*Journal of Network Intelligence*
**Taiwan Ubiquitous Information**
 ISSN 2414-8105
Volume **8**, Number **4**, November **2023**

# Spatial Indexing Algorithm for Big Data of Traffic Trajectories in Parallel Computing Mode

Ying Chen*

Information Technology and Engineering Department
Guangzhou Institute of Technology
Guangzhou 510075, P.R. China
70548@gzvtc.edu.cn

Xiao-Ling Wu

School of Information Technology and Engineering
St. Paul University Philippines
Tuguegarao 3500, Philippines
wuxiaoling@spup.edu.ph

*Corresponding author: Ying Chen
Received February 21, 2023, revised March 24, 2023, accepted May 20, 2023.

Abstract. *With the massive deployment of location acquisition devices, an increasing amount of traffic trajectory data is generated. Since traffic trajectory data is spatial data, querying it requires a large number of spatial computation operations. However, as the number of users becomes increasingly large, real-time querying of large data of traffic trajectories becomes very difficult. To solve this problem, this work designs an efficient spatial indexing algorithm based on Hilbert-R-tree using the cloud computing platform Hadoop. First, the spatial data set is pre-processed and the pre-processed data set is Hilbert coded. Then, the encoded spatial dataset is sorted and partitioned in parallel, and the partitioned dataset is sent to each host node. In order to solve the problem of uneven spatial distribution, the spatial index structure is constructed using parallel computing mode and the spatial index structure constructed in each host node is merged so as to ensure the adjacency of the spatial data. Finally, local indexes are constructed using R-trees and global indexes are constructed using improved Hilbert-R-tree to improve the efficiency of spatial data retrieval. A test analysis was conducted on a Hadoop distributed environment with real traffic data as an example. The results show that the spatial data indexing in the distributed environment has good performance and can meet the demand for real-time query of big data of traffic trajectories.*
**Keywords:** Traffic tracks; Big data; Parallel computing; Hadoop; Spatial indexing

1. **Introduction.** In recent years, with the rapid development and popularity of the Internet, spatial data query has played an increasingly important role in people's social life. At the same time, spatial application technologies based on Geographical Information System (GIS) [1,2] and Location-Based Social Networking Services (LBSNS) [3,4] have been widely used, and spatial query processing technology is the basic functional component to support these spatial application The spatial query processing technology is the basic functional component to support these spatial application technologies. Among them, spatial queries are mainly concerned with traffic and travel applications.

With the development of satellite communication and location acquisition technology, it has become increasingly convenient to obtain information about the spatial location of

a target at a certain moment in time, which has led to a multiplication of the amount of location information in recent years [5]. A large amount of trajectory data is generated around moving objects such as vehicles, ships, people and animals, and has typical object-related characteristics, such as the trajectory data of a vehicle for a day. These trajectory data, in turn, involve properties in both time and space dimensions, and can often be regarded as time series data of mobile objects in a specific space (e.g. traffic road network) [6]. Trajectory data has certain spatio-temporal correlation and distribution characteristics, and most trajectory analysis applications are based on these data with spatio-temporal properties. However, with the geometric multiplication of spatial data, the diversity of data types (e.g. multi-dimensional geographical data, etc.), and the increasing density of concurrent access by multiple users, existing spatial data query techniques are facing serious challenges [7,8].

Spatial data query techniques are widely used in many fields. With the development of data acquisition technologies, the volume of data has expanded dramatically and the variety of data has increased, making it difficult for traditional serial algorithms and methods to effectively cope with the high performance requirements of data retrieval [9,10], so parallel computing techniques can provide new solutions. Cloud Computing can use inexpensive servers to form a cluster system to manage and apply large spatial data through distributed technology. Cloud Computing has significant advantages over other technologies in the processing of spatial big data due to its versatile, distributed, large-scale and highly scalable characteristics. In the cloud computing environment, if the reasonable storage and efficient indexing of spatial data can be realised, it will make it more convenient for users to use spatial data quickly and conveniently. Currently, Hadoop and MapReduce are being used to solve the problems of parallel access and processing of spatial big data, enabling the processing of many types of large-scale data.

Therefore, this work attempts to implement parallel processing of spatial big data in a cloud computing environment. Specifically, how to execute spatial indexing algorithms on a Hadoop platform so as to achieve high performance querying of spatial data. Spatial indexing is crucial for spatial databases, which can provide suitable data structures to construct mapping relationships between spatial objects and spatial locations, enabling one to precisely locate and quickly access specified spatial objects from spatial databases, thus greatly improving data retrieval efficiency.

1.1. **Related Work.** Trajectory data has significant spatial characteristics and is data information generated by recording the movement of a moving object. As a kind of spatial data, trajectory data can be collected from various sources, such as GPS [11], mobile services [12], mobile phone base stations [13], POS [14], in-vehicle recorders [15] and so on. Spatial indexing as the basis of spatial databases, it has been extensively studied, such as binary trees, B-trees, space-filling curves and Hashing trees, etc.

Binary tree based indexes are set to have no more than two nodes per node, and are therefore only suitable for applications where the spatial data is evenly distributed, and less efficient if the spatial data is unevenly distributed. B-tree based indexes are derived from binary tree indexes and no longer set the number of leaf nodes per node to 2, which avoids the branch degradation that occurs in binary trees. Space-filling curve-based indexing is based on dividing the region of a spatial object into multiple grids and then connecting the grids with curves. Indexing based on Hashing tree [16] divides the spatial data into multiple regions and generates a Key value for each region by using spatial coordinates for Hash calculation, thus achieving fast location. However, there are some shortcomings in the above methods when dealing with high-dimensional data

and parallelisation, such as dimensionality reduction of high-dimensional data, division of data, etc.

