International Journal of Database Theory and Application
Vol.10, No.5 (2017), pp.33-46
http://dx.doi.org/10.14257/ijdta.2017.10.5.04

A Topology-concerned Spatial Vector Data Model for Column-oriented Databases

Kun Zheng[1], Mei-Po Kwan[2], Falin Fang[3*], Junjun Yin[4], Danpeng Gu[5] and Yanli Fu[6]

[1,3,5] China university of geoscience (wuhan)
[2]University of Illinois at Urbana-Champaign
[4]University of Illinois at Urbana-Champaign Author Affiliation(s)
[6]Jinan Research Institute of Geotechnical Investigation & Surveying
[1]Michael_Power@Sina.com; [2]mpk654@gmail.com; [3]cugwhlin2014@126.com;
[4]jyn@illinois.edu; [5]gdp12315@163.com; [6]fuyli6@126.com

Abstract

In today's "Big Data" era, the volume of spatial data grows rapidly. Addressing the challenges in efficient spatial Big Data storage and management becomes urgent. However, conventional row-based spatial databases have many limitations, such a slow data I/O efficiency, low data retrieval performance, poor scalability, and high maintenance costs. These conventional spatial databases are no longer suitable for today's spatial Big Data. On the other hand, column-oriented databases have several superior features, such as high reliability, scalability and fault tolerance. More importantly, they have better I/O efficiency for query processing. This paper presents a topology-concerned spatial vector data model for column-oriented databases and designed the physical storage model, which is a unified model for storing and managing information of geometry, attribute and topology of spatial objects. For the storage characteristics of column-oriented databases, the model designed a new Rowkey encoding schema with the Z-order filling curve approach. This encoding schema of Rowkey considering spatial proximity optimizes the organizational structure of spatial data models. It means nearby spatial objects are also closer to each other in the physical storage, which can further improve the efficiency of spatial data storage and enable spatial query capability in column-oriented databases. Three experiments were conducted including data storing, range query and K-NN query to analyze the efficiency and spatial query capability of the data model. The results of the experiments show that the data model has good scalability and efficiency on the vector data storage and spatial query. It is suitable for large-scale spatial vector data storage and management in column-oriented databases.

***Keywords**: Spatial Vector Data Model, Column-oriented Database, Topology, Spatial Proximity*

1. Motivation and Introduction

Recent advances in geospatial data acquisition technology have led to dramatic increase in large-volume, multi-scale and complex spatial datasets [1]. To efficiently deal with massive spatial datasets, it requires innovative spatial database management system (SDBMS) to support scalable data handling, such as data storage with flexible data schema and efficient database querying. Traditional file systems, in particular, SDBMS extended from relational databases, are becoming inadequate for efficient massive data storage and processing, where relational databases with a rigidly defined, schema-based approach make it difficult to quickly incorporate new types of data, and to achieve dynamic scalability while maintaining the performance users demand [2]. In this regard, non-relational (also known as NoSQL) databases are becoming the mainstream database technology for Big Data research and applications [3]. The state-of-the-art NoSQL databases in the market employ distributed technology to enhance efficient mass storage and database querying, where multiple instances of NoSQL databases work together across a network. There are several kinds of NoSQL databases according to the difference of data model, for example key-value database, column-oriented database and so on. In contrast to another NoSQL databases, Column-oriented database organize the 'value' with multiple columns. In addition, Column-oriented database provided a pattern named "column family" for improving the efficiency of data query by storing some columns often needed to access together in the same storage area [3, 18].

However the spatial data models developed based on relational databases are not suitable for NoSQL environment, many current research efforts focus on extending the data models in NoSQL database to support efficient spatial vector data storage and management, such as MD-HBase [4], HBaseSpatial [5],Dart [6].

All of them only store the geometry and properties but not the topology relation information, where the topology relationships are managed separately from the physical data storage, by using the spatial indexes, such as R-tree, KD-tree, Quad-tree, *etc.* [4, 7-9]. Then when new data are inserted into the database, the spatial indexes should be updated accordingly, which can be accumulated as a large computation overhead especially considering the massive number of spatial objects in dealing with spatial Big Data. Furthermore they do not consider the spatial proximity of spatial objects when storing data across different computing nodes in a network, where spatially distant objects can be stored in the same node or spatially close objects are stored in different nodes. Subsequently, it can potentially lead to inefficient I/O when performing spatial query processing, as the targets should be accessed and retrieved from multiple computing nodes.

Based on the above consideration, this paper proposes a topology-concerned spatial vector data model for column-oriented Databases. In particular, we designed and implemented a spatial vector data model by establishing a topological vector data model and integrating it with the physical storage structure of the column-oriented database. Also it designs a Rowkey encoding scheme with modified SFC (Space Filling Curve) to enable the capability of building topology relationships on large spatial vector data in column-oriented database and performing spatial query, such as range query and k-NN query. Following, Section 2 introduces the topology-concerned spatial data model by exploring the topological relationships and the topology-concerned logical data model. Section 3 focuses on the development of the spatial vector data model in column-oriented databases and the design of Rowkey encoding. Section 4 presents the experimental implementation for evaluating the data model using HBase. Finally, Section 5 concludes and summarizes this research.

