

Received April 21, 2019, accepted June 7, 2019, date of publication June 13, 2019, date of current version June 28, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2922693

Improving NoSQL Storage Schema Based on Z-Curve for Spatial Vector Data

DONGFANG ZHANG¹, YONG WANG¹, ZHENLING LIU, AND SHIJIE DAI

School of Computer Science, China University of Geosciences, Wuhan 430074, China

Corresponding author: Yong Wang (yongwang@cug.edu.cn)

ABSTRACT NoSQL database can provide massive, high concurrency, and scalable services for storing different types of data. HBase, a type of NoSQL database, in which columns are grouped into column families, is very suitable for storing semi-structured or unstructured spatial vector data. However, since there are few rules and constraints to be followed for the NoSQL database, the design of storage schema for spatial data based on NoSQL is difficult. In this paper, based on our early work, an improved Z-curve storage schema is proposed for spatial vector data. According to our new schema, row key of a geometric object is the Z-curve code of the spatial grids intersected with the geometric object. Moreover, geometric objects with the same row key are stored in a column family. Our proposed method has two features. First, geometric objects adjacent in the location are adjacent in physical storage. Second, redundant exists in storage for improving query accuracy. In our experiments, we compare the improved Z-curve storage schema with a Quadtree storage schema, an R-tree storage schema, and the previous Z-curve storage schema. Query response time, memory usage, and the query accuracy of spatial query on point and range are used to verify the validity of our proposed method. The experimental results show that the two storage schemas based on Z-curve achieve higher query efficiency than the two storage schemas based on tree—the Quadtree storage schema and the R-tree storage schema. More importantly, the query results of the improved Z-curve schema are completely correct, while the query results of the previous Z-curve schema are not.

INDEX TERMS Cloud computing, geographic information system (GIS), HBase, NoSQL, spatial index.

I. INTRODUCTION

Geographic information system (GIS) is mainly intended for spatial data collection, storage, and on-demand processing with the support of computer software and hardware [1]. In GIS, many services need be provided for spatial vector data. The rapid development of information technology, earth observation technology, and sensor technology, has resulted in a rapid increase in the accumulated volume of spatial vector data in various fields. Hence, GIS services are often provided via the Internet to speed up processing [2]. GIS service via the Internet requires a large number of concurrent read and write operations together with scalability. Consequently, efficient storage of massive spatial vector data based on the Internet has become the key to the development of GIS [3].

Traditional relational database is fit for storing structured data rather than spatial vector data, a type of unstructured data [4]. Fortunately, the new-generation IT model - cloud computing, which plays an important role in big data

processing, is designed to enable dynamic resource pooling, virtualization, and high availability. Not only SQL (NoSQL), which is a new type of database based on cloud computing, has the performance advantages on big data access, scalability, and high concurrency. The excellent scalability of NoSQL provides the ability to handle large amounts of data. Thus, performance problems that result from increased amount of data can be solved [5]. In short, the strong scalability and low-cost features of a NoSQL distributed database management system in the cloud-computing environment provide a new way of thinking for spatial vector data operations. Hence, NoSQL is a better choice for spatial vector data than relational database.

With the aim of providing improved support for search engines and other applications, Google has designed massive extensible infrastructure that addresses each issue individually on each level of the application stack. Their goal is to create a scalable infrastructure to process massive amounts of data including spatial vector data in parallel [6]. To this end, Google has established a complete set of cloud computing mechanisms, including a distributed file system GFS,

The associate editor coordinating the review of this manuscript and approving it for publication was Lu An.

distributed coordination system Chubby, column family oriented data storage Bigtable, and the parallel algorithm execution environment MapReduce. Basic principle and key details can be found in [7]. The above work of Google is an example of spatial vector data operations based on NoSQL in the cloud-computing environment.

Undoubtedly, storage schema is the foundation of spatial vector data operations based on NoSQL in the cloud-computing environment. Nevertheless, in design of storage schema, few rules and constraints can be followed. The literature on storage schema on NoSQL, especially storage schema for spatial polygon data on NoSQL, is quite limited [8]. In [9], two schemas for storing spatial vector data on HBase - a widely used NoSQL database - were proposed by us. One schema employs row key based on the Z curve and is named Z curve schema, whereas the other has row key based on geometric object identifiers. Experiments in [9] showed that the performance of the former schema is better than the performance of the latter schema. So far, the performance of the two schemas is not compared with the performance of any storage schemas based on classical indexing methods. In this paper, based on our previous work in [9], we propose an improved Z curve schema for storing spatial vector data on HBase. The current Z curve schema, differs from the last version in terms of the row key and column family. According to the current Z curve schema, the row key of a geometric object is the list of Z curve code of the spatial grids intersected with the geometric object. Moreover, geometric objects with the same row key are in the different column of the same column family. With the proposed schema, geometric objects adjacent in location are stored adjacently in physical position. In practice, such a feature can improve query efficiency because the geometric objects of one query result usually are adjacent in location. Meanwhile, under the control of the schema, a geometric object could possibly be stored in multiple grids. That is, redundancy may occur in storage. However, such redundancy based on high scalability of HBase may help to accurately locate geometric objects in less query time. Therefore, accuracy and speed of spatial query for special geometric objects, such as those with a large area or narrow shape, can be improved.

In our experiments, not only the current and previous Z curve schemas but also two other classical storage schemas for spatial vector data, one based on Quadtree, the other based on R-tree are executed. We assess their performance in terms of spatial point query and spatial range query. Experimental results show that both the previous and current Z curve schemas outperform R-tree schema and Quadtree schema. However, the previous Z curve schema does not work well when processing large or narrow geometric objects. In these cases, the query results do not completely meet the query condition.

The remainder of the paper is organized as follows. Section II provides background information and related work. Section III presents our proposed schema and its three peers. Section IV describes our experiments based on a

real-world vector dataset, followed by an analysis of the results. Section V concludes the paper.

II. BACKGROUND AND RELATED WORK

A. SPATIAL INDEX TECHNOLOGY AND NOSQL

In recent years, the ever-increasing scale of spatial data has attracted increasing attention and prompts the development of spatial index technologies that enable spatial data to be processed quickly. Spatial index relates not only to information about the location, shape, and attributes of spatial objects, but also the relationship between objects. For an object, the data structure of the spatial index may include its identification information, its minimum bounding rectangle (MBR), and its pointer to the spatial object entity [10]. Thus, spatial index acts as a bridge between algorithm performing the operation and spatial object entity. In fact, when certain spatial objects need to be retrieved, spatial index can help to exclude a large number of spatial objects, thereby reducing the scope of related spatial objects. Thus, spatial objects are quickly located and the query efficiency is consequently improved.

NoSQL, a new database model that differs from relational database, disposes of the limitation of ACID theory and belongs to non-relational database. NoSQL is seizing the dominant position occupied by relational database for a long time [11]. A NoSQL provides unstructured storage [12]. That is, data storage does not need the table structure to be designed in advance, because the join operation and horizontal segmentation between tables do not occur. In short, NoSQL are useful when processing an enormous quantity of data (especially big data) when a relational model is not necessary [13]. The growth in the scalability of spatial data has made the NoSQL database a suitable choice.

HBase, an open source column family NoSQL-type database, provides high reliability, high performance, and column-based storage. Furthermore, it is scalable and offers real-time read and write database system services derived from its underlying storage based on HDFS. All columns in a HBase column family are stored together. Column families of a table are stored separately. HBase is suitable for storing semi-structured and unstructured data that have no fixed pattern and are loosely coupled. In addition, HBase is particularly beneficial for physical decentralized storage in the case for spatial vector data collected in various locations.

