1,2 \ldots c \qquad (1)$$

$$\lambda_i^{(t)} = \sum_{k=1}^{n/p} w_k u_{ik}^m \quad i = 1,2 \ldots c \qquad (2)$$

Table 2. Reduce function Algorithm

```
Algorithm 2.map_reduce(s_key' ,s_list )

Problem: Given s_key' ,s_list as input to map_reduce function, it reduces to the point centre v_i.

Algorithm

    1.  Initialize an array record.
    2.  Set num_count as 0.
    3.  Repeat steps 4 through 6 until the function V.hasNext() returns true.
    4.  Using V.next() the sample instance is constructed.
    5.  Fill those values to the array.
    6.  Increment num_count by num_count.
    7.  Array entries are divided by num_count.
    8.  Set s_key as s_key'.
    9.  s_value' is constructed.
    10. Print v_i using equation (3) .
    11. End.
```

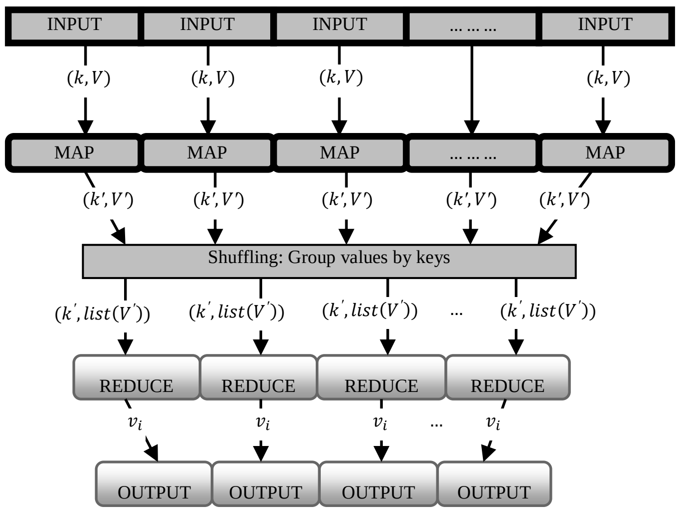

Fig.2.Map Reduce Programming Model

As a final point, the Map function outputs$c\langle key' , value'\rangle$, where c indicates the number of classes, key' indicates the identifier of the class and $value'$ indicates a vector that includes $\xi_{key}^{(t)}$ and $\lambda_i^{(t)}$. During the reduce phase, the Reduce function is intended to compute the clustering centers v_i. The input of the Reduce function is a key list, where key' indicates the identifier of the class and $list$ comprises the entire value's along with the same key' derived from the map function. The Reduce function is accountable for computing the cluster centers in accordance with the equation (3);

$$v_i = \frac{\sum_{t=1}^{p} \xi_i^{(t)}}{\sum_{t=1}^{p} \lambda_i^{(t)}} \tag{3}$$

Where, p indicates the quantity of the data nodes and i indicates the identifier of the class, which comprise the same explanation with $k\acute{e}y$.

3.2 Search phase

During this phase, initially the hierarchical structure of the Cluster Tree++ is introduced. Subsequently, a method is proposed for breaking down a cluster into multiple sub clusters with an algorithm to produce Cluster Tree ++ by means of braking down the clusters recursively. ClusterTree++ depends on the design of the ClusterTree+ and improves its ability to manage dynamic data insertions, queries and deletions. There are different methods to connect the time information with the original ClusterTree structure. The three different methods are: First one is, openly append time details into the ClusterTree+ as another dimension. Second, make use of an uncomplicated queue to process the time issue, The third one is to manage the time information an independent simple Prefix B++ tree similar structure is used. The key advantage of this method is that it makes the process of implementation easier and effective.The original algorithm can be slightly modified to maintain queries related to time details and deletions with reference to the time period indicated by users.

The disadvantage is that the clustering process results will considerably change when we directly take time details as an additional dimension of the data set. When adding the time dimensions two data points which are very close to each other might not even exist in the similar cluster, in view of the fact that the data are inserted into the indexing structure might be fairly far-away to each other. Subsequently ,the queries which are dependent on the data itself and its query results are corrupted. The concept of using an uncomplicated queue to process the time issue is easy.

