Distributed Storage and Index of Vector Spatial Data Based on HBase

Lin Wang[1], Bin Chen[1]*, Yuehu Liu[1]
[1]Institute of Remote Sensing and Geographic Information System,
Peking University, Beijing, China
*Corresponding author, e-mail: gischen@pku.edu.cn

Abstract—**In order to deal with the storage and query problems of massive vector spatial data, we design a vector spatial data model based on HBase. Then, a parallel method of building Hilbert R-tree index using MapReduce and packed Hilbert R-tree algorithm is proposed. In the end, the proposed vector spatial data model and the method of building Hilbert R-tree index in parallel are proven to be effective and efficient by experiments.**

Keywords-Hadoop; HBase; MapReduce; Vector Spatial Data; R-tree

I. Introduction

As GIS and IT industries boost and the scale of vector spatial data magnify, the traditional approaches of managing and storing vector spatial data could no longer fulfill the increasing requirements. However, Apache Hadoop, which is considered as one of the most important tools to deal with massive data, has received increasing attention. It is an open-source software project for reliable, scalable, distributed computing [1]. Hadoop provides distributed storage and management of massive data while preventing some traditional complicated problems, such as data partition and failure handling from the internal system.

Hadoop has two key components: Hadoop Distributed File System (HDFS) for reliable storage and MapReduce for high-performance parallel analysis, derived from the Google File System (GFS) and Google's MapReduce papers respectively. There are also other related projects in Hadoop, providing more services based on its kernel [2]. For example, HBase is the Hadoop database built on top of HDFS which stores a large amount of sparse data in a fault-tolerant way. Due to the powerful storage and computing abilities of Hadoop, it can offer a reliable solution to store and manage massive vector spatial data. Therefore, the main objective of this paper is to explore and accomplish the storage and index of vector spatial data using HBase and MapReduce.

II. Preliminaries

This section gives a quick overview of some core concepts about HBase, MapReduce and packed Hilbert R-tree.

A. HBase

HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable [3], supporting both structured and unstructured data. In HBase, Tables are composed by rows and columns. Columns are grouped into column families. Table cells are versioned using timestamps assigned by HBase automatically when inserted into table. Table cells including row keys are all byte arrays. Rows are always sorted in ascending order of row keys by default and the sort is byte-ordered. But timestamps are stored in descending order. As HBase is a column-oriented database, tables are stored on a per-column family basis, which means empty cells are not stored [4]. Table I is an example of an HBase table. A request for the values of "cf2: column1"in the row "row1" if no timestamp is specified would be the value from time stamp t3, that is r1cf2col1v2.

TABLE I. An Example of An HBase Table

Row Key	**Time Stamp**	**Column Family "cf1"**	**Column Family "cf2"**	
			column1	***column2***
row1	t3	r1cf1v1	r1cf2col1v2	
	t2		r1cf2col1v1	
	t1			r1cf2col2v1
row2	t5		r2cf2col1v1	r2cf2col2v2
	t4	r2cf1v1		r2cf2col2v1

HBase partitions tables horizontally into regions. Regions are the units of distribution across an HBase cluster. A region is defined by the key of its first row as the start key, inclusive, and the key of its last row as the end key, exclusive [5]. When an HBase table is created, there is only one region in the table by default. After the size of region exceeds a configurable size threshold, it splits into two regions automatically.

B. MapReduce

MapReduce is a programming model for processing large data sets proposed by Google. It is a parallel and distributed algorithm on a cluster. In short, a MapReduce program has two key functions: the map function and the reduce function, each function done in parallel. In the map phase, the framework distributes map tasks across nodes in the cluster. Each map task processes key/value pairs of its data fragment assigned by the framework and produces a set of intermediate key/value pairs. In the reduce phase, reduce function merges all intermediate values with the same intermediate key [6].

Apache Hadoop implements this MapReduce model. Fig. 1 shows the classes involved in the Hadoop implementation [7].

Supported by Key Technologies R&D Program of China funded by MOST(No. 2011BAH06B01), and NSFC (No. 41101347).

In Hadoop, MapReduce jobs can also use HBase as a source and/or sink.