Over time, however, the emergence of distributed computing technologies has opened up new opportunities for indexing spatial big data. With the rapid growth of spatial datasets, various computations under the cloud computing platform have also been rapidly developed. One of the most widely used is MapReduce [17]. MapReduce is a computational framework proposed by Google. Colosimo et al. [18] proposed to use MapReduce to construct R-trees and use Z-curves as the partition function. Although the construction time decreases significantly with increasing parallelism, the merging operation of R-trees tends to lead to lower spatial query efficiency. Zhu et al. [19] proposed to use a per-layer design on MapReduce to load R-trees in parallel, and the algorithm is easy to implement, highly reliable and easy to scale. Sinha et al. [20] proposed algorithms that can quickly construct R-tree space indexes based on the MapReduce parallel computing model. Bāders et al. [21] designed a two-tier distributed spatial index for pruning search spaces, and proposed a MapReduce-based data indexing architecture to improve the computational power of spatial queries.

Several of the approaches proposed above mostly construct indexes based on static data, without fully analysing the shortcomings of traditional spatial indexing mechanisms, especially the impact of the data partitioning process on individual host nodes in cloud computing.

1.2. **Motivation and contribution.** In this paper, we propose a novel spatial indexing algorithm for trajectory big data in a big data environment with the objective of optimising trajectory data query by taking massive traffic trajectory data from multiple sources as the research object. The challenge of the research work is to design a spatial query technique based on parallel processing, so as to efficiently handle the massive scale of trajectory data.

The main innovations and contributions of this work include:

(1) As spatial data also grows dramatically and with increasing complexity, traditional serial techniques become inefficient in handling spatial queries, so this paper proposes a Hilbert-R-tree construction algorithm based on a parallel computing model. The proposed algorithm uses the properties of Hilbert space curves to sort and partition spatial data in parallel, while it well balances the load of each host node in the data partitioning process.

(2) In order to overcome the shortcomings of the traditional spatial indexing mechanism, this paper uses R-tree to construct a local index for the divided spatial data in a bottom-up manner according to the distribution order of the geographic entity elements, and then constructs a global index according to the Hilbert-R-tree, storing the information of the local index and the global index on the DataNode and NameNode respectively, so as to improve the efficiency of spatial data retrieval.

## 2. Trajectory query problem analysis and definition.

2.1. **Definition of trajectory data.** A trajectory is a curve that describes the path of motion of a target object, e.g. a driving trajectory describes the path of a vehicle. The study of trajectories begins by considering the description of the trajectory. For descriptive convenience, the target object is abstracted into a target point. After a simple abstraction, the physics of a trajectory is defined below.

Firstly, the curve $l$ formed by the position $s$ of a moving point p in space varying continuously with time t is called a trajectory $T$. For the storage of trajectories, query operations, it is not enough to have an abstract definition description, but a more intuitive and concrete mathematical description is also needed. In mathematics, curves

can be described from two perspectives, continuous descriptions, such as curve equations. The other is a discrete description, such as representing a section of a curve by a finite number of points. Computers can only handle finite discrete problems, so trajectories are represented in the computer in discrete form, i.e. trajectories are represented by a finite number of ordered points. The definition of a spacetime point is shown below:

$$p = (x, y, t) \tag{1}$$

where $x$ represents longitude, $y$ represents latitude and $t$ represents the time attribute of the spatio-temporal point.

Suppose that a trajectory Traj $= \{p_0, p_1, \ldots, p_n\}$ is an ordered set of spatio-temporal points generated by a moving object [22].

$$p_i = (d_i, t_i), i \in \{1, 2, \ldots, n\} \tag{2}$$

A monitoring point *d (x, y)* is a deployment point of a collection device, which has spatial properties. The set of all monitoring points *d (x, y)* is called the set of monitoring points $\{D|d_i\,(x_i, y_i) \in D, i = 0, 1, 2, \ldots, n\}$ .

For two consecutive spatio-temporal points on a trajectory, if there is an unrecorded spatio-temporal point $p_{key}$ between them, then this spatio-temporal point $p_{key}$ is called the key point.

$$\frac{\sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2}}{t_{i+1} - t_i} < v_{\text{thresh}}\,, 0 \le i \le n-1 \tag{3}$$

where $v_{\text{thresh}}$ is the rate threshold.

We divide the trajectory with missing keypoints into two sub-trajectories: and . If for a sub-trajectory there is no key point pkey , this sub-trajectory is called a single-trip trajectory. A single trip trajectory is used to describe the movement of the target in one trip. For each trajectory, the format of the trajectory logic storage model is shown in Table 1.

Table 1. Track storage logic model.

| **Field** | **Traj ID** | **Car ID** | **P1** | | **P2** | | ... | **Pn** | |
|---|---|---|---|---|---|---|---|---|---|
| | | | **T1** | **G1** | **T2** | **G2** | ... | **Tn** | **Gn** |
| Explanation | Track ID | License plate number | Track point 1 | | Track point 2 | | ... | Track point n | |
| | | | Time 1 | Position 1 | Time 2 | Position 2 | ... | Time n | Location n |

2.2. **Trajectory query problem analysis.** The needs of trajectory query may be complex and diverse, but they are all combinations or variations of the following types of basic trajectory query. With the frequent increase in the types of trips, the demand for trip queries is increasing and is a new type of trajectory query.

The spatial data selection operations can be divided into two types of point queries and trajectory queries. The object of a trajectory query must be a trajectory, and the returned result must be a trajectory or a collection of trajectories [23]. For example, find all trajectories within 100m of a spatial point (114.0791000, 22.5458000). The parameters required for the point query and the query example are listed below.

**Method**: TrajPointQuery.

**Input**: Spatial-Point, Thresh.

**Output**: Traj [].

Example of a query.