However, as a linear structure, the effectiveness is the major complication. By means of an individual simple prefix B+ tree-similar structure to manage the time details can support both time-associated queries and time-unrelated queries, "simple prefix" points out that the index set encloses shortest separators, or prefixes of the keys more willingly than copies of the actual keys. In case of time-irrelevant queries, algorithms like those of the original ClusterTree are used. In case of time-related queries including range queries and by means of an individual B+-Nearest Neighbors queries, the intersection of searching result of both modified ClusterTree+ structure and B+ tree common structure can be used to obtain the ultimate result. These schemes can moreover efficiently handle user specified periodic deletions to throw away the outdated data in the dataset. As a result, the last category of scheme is selected to set up the ClusterTree indexing structure as the solution to solve the data update complication of high dimensional datasets.

Two independent structures are included, first one is a modified ClusterTree+ structure known as ClusterTree++, the additional simple prefix B+ tree. Hierarchical representation of the clusters is called as the Cluster Tree. The cluster tree includes two types of nodes namely internal and leaf nodes. The internal node is represented in the below format:

$$\boldsymbol{Internal\ Node}: \boldsymbol{Node_{id}, a, ht, ct, (Entry_1, Entry_2, \ldots}$$
$$\boldsymbol{(min_{node} \leq a \leq Max_{node})},$$
$$\boldsymbol{Entry_i: (SC_i, BS_i, SN_i)}$$

where $Node_{id}$ represents the node identifier, a indicates the quantity of the arrival in the node, ht represents the past time of when the data was introduced into the node or its descendants, and ct describe the present time when data are introduced into that node or its descendants, min_{node} and Max_{node} intimates the least and highest number of entries in the node. For every subclusters an entry is generated for which the current nonleaf node corresponds to. In case of entry $Entry_i$, SC_i shows a pointer to the $i-th$ sub clusters, BS_i indicates the bounding sphere for the sub cluster and SN_i indicates the amount of data points in the i-th sub clusters. The extreme final leaf nodes are given as follows:

$$\boldsymbol{Leaf\ Node}: \boldsymbol{Leaf_{id}, a, ht, ct, (Entry_1, Entry_2, \ldots \ldots Entry}$$
$$\boldsymbol{(min_{leaf} \leq a \leq Max_{leaf})},$$
$$\boldsymbol{Entry_i: (ADR_i, T_i, L_i)}$$

Where a indicates the number of data points enclosed in the leaf node, and min_{leaf} and Max_{leaf} indicates the lowest and highest amount of entries. The $Entry_i$ shows the address of the datapoint at the secondary storage includes the address of the data point exist at the secondary storage (ADR), the time related informations when the data point is introduced into the structure (T_i), and the connection to the time data point in the simple prefix B+ tree (L_i). For the simple prefix B+ tree indexes on the time data which is equivalent to the number of times the data were brought into the structure. It begins from the B+ tree with certain changes like: There is no lowest number requirement of entries in the place of internal and leaf nodes, in order that there will be no cases of underflow. This is equivalent to the character of the prefix B++ tree stating the time data in it will be removed collectively based on the user specified condition.

For the leaf nodes, all entry has an added field which is a interface to the data point it is connected with in the ClusterTree++. In this scenario, we can navigate from the simple prefix B+ tree back to the ClusterTree++ proficiently. The separators in the index set is lesser than the keys in the sequence set ==> Tree is even smaller. Consecutive insertions are not a better method since splitting and redistribution are reasonably costly and would be finest to make use of only for tree maintenance. Beginning from an arranged file, on the other hand, it is easy to place the records into sequence set blocks one by one, opening a new block when the current working block with fills up. Since, the changeover is done among two sequence set blocks, it is easy to determine the shortest separator for blocks. These separators are collected into an index set block that is constructed and added in memory until it is occupied. The benefits of filling a simple Prefix B+ Tree more or less constantly outweigh shortcomings related with chance of generating blocks that enclose very few records or very few separators.

A specific improvement is that the loading procedure goes more rapidly since: The output can be written in series; only one pass can be made over the data; No blocks required to be rearranged during the process. The major benefit after the tree is loaded is that the blocks are 100% occupied. Sequential loading generates a degree of spatial locality inside the file ==> Seeking can be reduced. The major three stipulations are: In case when blocks are divided in the sequence set, a fresh separator has to be introduced into the index set. When blocks are combined in the sequence set, a separator has to be eliminated from the index set. When records are re-allocated among blocks in the sequence set, the value of a separator in the index set have to be transformed.