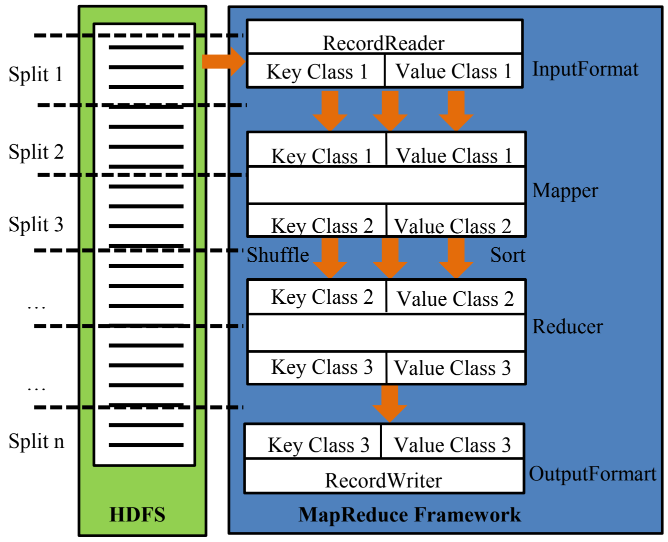

Figure 1. The MapReduce process[7]

C. *Hilbert R-tree*

Hilbert R-tree [8] uses points' Hilbert value, which is the length of the Hilbert curve from the origin to the point, to impose a linear ordering on the data rectangles. Let C represents the capacity of tree node and L represents the level in the tree, then the algorithm of packed Hilbert R-tree is described as follows [9]:

```
Phase 1. Calculate the Hilbert value of each data rectangle
Phase 2. Sort data rectangles on ascending Hilbert values
Phase 3. Create leaf nodes (L=0):
    While (there are more rectangles)
        Generate a new R-tree node
        Assign the next C rectangles to this node
        Calculate the Hilbert value of this node's MBR
Phase 4. Create non-leaf nodes at higher level:
    While (there are more than one node at level L)
        Sort nodes at level L on ascending Hilbert values
        Repeat Phase 3 to generate nodes at Level L+1
```

Packed Hilbert R-trees are suitable for static or rarely modified database and the algorithm is parallelizable [10].

III. METHODOLOGY

A. *Vector Spatial Data Model*

Vector spatial data usually contain geometric coordinates and attributes. According to this, table schema of vector spatial data layer is designed as Table II. Coordinates of geographic entities are stored in WKB format. Geo_bound is composed by coordinates of the lower left corner and the upper right corner of the entity's minimum bounding rectangle (MBR).

TABLE II. SCHEMA OF VECTOR SPATIAL DATA LAYER

	Time Stamp	**attributes**			**coordinates**	**geo_bound**
		atte1	***attr2***	***attr3***		
Row Key	t3			a3v2		
	t2		a2v2			
	t1	a1v1	a2v1	a3v1	geocoor1	geobound1

HV represents Hilbert value and E_ID represents entity ID.

In this model, row key is denoted by Hilbert value and entity ID sequentially. We use a 512*512 grid to calculate the Hilbert value of an entity, so Hilbert value will not be greater than 524288. As demonstrated in Fig. 2, row key is a byte array with a length of 16. The first 6 bytes are utilized to store Hilbert value and the last 10 bytes to store entity ID. Fig. 2 shows that the row key of the entity with ID 301050 and Hilbert value 6401 is 0064010000301050.

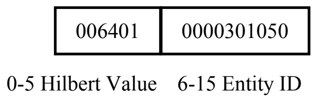

Figure 2. Row key

The Hilbert value of an entity is defined by the Hilbert value of the grid cell to which the center of the entity's MBR belongs.

In this data model, records are ranked in ascending order of Hilbert values prior to IDs. Row key designed in this way guarantees that entities close to each other in geographical space would be stored in adjacent rows in the table, thus it can speed up range query. Meanwhile, this design avoids utilizing the monotonically increasing row key, which might cause the overload problem of single machine [11]. On the other hand, records ordered by ascending Hilbert values are beneficial to create a spatial index. But this design of row key might cause inconvenience when the coordinates of the entity is modified. The change of the coordinates might cause the change of the MBR. Then, it might lead to change the Hilbert value and finally end in changing the row key. As the row key changes, this row must be deleted and the content of this row must be reinsert to create a new row using the new row key.

