IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 3, Issue 1, Feb-Mar, 2015
ISSN: 2320 – 8791 (Impact Factor: 1.479)
www.ijreat.org

Spatial Data Mining: Querying Over Spatial Data

Nipun Bansal

Department of Computer Science, SGGSCC, Delhi University, New Delhi, 110088

Abstract

Mining data from large amounts of spatial data is very difficult because of the huge amounts of spatial data collected from satellite or local cameras for many important and sophisticated areas like designing of road maps for different regions or states, countries, cloud cover, traffic control or GPS etc. Thus owing to urgent practical, social and environmental needs, knowledge discovery in spatial data has become important and one of the rapidly growing field. In this paper, an overview of common knowledge discovery algorithms and a comparative study of these algorithms have been done.

Keywords: Spatial Data, Data Mining, Map reduce.

1. Introduction

Spatial data are the data related to object that occupy space. Aspatial database stores the spatial objects represented by spatial data types and spatial relationships among such objects the number and size of spatial database e.g. for geo marketing, traffic control or environmental studies, medical diagnosis, weather prediction are rapidly growing which result in an increasing need to store huge amount of data and retrieve information that is both convenient and efficient.

A very interesting and efficient method has introduced for this purpose and it is called as Spatial Data Querying.

Thus spatial data querying can be summarized as:

1. Extracting interesting spatial patterns and features
2. Capturing intrinsic relationships between spatial and non-spatial data
3. Presenting data regularity concisely and at higher conceptual levels and
4. Helping to reorganize spatial databases to accommodate data semantics, as well as to achieve better performance

This report deals with spatial data structures for indexing and with their usability for knowledge discovery in spatial data. Huge amount of data processed in spatial data mining requires using some indexing structures to speed up the mining process which are described in this paper. Then the various algorithms and methods for processing and analyzing spatial data are described and a comparative study of these algorithms is done. We then explored Parallel Data Mining, an upcoming field. Three new frameworks Map Reduce, DBMS-X and Vertica are introduced and a comparison of these techniques has been done. Finally benchmark performances of them are presented.

2. Database sources and issues

With the advancement in technology and tools, spatial data maybe collected in huge amount from different resources for various applications ranging from remote sensing and satellite telemetry systems, to computer cartography, medical diagnosis, weather analysis and prediction, and all kinds of environmental planning. Various national and international agencies are also providing spatial data in different dimensions. Most common data sources are satellite images, medical images, human body's protein structure and all those data who can be represented in the form of cuboids, polygon, cylinder etc. Various sites are also available for collection of GIS data for example Google earth, visible earth (NASA),JSC digital image collection (NASA) ,Global land cover facility are also available for collection of spatial data. Basically spatial data include geographic data such as maps and associated information, and computer aided design data such as integrated circuit design or building designs. It has observed that 2D database are not more efficient in storing, indexing and queuing of data on the basis of spatial locations .Additionally for 2D databases, we cannot use standard index structures, such as B-trees or hash indices, to answer such a query efficiently. So it is recommended that we should work for higher dimensional data.

3. Spatial data mining and spatial data querying

Analysis is an important part of GIS which allows spatial operations with data (e. g. network analysis or filtering of raster data), measuring functions (e.g. distance, direction

between objects), statistical analyses or terrain model analysis (e. g. visibility analysis).
Spatial data mining is a special kind of data mining. The main difference between data mining and spatial data mining is that in spatial data mining tasks we use not only non-spatial attributes(as it is usual in data mining in non-spatial data), but also spatial attributes.
Basic tasks of spatial data mining are:

data set - First a particular application like weather forecasting, traffic control for which analyses needs to be performed has to be identified and then the data set may be collected using satellite or GPS or other sensors for processing.

Data processing - Analyst may select, filter, aggregate, sample, clean and/or transform data into much more understandable form. Unwanted and useless portions may be cut from the existing data hence to improve the productivity and applicability of the data.

Prediction - Prediction means to give some outcomes in advance on the basis of previous history or patterns of data items. Values of specific attributes of the data items may also be calculated accurately with iterative methods and different samples of data.

Regression - Given a set of data items, regression identifies dependency y of some attribute values upon the values of other attributes in the same item and apply these values on other data items or records.

Classification - Given a set of predefined categorical classes, determine to which of these classes a specific data item belongs. For example, in weather prediction system we classify satellite images into different classes on the basis of some common properties and patterns.

Clustering - given a set of data items, group items that are similar. For example, given a set of satellite images, identify subgroup of objects of patterns (colored, non colored, size, shape)and their behavior.

Link Analysis - Given a set of data items identify relationships b/w attributes and items such as the presence of one pattern implies the presence of another pattern.

Model visualization - Visualization plays a very important role in understanding and demonstration the desired task properly. Visualization techniques may range from simple scatter plots and histogram plots over parallel coordinates to 3D data items.

Result testing, verification and refinement - This step includes verifying and testing the results got from above steps and refining them for better accuracy and efficiency.
Spatial data querying on the other hand is a stage in the whole process of spatial data mining and can be explained in detail as:

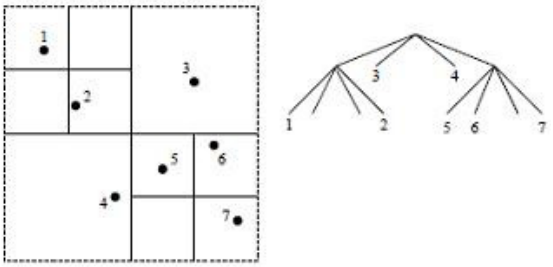


Figure 1: Quad Tree

- Classification: finds a set of rules which determine the class of the classified object according to its attributes e. g. "IF population of city = high AND economic power of city =high THEN unemployment of city = low" or classification of a pixel into one of classes, e. g. water, field, forest.

- Association rules: Association rules describe patterns, which are often in the database. The association rule has the following form: A! B(s %; c %), where s is the support of the rule (the probability, that A and B hold together in all the possible cases) and c is the confidence (the conditional probability that B is true under the condition of A e. g. "if the city is large, it is near the river (with probability 80 %)".

- Characteristic rules: describe some part of database e.g. "bridge is an object in the place where a road crosses a river."

- Discriminant rules: describe differences between two parts of database e. g. find differences between cities with high and low unemployment rate.

- Clustering: groups the object from database into clusters in such a way that object in one cluster are similar and objects from different clusters are dissimilar e. g. we can find clusters of cities with similar level of unemployment or we can cluster pixels into similarity classes based on spectral characteristics.

- Trend detection: finds trends in database. A trend is a temporal pattern in some time series data. A spatial trend is defined as a pattern of change of a non-spatial attribute in the neighborhood of a spatial object e. g. "when moving away from Brno, the unemployment rate increases" or we can find changes of pixel classification of a given area in the last five years.