Construction

The stages in the building of ClusterTree++ includes the production of ClusterTree++ and developing of simple prefix B+ tree in parallel. The developmet of the ClusterTree++ is same as the development of ClusterTree+ and tehn every internal node and leaf node has to set the *ct(current time)* and *ht* (*historic time*) as the current time.Since there is no information related to the insertion time of the original data points in the dataset, therefore it is mandatory to fix the information as a result it is essential to fix the insertion times of the entire original data points as the current one. When new data points are introduced into the structure it has to be recorded into the structure. Meanwhile,the current time datais incorporated by a leaf node using the simple prefix B+s tree. All the L fields of the entries in the leaf node of Cluster Tree+ has to be pointed to the generated leaf node in simple prefix B++tree.

Processing of the ClusterTree

Insertion,Query,Deletion are the most important processing of the Cluster Tree++

Insertion

For every new incoming data point, it's important to divide it into one of the three groups: *Cluster points*: they are the duplicates or extremely similar to particular data points in a cluster inside a specified threshold. *Close by points:* They are the data points which are certain points in the clusters within a specific threshold. *Random points*: They are the data points which are not neighbors and distant from the entire clusters and cannot be bounded, or even at every level they cannot be incorporated. But they do not mention any neighboring cluster points inside a specified threshold. finally, in agreement with the type of the new coming data point, its is required to apply the insertion algorithm of Cluster Tree to serially insert data point to a specific leaf node of Cluster Tree++ and with the insertion time in the T(time) field of the new entry of the leaf node, which includes the inclusion of time details into the simple prefix B++ tree. The L(link) is a link between the specific leaf node in Cluster Tree++ to the new entry the $L(link)$ field of the new entry in the specific leaf node in Cluster Tree++ to the new entry in leaf node.

Query

Queries are classified into two types based on time ,The first category is the time irrelevant queries which includes the range queries and Nearest neighbor queries, the second category is time related query which includes time related queries with certain time period constraint. For example, specific users may demand neighbors to a specific data point which are included into the structure in equal time as the insertion time of that data point. It is based on the original Cluster Tree query algorithm to solve the former category of queries.

Table 3.Algorithm- time-related queries

Problem: Given a data point and a time stamp, the algorithm calculates the set of data points

in ClusterTree which satisfies the query.

Algorithm

1. Calculate the set A of candidate data points in the ClusterTree++;
2. Locate an entry x in simple prefix B+tree where the time data of simple B+tree is close to the time stamp of query.
3. Locate the set of entries with specific threshold in time distance to the entry x in simple prefix B+ tree.
4. Construct a set B of equivalent data points in ClusterTree++ using the L field entries in simple prefix B+tree.
5. Intersection of sets A and B gives the resultant data sets.

Deletion

The irrelevant data has to be deleted from time to time from many systems. The data administrator might need to delete those data which are introduced to the system and in some situations the might want delete some date inserted at some stage in a specific period.

Table 4.Algorithm- time-related deletion1

Problem: Given a time stamp ts, the algorithm constructs a new ClusterTree after removing the outdated data.

Algorithm

1. Locate the entry x in the simple prefix B+tree where the time data of entry x is close to the time stamp ts.
2. Retrieve the entries, set A which are older than entry x from simple B+tree.
3. Locate the resultant set B of data points in ClusterTree++ using the L field entries in simple prefix B+tree
4. Recursively cut older entries than entry x from the simple prefix B+tree.
5. Set A is cut from simple prefix B+tree.
6. Set B is cut from ClusterTree++

In addition they can just point out to the data system to automatically fine-tune itself. The ClusterTree+ can support such user specified deletions.

Table 5.Algorithm- time-related deletion2

Problem: : Time stamp ts1, time stamp ts2 out puting the new ClusterTree specified time stamps ;

Algorithm

1. find the entry x in simple prefix B+ tree the time data of whose is right closest to the time stamp T1
2. find the entry y in simple prefix B+tree the time data of whose is left closest to the time stamp T2;
if the time data of entry x is newer compared to that of entry y,

exit;

3. get the set the entries in the simple prefix B+ tree
4. find the set b of similar data points in ClusterTree++ by means of the L field in the entries in simple prefix B+
5. cut those entries in the simple prefix B+tree
6. cut set a seen in the B simple prefix B+tree;
7. cut set b in the ClusterTree++.

Table 6.Algorithm: automatic adjustment

Problem: This recursive algorithm automatically adjust the new ClusterTree.