Spatial-Point (114.0791000,22.5458000), Thresh=100.00meter.

Spatial-Point (116.3464700,40.0200400), Thresh=100.00meter.

Spatial-Point (116.2802000,39.8956000), Thresh=100.00meter.

Suppose that a rectangle R can be represented by two endpoints S and T on its main diagonal.

$$R = (S, T), S = [s_1, s_2, \ldots, s_n], T = [t_1, t_2, \ldots, t_n] \tag{4}$$

The distance from a point $P$ to a rectangle $R$ in the same space is noted as *MINDIST (P, R)*.

$$\mathrm{MINDIST(P, R)} = \sum_{i=1}^{n} |p_i - r_i|^2 \tag{5}$$

If the points lie inside the rectangle, the distance between them is 0. If the points lie outside the rectangle, the distance between them is the square of the Euclidean distance. The square is used to reduce the computational cost, and the square of the distance is used as the metric later on.

$$r_i = \begin{cases} s_i, \text{ if } p_i < s_i \\ t_i, \text{ if } p_i > s_i \\ p_i, \text{ otherwise} \end{cases} \tag{6}$$

The minimum distance from a point P to an object o in space is expressed as

$$\|(P, o)\| = \min\left(\sum_{i=1}^{n} |p_i - x_i|^2, \forall X = [x_1, \ldots, x_n] \in O\right) \tag{7}$$

For a point $P$ and an MBR $R$ = *(S, T)* with the same dimension, MINMAXDIST is defined as shown below [24]:

$$\mathrm{MINMAXDIST}(P, R) = \min_{1 \le k \le n}\left(|p_k - rm_k|^2 + \sum_{\substack{i \ne k \\ 1 \le i \le n}} |p_i - rM_i|^2\right) \tag{8}$$

$$rm_k = \begin{cases} s_k & \text{if } \ p_k \le \frac{(s_k + t_k)}{2} \\ t_k & \text{otherwise} \end{cases} \tag{9}$$

$$rM_i = \begin{cases} s_i & \text{if } \ p_i \ge \frac{(s_i + t_i)}{2} \\ t_i & \text{otherwise} \end{cases} \tag{10}$$

An example of the calculation of MINMAXDIST for the query point p (8,0,0) for an n-dimensional object is shown in Figure 1.

## 3. **Hilbert-R-tree index design in parallel computing mode.**

3.1. **Fundamentals.** For R-trees, the overlap between nodes will lead to more invalid nodes being accessed when querying, increasing the number of data accesses and thus affecting its query performance. While Hilbert space filling curve has good clustering effect, it can effectively solve the clustering storage of data.

As spatial queries require a large number of computational operations, and the traditional serial processing method cannot provide results back to the user in an acceptable time when dealing with large amounts of data, this will lead to lower user satisfaction. Therefore, parallel processing of data is needed to improve the query efficiency. Hilbert space-filling curves can effectively reduce the overlap between nodes and reduce the dead space of index structure, thus facilitating the construction of indexes for each host node in a cluster. Therefore, this paper proposes a Hilbert-R-tree spatial index based on the parallel computing model, which can make full use of the Hilbert curve characteristics to reduce, sort and divide data, so as to build an efficient index and improve query performance.

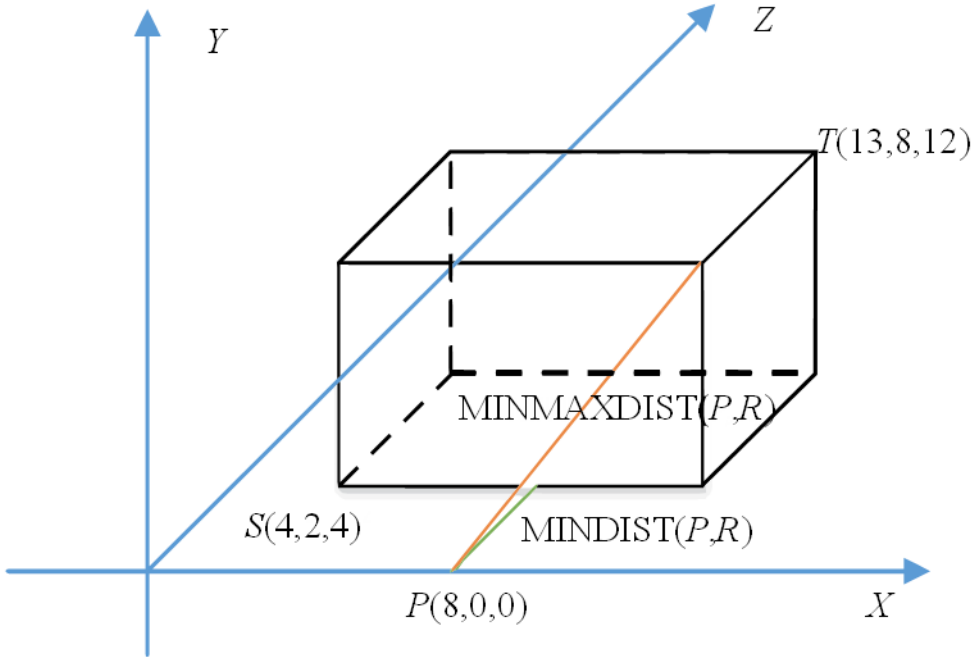


Figure 1. MINMAXDIST
Finally find MINDIST (P, R) =20 and MINMAXDIST (P, R) =105 for this example

First, for agenda parallelism problems where the number of tasks is greater than the number of processors, real parallel computers must be clustered, as shown in Figure 2, with each processor performing multiple tasks.