Generally speaking, the cost of a spatial index consists of two parts, the time required for data positioning and the time required for returning the data. The time for positioning is the time required to execute the query, whereas the time for data return is the time required to transmit the query result from the system storage layer to the processing layer. The time for data return is proportional to the amount of data to be returned. The computational cost of the spatial index is mainly the result of data positioning, which involves indexing the spatial data. The considerable cost associated with querying spatial data stored in a relational database is due to the complexity of these data. This explains why it is necessary to use an index in a non-relational database [14].

Spatial index is important to enable a spatial database to retrieve and display data efficiently; thus, the performance of the spatial index affects the overall performance of the spatial database directly [15]. From the perspective of the development and evolution of spatial index technology, four structures have been used to build the spatial index: a binary tree, B-tree, hash, and space-filling curve.

B. RELATED WORK

NoSQL databases are becoming popular for storing massive amounts of spatial data. For example, the KR^+ -spatial index was designed using the NoSQL database Cassandra to improve the efficiency of spatial data query [16]. Nishimura *et al.* used HBase to store spatial data and designed an index layer to address multi-dimensional query problems [17]. A spatial index structure based on the NoSQL database Accumulo was designed to conduct performance query [18]. A spatial index of raster data stored in HDFS was established on HBase [19]. A hybrid index structure combining Quadtree and a grid based on HBase was designed [20]. A map tile data storage strategy was designed on the NoSQL database MongoDB to improve the storage of massive amounts of spatial data and to allow concurrent access efficiency [21]. A data storage structure was designed on the NoSQL database CouchDB to enable spatial vector data to be stored in a hand-held application [22]. Hsu *et al.* [23] made use of the R^+ -tree index technique to optimize the design of row keys in HBase and Cassandra to improve the data access efficiency. Li *et al.* [24] used geohash coding technology to establish a spatial index on the NoSQL database OrientDB to improve the write data rate. Liu *et al.* [25] stored spatial big data in the NoSQL database MongoDB and designed a big data cloud storage solution based on MongoDB cluster architecture. Cho and Choi [26] used the column family database HBase to store information relating to GPS logistics of vehicles to improve query efficiency. Most of the above studies proposed storage schemas for spatial point data. Meanwhile, few studies concerned with storage schemas for polygon. Nevertheless, for polygon, we proposed two storage schemas implemented in Hbase based on Z curve in [9].

III. HBASE STORAGE SCHEMAS

As mentioned above, besides our early work, few storage schemas for polygon data in NoSQL database can be found. Thus, we also realize other two HBase storage schemas for our experiments. That is, we employ four HBase storage schemas for spatial vector data in experiments. Apart from our proposed Z curve schema, R-tree schema, Quadtree schema, and the previous version of the Z curve schema are given here. Realization of R-tree and Quadtree schemas requires two tables, one of which is for recording the established tree structure of the index and the other for recording spatial data. When a spatial query is performed, the approximate spatial index range should first be queried through the index table. Then, the spatial query can be executed based on the associative relationship between the row key

of the spatial data table and the index table. In the previous version of Z curve schema, Z curve is used to divide the entire index space into several grids, each of which is used as an index - row key. Both the barycentric coordinates of the MBR of the geometric objects comprising the grid and the geometric objects themselves are stored in the column according to row key. The improved Z curve schema proposed in this paper uses the list of Z curve code of the spatial grids intersected with a geometric object as the row key of the geometric object. Further, all geometric objects with the same row keys are sequentially stored in the corresponding column family. Additional details of the four storage schemas - row key design, column family design, and access method - are provided in the following subsections, respectively.

A. STORAGE SCHEMA BASED ON QUADTREE (QUADTREE SCHEMA)

1) ROW KEY DESIGN

In index structure table, information for structure of the established quadtree can be stored and indexed. Firstly, a row with the row key value "1" is used to store the index structure of quadtree. Then, each non-empty leaf node of the current quadtree indicates an index range of the MBR of a geometric object. Thus, in spatial data table, each non-empty leaf node is stored in a row. The serial number of non-empty leaf node is used as row key. In addition, the information of geometric object within the index range can be stored in different column of the same column family in the corresponding row.



FIGURE 1. MBR N of polygon M.

2) COLUMN FAMILY DESIGN

In index structure table, three column families are needed. The first column family, named Tree, is used to store information about quadtree structure - the height of quadtree, number of tree nodes, and number of geometric objects. The second column family, named Nodes, is used to store tree node data including the serial number of each non-leaf node and the four quadrant location information of its child nodes, which refers to the four quadrant positions in quadtree index - northeast (NE), northwest (NW), southwest (SW), southeast (SE). The four quadrants correspond to a four-digit binary number, where "0" means that the MBR of the geometric object is not in the quadrant, whereas "1" means that it is present. Fig. 1 shows geometric object M and its MBR N. The third column family, named Range, is used to store quadtree index range of each node.

Here, we give some examples. Firstly, a quadtree index is established based on the MBR data of the geometric object.

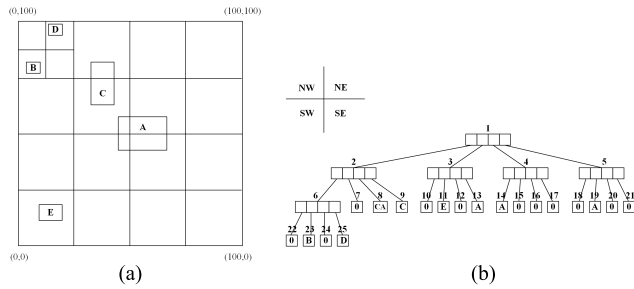


FIGURE 2. Instance of spatial data storage model based on quadtree. (a) Spatial data. (b) Quadtree index.

For example, Fig. 2 (b) can be established based on Fig. 2 (a). The column family design model of the storage schema is shown in Table 1. Fig. 2 (a) shows that both MBR A and MBR C cover multiple grids each corresponds to a leaf node. First, MBR A and MBR C need to be stored in the leaf nodes covered by them. If spatial query operation is required, the index range is obtained in the column family - Range - through the node number.

In fact, each spatial data table requires a column family to store geometric objects. The number of columns in a column family is changed according to the number of geometric objects. Based on quadtree index in Fig. 2, the design model of the column family of the spatial data shown in Table 2 can be obtained.

3) ACCESS METHOD

Based on the row key design and column family design mentioned above, spatial point query, which is used to query whether a point is included in a geometric object within the index space, and spatial range query, which is used to list all the geometric objects in a spatial range, can be done with the help of index structure table and spatial data table. When performing spatial point query, firstly, an intersection operation is performed on the pair of coordinates of point and the index range of the root node in the column family - Range - within the index structure table. If intersection exists, the index ranges of the child nodes stored in the column family - Nodes - is used for the intersection operation.

TABLE 1. Instance of column family design model of Quadtree schema for index structure table.

Row Key	Time Stamp	Column family "Tree"			Column family "Nodes"				Column family "Range"		
		Height	Node Num	Polygon Num	Node:1	Node:2	...	Node:6	Range:1	...	Range:25
1	t1	4	25	5	111112345	210116789	...	6010122232425	POLYGON((0 0, 0 100, 100 100, 100 0))	...	POLYGON((12.5 87.5, 12.5 100, 25 100, 25 87.5))

TABLE 2. Instance of column family design model of Quadtree schema for spatial data table.