2. Topology-concerned Spatial Vector Data Model

Compared to raster data, vector data has more complex data structures and contains spatial topological relations. A geospatial object contains not only information of its geometric elements, such as location and shape, but also the topological relationships among the geometric elements, such as edges with nodes, and lines with surfaces [10-12].

2.1. The Approach of Topological Relations

Spatial relationships are the fundamentals of GIS. They play an important role in spatial analysis and GIS applications [13].Topological relations are one of the most important elements in spatial relationships. It is the key for understanding how geospatial objects are organized in databases and it helps optimize data analysis, processing, and query.

9-Intersection Model (9-I Model) is the most commonly used topology model. Many researchers study and extend the topological relations depicted by the 9-I Model [14, 15]. However, the 9-I Model relies on a large complementary logical set to calculate all the topological relations of every spatial object, which can potentially generate a large amount of redundant data. Therefore, it is not suitable for direct use in spatial data models. On the other hand, border relationship between spatial objects are easier to express, in particular, the core topological relationships such as adjacent, intersection and contains that can be used to derive other types of topological relationships [10, 11, 16].

To illustrate the border relationships of two given spatial objects M and N, where B[M] and B[N] are the boundaries of M and N respectively, and I[M], I[N] represent the interior of M and N, the topological model can be expressed as the matrix M-1.

$$\begin{bmatrix} B[\mathrm{M}] \cap \mathrm{B[N]} & B[M] \cap I[N] \\ I[M] \cap B[N] & I[M] \cap I[N] \end{bmatrix} \qquad \text{(M-1)}$$

Each element in the matrix is valued either “0” or “1”. The boundary and the interior of a point is itself; the boundary of a line is the two nodes at both ends of the line segment, while the interior is the portion between these two nodes; the boundary of a polygon is the collection of ordered edges and the interior is the area enclosed by the edges. From the M-1 matrix, there are 16 relationships among different combinations of spatial objects. To be specific, the collection of spatial relationships include equal, contain, touch, adjacent, intersection and disjoint relations between points, lines and polygons.

2.2. Topology-concerned Logical Data Model

Based on the simple feature specifications of OGC (Open GIS Consortium), the topology-concerned logical data model consists of three components: Geometry, Property, and Topological relationships [17], which are illustrated in Figure 1.

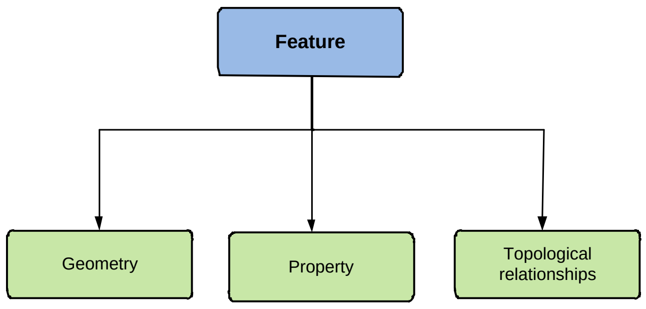


Figure 1. The Diagram of Topology-Concerned Data Model

In 2-dimensional space, the geometric shape of a spatial object can be described as a point, line, or polygon. To express the topological relationships accurately, the geometry of the topology-concerned data model not only consists of the Point, Line, and Polygon elements, but also includes the Node and Edge elements. As it is shown in Figure 2, the point class describes the shape of 0-D objects; Line describes the shape of 1-D objects; Polygon describes the shape of 2-D objects.

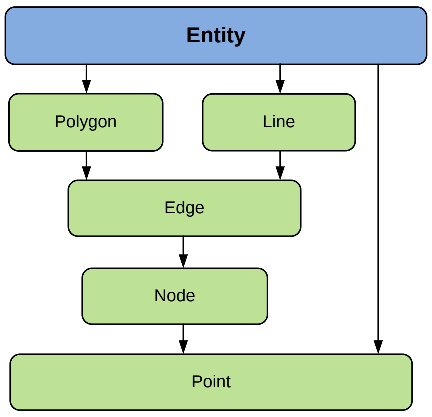


Figure 2. The Logical Geometrical Structure of the Topology-Concerned Data Model

A Point is represented by a single pair of coordinates, *e.g.*, (x, y), and an Edge is the line segment consisting of a series of ordered points. Each edge has two nodes where the node is the end point of the edge and it can be the end point of one or more edges. A Line consists of a series of ordered points, or a series connected edges. A Polygon is bounded by edges, where there is no self-intersection for each edge and an edge can be shared by two polygons. The topological relationships can be described with the following formulation.

$$Topo(A,B) =< A,B,R,V > \tag{2.1}$$

In this formulation, A and B can be any spatial objects; R is the topological relationship between A and B; V is the measurement of such a topological relationship, which can be a null value. The topological relationships in this data model are all about the boundary and interior relations between the spatial objects, which are expressed by using Node and Edge to store the topological information. For the topological relationships, Node can express the touch relation between points and lines, while Edge can express the line's topological relationships between Line and Line. In addition, Line and Polygon can be separated to an array of relationships between Edge and Edge, as Polygon is essentially enclosed by Lines.