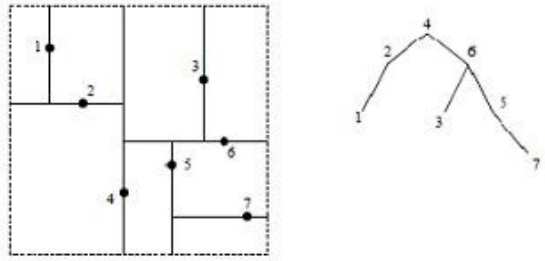


Figure 2: k-d Tree

4. Data structures used in querying spatial data

4.1 Quad tree

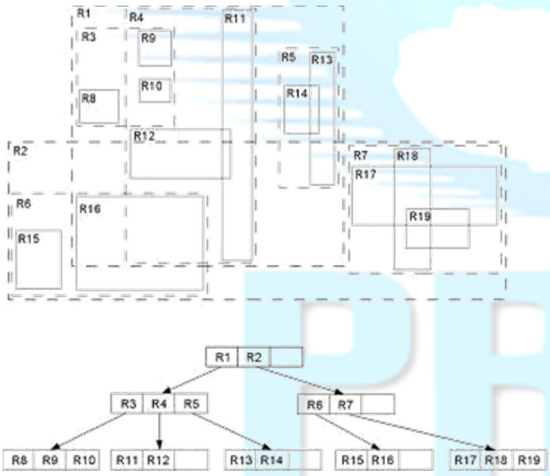


Figure 3: R-tree for 2D rectangles

The quad tree is used to index 2D space. Each internal node of the tree splits the space into four disjoint subspaces (called NW,NE, SW, SE) according to the axes. Each of these subspaces is split recursively until there is at most one object inside each of them. The quad tree is not balanced and its balance depends on the data distribution and the order of inserting the points.

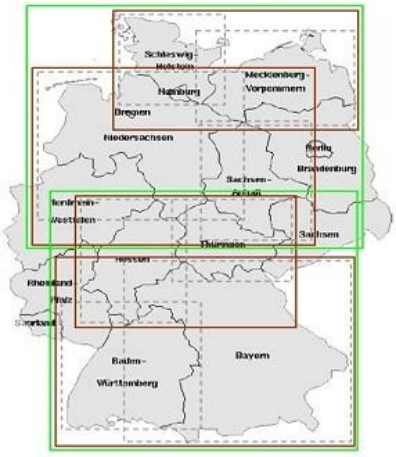


Figure 4

4.2 k -d -tree

This method uses a binary tree to split k - dimensional space. This tree splits the space into two subspaces according to one of the coordinates of the splitting point. Let level (nod) be the length of the path from the root to the node nod and suppose the axes are numbered from 0 to k - 1. At the level, level (nod) in every node the space is split according to the coordinate number (level (nod) mod k).

Inserting and searching are similar to the binary trees. We only have to compare nodes according to the coordinate number (level (nod) mod k). This structure has a disadvantage that it is sensitive to the order in which the objects are inserted.

4.3 R-Tree

A spatial database consists of a collection of tuples representing spatial objects, and each tuple has a unique identifier which can be used to retrieve it. An index based on object's spatial locations is desirable, but classical one dimensional database indexing structures are not appropriate to multi-dimensional spatial searching. Also, structures based on exact matching of values, such as hash tables, are not useful because a range search 1s requested. Since the search space is multidimensional, structures using one dimensional ordering of key values, such as B-trees and ISAM indexes, also fails.

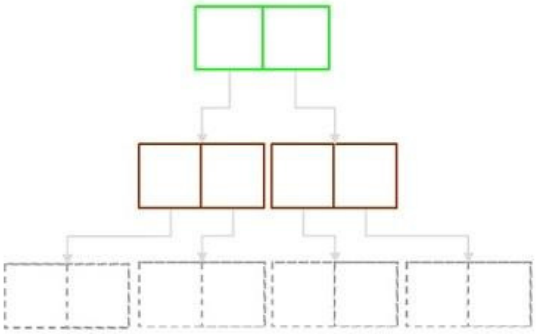


Figure 5: Data objects in the map are represented by MBR

Structure An R-tree 1s a height-balanced tree similar to a B-tree with index records in its leaf nodes containing pointers to data objects. Let M be the maximum number of entries that will fit at one node and let m = M/2 be a parameter specifying the maximum number of entries in a node

1. Every leaf node contains between m and M index records unless it is the root.
2. For each index record , (I, tuple-identifier) in a leaf node, I
   1s the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple.
3. Every non-leaf node has between m and M children unless it is the root.
4. For each entry (I, child-pointer) in a non-leaf node, I 1s the smallest rectangle that spatially contains the rectangles in the child node
5. The root node has at least two children unless it is a leaf.
6. All leaves appear on the same level.

Search

The input is a search rectangle (Query box). Searching is quite similar to searching in a B+ tree. The search starts from the root node of the tree. Every internal node contains a set of rectangles and pointers to the corresponding child node and every leaf node contains the rectangles of spatial objects (the pointer to some spatial object can be there). For every rectangle in a node, it has to be decided if it overlaps the search rectangle or not. If yes, the corresponding child node has to be searched also. Searching is done like this in a recursive manner until all overlapping nodes have been traversed. When a leaf node is reached, the contained bounding boxes (rectangles) are tested against the search rectangle and their objects (if there are any) are put into the result set if they lie within the search rectangle.

A recursive process starting from the root result = f
For a node N
if N is a leaf node, then result = result [*N else // N is a non-leaf node*
for each child N' of N
if the rectangle of N' contains q then recursively search N'

Insertion To insert an object, the tree is traversed recursively from the root node. At each step, all rectangles in the current directory node are examined, and a candidate is chosen using a heuristic such as choosing the rectangle which requires least enlargement. The search then descends into this page, until reaching a leaf node. If the leaf node is full, it must be split before the insertion is made. Again, since an exhaustive search is too expensive, a heuristic is employed to split the node into two. Adding the newly created node to the previous level, this level can again overflow, and these overflows can propagate up to the root node; when this node also overflows, a new root node is created and the tree has increased in height.

Splitting an overflowing node

Since redistributing all objects of a node into two nodes has an exponential number of options, a heuristic needs to be employed to find the best split. In the classic R-tree, Guttman proposed two such heuristics, called Quadratic Split and Linear Split. In quadratic split, the algorithm searches the pair of rectangles that is the worst combination to have in the same node, and puts them as initial objects into the two new groups. It then searches the entry which has the strongest preference for one of the groups (in terms of area increase) and assigns the object to this group until all objects are assigned (satisfying the minimum fill).

4.4 R+ Tree

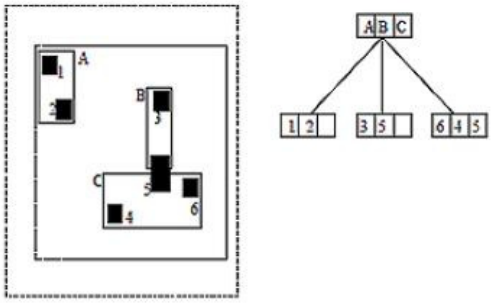


Figure 6: R+ tree

R+ tree is an extension of the R tree. In contrast to R tree bounding rectangles of the nodes at one level don't overlap in this structure. This feature decreases the number of searched branches of the tree and reduces the time consumption. In the R+-tree it is allowed to split data objects so that different parts of one object can be stored in more nodes of one tree level. If a rectangle over-laps another one, we decompose it into a group of non-overlapping rectangles which cover the same data objects. This increases a space consumption but allows zero overlap of the nodes and therefore reduces the time consumption.