Row Key	Time Stamp	Column family "Geometry"			
		Geometry:1	Geometry:2	...	Geometry:n
8	t2	POLYGON ((31.5 64, 31.5 82, 44 82, 44 64))	POLYGON((45 42.6, 45 57.8, 66 57.8, 66 42.6))
9	t1	POLYGON ((31.5 64, 31.5 82, 44 82, 44 64))			
11	t4	POLYGON((9.9 11, 9.9 18.7, 20 18.7, 20 11))			
13	t7	POLYGON((45 42.6, 45 57.8, 66 57.8, 66 42.6))			
14	t11	POLYGON((45 42.6, 45 57.8, 66 57.8, 66 42.6))			
19	t3	POLYGON((45 42.6, 45 57.8, 66 57.8, 66 42.6))			
23	t2	POLYGON((4.3 77, 4.3 82, 10.5 82, 10.5 77))			
25	t6	POLYGON((13 92, 13 99, 20 99, 20 92))			

Algorithm 1 Pseudocode of Spatial Point Query Based on Quadtree Schema

```

Input: Spatial point M
Output: The geometric object intersected with M
1 Get the input, M
2 Read the stored quadtree index structure for the row whose row key is "1" in the index structure table
3 Use the list collection to reconstruct quadtree to obtain the index range of each node of quadtree
4 Execute intersections between spatial point M and its root node
5 if intersection exists then
6     Query quadtree until a leaf node that intersects with spatial point M is found
7     Get the node number N of the above leaf node as row key to obtain the collection of geometric objects, Result, in the spatial data table
8     for traverse all the geometric objects in Result do
9         Execute intersections between the pair of coordinates of spatial point M and the current geometric object
10        if intersection exists then
11            Give the output
12        end
13    end
14 end
    
```

This occurs recursively until the index range of the leaf node that intersects with the spatial point is found. After obtaining the index range, the found index range of the leaf node is used as row key to query the spatial data table for all the geometric objects stored in the row key. Then, an intersection operation is performed on the obtained geometric objects and the pair of coordinates of spatial point to obtain the geometric object that intersects with the spatial point. The pseudocode of spatial point query is shown in Algorithm 1.

Similar to spatial point query, when performing spatial range query, firstly, an intersection operation is performed on the spatial range to be queried and the index range of all

quadtree root nodes to find quadtree in the spatial range to be queried. Then, recursively find the index range of the leaf node that intersects with the spatial range to be queried in quadtree. Subsequently, the index range is used as row key. The geometry objects stored in the row with the row key are queried. Finally, an intersection operation is performed on the obtained geometric objects and the spatial range to obtain a collection of geometric objects that intersect with the spatial range. The pseudocode of spatial range query is shown in Algorithm 2.

Algorithm 2 Pseudocode of Spatial Range Query Based on Quadtree Schema

Input: Spatial range R
Output: Geometric objects collection C

- 1 Get the input, R
- 2 Read the stored quadtree index structure for the row whose row key is “1” in the index structure table
- 3 Use the list collection to reconstruct quadtree to get the index range of each node of quadtree
- 4 Execute intersections between spatial range R and its root node
- 5 **if** intersection exists **then**
- 6 Query quadtree until all the leaf nodes that intersect with spatial range R are found
- 7 Get the node number collection Find of the above leaf nodes as row key, respectively, to obtain the collection of geometric objects, Result, in the spatial data table
- 8 Remove duplicate geometric objects in Result
- 9 **for** traverse all the geometric objects in Result **do**
- 10 Execute intersections between the current geometry object and query range R
- 11 **if** intersection exists **then**
- 12 Add the current geometric object to the collection C
- 13 **end**
- 14 **end**
- 15 **end**

B. STORAGE SCHEMA BASED ON R-TREE (R-TREE SCHEMA)

Compared with Quadtree schema, R-tree schema is different in terms of their row key design and column family design whereas their access methods are basically the same. Therefore, this section just introduces the row key design and column family design of R-tree schema.

1) ROW KEY DESIGN

In index structure table, information for structure of the established R-tree can be stored and indexed. Similar with quadtree index structure, firstly, a row with the row key value “2” is used to store the index structure of R-tree. Then, in spatial data table, the serial number of each non-empty leaf node is

used as row key. In addition, the information of geometric objects within the index range can be stored in different column of the same column family in the corresponding row.

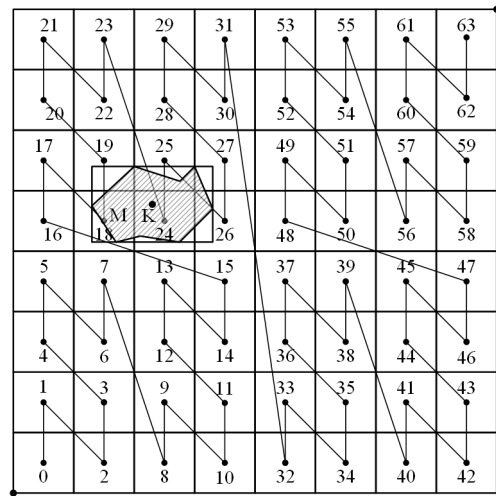


FIGURE 3. Third-order Z curve.

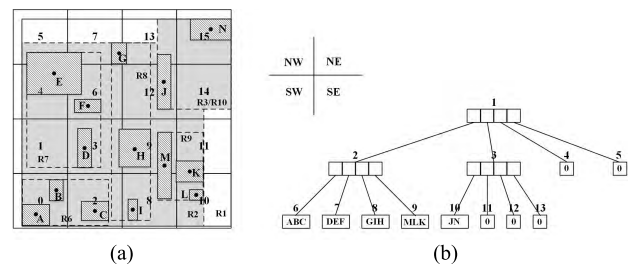


FIGURE 4. Instance of spatial data storage model based on R-tree. (a) Spatial data. (b) Structure of R-tree.

2) COLUMN FAMILY DESIGN

Here, similar to the storage structure of quadtree index, in index structure table, three column families are needed for storing information about R-tree structure, R-tree non-leaf node information, and nodes range, respectively. Nevertheless, difference exists. In detail, the geometric objects in the original index space needs to be mapped in the Z curve according to the barycentric coordinates of its MBR for obtaining its one-dimensional code. As shown in Fig. 1, the barycentric of the MBR of geometric object M is point K, which corresponds to the code 24 in the third-order Z curve shown in Fig 3. Thus, the one-dimensional code of M is 24. After the one-dimensional code of all the geometric objects is obtained in the original index space range, the code is sorted in ascending order. In theory, adjacent geometric objects in coding positions are adjacent in actual geographical positions. Therefore, all the MBRs with adjacent coding position are combined to form a large rectangular range as the leaf node of R-tree. Then, quadtree is built from bottom layer to top layer. In Fig 4 (a), the second-order Z-curve codes of geometric objects A-N are 0, 0, 2, 3, 4, 6, 7, 9, 8, 12, 11, 10, 9, 15, or 0, 0, 2, 3, 4, 6, 7, 8, 9, 9, 10, 11, 12, 15 if

TABLE 3. Instance of column family design model of R-tree schema for index structure table.

Row Key	Time Stamp	Column family "Tree"			Column family "Nodes"			Column family "Range"		
		Height	Node Num	Polygon Num	Node:1	Node:2	Node:3	Range:1	...	Range:13
1	t1	3	13	14	111002345	211116789	3100010111213	POLYGON((4.2 1, 4.2 96, 100 96, 100 1))	...	0

TABLE 4. Instance of column family design model of R-tree schema for spatial data table.

Row Key	Time Stamp	Column family "Geometry"		
		Geometry:1	Geometry:2	Geometry:3
6	t2	POLYGON((4.2 14, 4.2 11.3, 16.8 11.3, 16.8 1))	POLYGON((16.9 13, 16.9 22, 23.7 22, 23.7 13))	POLYGON((31.3 3.7, 31.3 12.6, 42 12.6, 42 3.7))
7	t1	POLYGON((30 28.2, 30 46, 36.3 46, 36.3 28.2))	POLYGON((5.1 63, 5.1 80, 32 80, 32 63))	POLYGON((30.9 53.7, 30.9 60, 39 60, 39 53.7))
8	t4	POLYGON((44 75, 44 85, 52 85, 52 75))	POLYGON((53 3.7, 53 14, 56.1 14, 56.1 3.7))	POLYGON((49.3 28.2, 49.3 45, 62.5 45, 62.5 28.2))
9	t7	POLYGON((65.4 13, 65.4 44.2, 71.2 44.2, 71.2 13))	POLYGON((80 12.5, 80 19, 87.5 19, 87.5 12.5))	POLYGON((75 21, 75 31, 88 31, 88 21))
10	t11	POLYGON((65.4 55, 65.4 80, 71 80, 71 55))	POLYGON((80 86, 80 96, 100 96, 100 86))	

TABLE 5. Instance of column family design based on Z curve schema.