In other words, in our topology-concerned logical data model, all topological relationships can be expressed or deduced by Node and Edge. Since V can be calculated on-demand once the topological relationships are identified, and hence formulation (2.1) can be simplified to the following:

$$Topo(A,B) =< A,B,R > \tag{2.2}$$

In terms of feature's property, it can be described as a set of attributes, where the values are paired with the corresponding attribute name. The formulation is shown below:

$$Property(F) = \{(Attr_1, Value_1), (Attr_2, Value_2), ..., (Attr_n, Value_n)\} \tag{2.3}$$

According to the designed topology-concerned logical data model, a feature can be defined by the following abstract class.

$$
\begin{aligned}
&Feature\{\\
&\quad GeometryType\ gType = \{Point, Line, Pologon, Node, Edge\};\\
&\quad Geometry\ geo\\
&\quad Topology\ Topo = \{Topo(A,B), Topo(A,C), ...\};\\
&\quad Property\ property(F)\\
&\}
\end{aligned} \tag{2.4}
$$

3. Data Storage Model based on Column-oriented Databases

The data storage model in column-oriented databases can be defined as a distributed, sparse and column-oriented mapping table. Compared to row-oriented database, column-oriented databases have sparse storage characteristics, and therefore are more efficient for operations like data query, insert, update, and delete in data intensive databases. An illustration of the logical and physical table structure and storage mode in a column-oriented database is shown as Figure 3[18].

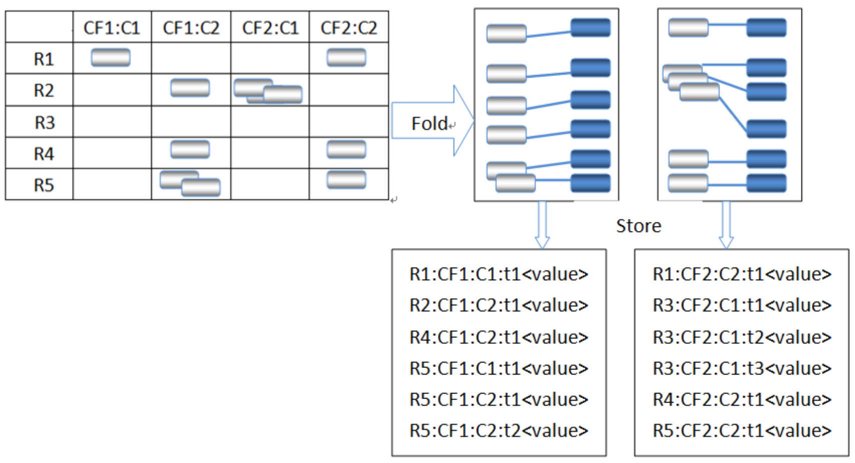


Figure 3. A Diagram for the Logical and Physical Table Structure in Column-Oriented Database

Column-oriented databases utilize Rowkey (row index) to organize the table into a two-dimensional matrix. Through the Rowkey, column families (CF) and columns, we can smoothly access the information in the corresponding cell. Moreover, multiple versions of data in the same cell will be stored concurrently, where each data is tagged with a timestamp when it is stored. These multiple versions of data are sorted in descending order so that the latest version will be accessed first. In summary, each cell now contains not only the information of column and value, but also Rowkey and timestamp, where the table is sorted according to the dictionary order of Rowkey.

3.1. Topology-concerned Spatial Vector Data Storage Model

(1) Logical Data Storage Model

Column-oriented databases only support "string" data type, and different column families are stored separately. All spatial objects and property information must be organized into tables, which consist of Rowkey, column family, column qualifier and timestamp. Data within a column family is addressed via its column qualifier, or simply, column. Column qualifiers, defined as "<family>:<qualifier>", do not need to be specified in advance nor should be consistent between rows.

For the topology-concerned data model, the vector data consist of geometry, property, and topology. The data model should be transformed to column-oriented organization before being directly used for the column-oriented table. Following the organization rules of column-oriented databases, the logical storage structure of the data model is shown as Figure 4.