Algorithm

1. Recursively check each subcluster

```
2.  Do
    2.1 Check whether the gap between
        subcluster's not filed exceeds specific
        threshold.
        If yes either delete the complete
        subcluster or move the complete
        subcluster into
        the secondary memory until it no new
        data is reported. Also remove the
        equivalent
        entries from simple prefix B+tree.
    2.2 Check whether a subcluster's old
        density exceeds specific threshold. If
        yes rearrange subcluster subsequently in
        order to get rid of old data part.
    2.3 Check for two closer subclusters and
        those with similar time nature. If found
        merge them into a single subcluster
        which gives a reasonable and more
        compact vision.
    Return.
```

4. EXPERIMENTAL RESULTS AND DISCUSSION

Here, distributed multi dimensional dataset is taken as input. Experiment demonstrates that DWPCM executes better than WPCM for multi-dimensional dataset having streaming activities. It is to be observed that, the proposed DWPCM algorithm is an enhancement of WPCM. The subsequent real-world datasets are employed for conducting the tests: CAR includes 2,249,727 road segments of California obtained from Tiger/Line datasets; (b) HYD includes 40,995,718 line segments representing rivers of China and (c) TLK includes up to 157,425,887 points obtained from the elevation data of China.

4.1 Memory Usage

Given that DWPCM process data as amount of chunks calculated as memory utilization of every chunk independently and get the largest value as the closing memory consumption for DWPCM. In view of the fact that the dataset is streaming in character, it is not necessary for DWPCM to access over one chunk at a time. Figure 1 shows the percentage of improvement in terms of memory consumption by proposed (DWPCM) as compared against the Baseline Algorithm (WPCM).

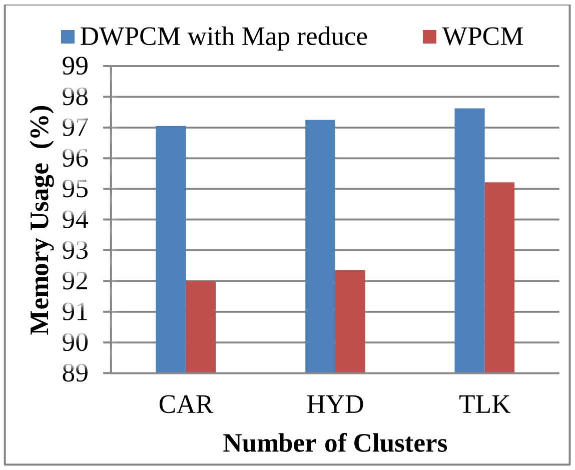

Fig.3. Memory Usage Comparison

It is clear from table 7, the improvement is in excess of 97% for the entire three datasets. WPCM makes use of the complete dataset at a time and that's why it needs adequate memory to hold the entire dataset. This is the cause why WPCM needs much higher memory than the proposed algorithm.

Table 7.Memory Usage Comparison

Input Dataset	DWPCM with Map reduce (%)	WPCM (%)
CAR	97.05	92
HYD	97.25	92.35
TLK	97.62	95.21

4.2 Selectivity

Figure 4 demonstrates the time and amount of page writes for the purpose of adding bounding boxes, with several piece of data of large sizes. In view of the fact that the data portion size reduces, then the amount of leaf nodes in the indexing trees increases, because only a predetermined size dataset undergoes partitioning. If the data portions size is 300×204 (or 300×205), approximately 40,000 data portions are inserted into the indexing trees, however when the data portion size is 20×204, approximately 560,000 data portions are present.

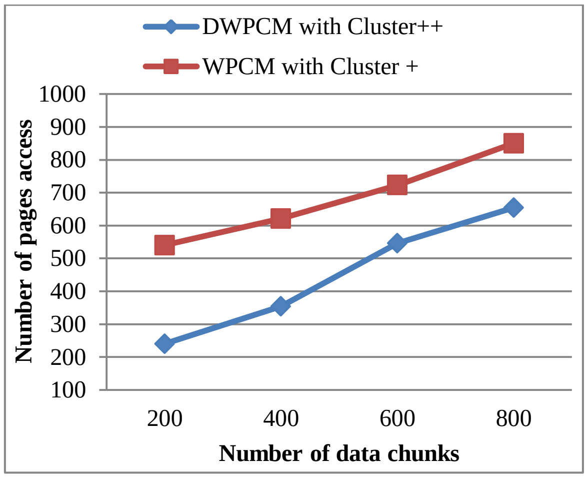

Fig.4. Selectivity Comparison

In actual fact, multi-dimensional indexing structures have to supply low amount of file access throughout the search since accessing to file pages diminishes the response time to a specific query and increase the selectivity. It is clear from the table 8 that the proposed indexing scheme of DWPCM with cluster tree++ is perform well than the WPCM with cluster tree+.