Row Key	Time Stamp	Column family "Geometry"		
		Geometry:1	Geometry:2	Geometry:n
32	t3	POLYGON ((292.61 487.12, 294.93 488.17, 298.03 487.11, 294.85 487.97))	POLYGON((294.42 489.05, 298.74 487.65, 294.76 488.26, 299.26 484.41, 295.71 488.03))	...
53	t5	POLYGON ((297.44 487.05, 298.76 489.61, 299.03 480.66, 294.32 488.31, 293.79 484.69))		...
71	t8	POLYGON ((298.66 489.65, 294.12 480.66, 298.26 488.13))	POLYGON ((295.19 488.87, 298.19 489.73, 296.21 489.78, 294.25 488.89))	...
...

sorted in ascending order. The column family design model of the storage schema is shown in Table 3. In each spatial data table, only a column family is needed for storing geometric object information. According to Fig. 4, quadtree index can generate the column family of the spatial data table presented in Table 4.

C. STORAGE SCHEMA BASED ON Z CURVE (Z CURVE SCHEMA)

Because foreign key is not used in HBase, it is necessary to visit the HBase data table twice when using a storage schema with tree, a structure may affects the efficiency of a spatial query. The deeper the index structure is, the higher the time complexity becomes. Hence, design of the index tree is a key step. Against the above background, the storage schema based on Z curve is proposed by us.

1) ROW KEY DESIGN

First, the index space is divided into 4^m grids by the m-th order Z space-filling curve. Meanwhile, all grids are numbered according to the order of the curves. Then, the MBR of each geometric object needs to be obtained to find the covered grid according to the barycentric coordinates of the MBR. After that, the grid number is used as row key. For the geometric object M shown in Fig. 3, its barycentric of the MBR is point K and the Z curve code of the grid where point K is located is 24. Therefore, the row key of the geometric object stored in HBase is 24.

2) COLUMN FAMILY DESIGN

Adjacent geographical positions lead to adjacent codes for the Z space filling curves. Therefore, the use of the Z curve to encode the row keys can partially ensure that spatial vector data with adjacent geographical location have adjacent locations in physical storage in Hbase. However, if the spatial

vector data, geometric objects, around a geographical location are very dense, there are usually many geometric objects in the range of the Z curve code although the Z curve order is sufficiently high. Therefore, the Z curve code of multiple geometric objects may be the same. To solve this problem, when the columns of the Z curve schema are designed, it is necessary to design multiple columns in a column family. Instead of being predefined, columns are stored sequentially in different columns of the same column family when the same encoded geometric object is acquired. The column names are encoded incrementally. The column family design model for the Z curve schema appears in Table 5.

3) ACCESS METHOD

In Z curve schema, row key is the Z curve code of the grid where the barycentric of the MBR of the stored geometric object is located. Therefore, when performing spatial point query, the grid code corresponding to the spatial point first needs to be obtained. Then, all the geometric objects stored in the spatial grid are obtained. Finally, intersections between the obtained geometric objects and the spatial point are done to obtain the geometric objects that intersect accurately with the spatial point. The pseudocode of this access method is shown in Algorithm 3. Spatial range query is similar to spatial point query. The pseudocode of this access method is shown in Algorithm 4.

D. STORAGE SCHEMA BASED ON IMPROVED Z CURVE (IMPROVED Z CURVE SCHEMA)

In Z curve schema, row key is the grid code where the barycentric of the MBR of the geometric object is located. Further, each geometric object is stored only once. However, in this storage schema, the geographical location information of each geometric object is transformed into the barycentric coordinates of its MBR for storage. To some extent, part of

Algorithm 3 Pseudocode of Spatial Point Query for Z Curve Schema

Input: Index space range IR and query spatial point M
Output: A geometric object or None

- 1 Get the input IR and M
- 2 Divide IR into orders
- 3 Obtain the spatial grid code of the spatial point under query as row key
- 4 Get the Result collection that stores the geometric object information according to the row key
- 5 **for** traverse all the geometric objects in Result **do**
- 6 Execute intersections between the pair of coordinates of spatial point M and the current geometric object
- 7 **if** intersection exists **then**
- 8 Give the output
- 9 **end**
- 10 **end**

Algorithm 4 Pseudocode of Spatial Range Query for Z Curve Schema

Input: Index space range IR and query range QR
Output: Geometric objects collection C

- 1 Get the input IR and QR
- 2 Divide IR into orders
- 3 Get the spatial grid code collection, Code, covered by the query range
- 4 Cluster all the codes to get the scan coding range collection, Find
- 5 **for** traverse each scan in Find **do**
- 6 Query the data table to get the collection Result that stores the geometric object information
- 7 **for** Traverse all the row key in Result **do**
- 8 **if** Row key exists in Code **then**
- 9 Add geometric object to the collection Sum_geoms
- 10 **end**
- 11 **end**
- 12 **end**
- 13 Remove duplicate geometric objects in Sum_geoms
- 14 **for** Traverse all the geometric objects in Sum_geoms **do**
- 15 Execute intersections between all geometric objects and query ranges
- 16 **if** intersection exists **then**
- 17 Add geometric object to the collection C
- 18 **end**
- 19 **end**

the information of the original geometric object is lost in this course. For example, Fig. 5 reveals several problems that occur when storing special geometric objects such as large or narrow-shaped objects.

According to the storage schema shown in the previous section, the Z curve code of Polygon A, the larger polygon,

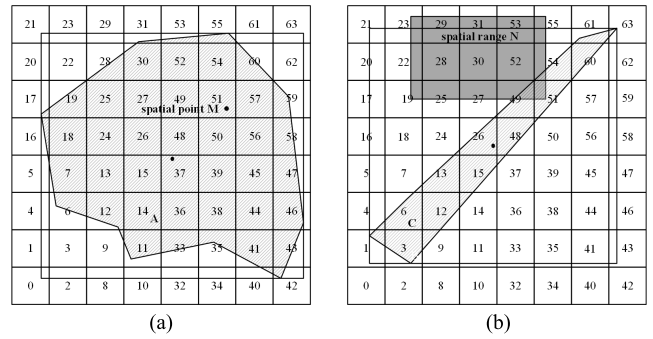


FIGURE 5. Storage instance of (a) large area and (b) narrow shape geometric objects based on Z curve schema.

in Fig. 5 (a) is 37, whereas the grid code of the spatial point M is 51. Actually, it can be seen that spatial point M is intersected with polygon A. However, they are assigned different Z curve codes. Since the geometric information of polygon A is only stored in the column numbered 37, M cannot intersect with polygon A in response to a query. That is, query result cannot reflect the fact. Similarly, in Fig. 5 (b), the Z curve code of polygon C - a narrow and long polygon - is 26. Actually, an intersection exists between spatial query range N and polygon C. Because the grid codes covered by query range N do not contain 26, in the query result of this storage schema, query range does not intersect with polygon C.

The inability of the Z curve storage schema to effectively store geometric objects with a larger area or a longer shape led us to propose an improved Z curve storage schema to avoid these errors. The improved schema mainly improves the row key design and the column family design of the Z curve schema, and its access method is similar to the traditional Z curve storage schema.