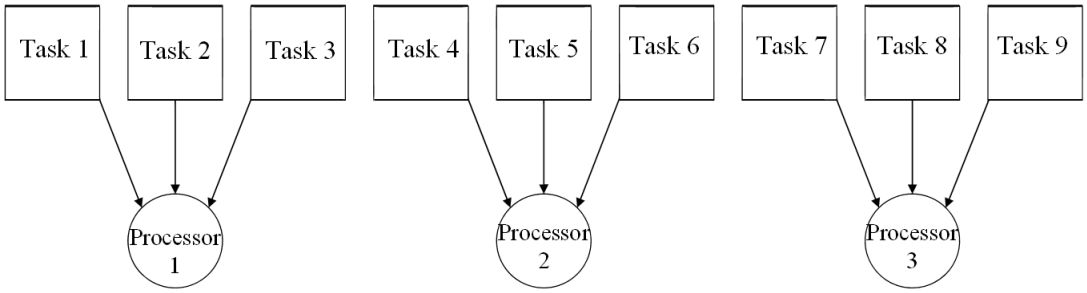


Figure 2. Parallel aggregation of agendas

Conceptually, agenda parallelism takes the form of backpack tasks, where each processor repeatedly takes tasks out of the backpack and executes them until the backpack is empty, as follows:

**Processor** 1: Takes out tasks and executes them when they become available;
**Processor** 2: Takes out tasks and executes them when they become available;
....
**Processor** K: takes out tasks and executes them when they are available;

In cluster parallel computers, the agenda parallelism problem of clustering is usually implemented in the Master-Worker model, as shown in Figure 3. First, the Master sends the task to the Worker. then, the result of the execution is obtained from the Worker, thus tracking the result of the whole program. Each Worker receives the task from the

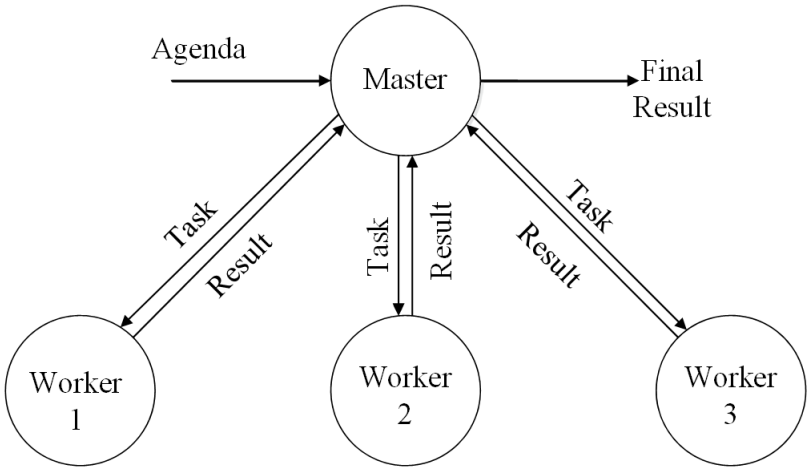


Figure 3. Master-Worker mode

Master and computes the result. In a cluster parallel program, the Master-Worker model is used exclusively for load balancing.

3.2. **Hilbert-R-tree coding.** The construction of a Hilbert-R-tree first requires the encoding of spatial data, and the Hilbert-R-tree encoding method is described below.

The operation that transforms spatial coordinate data into Hilbert-encoded [25] is called the forward operation. The operation that transforms Hilbert-encoded data into spatial coordinate data is called the inverse operation. The calculation procedure for the positive operation is shown in Table 2.

Table 2. Positive Arithmetic.

| $x$[i] | $y$[i] | $s$[2i:2i-1] | $x$[i-1:1] | y[i-1:1] |
|---|---|---|---|---|
| 0 | 0 | 00 | $y$[i-1:1] | $x$[i-1:1] |
| 0 | 1 | 01 | $x$[i-1:1] | $y$[i-1:1] |
| 1 | 0 | 11 | $y$[i-1:1] | $x$[i-1:1] |
| 1 | 1 | 10 | $x$[i-1:1] | $y$[i-1:1] |

For example, for a point (5,4) on an 8×8 plane. By checking the table, we can get $s$[6:5]=10, $x$[2]=0, $y$[2]=0. $x$[1]=0, $y$[1]=1, giving $s$[2:1]=01, so the final $s$=100001, i.e. $s$=33. Conversely, the inverse operation is calculated as shown in Table 3.

Table 3. Inverse operations

| $s$[2i:2i-1] | $x$[i] | $y$[i] | $x$[i-1:1] | y[i-1:1] |
|---|---|---|---|---|
| 00 | 0 | 0 | $y$[i-1:1] | $x$[i-1:1] |
| 01 | 0 | 1 | $x$[i-1:1] | $y$[i-1:1] |
| 10 | 1 | 0 | $y$[i-1:1] | $x$[i-1:1] |
| 11 | 1 | 1 | $x$[i-1:1] | $y$[i-1:1] |

3.3. **Construction of the Hilbert-R-tree.** First, the spatial data set is encoded and partitioned. Second, the partitioned data is sent to each Worker host node to build the spatial index structure in parallel. Finally, the Master node maintains a reference to the

root node of the spatial index structure constructed in each host node to facilitate spatial queries.

The construction of a Hilbert-R-tree in parallel computing mode consists of three main stages:

(1) Encode and partition the spatial dataset. The spatial data set D is partitioned into k equal parts of equal size, where a small error is acceptable in practice. Hilbert encoding is performed on the spatial data set, i.e. Hilbert encoding is calculated for the centroid of the MBR of each spatial object, and the spatial objects are sorted in parallel according to Hilbert encoding, and then the ordered spatial objects are divided into groups, thus dividing the spatial neighbouring objects into the same partition and facilitating the subsequent construction of the spatial index structure.