R+ trees differ from R trees in that

- Nodes are not guaranteed to be at least half filled
- The entries of any internal node do not overlap
- An object ID may be stored in more than one leaf node

Advantages over R tree

- Because nodes are not overlapped with each other, point query performance benefits since all spatial regions are covered by at most one node.
- A single path is followed and fewer nodes are visited than with the R-tree.

Disadvantages over R tree

- Since rectangles are duplicated, an R+ tree can be larger than an R tree built on same data set.
- Construction and maintenance of R+ trees is more complex than the construction and maintenance of R trees and other variants of the R tree.

4.5 R* - Trees

R*-trees are a variant of R-trees used for indexing spatial information.

Operations on R* tree in SQLite Database

1. Creating An R*Tree Index
   Consider creating a 2D R*Tree index for use in spatial queries:

```
CREATE VIRTUAL TABLE demoindex USING rtree(
id, – Integer primary key
minX, maxX, – Min and Max X coordinate
minY, maxY – Min and Max Y coordinate);
```

2. Populating An R*Tree Index

```
INSERT INTO demoindex
VALUES( 1, – Primary key
-80.7749, -80.7747, – Longitude range
35.3776, 35.3778 – Latitude range);
```

3. Querying An R*Tree Index
   To find all elements of the index that are contained within the vicinity of given point we can write query like:

```
SELECT id FROM demoindex
WHERE maxX>=-81.08 AND minX<=-80.58
AND maxY>=35.00 AND minY<=35.44;
```

Difference between R*-trees and R-trees

1. Optimization in ChooseSubTree module for leaf nodes
2. Revised Node-Split Algorithm. The split heuristic is improved to produce pages that are more rectangular and thus better for many applications.
3. Forced Reinsertion at Node Overflow which optimizes the existing tree, but increases complexity.
4. Completely Dynamic.
5. Supports point and spatial data efficiently at the same time.
6. Implementation cost of R* tree is slightly higher than that of other R-trees.

4.6 Neighborhood graphs

Neighborhood graphs and neighborhood paths

Definition: Neighborhood graph G for spatial relation neighbor=2is a graph G(U,H) where U is a set of nodes and H is a set of edges. Each node represents an object and two nodes N1, N2 are connected by edge iff the objects corresponding to N1 and N2 are in the relation neighbor. The relation neighbor can be:

1. *topological relation*, e. g. two objects touch, cover, are equal
2. *metric relation*, e. g. distance of the objects is less than d
3. *direction relation*, e.g. north, south, east, west
4. any conjunction or disjunction of previous relations

Neighborhood graph is oriented. Thus it can happen that object A is a neighbor of the object B but object B is not a neighbor of the object A.

Definition: Neighborhood path for the neighborhood graph G is an ordered list of nodes from G where every two following nodes from the path are connected by some edge from G, i. e. for the path [n_0, n_1,....., n_{k1}] there must be edges (n_i , n_{i}Å1) for every 0· i < k-1. Length of the path is a sum of edges in the path.

Elementary operations on the neighborhood graphs

Elementary operations on the neighborhood graphs are:

getGraph(data, neighbor)- returns the neighborhood graph G representing the relation neighbor on the objects from the table data. The relation neighbor can be one of the spatial relations listed in the definition of the neighborhood graph.

getNeighborhood(G, o, pred) - returns the set of the objects connected to the object o by some of the edges from the graph G. The predicate pred must hold for these objects. This condition is used if we want to get only some specific neighbors of the object o. The predicate pred may not necessarily be spatial.

createPath(G, pred, i)- returns the set of all paths which consist of the nodes and edges from the graph G, their length is less than or equal to i and the predicate pred holds for them. Moreover these paths must not contain any cycles, i.e. every node from G can appear at most once in each path.

4.7 MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. A Map Reduce program consist of only two functions Map and Reduce writ-ten by user to process key/value data pairs.

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

The Reduce function, also written by the user, accepts an inter-mediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allow programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Input and Output types of a MapReduce job:

(input) <k1, v1> -> map -><k2, v2> -> combine -><k2, v2> -> reduce -><k3, v3> (output)

4.8 Parallel Database Systems

The two key aspects that enable parallel execution are that (1) most (or even all) tables are partitioned over the nodes in a cluster and that (2) the system uses an optimizer that translates SQL commands into a query plan whose execution is divided amongst multiple nodes. Because programmers only need to specify their goal in a high level language, they are not burdened by the under-lying storage details, such as indexing options and join strategies. Consider a SQL command to filter the records in a table T1 based on a predicate, along with a join to a second table T2 with an aggregate computed on the result of the join. A basic sketch of how this command is processed in a parallel DBMS consists of three phases. Since the database will

have already stored T1 on some collection of the nodes partitioned on some attribute, the filter sub-query is first performed in parallel at these sites similar to the filtering performed in a Map function. Following this step, one of two common parallel join algorithms are employed based on the size of data tables. For example, if the number of records in T2 is small, then the DBMS could replicate it on all nodes when the data is first loaded. This allows the join to execute in parallel at all nodes. Following this, each node then computes the aggregate using its portion of the answer to the join. A final "roll-up" step is required to compute the final answer from these partial aggregates.

If the size of the data in T2 is large, then T2's contents will be distributed across multiple nodes. If these tables are partitioned on different attributes than those used in the join, the system will have to hash both T2 and the filtered version of T1 on the join attribute using a common hash function. The redistribution of both T2 and the filtered version of T1 to the nodes is similar to the processing that occurs between the Map and the Reduce functions. Once each node has the necessary data, it then performs a hash join and calculates the preliminary aggregate function.

5. Queries on spatial data

The key characteristic that makes a spatial database a powerful tool is its ability to manipulate spatial data, rather than simply to store and represent them. The basic form of such a database is answering queries related to the spatial properties of data. Some typical spatial queries are the following:

1. A "Point Location Query" seeks for the objects that fall on a given point (e.g. the country where a specific city belongs).
2. A "Range Query" seeks for the objects that are contained within a given region, usually expressed as a rectangle or a sphere (e.g. the pathways that cross a forest).
3. A "Join Query" may take many forms. It involves two or more spatial datasets and discovers pairs (or tuples, in case of more than two datasets) of objects that satisfy a given spatial predicate (e.g. the pairs of boats and stormy areas, for boats sailing across a storm).
4. The distance join was recently introduced to compute a subset of the Cartesian product of two datasets, specifying an order on the result based on distance (e.g. the pairs of hotels and archeological sites, ordered by driving distance up to 50 km between them).

Finally, very common is the "Nearest Neighbor Query" that seeks for the objects residing more closely to a given object. In its simplest form, it discovers one such object (the nearest neighbor). Its generalization discovers K such objects (K nearest neighbors), for a given K (e.g. the K ambulances closer to a spot where an accident with K injured persons occurred).

6. Algorithms and methods

In this section we have presented the various queries possible on Spatial Data in detail. Also the experimental results are analyzed for spatial queries.