1) ROW KEY DESIGN

In our improved Z curve schema, grid codes that intersect with geometric object is used as row keys. That is, first, MBR of geometric object is used to conduct intersection operation with each grid to obtain the grids intersect with it. Then, the actual geometric object is used to conduct the intersection operation with the grids obtained in the previous step. After obtaining grids that intersect with the geometric object, grid codes are used as row keys to store geometric object. In the improved Z curve schema, there is a possibility that a geometric object is stored in multiple grids provided that these grids intersect with it.

This schema effectively avoids the problem described in the previous section when storing objects that are either geometrically large or have a narrow shape. As shown in Fig. 6 (a), first, the MBR of polygon A is used to conduct an intersection operation with each grid to find the grids that intersect with it. The Z curves of these grids are encoded from 0 to 63. Then, polygon A conducts an intersection operation with the index region represented by all 64 spatial grids. Removing the grids that do not intersect with polygon A - 0, 1, 2, 3, 4, 8, 10, 20, 21, 22, 23, 29, 32, 34, 62, and 63

TABLE 6. Instance of column family design based on the improved Z curve schema.

Row Key	TimeStamp	Column family "Geometry" Geometry:1	Geometry:2	...	Geometry:n
39	t7	POLYGON((292.61 487.12, 294.93 488.17, 298.03 487.11, 294.85 487.97))	POLYGON((294.42 489.05, 298.74 487.65, 294.76 488.26, 299.26 484.41, 295.71 488.03))
45	t5	POLYGON((294.42 489.05, 298.74 487.65, 294.76 488.26, 299.26 484.41, 295.71 488.03))	
...

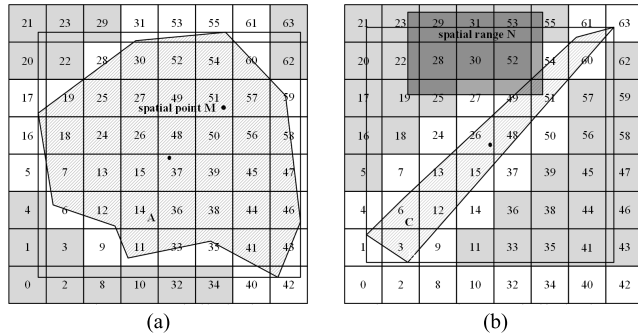


FIGURE 6. Storage instance of geometric objects with a (a) large area and (b) narrow shape based on the improved Z curve schema.

(as shown in the gray rectangle in Fig. 6 (a)), the remaining 48 spatial grids are used as the grid range that intersects with polygon A. At the same time, the Z curve code of each of these grids is used as a row key to store polygon A in each grid, respectively. When performing spatial point query as shown in Fig. 6 (a), the grid code covered by the spatial point M is 51, while the storage unit with row key 51 contains polygon A. The intersection operation can be performed to determine that point M is intersected with polygon A. Similarly, for polygon C, as shown in Fig. 6 (b), the Z curve code of each of the 22 grids with which it intersects is used as a row key to store polygon C in each grid, respectively. When performing spatial range query, the grid codes covered by query range N are 19, 22, 23, 25, 28, 29, 27, 30, 31, 49, 52, 53, 51, 54, and 55. The codes of the common grid of spatial query range N and polygon C are 49, 51, and 54. Querying the geometric objects in the corresponding storage unit can find both spatial query range N and polygon C to intersect with each other. This storage schema can accurately determine the spatial location of geometric objects. Therefore, the accuracy of spatial queries relating to special geometric objects such as those with a large area or narrow shape is improved.

2) COLUMN FAMILY DESIGN

In practice, one grid could be covered by more than one geometric object. Therefore, in our improved Z curve schema, row keys are the Z curve codes of the spatial grids covered by the geometric object. In this schema, a geometric object may be stored in multiple grids provided that these grids all intersect with it. That is, there is a possibility of redundancy in storage. Similar to Z curve schema, the number of columns in the column family named Geometry is dynamic in our proposed improved Z curve schema. Moreover, the names of these columns are sorted in ascending order. The model of the column family design in this storage schema is presented in Table 6.

TABLE 7. Configuration of experimental environment.

Items	Parameters
CPU	Inter® Xenon® CPU E5-2620 @2.00 GHz (24 CPUs)
RAM	32768 M
Disk	DELL H310 SCSI Disk Device 500 GB * 4
Network adapter	Broadcom NetXtreme Gigabit Ethernet * 4
Switch	1000M switches
Number of nodes	12
Node configuration	2 CPUs, 2 GB memory, 80 GB Disk, 100M network
Node operating system	Ubuntu 12.04 Server
Hadoop version	1.2
HBase version	0.94

IV. EXPERIMENTAL STUDY

A. EXPERIMENTAL ENVIRONMENT

In our experiments, Infrastructure as a Service (IaaS) is provided by two DELL R720 servers. Details of the hardware and software configuration are provided in Table 7. Virtualization is employed to provide 12 nodes for our IaaS, ten of which are used as DataNodes to provide the HDFS service.

B. EXPERIMENTAL DATA

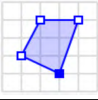

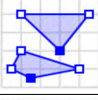

Land changing survey in China produces a tremendous number of polygon parcels every day. Data of land changing survey, which covers an area of approximately 166 square kilometers, is chosen as the spatial vector data in our experiment. About four million polygons are contained in our chosen data, the format of polygon is specified as Well-known text (WKT), a text markup language defined by the Open Geospatial Consortium(OGC) [27]. Fig. 7 shows a part of the data. Table 8 shows the details of store format of polygon and multipolygon.

C. EXPERIMENTAL METHOD

In our experiment, for point query and region query, on different scales, the query response time, memory usage, and query accuracy is measured under the control of different storage schemas. This approach for comparison is expected to verify whether the improved Z curve schema could provide an effective storing and indexing service for massive amounts of spatial vector data.

Twenty spatial points are selected for our spatial point query experiment. The query is repeated 30 times to calculate the average for each spatial point. The 20 selected spatial points are listed in Table 9.

TABLE 8. Storage format of a polygon.

Type	Examples
Polygon	 POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
	 POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))
MultiPolygon	 MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
	 MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))

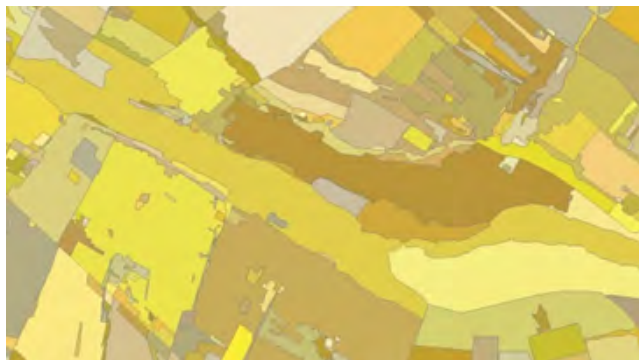


FIGURE 7. Part of the polygons in the district.

TABLE 9. Points selected for the experiments.

Number of points	X coordinate	Y coordinate
1	29490531.369219	4908616.887721
2	29483445.371855	4908505.121192
3	29427899.403817	4816322.987430
4	29461287.353615	4908459.247608
5	29485806.853477	4908724.060198
6	29496579.058309	4908434.852895
7	29496834.391533	4908542.554553
8	29482015.187783	4908357.338808
9	29492949.380309	4908703.278788
10	29471704.883041	4908723.312527
11	29506773.231479	4908664.170541
12	29494549.851274	4908173.869755
13	29497434.069225	4908426.559765
14	29504005.304232	4909026.571307
15	29467789.052651	4908336.512673
16	29467119.375066	4908387.312877
17	29508380.026848	4908350.599367
18	29484423.208778	4908123.898411
19	29472874.872673	4908176.214107
20	29490497.831655	4908202.402972

For spatial range query, we randomly select 20 spatial points as the lower left corner of a square query range. The basic side length of each square s is 200 meters. In this experiment, side length of query is s , $2 \cdot s$, $4 \cdot s$, or $8 \cdot s$. Query is repeated 30 times to calculate the average. The selected spatial points are listed in Table 10. Using spatial

TABLE 10. Chosen points of the square query range.