(2) Send the divided data to each Worker host node and build the spatial index structure in parallel. The specific design of the spatial index structure is described in detail later.

(3) Merging of Hilbert-R-tree indexes. the reference to the root node of the spatial index structure constructed in each host node is maintained in the Master host node to facilitate spatial queries. The pseudo-code for constructing a Hilbert-R-tree in parallel computing mode is shown in Algorithm 1.

**Algorithm 1** Build the Hilbert-R-tree

1: initial the cluster//initialise cluster information
2: **if** the host node is Master
/* A parallel team of two threads executes two parallel sections concurrently the master section and the worker section. */
3: ParallelTeam().execute(ParallelRegion())
4: MASTER:
5 **masterSection()**
6: barrier();
7: WORKER:
8: barrier();
9: **workerSection()**
10: **else**
11: myslices =world.receive()
12: **workerSection()**
13: **endif**
14: **for each** myslice in myslices do//master node and worker node build Hilbert-R-tree in parallel
15: tree.insert (myslice);
16: Merge the root node in all hosts;:// Merge the Hilbert-R-tree root nodes in all hosts

After initialising the cluster information, if a node in the cluster is the master node, two threads are opened to execute two sections in parallel, the masterSection and workerSection sections, as shown in Algorithm 2 and Algorithm 3 respectively. At this point, the workerSection section in the master node and the workerSection section in the worker node are in a blocking state and become active as soon as data is sent.

**Algorithm 2** masterSection()

1: read the dataset and compute Hilbert value for every data rectangle;
2: S=sort(records); /sort datarectangles by Hilbert values on the ascending orders.

3: myslices=partition(S);//partition the records according to the number of hosts.

The masterSection() function first reads the dataset in parallel and calculates the Hilbert value for each record, then sorts the dataset in parallel according to the Hilbert value, after which the data can be divided.

**Algorithm 3** workerSection()

1: **if** the host node is Master
2: worldscatter (root slices, myslices); // send data to master node
3: **else**
4: world.scatter(root,null, myslices)// send data to worker node
5: **endif**

The workerSection() function sends the above divided data to each host node.

## 4. Design of parallel spatial indexes for spatial data.

4.1. **Local index construction.** After partitioning the data, the number of feature targets contained in each data block does not vary greatly, so how to build an index on this already partitioned data is the next major problem to be solved.

Assuming that the number of geographic elements contained in a divided data block is N, then the number of outsourcing rectangles to be created for these geographic elements is N. For each node, the maximum number of geographic elements that can be contained is M, then the number of nodes required for all geographic elements is P = N/M. For spatial big data, it is assumed that when creating an R-tree, the empty tree is inserted one by one with geographic elements until the indexing is completed, which will lead to inefficiency and mismanagement [26]. Therefore bulk processing of spatially large data is required. To construct an R-tree index, the geospatial dataset needs to be sorted according to a dichotomous rule. The ordered geospatial elements are then pressed into the leaf nodes in the R-tree. By recursive inter-party, it is also ensured that spatially adjacent geographic entities are assigned to the same R-tree, reducing the cost of data retrieval and data exchange. The spatial index of each data block is constructed as follows:

(1) Before constructing the R-tree, assume that the degree of the R-tree is D. After MapReduce divides the spatial data, it already constitutes ordered data, so the spatial data can be inserted into the R-tree nodes in each group in the order of smallest to largest.

(2) For geographic entities, separate R-tree node insertion operations are performed so that new leaf nodes are generated and used to store the corresponding unique ID value and minimum outlier rectangle (id, MBR) for each group, returning (id, MBR) as a key-value pair. The unique identifier used by the parent node to locate its children is the id.

(3) Recursive processing of the previous step. Firstly, the minimum outer rectangle MBR is sorted, then the parent node is generated based on that sort and the geographic entity elements, and this step is repeated, stopping when a root node is finally generated.

This spatial index is built from the bottom up, and the R-tree is constructed based on the spatial distribution of geographical elements. If the nodes are adjacent in spatial distribution, then this index construction ensures that they are located on the same parent node, thus saving a lot of computing time in subsequent operations such as spatial analysis and improving the efficiency of spatial big data management. The pseudo-code for local index construction is shown in Algorithm 4.

**Algorithm 4** CreatRTree()

```
1: public class CreatRTree
2: Data d; // input data, a new leaf node
3: Long id:// Unique identifier of the geographic entity element
4: MBR r// Minimum outsourcing rectangle for geographic entity elements
5: public static Node RtreeCreat()
6: for (int i=0; i¡numRects; i++)
7:   d.getMBR(); // get the rectangular coordinates
8: r.insert(d); // get the minimum outer wrapping rectangle of the geographic entity
elements
9:
10: InsertNode(); //Insert a new leaf node
10: sortMbr(); // Sort the outsourced rectangle
11: BuildRTree(); // select generation to process until root node is generated
12:
13:
```

4.2. **Global index construction.** Due to the unusually large amount of spatial data data, all operations on delineated geographical data should go through the global index first, followed by the local index, which can improve the efficiency of data retrieval and optimise the query results.

For global indexes, which are accessed more frequently by users and have a smaller data volume than local indexes, the global index can be placed on the name node NameNode and store the relevant information. The only difference is that the R-Tree data node stores a storage path to each HDFS, rather than a pointer to a leaf node. The HDFS storage path Path and the MBR information of the local index R-Tree allows the overall index to be linked to the local index in a way that is more efficient and less time consuming when performing spatial data queries.

The global index needs to store the HDFS path information and the minimum outsourcing rectangle information, so the global index needs to be constructed according to the parallel Hilbert-R-tree method after the local index has been built.