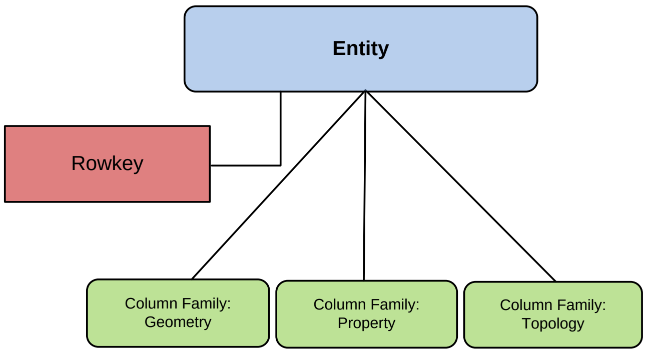


Figure 4. Logical Structure of Data Storage Model

In column-oriented databases, any spatial vector object from the data storage model can be defined by the following formulation:

$$M(f) = < Rowkey, Timestamp, \{Geometry; Property; Topology\} > \quad (4.1)$$

Where Rowkey is the unique identifier of the spatial object in the database; Timestamp records the time when the particular object gets updated. This model contains three column families: Geometry, Property, and Topology, where Geometry can be defined by formulation (4.2); geoType is any of “Point”, “Line”, “Polygon”, “Node”, or “Edge”; Shape is an(x,y) coordinate sequence or an array of other object’s ID such as Edge, Node; and ProCoorSys is the spatial reference information.

$$Geometry(f) = [geoType, ProcoorSys, Shape] \quad (4.2)$$

$$geoType = \{"Point", "Line", "Polygon", "Edge", "Node"\} \quad (4.3)$$

Property column family can be described by <Field, Value>, and Topology column family can be define as formulation (4.4)

$$Topology(f) =< TopoR, Rowkey > \quad (4.4)$$

Within the topology-concerned spatial vector data model, the boundary TopoR (topological relations) of Point, Line and Polygon types has been converted to the relations between Node and Edge. For Node type, the topology information is its touched Edge, which is recorded as $< Touch, Rowkey_{Edge} >$; the topology information of Edge consists of two types: the first is the relationship between Edge and Node record by $< Touch, Rowkey_{Node} >$; the second is the adjacent relation including Line with Line, Line with Polygon, and Polygon with Polygon, recorded by $< Adjacent, Rowkey_{Line} >$ or $< Adjacent, Rowkey_{Edge} >$ The interior topological relations of Point, Line and Polygon types can be recorded by $< Contains, RowkeyArray >$,where RowkeyArray is the array of Rowkey values of the contained spatial objects. So TopoR is valued one of the set $\{"Touch", Adjacent, "Contain"\}$.

Based on the above analysis for the logical structure of the data storage model in column-oriented databases, the logical view of the model is illustrated in Table. 1:

Table 1. Logical View of Storage Data Model

Rowkey	Timestamp	Column Family: Geometry		Column Family: Attribute		Column Family: Topology	
		Info	Value	Info	Value	Info	Value
ID1	T1	Shape	(x,y)	Atrrib:1	Value1		
	T2	ProCoorSys	SR1	Atrrib:2	Value2		
	T3	Type	Point Node				
ID2	T1	Shape	(x,y)	Atrrib:1	Value3		
	T2	ProCoorSys	SR1	Atrrib:2	Value4		
	T3	Type	Point Node				
ID3	T1	ProCoorSys	SR1	Atrrib:1	Value5		
	T2	Type	Line Edge	Atrrib:2	Value6		
	T3	Shape	(x,y)			Adjacent	ID4, ID5,…
	T4					Touch	ID1, ID2,…
ID4	T1	ProCoorSys	SR1	Atrrib:1	Value7		
	T2	Type	Polygon	Atrrib:2	Value8		
	T3	Shape	ID3,…			Contain	… …
ID5	T1	ProCoorSys	SR1	Atrrib:1	Value9		
	T2	Type	Polygon	Atrrib:2	Value10		
	T3	Shape	ID3,…			Contain	… …

Note that every spatial object has three column families in addition to the Rowkey and the timestamp. In this table view, all data is ordered and identified by the Rowkey. Rowkey is always treated as a “byte” array. The Rowkey maps a list of column families, where each column family consists of a list of column qualifiers with corresponding timestamp and value. In other words, those rows with the same Rowkey belong to the same spatial object. The default version number of a spatial object is the timestamp. To access a specific version of spatial object, if the timestamp is not specified, the latest one will get returned; if a timestamp is not recorded when the data was written, the current timestamp of the system is used.

(2) Physical Data Storage Model

In column-oriented database, the physical storage of logical tables is organized by the column family only. One column family can be stored in many files, but each file can only store one column family data, where empty columns are not stored to improve efficiency. Table 2 provides the overall view of the physical storage for our data model.

Table 2. Overall View for the Physical View of Column Family

Rowkey	Timestamp	Column Family	
		info	value
ID	T1	*<family>:<qualifier1>*	value1
	T2	*<family>:<qualifier2>*	value2

Based on the above illustrations, the spatial objects can be defined as below:

$$\begin{aligned}
&\text{Point:}\{\text{ Rowkey, Geometry: Type} = \text{Point, Geometry: Coordinates, Attribute}\}\\
&\text{Line:}\{\text{ Rowkey, Geometry: Type} = \text{Line, Geometry: Coordinates, Topology : Contain, Attribute}\}\\
&\text{Ploygon: }\{\text{Rowkey, Geometry: Type} = \text{Polygon, Geometry: Coordinates, Topology}\\
&\text{: Contain, Attribute }\}\\
&\text{Node:}\{\text{ Rowkey, Geometry: Type} = \text{Node, Geometry: Coordinates, Topology : Touch, Attribute }\}\\
&\text{Edge: }\{\text{ Rowkey, Geometry: Type} = \text{Edge, Geometry: Coordinates, Topology}\\
&\text{: }\{\text{Adjacent, Touch }\}\text{, Attribute }\}
\end{aligned}$$

In addition, every table is initialed with a Region and can be further separated into several Regions, where each Region is recognized as a storage node. When the data volume increases and reaches a threshold value, the table will be split into two Regions automatically by the distributed storage model and new Region will be stored in another physical storage node. Naturally, when the table keeps growing, it will be divided into multiple Regions.