6.1 Nearest Neighbor Queries

A very common type of query in spatial data is to find the k nearest neighbor to a given object or point in space. A naive approach to solve this problem requires O(n2) time with no preprocessing to find the neighbor of all the points in the data set, S. The author has proposed a much better and efficient search algorithm using R- tree for processing exact k - nearest neighbor queries and introduced several metrics for ordering and pruning the R tree.

NEAREST NEIGHBOR SEARCH USING R-TREES

Metrics for Nearest Neighbor Search

Let N_P and N_Q be two internal nodes of R_P and R_Q, respectively. Each of these nodes has an MBR that contains all the points that reside in the respective sub tree. In order for this rectangle to be the minimum bounding one, at least one point is located at each edge of the rectangle. Let M_P and M_Q represent the MBRs of N_P and N_Q, respectively. Let r1, r2, r3 and r4 be the four edges of M_P and s1, s2, s3 and s4 be the four edges of M_Q. By MINDIST(r_i , s_i) we denote the minimum distance between two points falling on $r_i$and s_i. Accordingly, by MAXDIST(r_i, s_i) we denote the maximum distance between two points falling on r_i and s_i . In the sequel, we extend definitions of metrics between a point and an MBR that appear in and define a set of useful metrics between two MBRs. In case M_P and M_Q are disjoint we can define a met-ric that expresses the minimum

possible distance of two points contained in different MBRs:

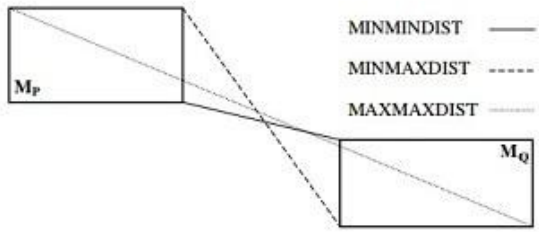


Figure 7: Two MBRs and their MINMINDIST, MINMAXDIST and MAXMAXDIST

$MINMINDIST(M_P , M_Q) = \min_{i,j} MINDIST(ri, sj)$

In case the MBRs of the two nodes intersect, then $MINMINDIST(M_P , M_Q)$ equals 0. In any case (intersecting or disjoint MBRs) we can define the metrics
$MINMAXDIST(M_P , M_Q) = \min_{i,j} MAXDIST(r_i , s_i)$ and

$MAXMAXDIST(M_P , M_Q) = \max_{i,j} MAXDIST(r_i , s_i)$

MAXMAXDIST expresses the maximum possible distance of any two points contained in different MBRs.

MINMAXDIST expresses an upper bound of distance for at least one pair of points. More specifically, there exists at least one pair of points (contained in different MBRs) with distance smaller than or equal to MINMAXDIST. In Figure, two MBRs and their MIN-MINDIST, MINMAXDIST and MAXMAXDIST distances are depicted. At least one point is located on each edge of each MBR.
To summarize, for each pair (p_i , q_j) of points, pi enclosed by M_P and q_j enclosed by M_Q, it holds that

$MINMINDIST(M_P , M_Q) \cdot dist(p_i , q_j) \cdot MAXMAXDIST(M_P , M_Q))$

Moreover, there exists at least one pair (p_i , q_j) of points,pi enclosed by M_P and q_j enclosed by M_Q, such that

$dist(p_i , q_j) \cdot MINMAXDIST(M_P , M_Q)$

Theorem 1 For a point P and an MBR R enclosing a set of objects say o_1, o_2, o_3 , o_4 ....., o_m then for any object o , MINDIST (P , R) · (P ,o).

Theorem 2 For a point P and an MBR R enclosing a set of objects say o_1, o_2, o_3 , o_4 .... , o_m then for any object o , MINMAXDIST (P , R) ¸ (P ,o).

Nearest Neighbor Search Algorithm

We utilize the two theorems we developed to formulate the following three strategies to prune MBRs during the search:

1. An MBR M with MINDIST(P,M) ¸ MINMAXDIST(P,M') of another MBR M' is discarded because it cannot contain the NN (theorems 1 and 2).We use this in downward pruning.
2. An actual distance from P to a given object O which is greater than the MINMAXDIST(P,M) for an MBR M can be discarded.

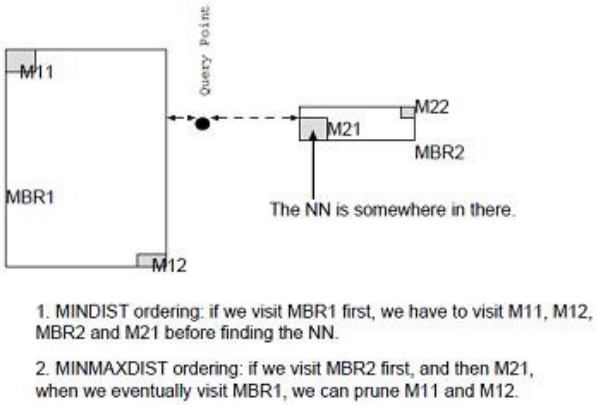


Figure 8: MINDIST and MINMAXDIST in 2-Space

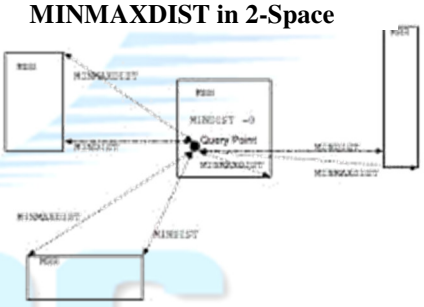


Figure 9: MINDIST is not always the better ordering

because M contains an object O' which is nearer to P (theorem 2). This is used in upward pruning.

3. Every MBR M with MINDIST(P,M) greater than the actual distance from P to a given object O is discarded because it cannot enclose an object nearer than O (theorem 1). We use this in upward pruning.

The algorithm presented above can be easily generalized to answer queries of the type: Find the k Nearest Neighbors to a given Query Point, where k is greater than zero.
The only differences are:

1. A sorted buffer of at most k current nearest neighbors is needed.
2. The MBRs pruning is done according to the distance of the furthest nearest neighbor in this buffer.

Best First Search

- Calculates minmax distance for all objects
- Sort R-Tree by minmax distance
- Removes nodes from sorted tree. If node has no children , then it is the nearest neighbor

Algorithms for finding Closest Pairs (CPQs)

In the following, a number of different algorithmic approaches for discovering the 1-CP and the K-CPs between points stored in two R-trees are presented.

Naive Algorithm

The simplest approach to the problem of Closest Pair Queries is to follow a recursive naive solution for the 1- CP subproblem and for two R-trees of the same height. Such an algorithm consists of the following steps.
CP1 Start from the roots of the two R-trees and set the minimum distance found so far, T^{-}.
CP2 If you access a pair of internal nodes, propagate downwards recursively for every possible pair of MBRs.
CP3 If you access two leaves, calculate the distance of each possible pair of points. If this distance is smaller than T, update T.

Exhaustive Algorithm

An improvement of the previous algorithm is to make use of the left part of Inequality 1 and prune some paths in the two trees that are not likely to lead to a better solution. The CP2 step of the previous algorithm would now be:

CP2: If we access a pair of internal nodes, calculate MINMINDIST for each possible pair of MBRs. Propagate downwards recursively only for those pairs that have MINMINDIST · T.