## 5. Experimental results and analysis.

5.1. **Data pre-processing.** All experiments were implemented in a cloud Hadoop environment, using components such as HDFS, MapReduce and Hbase. The original trajectory data and the normalised trajectory data were stored on Hbase. The pre-processing of the vehicle trajectory data was based on the implementation of MapReduce, using PIG scripts for data cleaning, normalisation and trajectory segmentation. The data cleaning part mainly filters the data for abnormal values, null data and invalid data to ensure that the processed data is standardized and reasonable. The standardisation of the data is to adjust the storage format of the trajectories.

5.2. **Experimental environment.** The implementation of trajectory segmentation and spatial indexing relies on various components of the Hadoop environment, so the experimental base environment consists of a Hadoop cluster containing three Linux servers, each server being one node. the Hadoop set cluster consists of one Master node and two Slave nodes with CPUs of 24, 8 and 8 cores, respectively, and CPU main frequencies of 2GHz,

2.4GHz and 2.5GHz, and memory of 16GB, 8GB and 8GB, respectively. The Hadoop cluster consists of one Master node and two Slave nodes with 24-core, 8-core and 8-core CPUs and CPU main frequencies of 2GHz, 2.4GHz and 2.5GHz respectively, and 16GB, 8GB and 8GB of RAM respectively.

5.3. **Experimental data set.** The data used in this paper is derived from the license plate recognition data of Shanghai. The data contains the travel records of 1760776 vehicles for 13 days. To validate the proposed method, the dataset was divided into two time bounded segments, the first 7 days were used as training set to generate the threshold dataset and the last 6 days were used as test set to verify the correctness of the method. The statistical information of the experimental data is shown in Table 4. For the purpose

Table 4. Statistical information on experimental data.

| Data category | Number |
|---|---|
| Number of training set records | 9909235 |
| Number of vehicles in the training set | 1191990 |
| Number of test set records | 6741483 |
| Number of vehicles in the test set | 915734 |

of evaluation, only vehicles in the test set with more than 100 track points recorded were selected for the split.

5.4. **Evaluation indicators.** The experimental evaluation mainly considers the accuracy and coverage of the identification results.

Defining $R$ to denote the set of actual keypoints and $D$ to denote the set of keypoints annotated by the algorithm, then the accuracy rate can be expressed as

$$P_{acc} = \frac{R \cap D}{D} \times 100\% \tag{11}$$

The rate of detection of completeness can be expressed as

$$P_{rec} = \frac{R \cap D}{R} \times 100\% \tag{12}$$

5.5. **Analysis of experimental results.** The experimental results contain two parts, which contain the results of the index construction experiment and the results of the trajectory query experiment.

Index construction is performed for data before and after segmentation, as shown in Figure 4. When the amount of data is small, segmentation has a small impact on the construction time of the three indexing methods, but as the amount of data increases, the indexing construction time of the segmented trajectory data consumes less than that of the unsegmented trajectory data. All three indexing methods are in parallel computing mode. As shown in Figure 5, for the same data set, the R-tree index always takes better time to build than the Hashing tree, due to the addition of trip fields to the Hashing tree index. the Hilbert-R-tree index takes the fastest time to build and takes less time incrementally than the Hashing and R-trees as the data volume grows. As shown in Figure 6, for point queries, each of the three indexing methods was used and the Hilbert-R-tree index was found to be the best and the most efficient for queries. Similar results are shown in Figure 6 for the track segment query, using each of the three indexing methods, and it was found that the Hilbert-R index was the best, followed by the R-tree query, and the Hashing tree was the lowest.