As data is stored in byte arrays in HBase, when we read them from HBase tables, we must assign correct data types to them. Therefore, we need create a layer management table to store layer name, all attributes' data types and the layer's MBR. The design is shown in Table III.

TABLE III. SCHEMA OF LAYER MANAGEMENT

Row Key	**Time Stamp**	**layer_name**	**attributes_type**			**geo_bound**
			atte1	***attr2***	***attr3***	
L_ID	t2			Int	String	
	t1	buildings	String			geobound1

L_ID represents layer ID.

B. *Vector Spatial Data Index*

So far, HBase has not provided efficient approaches to deal with spatial query. Thus, this paper proposes a method of building spatial index in parallel based on MapReduce and packed Hilbert R-tree.

The design of this algorithm was inspired by packed Hilbert R-tree [9] and [12], which use Z-order space-filling curve to map two-dimensional points into single dimension value and build R-tree in parallel using MapReduce. The core idea of building Hilbert R-tree in parallel is described as follows: First, sort spatial entities on ascending Hilbert values, then divide them into fragments as equally as possible; secondly, each fragment generates a local R-tree from bottom to top according to packed Hilbert R-tree algorithm given in Section II Part C; finally, merge local R-trees into a global R-tree.

The detailed algorithm of constructing Hilbert R-tree based on HBase in parallel is described in following steps.

1) *Sampling*

In order to get split points as the standard for data partition, we need take samples from large dataset. Due to the design of row key, spatial entities in the table are automatically sorted on ascending Hilbert values. In this scenario, interval sampling method provided by Hadoop is utilized to obtain samples faster than random sampling. After interval sampling, we need to write partition file and add it into distributed cache. The partition file stores sorted split points that are obtained from sorted samples. The number of split points is one less than the number of reducers.

2) *Partition and Sort*

This step is to sort spatial entities on ascending Hilbert values and divide them into fragments as equally as possible in preparation for building local R-trees. The process of this step is shown in Fig. 3 described as follows.

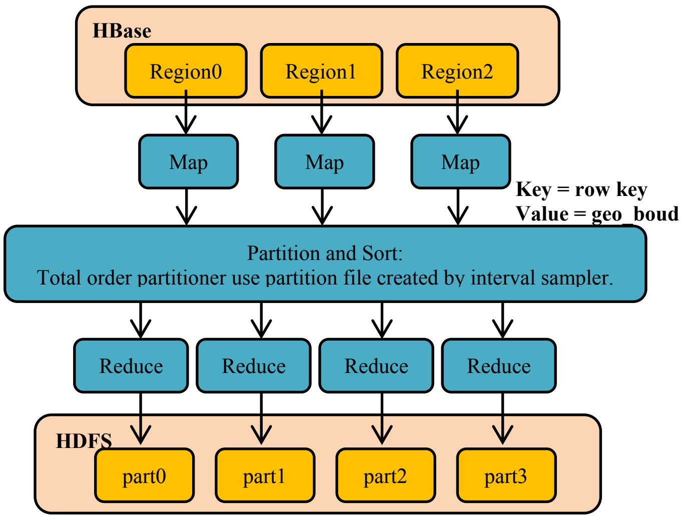

Figure 3. Partition and Sort

Map: Map function extracts the corresponding row key and geo_bound from each row in the table as the key and the value respectively. The number of map tasks depends on the number of the table's regions.

Partition: After map function exports certain key/value pair, partition function would assign it to the correct reducer as its input. Here, we use TotalOrderPartitioner class provided by Hadoop, and the assignment is based on the split points provided by the partition file in the distributed cache. TotalOrderPartitioner makes Hilbert values of the spatial entities assigned to reducer i smaller than the ones assigned to reducer $i+1$. For example, if R is the number of reducers, sorted split points is represented as $S[0\colon R-2]$ and reducers are numbered from 0 to $R-1$ in sequence, then the key/value pair whose key in $(S[i], S[i+1]]$ will be sent to reducer $i+1$.