Simple Recursive Algorithm

A further improvement is to try to minimize the value of T as soon as possible. This can be done by making use of Inequality 2. That is, when a pair of internal nodes is visited, to examine if Inequality 2 applied to every pair of MBRs, can give a smaller T value. Since Inequality 2 holds for at least one pair of points, this improvement is sound for the 1-CP problem. The CP2 step would now be:

CP2: If you access a pair of internal nodes, calculate the minimum of MINMAXDIST for all possible pairs of MBRs. If this minimum is smaller than T, update T. Calculate MINMINDIST for each possible pair of MBRs. Propagate downwards recursively only for those pairs that have MINMINDIST · T.

Sorted Distances Recursive Algorithm

A heuristic that aims at improving our algorithms even more when two internal nodes are accessed, is to sort the pairs of MBRs ac-cording to ascending order of MINMINDIST and to obey this or-der in propagating downwards recursively. This order of processing is expected to improve pruning of paths. The CP2 step of the previous algorithm would be:

CP2: If we access a pair of internal nodes, calculate the mini-mum of MINMAXDIST for all possible pairs of MBRs. If this minimum is smaller than T, update T. Calculate MINMINDIST for each possible pair of MBRs and sort these pairs in ascending order of MINMINDIST. Following this order, propagate downwards recursively only for those pairs that have MINMINDIST <= T.

Heap Algorithm

The overall algorithm is as follows.

1. Start from the roots of the two R-trees, set T to infinity and initialize the heap.
2. If we access a pair of internal nodes, calculate the mini-mum of MINMAXDIST for all possible pairs of MBRs. If this minimum is smaller than T, update T. Calculate MIN-MINDIST for each possible pair of MBRs. Insert into the heap those pairs that have MINMINDIST · T
3. If we access two leaves, calculate the distance of each possible pair of points. If this distance is smaller than T, update T.

4. If the heap is empty then stop.

5. Get the pair on top of the heap. If this pair has MINMINDIST > T, then stop. Else, repeat the algorithm from Step 2 for this pair.

6.2 An index structure for efficient reverse nearest neighbor(RNN) queries

.Introduction

RNN query finds all the points in the given data set S, having a query point q as their nearest neighbor. the author in this paper has proposed a new structure called Rdnn - tree i.e. R - tree containing Distance of Nearest Neighbors differing from standard R tree structure in terms of storing extra information about nearest neighbor of the points in each node.

Motivation

R - tree faces the following limitations :-

1. There is a large overlap of MBR (Minimum Bounding Region) of parent nodes hampering the RNN search performance
2. For dynamic cases, a second tree is required for storing the index structure of the spherical regions resulting in more time to compute RNN queries and added maintenance cost.

Thus there is a need to propose a better structure that can eliminate the above limitations and can provide better features for faster execution of both NN and RNN queries.

Rdnn - Structure and Proposed Algorithm

In a Rdnn - tree consist of leaf node and internal node. An internal node consist of an array of branches of the form (ptr; Rect; maxdnn) where ptr is the address of a child node in the tree and if the ptr points to a leaf node, Rect is the minimum bounding rectangle of all points in the leaf node otherwise Rect is the minimum bounding rectangle of all rectangles that are entries in the child node. Whereas the leaf node is of the form (ptid;dnn), where ptid refers to a d-dimensional point in the data set and dnn is the distance from the point to its nearest neighbors in the data set.

Algorithms Proposed

RNN search

Let say we need to do reverse nearest neighbor search on Rdnn tree for a query point q, then

1. For a leaf node, examine each point p in the node such that the distance of the point p to q is less than or equal to the nearest neighbor distance of point p in the dataset, S. From this we can say p is at least as close to q as to its nearest neighbor, then p is one of the reverse nearest neighbors

2. For an internal node, compare the query point q with each branch B=(ptr;Rect;maxdnn).By definition of Rdnn - tree we know that all the points in the subtree rooted at B are contained in Rect and the distance from each point to its nearest neighbor is not greater than maxdnn since maxdn-nis the largest of them. Hence if D(q;Rect)> maxdnn, then branch B need not to be visited otherwise call RNN-Search(B.ptr , q).

NN Search

The algorithm applied to find NN for R- tree can be applied to Rdnn - tree for the reason that Rdnn tree has all the properties of R-tree. Moreover in Rdnn tree , if the distance of the point p from the querying point q is less than half the distance of nearest neighbor of point p then we can safely say p is the nearest neighbor of q in data set S. This helps in pruning the extra branches during the branch-and-bound search making the algorithm more efficient and faster.

Experiments and Results

The author implemented and compared the RNN - tree method using R-tree and Rdnn-tree on systems with configurations as 2.5 GHz , Pentium II processors , 512 RAM and running SCO UNIX as operating system. The results are shown in the table.

1. Rdnn structure significantly outperformed the index structures in, and typically requires only 1-2 leaf access to locate the RNNs. From the figure 1 we can observe, in 2D cases the RNN tree approach took 20 leaf access for the data set having 1 lakh items whereas Rdnn -tree took only 2 leaf

access, thereby a significant improvement of over 90 percentage.

2. From the data in the Table 1, it is clear that Rdnn-tree can perform NN queries efficiently. Thus in dynamic cases , only one tree is required for both the Nearest Neighbor(NN) and RNN queries
3. The Rdnn-tree allows to execute multiple NN and RNN queries in one traversal of the tree, further enhancing performance in the dynamic case.

In terms of disk access, the Rdnn tree structure provided 4-5 times better efficiency than the RNN tree in the two dimensional data.

6.3 A density based algorithm for discovering clusters in spatial databases with noise

The task considered in this paper is class identification i.e. the grouping of the objects of a database into meaningful subclasses
.It requires one input parameter and supports the user in determining an appropriate value for it. It discovers clusters of arbitrary shape. Here DBSCAN has implemented on the basis of R*- tree.

	2-D data sets					4-D data sets	
Number of points	10,000	25,000	50,000	75,000	100,000	5,000	50,000
Rdnn-tree	2.098	2.120	3.307	3.388	3.452	4.376	6.464
R-tree	2.11	2.2.0	3.360	3.460	3.48	4.436	6.82

Figure 10: Comparison of NN queries performance

All experiments have been run on HP 735/100 workstations with the help of synthetic data and the database of the SEQUOIA 2000 benchmark.
Positive aspects-
i. Faster ii. Efficient iii. Applicable for large database
iv. Applicable on arbitrary shape. v. Extendable for polygons over point objects.
Points missed: i. High dimensional data not considered. ii. It is only about static rather than moving obstacles.

6.4 Algorithm for characterization and trend detection in spatial databases

In this algorithm it has observed that for spatial characterization, it is important that class membership of a database object is not only determined by its non-spatial attributes but also by the at-tributes of objects in its neighborhood. In this paper neighbor-hood relationship is considered as centered point of discussion. With the help Of different databases, various local and global trends have detected.