3.2. Design of Rowkey Encoding

Rowkey is a unique identifier for spatial objects in column-oriented databases. All rows are sorted lexicographically based on the Rowkeys. In terms of physical storage, a well-designed Rowkey encoding will make significant difference for efficient database query. For example, a poorly designed Rowkey encoding approach that uses ill-suited schema is usually a common source of hotspotting. Hotspotting occurs when a large amount of client traffic is directed at one node, or at only a few nodes of a cluster. Hotspotting overwhelms the node that is responsible for hosting that particular region and causes performance degradation and potentially leads to region unavailability. It can also have adverse effects on other regions hosted by the same region server as the host is unable to serve the requested load. It is important to design data access patterns so that a cluster is fully and evenly utilized [18, 19].

Although the design of Rowkey encoding has been proposed in many mature frameworks and data models, column-oriented databases mainly use monotonically increased Rowkeys (*i.e.*, using a timestamp or a sequence). For time series data, Rowkey is usually designed as $[metric\_type][event\_timestamp]$, which contradicts previous studies in which timestamp is not used as a key[18]. Nevertheless, all these Rowkey encoding methods do not deal with spatial data. In other words, these methods do not consider the spatial characteristics of the spatial vector datasets, where some spatial objects are (geographically) closer to others and some are farther away from others. In this paper, the design and implementation of the Rowkey encoding for our spatial vector data model in column-oriented databases considers the following two conditions:

(1)Column-oriented databases do not directly support spatial vector data, and database query relies solely on the Rowkey. Usually, a Rowkey is only a unique ID sequence, which does not have any spatial information. In addition, conventional Rowkeys do not have spatial indexing capability. To support efficient spatial database query, a complex spatial index structure must be built.

(2) By simply using the ID sequence or time stamp as the Rowkey, all spatial objects would be separately stored in various storage nodes according to the order of the sequence. In case of dealing with large volume of data, this will reduce the performance of spatial query and increase the cost for database maintenance.

Therefore, our solution for the design of the Rowkey encoding tries to avoid these two problems: (1) Convert the Rowkey encoding from 2 dimensions to1 dimension, and (2) Consider the spatial proximity of spatial objects and integrate it into the encoding algorithm. In this case, we chose the space filling curves (SFCs) method, which has been extensively used as a mapping schema from the n-D space into the 1-D space. Its act like a thread that pass through every cell element in D-dimension space so that every cell is visited exactly once [20, 21].More importantly, in term of space, the index encoded by the SFC method tends to order objects with smaller spatial proximity closely in the sequence.

With these characteristics of space-filling curves, the encoding of Rowkey can combine the conventional Rowkey pattern with SFCs. So, we designed a new Rowkey schema as following formulation (4.5):

$$Rowkey(f) = Geocode(f.geometry) + FID(f) \tag{4.5}$$

The $Geocode(f.geometry)$ is the encoding function of space-filling curves, which can be shown as following formulation (4.6):

$$Geocode(f.geometry) = ZG(P)_1 + ZG(P)_2 + ... + ZG(P)_m \tag{4.6}$$

Where the parameters are the spatial object's geometry coordinates and the $ZG(P)_m$ is the encoding of space-filling curves in each layer. The function $FID(f)$ generates unique sequential IDs in the database. Encoding by the space-filling curves approach, the Rowkey of various spatial objects in same spatial partition has the same prefix code. Coupled with a unique sequence ID number as a suffix, not only avoid the duplication of Rowkey, but also overcome the shortcomings of Rowkey which encode only by unique sequence ID. Using formulation (4.5), the data which are nearby together can also be aggregately stored. This encoding makes data storage take space proximity into account. Furthermore, Rowkey have the basic characteristics of spatial index, which makes it possible for making spatial query directly in Database.

Because of holding the characteristic of easy calculation, Z-order curve approach will be used in the progress of Rowkey encoding with consideration of high efficiency requirements in the vector spatial Big Data storage and query process,

Definition 4.1:

In a d-dimensional space, given $(P_1,...,P_d) \in D_1 \times ... \times D_i \times ... \times D_d,\ D_i = [L_i, U_i](i = 1....d)$ the number of partitions in each dimension are $2^m (m \in N)$ and the Z-order curve value of P is $ZG(P)_m = (zg_1,...,zg_d)$, where $zg_i = \lfloor p_i / ((U_i - L_i)/2^m) \rfloor$ .From the Definition 4.1, the zg_i can be represented by using length of m binary bit long string.