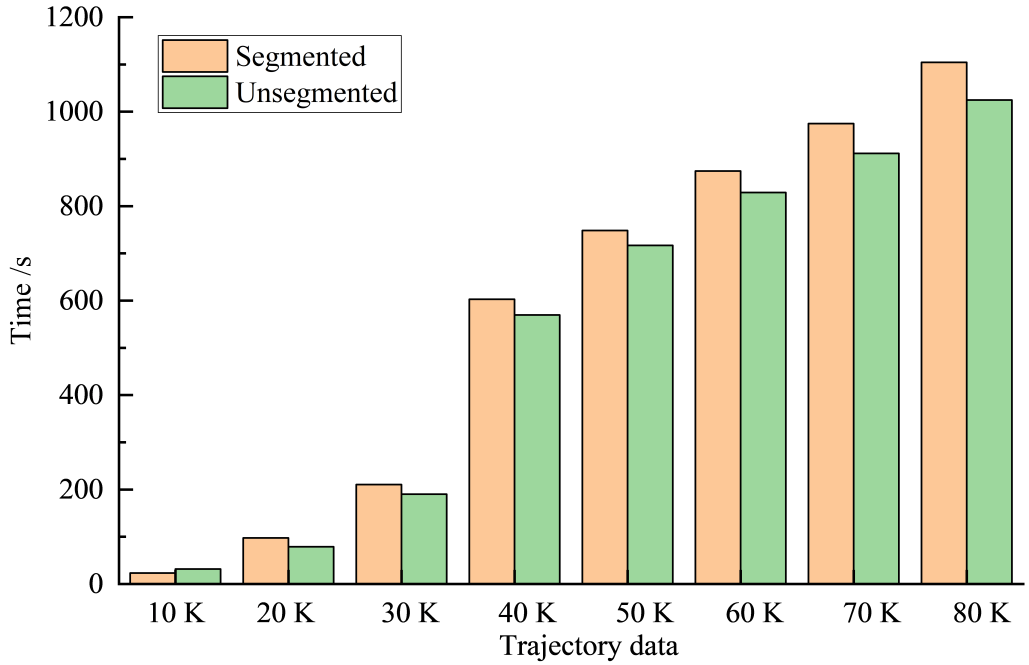


Figure 4. Impact of track segmentation on index construction time

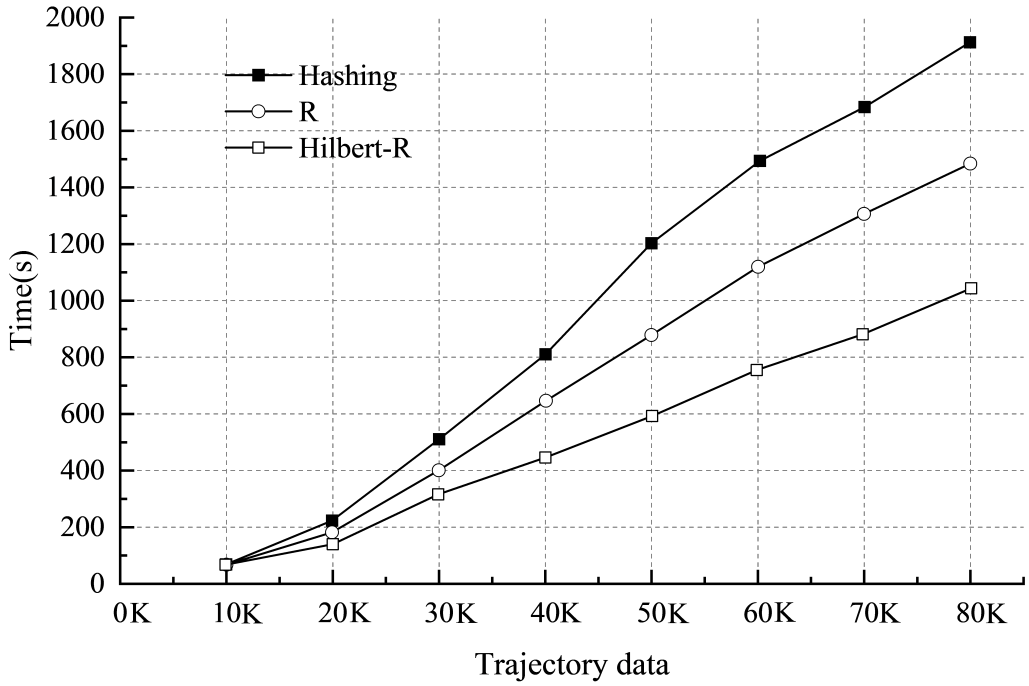


Figure 5. Time-consuming comparison of the three index construction methods

Finally, R-trees and Hilbert-R-trees were selected for a comparison of trajectory identification, as shown in Table 5. The Hilbert-R-tree based method is 20% higher than the R tree based method in terms of detection accuracy and 51% higher in terms of detection completeness. It seems that the method proposed in this paper is more effective for trajectory recognition of large data trips.

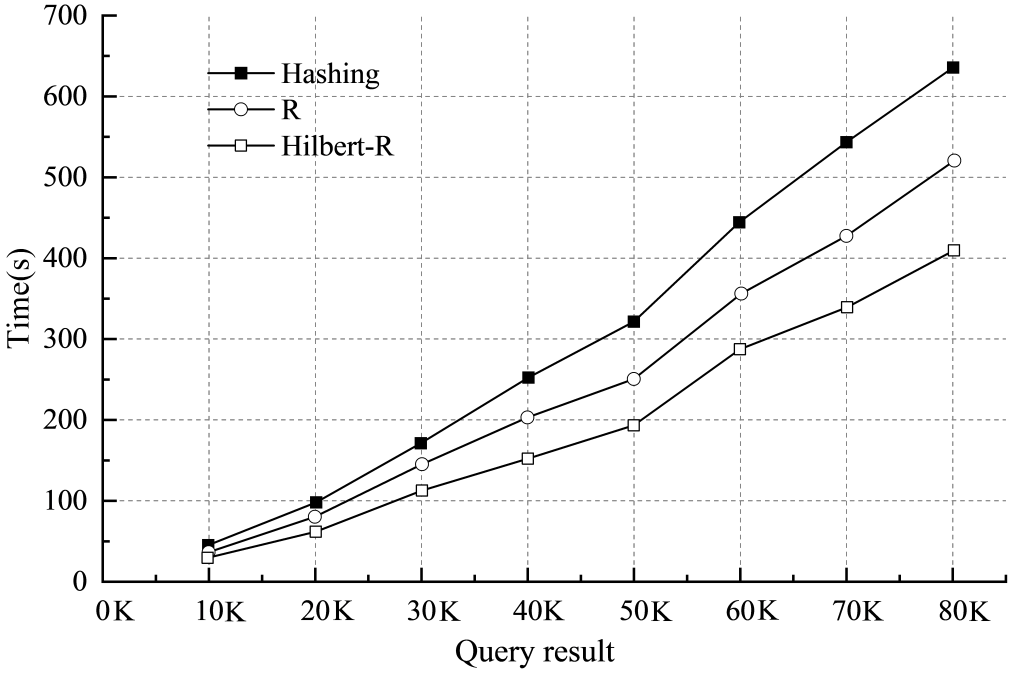


Figure 6. Point search experiment results

Table 5. Track Recognition Comparison Results.

| Spatial indexing algorithms | Check accuracy | Search completeness rate |
|---|---|---|
| Hilbert-R-trees | 82.169% | 89.539% |
| R-tree | 62.301% | 38.785% |

6. **Conclusion.** In this paper, we have conducted an in-depth study on the indexing method of trajectory data in a big data environment. In response to the fact that traditional serial techniques become inefficient in handling spatial queries, this paper proposes a Hilbert-R-tree construction algorithm based on a parallel computing model. The spatial data is sorted and partitioned in parallel using the characteristics of Hilbert space curve, thus well balancing the load of each host node in the data partitioning process. For the massive spatial data, due to the characteristics of uneven spatial distribution and data adjacency, it is necessary to divide it reasonably, so the global index and local index are constructed and the construction process is elaborated to further improve the efficiency of spatial data retrieval. The results show that the spatial data indexing in the distributed environment has good performance and can meet the demand for real-time query of big data of traffic trajectories. After the efficient indexing of spatial data, how to continue to use the Hadoop platform to perform a series of spatial analysis on it, such as network analysis and overlay analysis, so as to improve its efficiency, is the next direction that can be carried out to focus on research.