Spatial Trend Detection

Spatial trend is defined as a regular change of one or more non-spatial attributes when moving away from a given start object o.A trend can be positive trend, negative (linear) trend as well as a situation where no significant (linear) trend is observed. Neighborhood paths starting from o are considered and linear regression analysis is performed on the respective attribute values for the objects of a neighborhood path to describe the regularity of change. A Linear regression is used, since it is efficient and often the influence of some phenomenon to its neighborhood is either linear or can be transformed into a linear model, e.g. exponential regression. The correlation of the observed attribute values with the values predicted by the regression function yields a measure of confidence for the discovered trend.

Algorithm
Let g be a neighborhood graph, o an object (node) in *g*, attr be a subset of all non-spatial attributes, *t* be a type of function, e.g. linear or exponential, used for the regression and let filter be one of the filters for neighborhood paths.

	Algorithm *global-trend*	
correlation	Ø *neighbors* operations	Ø runtime (sec.)
0.60	56.6	91
0.70	55.0	90
0.80	54.3	85
0.90	53.7	84

	Algorithm *local-trends*	
correlation	Ø *neighbors* operations	Ø runtime (sec.)
0.60	8.4	14.3
0.70	8.3	13.8
0.80	8.1	12.9
0.90	7.1	11.3

Figure 11: Performance of both trend algorithm

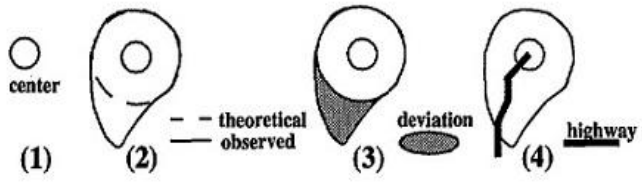


Figure 12: The steps of trend detection in a geographic information system

Spatial trend detection Algo discovers the set of all neighborhood paths in g starting from o and having of type t in attributes attr with a correlation of at least min-conf. The paths must satisfy the filter and their length must be between min-length and max-length.

1. Global Trend Algo creates all neighborhood paths of the same length simultaneously - starting with min-length and continuing until max-length. The regression is performed once for each of these sets of all paths of the same length. If no trend of length l with correlation ¸ min-conf is detected then the path extensions of length l+1, l+2, ---, max-length are not created. The algorithm returns the significant spatial trend with the maximum length.

2. Local Trend Algo performs a regression once for each of the neighborhood paths with length min-length and a path is only extended further if it has a significant trend. The algorithm returns two sets of paths showing a significant spatial trend, a set of positive trends and a set of negative trends.

6.5 Density connected sets and their application for trend detection in spatial databases.

In this paper, the concept of density connected sets has been introduced to discover the trends in a spatial database. Also an algorithm which is generalized form of DBSCAN has been pro-posed having following properties.(1) any symmetric predicate can be used to define the neighborhood of an object allowing a natural definition in the case of spatially extended objects such as polygons, and (2) the cardinality function for a set of neighboring objects may take into account the non-spatial attributes of the objects as a means of assigning application specific weights.

Algorithm and Trend Detection in GIS (Geographic Information System)

A geographic database having both spatial and non-spatial data information with its administrative units such as communities, its natural facilities such as the mountains and its infrastructure such as roads has been considered. The database contains the ATKIS 500 data and the Bavarian part of the statistical data obtained by the German census of 1987.For trend detection , SAND (partial And Non-spatial Database) architecture is used to store spatial extension of all objects using R* tree structure and the non-spatial attributes of the communities managed by a relational database management system.

The Bavaria database may be used, e.g., by economic geographers to discover different types of knowledge. In the following, we discuss the tasks of spatial classification and spatial trend detection.

Spatial classification should discover rules predicting the class membership of some object based on the spatial and non-spatial attributes of the object and its neighbors. for example:

if there is some agglomeration of cities, then this agglomeration neighbors a highway (confidence 75 %)

A spatial trend as a pattern of systematic change of one or several non-spatial attributes in 2D or 3D space. To discover spatial trends of the economic power, following steps needs to be followed as follows:

1. To discover local extrema of some non-spatial attributes such as the rate of unemployment. Initially all the areas with a locally minimal rate of unemployment are determined which are called centers, e.g. the city of Munich. The theory of central places claims that the attributes of such centers influence the attributes of their neighborhood to a degree which decreases with increasing distance. E.g., in general it is easy to commute from some community to a close by center thus implying a low rate of unemployment in this community.

2. To determine both theoretical and observed trend of non-spatial attributes when moving away from the centers the theoretical trend of the rate of unemployment in the neighborhood of the centers

is calculated.

when moving away from Munich rate of unemployment increases (confidence 86 %)

3. To discover deviations of the observed trend from the theoretical trend
   If the deviations from the theoretical trends significantly differ in length, then the longer one discovered is returned, e.g. indicating the direction of a deviation.

   when moving away from Munich in south-west direction, then the rate of unemployment is stable (confidence 97 %)

4. To explain the deviations by other spatial objects (e.g. by some infrastructure) in that area and direction.
   The goal of the fourth step is to explain these deviations. E.g. if some community is relatively far away from a center, but is well connected to it by train, the rate of unemployment in this community is not as high as theoretically expected.

A simple method for detecting spatial trends based on GDBSCAN has been explained. GDBSCAN is used to extract density-connected sets of neighboring objects having a similar value of the non-spatial attribute(s). In order to define the similarity on an at-tribute, the domain is partitioned into a number of disjoint classes and only the values in the same class similar to each other are considered. The sets with the highest or lowest attribute value(s) are most interesting and are called influence regions, i.e. the maximal neighborhood of a center having a similar value in the non-spatial attribute(s) as the center itself. Then, the resulting influence region is compared to the circular region representing the theoretical trend to obtain a possible deviation. Different methods may be used to accomplish this comparison, e.g. difference-based or approximation-based methods. A different-based method calculates the difference of both, the observed influence region and the theoretical circular region, thus returning some region indicating the location of a possible deviation. An approximation-based method calculates the optimal approximating ellipsoid of the observed influence region. If the two main axes of the ellipsoid significantly differ in length, then the longer one is returned indicating the direction of a deviation.

Conclusion

GDBSCN algorithm is proposed to find interesting regions for trend detection in a geographic database on Bavaria. A spatial trend was defined as a pattern of systematic change of one or several non-spatial attributes in 2D or 3D space. On the basis of repeated trends of the databases certain predictions are explained. Somewhere it has observed that given algorithm is not able to give clear cut relations between different datasets.

6.6 Extended algorithm for spatial characterization and discrimination rules

In this paper, a new spatial data mining algorithm for both characterization and discrimination rules have been proposed. A characterization rule is an assertion which characterizes a concept satisfied by all or a majority number of the examples in the class undergoing learning (called the target class).A discrimination rule is an assertion which discriminates a concept of the class being learned from other classes (called contrasting classes).In medical science for diagnosis of diseases, it is very important and usable.
In this paper, proposed algorithm is very much suitable for identification of weather patterns. In this algorithm, the characteristics of some spatial objects can be found as well as what the characteristics of that spatial objects discriminate from other contrast spatial objects can also be found.

Positive aspects of this algorithms are:

1. It extracts not only the properties of target and contrast objects but also the properties of their neighbors as they impact on the characteristics of all objects.
2. It shows successful implementation of general frame work for SPDM.
3. This algorithm is more suitable for medical and weather applications.