Definition 4.2:

Grid of Z-order curve is defined as the d-dimensional grid of Zcurve of order min the $k(1 \le k \le m)$ times to split the space $R^d = [0,1]^d$ into hyper cubes, which can be determined by the two endpoints of the diagonal $S=(s_1,...,s_d)$ and $T=(t_1,...,t_d)$, and it is denoted as: $ZGrid(k)=<S,T>$,with side length $r=(t_i-s_i)\times 2^{-k}$ .In the 2D space, Zcurve of order m split the edge length square into $2^{2k}(1\le k\le m)$ equal-sized grids, where the length of each grid is $r=2^{-k}$. Any spatial objects in the same grid have the same Z-order curve code.

Definition 4.3:

The minimum grid of spatial object G is defined as the smallest grid containing all of the points of the object, which is denoted as $Mingrid(G)$ .According to the definitions above, the Z-order curve encoding algorithm is as below:

Given a spatial data object G, the highest order of Z-order curve is m. First, from the first (1) order of Z-order curve to m, we calculate the highest order k for $Mingrid(G)$, and convert G to Z-order curve value $ZG(G)$ in binary bits for each layer. Finally the value of Z-order curve will be jointed. For Point objects, we calculate its coordinates directly, while for other spatial objects, the center point or the minimum bounding box (MBR)of the geometry will be used. Finally, $ZG(G)$ is converted to the corresponding encoding. Below is the pseudo-code for 2D Z- curve of m-order encoding algorithm:

```
Algorithm 4.1: Encoding of 2D Z-order curve
Input:        spatial object G, Highest order m
Output:       Encoding of G
0   Initial order k to 1
1   for k < m do
2           if ( $ZGrid(k).r$ <G.Range | every vertex in G not in same zg):
3                   goto 5
4   end for
5   for order =1 to k do
6     /*convert xy coordinate value into 2 dimension array*/
7   zg[order][2]= $(\lfloor G.CenterPoint/((U_i-L_i)/2^{order})\rfloor)_2$
8       for i=1 to order do
9           for j=1 to 2 do
10              $ZG(P)_i = ZG(P)_i + 2^{2(i-1)+j-1}$ *zg[i][j]
11          end for
12      end for
13      Zcode:= $2^{2i}$ *Zcode + $ZG(P)_i$
14  end for
```

4. Experimental Implementation

For this study, the distributed computing environment is established on 13 PCs. Each PC installs a VMware software (version 8.0.4) with a total of 13 virtual Linux operating systems (Ubuntu 14.04.2 LTS) to accommodate Apache Hadoop(version2.5.2) and HBase(version1.0.1.1), where the parallel compute framework is Yarn. The system consists of 2 Master nodes (one is active node, another is backup node), 8Slave nodes (Region Server, Data node) and 3 Zookeeper nodes (Zoo1, Zoo2, Zoo3).

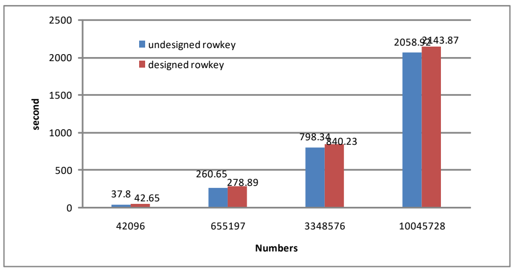
As there is no vector data model in HBase, no existing software or tools are available for storing the spatial vector data in a column-oriented database. Therefore, a special toolkit is developed that stores the spatial vector dataset in HBase, based on the Yarn framework. In order to analysis the designed data model, three experiments on data storing, range query and k-NN query were be conducted.

Firstly, for comparing the change in efficiency of data storing, different size data were imported with two different Rowkey encoding. One is designed Z-curve Rowkey encoding, and another is undesigned Rowkey which use sequence number from 1 to n.

Four data sets with different volumes were selected, from 42,096 to more than 10 million road lines which maximum size is over 21.6GB.The result is shown in Table 3 and illustrated in Figure 5-1 and Figure 5-2. Seeing from the experiment result, the designed Rowkey has the same performance level with sequence number Rowkey, and the time consumed is increased slowly even with the volume of data increasing.

Table 3. Time Consumed of Data Storing

Numbers	designed Rowkey	undesigned Rowkey
42096	42.65s	37.80s
655197	278.89s	260.65s
3348576	840.23s	798.34s
10045728	2143.87s	2058.92s



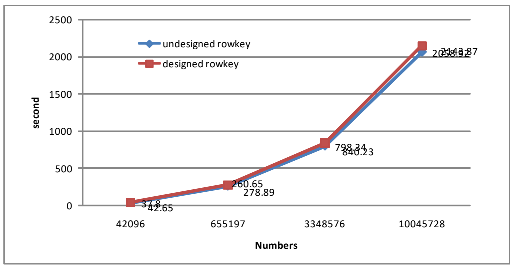
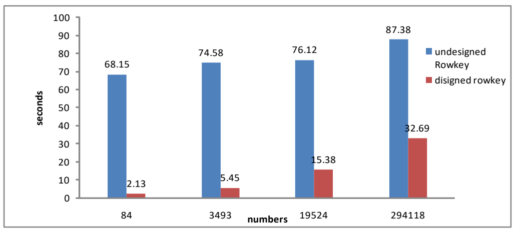