Sort: Before each reducer processes data pulled from nodes where map tasks run, the default sort function sorts input records of each reducer in ascending order of keys.

Reduce: At last, each reducer exports a file that stores sorted key/value pairs assigned to it.

3) *Local R-tree Construction*

We utilize map function's feature of local reading to process the output file of each reducer in the first step. In order to import the file as a whole into map function in the second step, we customize WholeFileInputFormat class [13]. The process of local R-tree construction is show in Fig. 4.

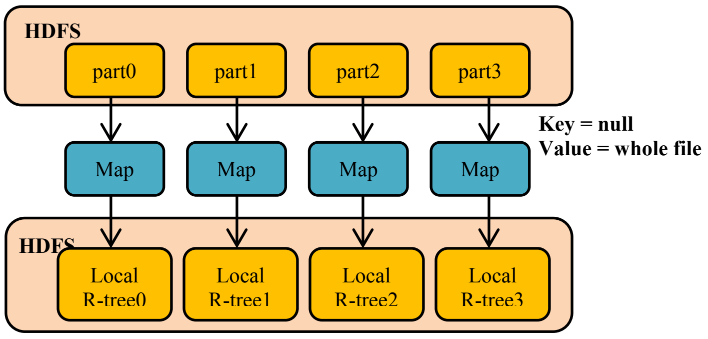

Figure 4. Local R-tree Construction

Local R-tree construction algorithm references packed Hilbert R-tree algorithm described in Section II Part C. Based on sufficient sampling and partition, each local R-tree has the same height in general. Exceptions are ignored here.

4) *Global R-tree Construction*

Merging local R-trees is executed outside the cluster. If the node capacity is greater than or equal to the number of reducers in the first step, each local R-tree becomes a subtree of a common root node, forming a global R-tree. If the node capacity is smaller than the number of reducers, we need to add an extra level under the root node to merge local R-trees. In the end, global R-tree index is written into HDFS.

IV. EXPERIMENTS

A. *Datasets and Setup*

The experiments use shape file (.shp) spatial data of buildings and roads of Italy, which are OpenStreetMap derived data made by Geofabrik GmbH. The longitude range of these two datasets is from 6.605773 to 18.516541, and the latitude range is from 35.495028 to 47.108010. Detailed descriptions of these two datasets are shown in Table IV.

TABLE IV. VECTOR SPATIAL DATASETS

Dataset	Data Size(GB)	Spatial Entities	Geometry Type
roads	0.68	1937972	Polyline
buildings	1.15	4808986	Polygon

Data up to 2013-03-12

In this experiment, we use Hadoop 0.20.2 and HBase 0.94.2. The Hadoop cluster consists of seven VMware virtual nodes running Ubuntu (32-Bit) 10.04 operating system. One of them is the master, acts as name node, HMaster and job tracker. The other six nodes are slave nodes named slave01~slave06 in turn. Each slave node acts as data node, compute node, HRegionServer and task tracker. The client is also a VMware virtual node running Ubuntu (32-Bit) 10.04 operating system. All these VMware virtual nodes are created from four host computers. Detailed configurations are shown in Table V.

TABLE V. CONFIGURATIONS OF NODES

Node	CPU	P	CP	RAM (GB)	Host Computer
client master	Inter Core i7-E2600k, 3.40GHz	2	2	4	host01
slave01 slave02	Inter Core i3-530, 2.93GHz	1	2	1	host02
slave03 slave04	Inter Core i3-530, 2.93GHz	1	2	1	host03
slave05 slave06	Inter Core i3-530, 2.93GHz	1	2	1	host04

P = number of processors, CP = number of cores per processor

B. *Importing Vector Spatial Data*

Vector spatial data are imported into HBase using multithreading. Each thread imports data from different position of the dataset. We set parameter writebuffersize to 5M, close WAL, use default values for the other parameters. The spatial range of the datasets is partitioned into 512*512 grid cells to calculate Hilbert values. As shown in Table VI, data import using multithreading has much better time performance than that using single thread. In current testing environment, time performance has very obvious improvement when data import uses 2 threads compared with single thread. But the improvement is not obvious from 2 threads to 3 or 4 threads; sometimes time performance even deteriorates a little.