Table 8.Selectivity Comparison

Number of Data Chunks	DWPCM with Cluster++	WPCM with Cluster +
200	240	540
400	354	621
600	546	723
800	654	850

4.3 Scalability with query

Figure 5 demonstrates the query experiment result. It is obvious from the results that Cluster Tree+ can resolve the multi-dimensional query inefficient setback, however Cluster Tree+ ++ execute much better than Cluster Tree+.

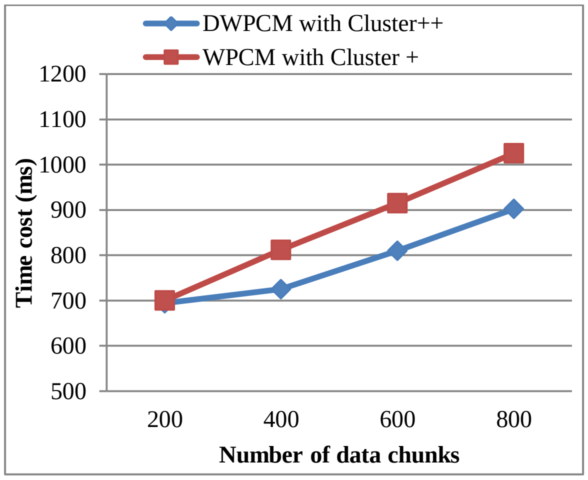

Fig.5. Scalability Comparison

Table 9 shows the values of scalability of the methods. It proves that the multi-dimensional distributed index ranges approximately linear with the number of nodes in the system.

Table 9.Scalability Comparison

Number of Data Chunks	DWPCM with Cluster++ (ms)	WPCM with Cluster + (ms)
200	694	700
400	725	812
600	810	915
800	902	1025

5. CONCLUSION

In the scenario of distributed multi dimensional data, clustering conventional similarity measures are typically will not provide the significant result. In order to overcome this issue, a Distributed Weighted Possibilistic Clustering Algorithm (DWPCM) is proposed here. DWPCM can be utilized for high dimensional datasets having streaming activities. Additionally, a customized edition of ClusterTree+ called as ClusterTree++ is proposed to competently maintain the time-based queries and deletions specified by the user. The ClusterTree++ can maintain the set of data constantly in the most rationalized condition to uphold the competence and effectualness of data insertion, query and update. It is a ordered structure of clusters and subclusters which integrates the representation of cluster into the index configuration to realize efficient and well-organized recovery of data. Experiments are done to assess the ClusterTree++. This scheme will be supportive in the areas of data fusion wherever the data change with dynamism and existing schemes frequently not succeed in solving the problem of keeping a certain structure constantly holding the most efficient data. This scheme can vigorously monitor the status of data of the system and competently evade the outdated data and simultaneously, reorganize the structure of the set of data.

REFERENCES

[1] B.S. Manjunath and W.Y. Ma. Texture Features for Browsing and Retrieval of Image Data. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(8):837–842, August 1996.

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

[3] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An index structure for high-dimensional data. In Proceedings of 22th International Conference on Very Large Data Bases, VLDB’96, pages 28–39, Bombay, India, 1996.

[4] D. M. Gavrila. R-tree index optimization. In T. Waugh and R. Healey, editors, Advances in GIS Research. Tayor and Francis, 1994.

[5] I. Kamel and C. Faloutsos. On packing R-trees. In Proceedings of 2nd International Conference on Information and Knowledge Management(CIKM-93), pages 490–499, Arlington, VA, November 1993.

[6] G. Sheikholeslami, W. Chang, and A. Zhang. Semantic clustering and querying on heterogeneous features for visual data. In The proceedings of the 6th ACM International Multimedia Conference (ACM Multimedia ’98), pages 3–12, Bristol, UK, September 1998.

[7] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pages 103–114, Montreal, Canada, 1996.