Figure 5-(1, 2). Time Consumed Of Data Storing

For the performance of spatial query in HBase, spatial range query and K-NN query experiments are conducted with two different Rowkey. In the processes of range query, time consumed were recorded with the different query range, and whole spatial objects data were get from HBase. The recorders shown in Table 4 are time cost of the undersigned Rowkey data and designed Rowkey data. Two query methods results are illustrated in Figure 6-1 and Figure 6-2. From Figure 6-1 and Figure 6-2, it shows that the model with designed Rowkey has more highly efficiency than undersigned one. It’s clear that efficiency of the spatial query did not drop down even the range expanded. And the cost of time for spatial query increased more slowly than the volume of query results. It proves that the designed data model not only can make HBase support range query, but also speed up the ratio of spatial query in HBase.

Table 4. Time Consumed of Spatial Query

Range	Number	undersigned Rowkey	dersigned Rowkey
(-201,-11; 4.3,133.6)	84	68.15s	2.13s
(-201,-11; 450,382)	3493	74.58s	5.45s
(-201,-11; 758,544)	19524	76.12s	15.38s
(-201,-11; 2362.7,1776.9)	294118	87.38s	32.69s



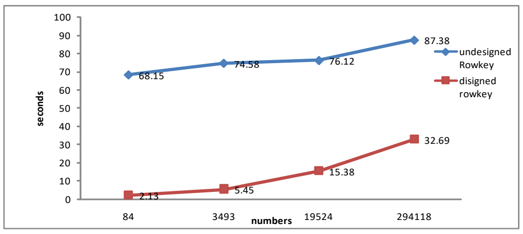
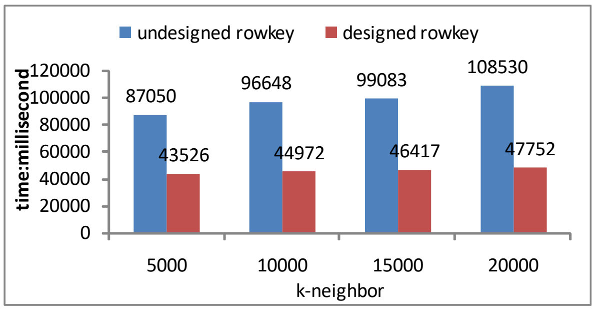


Figure 6-(1, 2). Time Consumed of Spatial Range Query

In the process of k-NN query, more than 4 million points were stored in HBase with different Rowkey encoding. In process of k-NN query, the cost of time for K-NN were recorded with different Rowkey encoding and shown in Table 5, Figure 7-1 and Figure 7-2. From Table 5, Figure 7-1 and Figure 7-2, it presents that designed Rowkey can accelerate query speed for k-NN. Form all of the analysis of experiments, it cleared that designed data model has good ability on the storage and spatial query in the column-oriented databases.

Table 5. Time Consumed of K-NN Query

The value of K	undersigned Rowkey	dersigned Rowkey
5000	87.05s	43.52s
10000	96.64s	44.97s
15000	99.08s	46.41s
20000	108.53s	47.75s



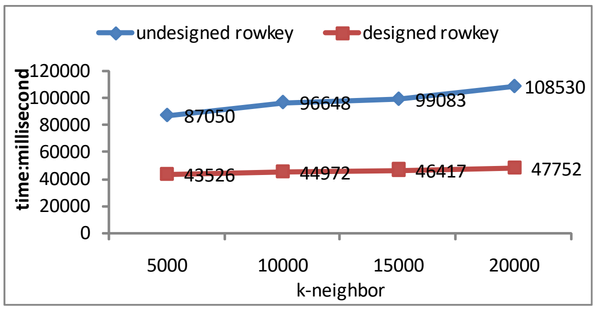


Figure 7-(1, 2). K-NN Query Performance

5. Conclusions

This paper presents a topology-concerned geospatial vector data storage model for column-oriented databases. The development of the data model is motivated by the urgent and enormous need to transform the storage and management of vector Big Data from RDBS to cloud storage. This transformation is primarily driven by rapidly growing spatial data in many fields to solve vector Big Data problems in GIS and spatial analysis. The results of this study show the following: (1) the study establishes a topology-concerned geospatial vector data storage model for column-oriented databases. This data model helps to achieve unified data management for handling geometry information, attribute information and topological information at the same time. (2) The efficiency of the spatial vector data storage model is optimized by utilizing the Z-order curve encoding with consideration of both the characteristics of Rowkey and the proximity of spatial data. (3) With the realized spatial proximity enabled Rowkey, the column-oriented database has a basic spatial indexing capability, and can directly support spatial queries. (4)This study leads important directions for future research on applications of vector Big Data management.