Number of points	X coordinate	Y coordinate
1	29465757.845621	4887395.665123
2	29563105.751125	4814204.745065
3	30024804.012578	4817266.345274
4	29565132.451122	4744463.984502
5	28595604.861274	4306272.054819
6	28897705.044578	4157456.741236
7	29439802.456387	4564724.784131
8	29284786.546419	4976595.886543
9	29231604.565298	5003674.546577
10	29586006.454103	4157456.963874
11	29000586.315165	4804395.645341
12	29725486.945127	4870107.554893
13	29146403.025248	4809052.451582
14	30084004.754102	4739772.941056
15	29185202.487125	4952107.654712
16	29460804.874456	4537172.124788
17	28928357.451203	4748095.774516
18	29569232.897451	4324963.784567
19	28538804.741125	4428963.852366
20	28518805.784501	4234495.996547

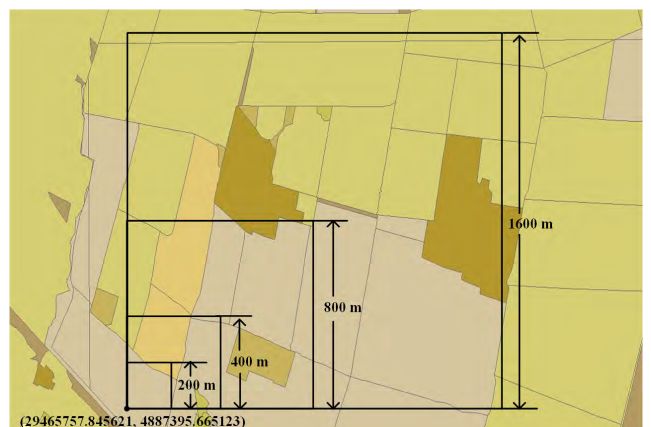


FIGURE 8. Example of different query rectangles.

point (29465757.845621, 4887395.665123) as an example, Fig. 8 shows the different square query ranges.

As mentioned above, the three evaluation indicators, the query response time, memory usage, and query accuracy

are used to test the efficiency of spatial point query and spatial range query under different storage schemas. As Formula 1, query response time T consists of the time for accessing the HBase data table T_t , and the time for performing the intersection operation T_i on the spatial point or the spatial range with the geometric object.

$$T = T_t + T_i. \tag{1}$$

Memory usage in spatial point query and spatial range query under the control of the four storage schemas is calculated, respectively. Since query result of R-tree schema is always correct, query accuracy of the Z curve schema and the improved Z curve schema is calculated based on the query result of R-tree schema for comparison. As Formula 2, query accuracy S is represented by the ratio of N_z to N_r , where N_z and N_r are the number of geometric objects obtained by the Z curve schema and R-tree schema, respectively.

$$S = N_z/N_r. \tag{2}$$

TABLE 11. Response time of spatial point query.

Storage schema	T_t	T_i	T
Quadtree schema	4.22	0.61	4.83
R-tree schema	4.64	0.88	5.52
Z curve schema	3.92	0.47	4.39
Improved Z curve schema	3.99	0.57	4.56

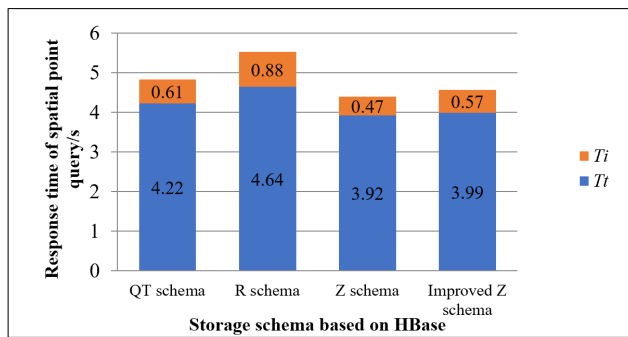


FIGURE 9. Response time of spatial point query.

D. ANALYSIS ON EXPERIMENTAL RESULTS

1) ANALYSIS FOR SPATIAL POINT QUERY

a: QUERY RESPONSE TIME T

Qesponse time for spatial point query is listed in Table 11. Based on Table 11, Fig. 9 is given. As shown in Fig. 9, response time of spatial point query is mainly affected by T_t . The value of T_t mainly changes with the number of geometric objects involved in query, whereas the value of T_i depends on the number of geometric objects involved in query. Both T_t and T_i of different schemas can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. So does response time T .

TABLE 12. Memory usage of spatial point query.

Quadtree schema/kb	R-tree schema/kb	Z curve schema/kb	Improved Z curve schema/kb
66.36	110.69	46.22	52.38

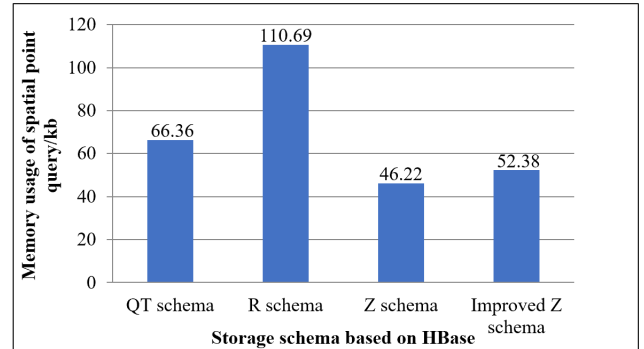


FIGURE 10. Memory usage of spatial point query.

TABLE 13. Query accuracy of spatial point query.

Storage schema	The number of geometric objects that are actually queried	The number of geometry objects that are correctly queried	accuracy
Z curve schema	3.9	4.3	90.7%
Improved Z curve schema	4.3	4.3	100%

b: MEMORY USAGE M

Memory usage for spatial point query is included in Table 12. Based on Table 12, we give Fig. 10. As shown in Fig. 10, memory usage of the four storage schemas can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. Memory usage is related to the number of geometric objects that satisfy the query conditions.

According to the memory usage results and query response time results, memory usage is proportional to query response time. Moreover, the efficiency of spatial point query of either tree-based index is lower than that of two storage schemas based on Z curve. The efficiency of the four storage schemas can be sorted in descending order as follow: R-tree schema, Quadtree schema, our improved Z curve schema, and Z curve schema.

c: QUERY ACCURACY S

Query accuracy for spatial point query is presented in Table 13 and are plotted in Fig. 11. As shown in Fig. 11, the average accuracy of spatial point query based on Z curve schema is 90.7%. That is, Z curve schema may lead to query error at a rate. Meanwhile, the query results of the proposed improved Z curve schema are completely correct. In short, although spatial point query by Z curve schema has the highest efficiency among the four storage schemas, this schema produces query error under certain conditions. Therefore, our improved Z curve schema delivers superior performance.

TABLE 14. Query response time of spatial range query.