7. Spatial data analysis using map Reduce (MR) and parallel DBMS

Two paradigms exist to query over large scale spatial data - Map Reduce and parallel database. In our paper, we have described and compared both the paradigms in terms of performance and complexity. Map Reduce is one of the

simplest and best known tool available through which one can easily write distributed programs. But at the same time , there are dozens of parallel database systems like Teredata , AsterData , Netezza , Dataupio , Oracle(via Exadata) are available in the market over last two decades and provide high level programming environment. Also it is possible to rewrite any parallel processing task as either a set of database queries or a set of Map reduce jobs. So the question arises what are the key difference between these two approaches - MapRe-duce and parallel database systems.

7.1 Analysis

7.1.1 Schema Support

Parallel DBMSs require data to fit into the relational paradigm of rows and columns. In contrast, the MR model does not require that data files adhere to a schema defined using the relational data model. That is, the MR programmer is free to structure their data in any manner or even to have no structure at all.

One might think that the absence of a rigid schema automatically makes MR the preferable option. For example, SQL is often criticized for its requirement that the programmer must specify the "shape" of the data in a data definition facility. On the other hand, the MR programmer must often write a custom parser in order to derive the appropriate semantics for their input records, which is at least an equivalent amount of work. But there are also other potential problems with not using a schema for large data sets.

Whatever structure exists in MR input files must be built into the Map and Reduce programs. Existing MR implementations pro-vide built-in functionality to handle simple key/value pair for-mats, but the programmer must explicitly write support for more complex data structures, such as compound keys. This is possibly an acceptable approach if a MR data set is not accessed by multiple applications. If such data sharing exists, however, a second programmer must decipher the code written by the first programmer to decide how to process the input file. A better approach, followed by all SQL DBMSs, is to separate the schema from the application and store it in a set of system catalogs that can be queried.

But even if the schema is separated from the application and made available to multiple MR programs through a description facility, the developers must also agree on a single schema. This obviously requires some commitment to a data model or models, and the input files must obey this commitment as it is cumbersome to modify data attributes once the files are created

Once the programmers agree on the structure of data, something or someone must ensure that any data added or modified does not violate integrity or other high-level constraints (e.g., employee salaries must be non-negative). Such conditions must be known and explicitly adhered to by all programmers modifying a particular data set; a MR framework and its underlying distributed storage system has no knowledge of these rules, and thus allows input data to be easily corrupted with bad data. By again separating such constraints from the application and enforcing them automatically by the run time system, as is done by all SQL DBMSs, the integrity of the data is enforced without additional work on the programmer's behalf

In summary, when no sharing is anticipated, the MR paradigm is quite inaccessible. If sharing is needed, however, then we argue that it is advantageous for the programmer to use a data description language and factor schema definitions and integrity constraints out of application programs. This information should be installed in common system catalogs accessible to the appropriate users and applications.

7.1.2 Data Distribution

Parallel DBMSs use knowledge of data distribution and location to their advantage: a parallel query optimizer strives to balance computational workloads while minimizing the amount data trans-mitted over the network connecting the nodes of the cluster

Aside from the initial decision on where to schedule Map instances, a MR programmer must perform these tasks manually. For ex-ample, suppose a user writes a MR program to process a collection of documents in two parts. First, the Map function scans the documents and creates a histogram of frequently occurring words. The documents are then passed to a Reduce function that groups files by their site of origin. Using this data, the user, or another user building on the first user's work, now wants to find sites with a document that contains more than five occurrences of the word 'Google' or the word 'IBM'. In the naive implementation of this query, where the Map is executed over the accumulated statistics, the filtration is done after the statistics for all documents are computed and shipped to

reduce workers, even though only a small subset of documents satisfy the keyword filter
SQL view and select queries perform a similar computation:

```
CREATE VIEW Keywords AS
SELECT siteid, docid, word, COUNT(*) AS
wordcount FROM Documents GROUP BY
siteid, docid, word; SELECT DISTINCT siteid
FROM Keywords
WHERE (word = 'IBM' OR word = 'Google') AND
wordcount > 5;
```

A modern DBMS would rewrite the second query such that the view definition is substituted for the Keywords table in the FROM clause. Then, the optimizer can push the WHERE clause in the query down so that it is applied to the Documents table before the COUNT is computed, substantially reducing computation. If the documents are spread across multiple nodes, then this filter can be applied on each node before documents belonging to the same site are grouped together, generating much less network I/O.

7.1.3 Indexes

All modern DBMSs use hash or B-tree indexes to accelerate access to data. If one is looking for a subset of records (e.g., employees with a salary greater than Rs 100,000), then using a proper index reduces the scope of the search dramatically. Most database systems also support multiple indexes per table. Thus, the query optimizer can decide which index to use for each query or whether to simply perform a brute-force sequential search. Because the MR model is so simple, MR frameworks do not pro-vide built-in indexes. The programmer must implement any indexes that they may desire to speed up access to the data inside of their application. This is not easily accomplished, as the frame-work's data fetching mechanisms must also be instrumented to use these indexes when pushing data to running Map instances. Once more, this is an acceptable strategy if the indexes do not need to be shared between multiple programmers, despite requiring every MR programmer re-implement the same basic functionality

If sharing is needed, however, then the specifications of what indexes are present and how to use them must be transferred between programmers. It is again preferable to store this index information in a standard format in the system catalogs, so that programmers can query this structure to discover such knowledge.

7.1.4 Performance

There is a potentially serious performance problem related to MR's handling of data transfer between Map and Reduce jobs. Recall that each of the N Map instances produces M output files, each destined for a different Reduce instance. These files are written to the local disk on the node executing each particular Map instance. If N is 1000 and M is 500, the Map phase of the program produces 500,000 local files. When the Reduce phase starts, each of the 500 Reduce instances needs to read its 1000 in-put files and must use a file-transfer protocol to transfer each of its input files from the nodes on which the Map instances were run. With 100s of Reduce instances running simultaneously, it is inevitable that two or more Reduce instances will attempt to read their input files from the same map node simultaneously, inducing large numbers of disk seeks and slowing the effective disk transfer rate. This is why parallel database systems do not materialize their split files and instead use a push approach to transfer data instead of a pull.

7.1.5 Flexibility

Despite its widespread adoption, SQL is routinely criticized for its insufficient expressive prowess. Some believe that it was a mistake for the database research community in the 1970s to focus on data sub-languages that could be embedded in any programming language, rather than adding high-level data access to all programming languages. Fortunately, new application frame-works, such as Ruby on Rails and LINQ have started to reverse this situation by leveraging new programming language functionality to implement an object-relational mapping pattern. These programming environments allow developers to benefit from the robustness of DBMS technologies without the burden of writing complex SQL. Proponents of the MR model argue that SQL does not facilitate the desired generality that MR provides. But almost all of the major DBMS products (commercial and open-source) now provide support for user-defined functions, stored procedures, and user-defined aggregates in SQL. Although this does not have the full generality of MR, it does improve the flexibility of database systems.