Acknowledgements

The authors would like to thank the National Key Research Program Plan of China (No. 2016YFB0502603), the Natural Science 437 Foundation of Hubei Province of China (No. ZRY2015001543) and Fundamental Research Founds for National University, China University of Geosciences (Wuhan) (1610491B20).

References

[1] Wright, D.J. and S. Wang, The emergence of spatial cyberinfrastructure. Proceedings of the National Academy of Sciences of the United States of America, 2011. 108(14): p. 5488-5491.
[2] Han, J., Survey on NoSQL database. in International Conference on Pervasive Computing and Applications. 2011.
[3] Chandra, D.G., BASE analysis of NoSQL database. Future Generation Computer Systems, 2015. 52(C): p. 13-21.
[4] Nishimura, S., MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. in IEEE International Conference on Mobile Data Management. 2011.
[5] Zhang, N., HBaseSpatial: A Scalable Spatial Data Storage Based on HBase. in IEEE International Conference on Trust, Security and Privacy in Computing and Communications. 2014.
[6] Zhang, H., Dart: A Geographic Information System on Hadoop. in IEEE International Conference on Cloud Computing. 2015.
[7] Cary, A., Leveraging Cloud Computing in Geodatabase Management. in IEEE International Conference on Granular Computing, Grc 2010, San Jose, California, Usa, 14-16 August. 2010.
[8] Zhong, Y., A novel method to manage very large raster data on distributed key-value storage system. 2011.
[9] Hsu, Y.T., Key Formulation Schemes for Spatial Index in Cloud Data Managements. in IEEE International Conference on Mobile Data Management. 2012.
[10] Anselin, L. and A. Getis, Spatial statistical analysis and geographic information systems. The Annals of Regional Science, 1992. 26(1): p. 19-33.
[11] Bhaskaran, S., Introduction to Geographic Information System. 2015. 544-546.
[12] Kwan, M.P., A combinatorial data model for representing topological relations among 3D geographical features in micro-spatial environments. International Journal of Geographical Information Science, 2005. 19(10): p. 1039-1056.
[13] PAPADIAS, D. and Y. THEODORIDIS, Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science, 1997. 11(2): p. 111-138.
[14] Papadias, D.,Topological Relations in the World of Minimum Bounding Rectangles: a Study with R-trees. Acm Sigmod Record, 1995. 24(2): p. 92-103.
[15] Laurini, R., A conceptual framework for geographic knowledge engineering. 2014: Academic Press, Inc. 2-19.
[16] Pouliot, J.,Reasoning about geological space: Coupling 3D GeoModels and topological queries as an aid to spatial data selection. Computers & Geosciences, 2008. 34(5): p. 529-541.
[17] Zheng, K. and Y. Fu, Research on Vector Spatial Data Storage Schema Based on Hadoop Platform. International Journal of Database Theory & Application, 2013. 6(5): p. 85-94.
[18] George, L., HBase The Definitive Guide. 2011: O'Reilly Media.
[19] Patel, A.B., M. Birla and U. Nair. Addressing big data problem using Hadoop and Map Reduce. in Nirma University International Conference on Engineering. 2013.
[20] Mokbel, M.F., Aref, W.G. & Kamel, I. GeoInformatica (2003) 7: 179. doi:10.1023/A:1025196714293.
[21] Couch, P.J., B.D. Daniel and T.H. Mcnicholl, Computing Space-Filling Curves. Theory of Computing Systems, 2012. 50(2): p. 370-386.

Authors

Kun Zheng, received his Ph.D from China University of geoscience. He is currently an associate professor with the Faculty of Information Engineering, China University of Geosciences, Wuhan, China. His current research interests involve Spatio-Temporal data Visual Analytics, Storage and management of spatial big data.

Mei-Po Kwan, received her Ph.D from University of California, Santa Barbara. She is the editor, Annals of the Association of American Geographers, the president, International Association of Chinese Professionals in Geographic Information Sciences and so on. Her research addresses health, social, transportation, economic, and environmental issues in urban areas through the application of innovative geographic information system (GIS) methods.

Falin Fang, received the B.E. degree from China University of Geosciences. He is currently master degree candidate with Faculty of Information Engineering, China University of Geosciences, Wuhan, China. His current research interest is the storage and management of spatial data and parallel processing for massive spatial data

Junjun Yin, obtained his PhD of Spatial Information Science from Dublin Institute of Technology, Ireland. He is currently a postdoctoral research associate at the CyberGIS Center. His main research interests focus on spatial analytics for large scale geopstial datasets, Location Based Services and human mobility patterns within the context of Urban Informatics.

Danpeng Gu, received the B.E. degree from China University of Geosciences. He is currently master degree candidate with Faculty of Information Engineering, China University of Geosciences, Wuhan, China. His current research interest is Spatio-Temporal data Visual Analytics and parallel processing for massive spatial data

Yanli Fu, received the Master degree from China University of Geosciences. She currently works at JiNan Geotechnical Investigation and Surverying Institute, JiNan, China. Her current research interest is the storage of spatial data and management of geographic information.