Side length of spatial range/meter	Quadtree schema/s			R-tree schema/s			Z curve schema/s			Improved Z curve schema/s		
	T_t	T_i	T	T_t	T_i	T	T_t	T_i	T	T_t	T_i	T
200	6.28	1.25	7.53	11.07	1.45	12.52	5.96	1.03	6.99	6.01	1.11	7.12
400	7.35	2.17	9.52	12.33	2.31	14.64	6.24	1.28	7.52	6.36	1.44	7.8
800	10.84	2.92	13.76	18.25	3.17	21.42	8.34	1.95	10.29	8.43	2.58	11.01
1600	18.01	4.38	22.39	21.32	5.54	26.86	15.55	3.05	18.6	16.27	3.58	19.85

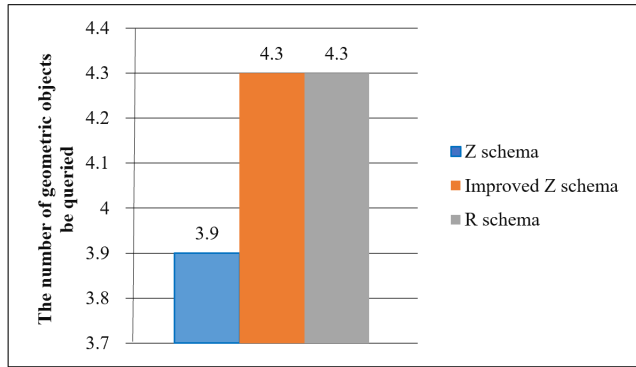


FIGURE 11. Query accuracy of spatial point query.

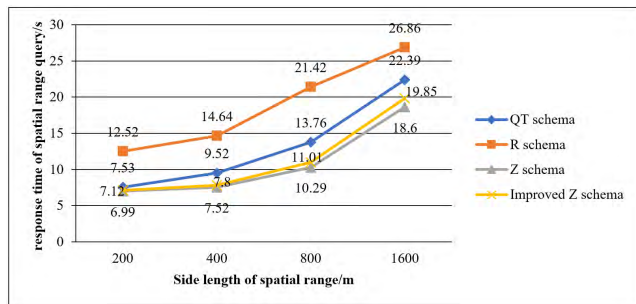


FIGURE 12. Response time of spatial range query (line chart).

2) ANALYSIS FOR SPATIAL RANGE QUERY

a: QUERY RESPONSE TIME T

Response time for spatial range query is listed in Table 14 and are additionally compared visually in Fig. 12 and Fig. 13. As shown in Fig. 12, response time of the four storage schemas becomes longer when the spatial range to be queried increases. For all the spatial ranges, query response time of different schemas can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. For each query range, although response times of the improved Z curve schema is larger than Z curve schema, the values are very close. As shown in Fig. 13, for all the spatial ranges, both T_t and T_i of different schemas can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema.

b: MEMORY USAGE M

Memory usage for spatial range query is listed in Table 15 and drawn in Fig. 14. As shown in Fig. 14, memory usage of the four storage schemas increases when the spatial range

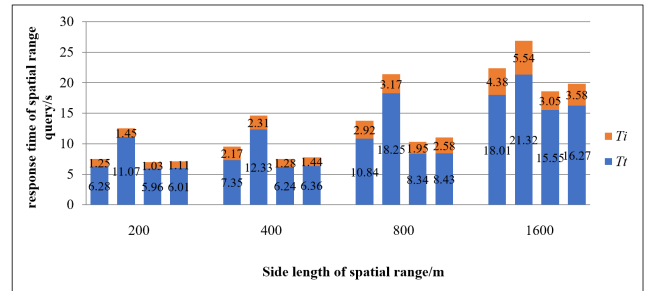


FIGURE 13. Response time of spatial range query (bar chart).

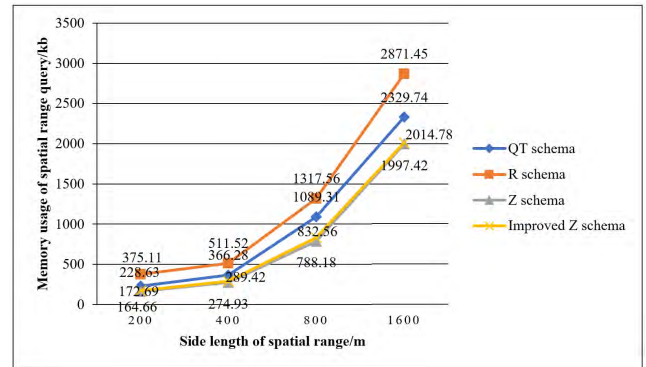


FIGURE 14. Memory usage of spatial range query (line chart).

to be queried increases. For all the query range, memory usage can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. For each query range, although memory usage of the improved Z curve schema is larger than Z curve schema, the values are very close. Since the four schemas have difference in thinking, for the same query task, difference number of geometric objects need be involved in query. In detail, number of involved geometric objects can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. That is, the efficiency of spatial range query can be sorted in descending order as follow: Z curve schema, improved Z curve schema, Quadtree schema, R-tree schema.

c: QUERY ACCURACY S

Query accuracy for spatial range query is presented in Table 16 and plotted in Fig. 15. Query results in Table 16 indicate that Z curve schema may produce query errors at a rate. Moreover, there is no significant linear relationship between accuracy and query range. The same as R-tree

TABLE 15. Memory usage of spatial range query.

Side length of spatial range/meter	Quadtree schema/kb	R-tree schema/kb	Z curve schema/kb	Improved Z curve schema/kb
200	228.63	375.11	164.66	172.69
400	366.28	511.52	274.93	289.42
800	1089.31	1317.56	788.18	832.56
1600	2329.74	2871.45	1997.42	2014.78

TABLE 16. Accuracy of spatial range query.

Side length of spatial range/meter	The number of geometric objects obtained by spatial range query of R-tree schema	Z curve schema		Improved Z curve schema	
		The number of geometric objects obtained by spatial range query	accuracy	The number of geometric objects obtained by spatial range query	accuracy
200	119	114	95.8%	119	100%
400	235	217	92.3%	235	100%
800	601	572	95.1%	601	100%
1600	1557	1472	94.5%	1557	100%

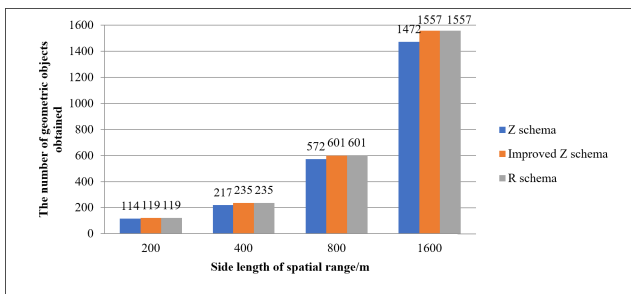


FIGURE 15. Accuracy of spatial range query.

schema, the query result of the improved Z curve schema is 100%. Similarly as results of spatial point query, for spatial range query, query efficiency and accuracy of the improved Z curve schema proposed in this paper are more accurate than those of the other three schemas.

It can be seen from the above results that, for either spatial point query or spatial range query, query response time and memory usage can be sorted in descending order as follow: R-tree schema, Quadtree schema, improved Z curve schema, and Z curve schema. In tree-based schemas, multidimensional structures is needed. Nevertheless, NoSQL databases are not fit for realizing multidimensional structures compared with relational databases. Z curve, one of the commonly used space filling curves, can reduce dimensionality of multidimensional structures and enhance query efficiency. Therefore, schemas based on Z curve are better choices than tree-based schemas. Most importantly, the query results of the improved Z curve schema are completely correct, while there exist errors in the results of Z curve schema. Based on an overall consideration, the improved Z curve schema performs best among the schemas.