7.1.6 Fault Tolerance

The MR frameworks provide a more sophisticated failure model than parallel DBMSs. While both classes of systems use some form of replication to deal with disk failures, MR is far more adept at handling node failures during the execution of a MR computation. In a MR system, if a unit of work (i.e., processing a block of data) fails, then the MR scheduler can automatically restart the task on an alternate

node. Part of the inaccessibility´ is the result of the fact that the output files of the Map phase are materialized locally instead of being streamed to the nodes running the Re-duce tasks. Similarly, pipelines of MR jobs, materialize intermediate results to files each step of the way. This differs from parallel DBMSs, which have larger granules of work (i.e., transactions) that are restarted in the event of a failure. Part of the reason for this approach is that DBMSs avoid saving intermediate results to disk whenever possible. Thus, if a single node fails during a long running query in a DBMS, the entire query must be completely restarted.

8. Performance analysis

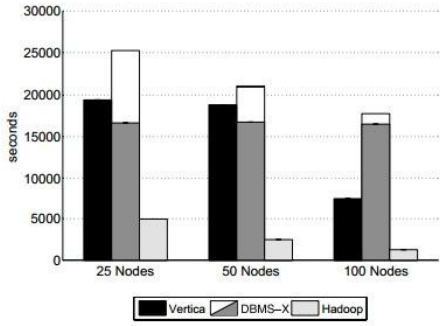


Figure 13: Data Loading

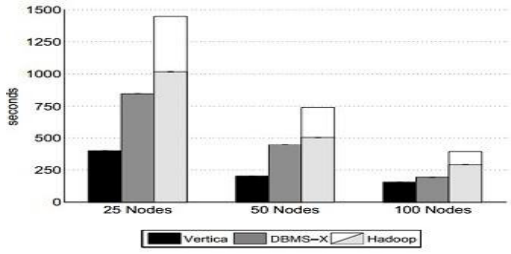
The three major systems Hadoop for Map Reduce(MR), DBMS-X for parallel DBMS and Vertica were deployed on a 100-node cluster such that each node has a single 2.40 GHz Intel Core 2 Duo processor running 64-bit Red Hat Enterprise Linux 5 with 4GB RAM.

8.1 Benchmark Test

Data Loading

The results for loading 1TB/cluster data sets is shown in the figure 13.

The most striking feature that can be observed from the figure is the difference in performance of DBMS-X compared to Hadoop and Vertica. Despite issuing the LOAD command on each node in parallel, the data is actually loaded on each node sequentially resulting in increased load time with increasing amount of data.



Task Execution

The performance results for the three systems for this task is shown as below:

It can be clearly observed that DBMS-X and Hadoop performed slower than Vertica by a factor of more than two. However, Hadoop and DBMS-X performs approximately the same, since Hadoop's startup cost is amortized across the increased amount of data processing for this experiment.

Selection Task

The Selection task is a lightweight filter to find the pageURLs in the Rankings table (1GB/node) with a pageRank above threshold, 10.

The DBMSs execute the selection task using the following simple SQL statement:

```
SELECT pageURL, pageRank
    FROM Rankings WHERE pageRank > X;
```

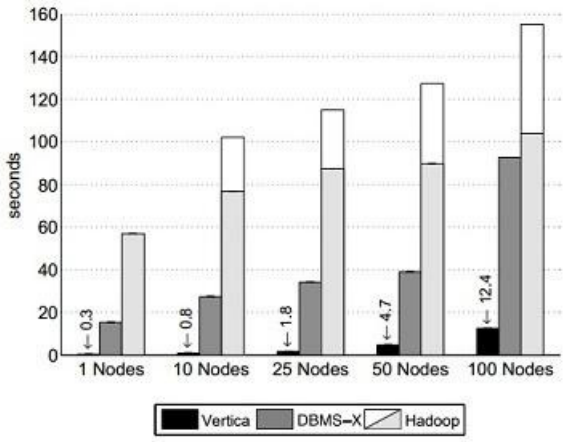


Figure 15: Selection Task Results

Figure demonstrate that the parallel DBMSs outperform Hadoop by a rather significant factor across all cluster scaling levels. Although the relative performance of all systems degrade as both the number of nodes and the total amount of data increase, Hadoop is most affected.

This is due to Hadoop's increased start-up costs as more nodes are added to the cluster, which takes up a proportionately larger fraction of total query time for short-running queries. Another reason can be is that both Vertica and DBMS-X use an index on the pageRank column and store the Rankings table sorted by pageRank

4. **Aggregation Task**

This task requires each system to calculate the total adRevenue generated for each sourceIP in the UserVisits table (20GB/node), grouped by the sourceIP column.

The SQL commands to calculate the total adRevenue is straight-forward:

```
SELECT sourceIP, SUM (adRevenue) FROM UserVisits
GROUP BY sourceIP;
```

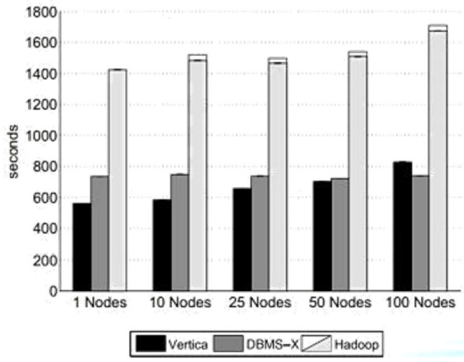


Figure 16: Aggregation Task Results

Clearly both DBMSs i.e. DBMS-X and Vertica outperformed Hadoop. The DBMSs execute these queries by having each node scan its local table, extract the sourceIP and adRevenue fields, and perform a local group by. These local groups are then merged at the query coordinator, which outputs results to the user. Thus, unlike Hadoop, their runtime is dominated by the cost to transmit the large number of local groups and merge them at the coordinator.

9. Discussions

In this report we have presented the various data structures for indexing with their usability for querying in spatial data. Various algorithms and methods developed have been discussed in brief and for detail explanation bibliography may be referred. Finally we dived into parallel data mining and performed benchmark performance test of three techniques i.e. Map Reduce, Vertica and DBMS-X.

References

[1] A Comparison of Approaches to Large-Scale Data Analysis SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data
[2] Algorithms for processing K-closest-pair queries in spatial databases. Data and Knowledge Engineering 49 (2004).
[3] An index structure for efficient reverse nearest neighbor queries in Data Engineering, 2001. Proceedings. 17th International Conference on 2001.
[4] Nearest Neighbor Queries SIGMOD '95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data
[5]Closest Pair Queries in Spatial Databases. SIGMOD '00 Proceedings of the 2000 ACM SIGMOD international conference on Management of data
[6] R-TREES. A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING.SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data
[7] R*-tree: An Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Int. Con. on Management of Data, Atlantic City, NJ
[8] Algorithms for Characterization and Trend Detection in Spatial Databases Published in Proceedings of 4th International Conference on Knowledge Discovery and Data Mining
[9] Density-Connected Sets and their Application for Trend Detection in Spatial Databases.PROC. 3RD INT. CONF. KNOWLEDGE DISCOVERY AND DATA MINING (KDD'97)
[10] An Introduction to Spatial Database Systems. Special Issue on Spatial Database Systems of the VLDB Journal (Vol. 3, No. 4, October 1994)
[11] A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Published in Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)
[12]Gueting, R.H. 1994. An Introduction to Spatial Database Systems. VLDB Journal 3(4).