[8] Dantong Yu and Aidong Zhang. ClusterTree: Integration of Cluster Representation and Nearest Neighbor Search for Image Databases. In IEEE International Conference On Multimedia and Expo, New York City, July 2000.

[9] Ester M., Kriegel H., Sander J., Xiaowei Xu (1996), “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, KDD‟96, Portland, OR, pp.226-231.

[10] Raymond T. Ng and Jiawei Han (2002), “CLARANS: A Method for Clustering Objects for Spatial Data Mining”, IEEE

[11] Juha Vesanto and Esa Alhoniemi, "Clustering of the Self-Organizing Map", Transactions on Knowledge and Data Engineering, Vol. 14, No. 5,pp.586-600,2000.

[12] Guha S, Rastogi R, Shim K (1998), "CURE: An efficient clustering algorithm for large databases", In: SIGMOD Conference, pp.73~84.

[13] Ankerst M., Markus M. B., Kriegel H., Sander J(1999), "OPTICS: Ordering Points To Identify the Clustering Structure", Proc.ACM SIGMOD"99 Int. Conf. On Management of Data, Philadelphia, PA, pp.49-60.

[14] Kaufman L. and Rousseeuw P. J (1990), "Finding Groups in Data: An Introduction to Cluster Analysis", John Wiley & Sons.

[15] John A. Bullinaria, "Self Organizing Maps: Algorithms and Applications",Introduction to Neural Networks : Lecture 17,2004

[16] V. Gaede and O. Gunther. Multidimensional access methods. ACM Computing Surveys, 30(2):170–231, 1998.

[17] R. Orlandic and B. Yu. A retrieval technique for high-dimensional data and partially specified queries. Data Knowl. Eng., 42(1):1–21, 2002.

[18] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, pages 194–205, 1998.

[19] Fodor I.K. A Survey of Dimension Reduction Techniques. Technical Report Lawrence Livermore National Laboratory (LLNL),UCRL. ID-148494, 2002.

[20] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, R. Bayer, B. Forschungszentrum, and T. Mnchen. Integrating the ub-tree into a database system kernel. In VLDB '00 Proceedings of the 26th International Conference on Very Large Data Bases, pages 263 – 272, 2000.

[21] Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18:509–517, 1975.

[22] D. Lomet and B. Salzberg. The hb-tree: A robust multiattribute search structure. In Proc. IEEE international conference on data enginerring, 5:296–304, 1989.

[23] M. S. Y. Ohsawa. Bd-tree: A new n-dimensional data structure with efficient dynamic characteristics. Proceedings of the Ninth World Computer Congress, IFIP, pages 539–544, 1983.

[24] S. M. K. Chakrabarti. The hybrid tree: An index structure for high dimensional feature spaces. In Proceedings of the 15th International Conference on Data Engineering (ICDE'99), pages 440–447, 1999.

[25] B. Stantic, R. W. Topor, J. Terry, and A. Sattar. Advanced indexing technique for temporal data. COMSIS - Journal of Computer Science and Information Systems, 7(4):679–703, 2010.

[26] Wang W., Yang J., Muntz R(1997), "STING: A statistical information grid approach to spatial data mining", In: Proc. of the 23rd VLDB Conf. Athens, pp.186~195.

[27] Rakesh A., Johanners G., Dimitrios G., Prabhakar R(1999), "Automatic subspace clustering of high dimensional data for data mining applications", In: Proc. of the ACM SIGMOD, pp.94~105.

AUTHORS

1. Sunil Sunny Chalakkal is working as an Asst. Professor in the Dept. of Computer Science ,

St Thomas college Trichur . He received MCA & MPhil from Bharathiar University Coimbatore with specialization in Data Mining. He has more than 15 years of teaching experience. Sunil sunny is a research scholar in Anna University

2. Ms.M.Rajalakshmi is currently working as an Associate Professor at Coimbatore Institute of Technology, Coimbatore in the Department of Computer Science Engineering & Information Technology. She did her Ph.D. in Computer Science and Engineering at Anna University, Chennai and she obtained M.E. Computer Science and Engineering from P.S.G.College of Technology, Coimbatore with distinction in the year 2005. . She has 19 years of teaching experience and published four papers in international journal and one paper in international conference.

3. Dr. R. Vijayakumar is currently working as the Professor At School of Computer Science, M.G.University, Kottayam. He has 32 years of teaching experience and published 11 papers in international journals and 31 papers in national journals