TABLE VI. TIME PERFORMANCE OF DATA IMPORT WITH VARIOUS NUMBER OF THREADS

Dataset	Number of Threads	Time (s)
roads	1	946.064
	2	587.952
	3	728.962
	4	837.257
buildings	1	2460.651
	2	1173.932
	3	1135.934
	4	1028.730

C. *Building Hilbert R-tree Index in Parallel*

The dataset of buildings is used to analyze time performance of building Hilbert R-tree index in parallel. The regions in the table of buildings are shown in Table VII.

TABLE VII. REGIONS BY REGION SERVER

Region Server	Region Count
slave01	1
slave02	1
slave03	2
slave06	2

In order to get more balanced data distribution to ensure time performance of building local R-trees, two experiments are performed. The first experiment is to find how interval sampling frequency impacts on sampling time and data distribution. The number of reducers is fixed at 12. Standard deviation (StdDev) of the total number of input records for each reducer is used to evaluate data distribution:

$$StdDev = \sqrt{\frac{1}{R}\sum_{i=0}^{R-1}\left(R_i - \frac{D}{R}\right)^2}$$

R is the number of reducers, D is the number of dataset records, and R_i is the number of input records for reducer i.

As shown in Fig. 5, the value of StdDev largely decreases and tends towards stable when sampling frequency increases to 2%. Fig. 6 indicates that sampling frequency has a negligible effect on sampling time. The reason is that interval sampling must scan the whole table regardless of the value of sampling frequency.

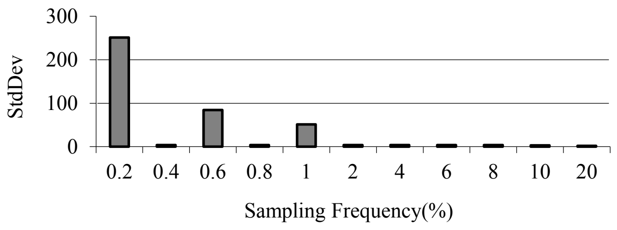

Figure 5. Standard deviation (StdDev) of the total number of input records for each reducer by various sampling frequency when R = 12

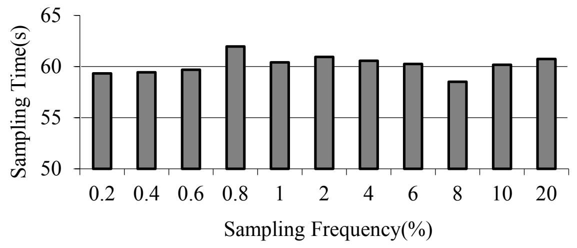

Figure 6. Sampling time by various sampling frequency when R = 12

In the second experiment, sampling frequency is fixed at 6% and the number of reducers varies from 6 to 36. As shown in

Fig. 7, the number of reducers has no obvious effect on the value of StdDev. The reason is that the number of records in the dataset is very large and the number of reducers is relatively very small.

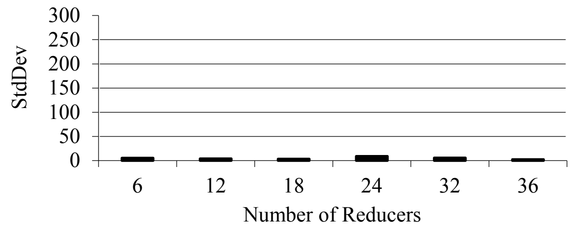

Figure 7. Standard deviation of the total number of input records for each reducer by various number of reducers when sampling frequency is 6%

At last, time performance of building index by various number of reducer is tested. Sampling frequency is fixed at 6% and tree node capacity is set to 40. The total time contains the time of sampling, partition and sort, and local R-tree construction. The time of merging subtrees is not included because it is completed in the client and has little significance compared with the other steps. As a reference, a single-process (SP) Hilbert R-Tree index (references packed Hilbert R-tree given in Section II Part C) is created on the node slave04.