V. CONCLUDING REMARKS

Compared with tree-based methods, storage schemas for spatial vector data based on Z curve are better choices in NoSQL databases. Based on our early Z curve schema, we propose the improved Z curve schema in this paper. Now that the

geometric objects of one query result usually are adjacent in location, the main idea of our method is that adjacent geometric objects in location are adjacent in physical storage. Moreover, redundant policy is used for design of column family avoiding the defects of the previous Z curve storage schema based on the excellent scalability of HBase for store large amounts of redundant data. Therefore, the limitations in our early work are removed. For example, geometric objects with a large area or narrow shape do not lead to failure in query any more. Experimental results show that, compared with tree-based methods, storage schemas based on Z curve are better choices in NoSQL databases. Further, under the control of the improved Z curve schema, correct rate of query is improved to 100%.

The proposed storage method can enlighten the design for operations other than storage of spatial vector data in NoSQL Database. For example, index structure based on the non-primary key, hierarchical storage schema, and query interface with a higher-level query engine such as Hive [28] are required to be addressed. We will study these topics later.

REFERENCES

- [1] L. Zhao, L. Chen, R. Ranjan, K.-K. R. Choo, and J. He, "Geographical information system parallelization for spatial big data processing: A review," *Cluster Comput.*, vol. 19, no. 1, pp. 139–152, Mar. 2016. doi: 10.1007/s10586-015-0512-2.
- [2] M. Fargher, "WebGIS for geography education: Towards a GeoCapabilities approach," *ISPRS Int. J. Geo-Inf.*, vol. 7, no. 3, p. 111, 2018.
- [3] Y. Wang, Z. Liu, H. Liao, and C. Li, "Improving the performance of GIS polygon overlay computation with mapreduce for spatial big data processing," *Cluster Comput.*, vol. 18, no. 2, pp. 507–516, 2015.
- [4] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proc. 6th Int. Conf. Pervasive Comput. Appl. (ICPCA)*, 2011, pp. 363–366.
- [5] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.
- [6] S. Tiwari, *Professional NoSQL*. Hoboken, NJ, USA: Wiley, 2011.
- [7] J. Jin, A. Song, H. Gong, Y. Xue, M. Du, F. Dong, and J. Luo, "Distributed storage system for electric power data based on HBase," *Big Data Mining Anal.*, vol. 1, no. 4, pp. 324–334, 2018.
- [8] N. Zhang, G. Zheng, H. Chen, J. Chen, and X. Chen, "HBasespatial: A scalable spatial data storage based on HBase," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Sep. 2014, pp. 644–651.

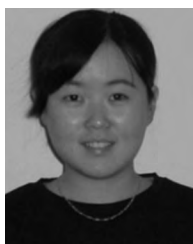
- [9] Y. Wang, C. Li, M. Li, and Z. Liu, "HBase storage schemas for massive spatial vector data," *Cluster Comput.*, vol. 20, no. 4, pp. 3657–3666, 2017.
- [10] A. Eldawy and M. F. Mokbel, "A demonstration of SpatialHadoop: An efficient mapreduce framework for spatial data," *Proc. VLDB Endowment*, vol. 6, no. 12, pp. 1230–1233, Aug. 2013.
- [11] S. Ceri, A. Bernasconi, A. Canakoglu, A. Gulino, A. Kaitoua, M. Masseroli, L. Nanni, and P. Pinoli, "Overview of GeCo: A project for exploring and integrating signals from the genome," in *Proc. Int. Conf. Data Anal. Manage. Data Intensive Domains*. Cham, Switzerland: Springer, 2017, pp. 46–57.
- [12] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL database: New era of databases for big data analytics—Classification, characteristics and comparison," 2013, *arXiv:1307.0191*. [Online]. Available: <https://arxiv.org/abs/1307.0191>
- [13] C. Strauch, U.-L. S. Sites, and W. Kriha, "NoSQL databases," Stuttgart Media Univ., Stuttgart, Germany, Tech. Rep. 8, 2011, vol. 20.
- [14] C. de Souza Baptista, C. E. S. Pires, D. F. B. Leite, and M. G. de Oliveiraa, "NoSQL geographic databases: An overview," in *Geographical Information Systems: Trends and Technologies*, vol. 73. London, U.K.: CRC Press, 2014.
- [15] G. P. O. Reddy, "Spatial data management, analysis, and modeling in GIS: Principles and applications," in *Geospatial Technologies in Land Resources Mapping, Monitoring and Management*. Cham, Switzerland: Springer, 2018, pp. 127–142.
- [16] L.-Y. Wei, Y.-T. Hsu, W.-C. Peng, and W.-C. Lee, "Indexing spatial data in cloud data managements," *Pervasive Mobile Comput.*, vol. 15, pp. 48–61, Dec. 2014.
- [17] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi, "MD-HBase: A scalable multi-dimensional data infrastructure for location aware services," in *Proc. IEEE 12th Int. Conf. Mobile Data Manage. (MDM)*, vol. 1, Jun. 2011, pp. 7–16.
- [18] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, "Spatio-temporal indexing in non-relational distributed databases," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2013, pp. 291–299.
- [19] Y. Zhong, J. Han, T. Zhang, and J. Fang, "A distributed geospatial data storage and processing framework for large-scale WebGIS," in *Proc. 20th Int. Conf. Geoinform.*, 2012, pp. 1–7.
- [20] D. Han and E. Stroulia, "HGrid: A data model for large geospatial data sets in HBase," in *Proc. IEEE 6th Int. Conf. Cloud Comput. (CLOUD)*, Jun./Jul. 2013, pp. 910–917.
- [21] Q. Ruqiong, "Research on the storage and services of digital cached map tiles based on mongodb," *Geospatial Inf.*, vol. 6, p. 060, Jun. 2014.
- [22] M. Miler, D. Medak, and D. Odošćić, "Two-tier architecture for Web mapping with NOSQL database couchdb," in *Proc. GI Forum*, 2011, pp. 1–10.
- [23] Y.-T. Hsu, Y.-C. Pan, L.-Y. Wei, W.-C. Peng, and W.-C. Lee, "Key formulation schemes for spatial index in cloud data managements," in *Proc. IEEE 13th Int. Conf. Mobile Data Manage. (MDM)*, Jul. 2012, pp. 21–26.
- [24] Y. Li, G. Kim, L. Wen, and H. Bae, "MHB-Tree: A distributed spatial index method for document based NoSQL database system," in *Ubiquitous Information Technologies and Applications*. Dordrecht, The Netherlands: Springer, 2013, pp. 489–497.
- [25] Y. Liu, Y. Wang, and Y. Jin, "Research on the improvement of MongoDB auto-sharding in cloud environment," in *Proc. 7th Int. Conf. Comput. Sci. Educ. (ICCSE)*, 2012, pp. 851–854.
- [26] W. Cho and E. Choi, "Spatial big data analysis system for vehicle-driving GPS trajectory," in *Advanced Multimedia and Ubiquitous Engineering*. Singapore: Springer, 2017, pp. 296–302.
- [27] Y. Wang, S. Wang, and D. Zhou, "Retrieving and indexing spatial data in the cloud computing environment," in *Proc. IEEE Int. Conf. Cloud Comput.* Berlin, Germany: Springer, 2009, pp. 322–331.
- [28] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," *BMC Bioinf.*, vol. 11, p. S1, Dec. 2010.



DONGFANG ZHANG received the B.Sc. degree from the China University of Geosciences, Wuhan, China, where he is currently pursuing the Ph.D. degree with the School of Computer Science. His current research interest includes the understanding of remote sensing images.



YONG WANG received the Ph.D. degree from the China University of Geoscience, Wuhan, China, where he is currently an Associate Professor with the School of Computer Science. His current research interests include parallel storage and processing of spatial data, and the understanding of remote sensing images.



ZHENLING LIU received the master's degree from the China University of Geosciences. Her current research interest includes parallel processing for massive spatial data.



SHIJIE DAI received the B.Sc. degree from the China University of Geosciences, Wuhan, China, where he is currently pursuing the master's degree with the School of Computer Science. His current research interest includes remote sensing image segmentation.

...