TABLE VIII. TIME PERFORMANCE OF PARALLEL CONSTRUCTION OF HILBERT R-TREE BY VARIOUS NUMBERS OF REDUCERS

Number of Reducers	Time (s)			
	Get Data		Local R-tree	Total
	Sampling	Partition and Sort		
6	61.325	57.500	24.209	143.034
12	60.646	58.442	20.232	139.320
18	61.286	66.826	23.237	151.349
24	61.239	77.192	23.233	161.664
32	61.248	82.901	27.325	171.474
36	60.031	91.871	26.260	178.162
SP	88.997		642.053	731.050

Table VIII shows that single-process construction of Hilbert R-tree costs less time in the phase of getting data, because single-process construction only need read sorted records from the table, whereas parallel construction need partition and sort data besides reading data. In the phase of building Hilbert R-trees, parallel construction costs much less time than single-process construction, mainly because single-process construction must deal with all records in memory at once. In general, during the whole process, parallel construction has much better time performance than single-process construction. Moreover, a reasonable number of reducers can get better time performance.

D. *Querying Vector Spatial Data*

The steps of range query are as follows: First, load the Hilbert R-tree index from HDFS; second, find the entities whose MBR intersects the query rectangle through the index and return their row keys; last, get all data from the table through their row keys and write the results into the local disk.

Table IX illustrates time performance of range query using the index built by 12 reducers.

TABLE IX. TIME PERFORMANCE OF RANGE QUERY

Query Range	Time (s)				Number of Query Results
	Load Index	Query Index	Query Table and Save	Total	
(14.349, 41.945, 15.851, 42.493)	8.449	0.000	5.717	14.166	1925
(8.912, 39.096, 9.708, 40.543)		0.005	19.545	27.999	10837

V. CONCLUSION AND FUTURE WORK

This paper researches the distributed storage and index of vector spatial data based on HBase and MapReduce. A spatial data model based on HBase and a parallel method of building Hilbert R-tree spatial index using MapReduce are designed. Also, the effectiveness and efficiency of this spatial data model and index are verified by experiments. The future work is to improve the design and performance of the data model and index.

ACKNOWLEDGMENT

Thanks are due to colleagues in our laboratory for their valuable comments and other helps. The authors are also grateful to the reviewers in the Program Committee.

REFERENCES

[1] Apache Hadoop project, http://hadoop.apache.org/

[2] T. White, Hadoop: The Definitive Guide,1st ed., O'Reilly Media, Inc.,2009,pp.4-12.

[3] Apache HBase, http://hbase.apache.org/

[4] L. George, Hbase: the Definitive Guide, 1st ed., O'Reilly Media, Inc., 2011, pp. 17-20.

[5] T. White, Hadoop: The Definitive Guide,1st ed., O'Reilly Media, Inc., 2009, pp. 345.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol.51(1), 2008, pp.107-113.

[7] L. George, Hbase: the Definitive Guide, 1st ed., O'Reilly Media, Inc., 2011, pp. 290.

[8] I. Kamel and C. Faloutsos, "Hilbert R-tree: An improved R-tree using fractals," In Proc. of VLDB Conf.,Santiago, Chile, September 1994, pp. 500-509.

[9] I. Kamel and C. Faloutsos, "On Packing R-trees," Second International ACM Conference on Information and Knowledge Management (CIKM), Washington D.C., 1993, pp. 490-499.

[10] I. Kamel and C. Faloutsos, "Parallel R-Trees," In Proc. of ACM SIGMOD Conf., San Diego, CA, June 1992, pp. 195–204.

[11] T. White, Hadoop: The Definitive Guide,1st ed., O'Reilly Media, Inc.,2009, pp.357.

[12] A. Cary, Z. Sun, V. Hristidis and N. Rishe, "Experiences on processing spatial data with MapReduce," Proceedings of 2009 Statistical And Scien tific Database Management, New Orleans: Springer-Verlag, 2009, pp.302-319.

[13] T. White, Hadoop: The Definitive Guide,1st ed., O'Reilly Media, Inc., 2009, pp. 192.