**Acknowledgement.** This article was supported by the 2022 Guangdong Provincial Higher Education Research Platforms and Projects (2022ZDZX1067)

## REFERENCES

[1] T.-Y. Wu, X. Guo, L. Yang, Q. Meng, and C.-M. Chen, "A Lightweight Authenticated Key Agreement Protocol Using Fog Nodes in Social Internet of Vehicles," *Mobile Information Systems*, vol. 2021, pp. 1-14, 2021.

[2] T.-Y. Wu, Z. Lee, L. Yang, and C.-M. Chen, "A Provably Secure Authentication and Key Exchange Protocol in Vehicular Ad Hoc Networks," *Security and Communication Networks*, vol. 2021, pp. 1-17, 2021.

[3] T.-Y. Wu, F. Kong, Q. Meng, S. Kumari, and C.-M. Chen, "Rotating behind security: an enhanced authentication protocol for IoT-enabled devices in distributed cloud computing architecture," *EURASIP Journal on Wireless Communications and Networking*, vol. 2023, no. 1, 36, 2023.

[4] T.-Y. Wu, L. Wang, X. Guo, Y.-C. Chen, and S.-C. Chu, "SAKAP: SGX-Based Authentication Key Agreement Protocol in IoT-Enabled Cloud Computing," *Sustainability*, vol. 14, no. 17, 11054, 2022.

[5] A. Fural, S. Kükrer, and S. Cürebal, "Geographical information systems based ecological risk analysis of metal accumulation in sediments of Kizcetepeler Dam Lake (Turkey)," *Ecological Indicators*, vol. 119, no. 19, 106784, 2020.

[6] D. Yha, D. Bca, T. C. Jing, and B. Yza, "Privacy-preserving point-of-interest recommendation based on geographical and social influence," *Information Sciences*, vol. 543, pp. 202-218, 2021.

[7] W. Gan, L. Chen, S. Wan, J. Chen, and C.-M. Chen, "Anomaly Rule Detection in Sequence Data," *IEEE Transactions on Knowledge and Data Engineering*, 2022. [Online]. Available: `https://doi.org/10.1109/TKDE.2021.3139086`

[8] C.-M. Chen, Z. Tie, E. K. Wang, M. K. Khan, S. Kumar, and S. Kumari, "Verifiable dynamic ranked search with forward privacy over encrypted cloud data," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2977-2991, 2021.

[9] C.-M. Chen, L. Chen, W. Gan, L. Qiu, and W. Ding, "Discovering high utility-occupancy patterns from uncertain data," *Information Sciences*, vol. 546, pp. 1208-1229, 2021.

[10] K. N. Qureshi, S. Din, and A. Ahmed, "Beaconless Traffic-Aware Geographical Routing Protocol for Intelligent Transportation System," *IEEE Access*, vol. 8, no. 2020, pp. 187671-187686, 2020.

[11] B. R. Senapati, P. M. Khilar, T. Dash, and R. R. Swain, "AI-assisted Emergency Healthcare using Vehicular Network and Support Vector Machine," *Wireless Personal Communications*, vol. 130, no. 3, pp. 1929-1962, 2023.

[12] Z. Wang, J. Wang, D. Yu, and K. Chen, "Groundwater potential assessment using GIS-based ensemble learning models in Guanzhong Basin, China," *Environmental Monitoring and Assessment*, vol. 195, no. 6, 690, 2023.

[13] N. Firouraghi, R. Bergquist, M. Fatima, A. Mohammadi, D. H. Hamer, M. R. Shirzadi, and B. Kiani, "High-risk spatiotemporal patterns of cutaneous leishmaniasis: a nationwide study in Iran from 2011 to 2020," *Infectious Diseases of Poverty*, vol. 12, no. 1, 49, 2023.

[14] S. Mackay, V. Ta, S. Dewez, and A. Körner, "Evidence-Based Practice in Psychosocial Oncology from the Perspective of Canadian Service Directors," *Current Oncology*, vol. 30, no. 4, pp. 3998-4020, 2023.

[15] A. Öztürk, A. U. Özcan, İ. Aytaş, G. Tuttu, D. Gülçin, J. Mongil-Manso, V. Rincón, and J. Velázquez, "Simulating with a combination of RUSLE GIS and sediment delivery ratio for soil restoration," *Environmental Monitoring and Assessment*, vol. 195, no. 6, 719, 2023.

[16] X. Liu, T. Zhang, Z. Tan, A. R. Warden, S. Li, E. Cheung, and X. Ding, "A Hashing-Based Framework for Enhancing Cluster Delineation of High-Dimensional Single-Cell Profiles," *Phenomics*, vol. 2, no. 5, pp. 323-335, 2022.

[17] W. Ding, C.-T. Lin, and W. Pedrycz, "Multiple Relevant Feature Ensemble Selection Based on Multilayer Co-Evolutionary Consensus MapReduce," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 425-439, 2020.

[18] M. E. Colosimo, M. W. Peterson, S. Mardis, and L. Hirschman, "Nephele: genotyping via complete composition vectors and MapReduce," *Source Code for Biology and Medicine*, vol. 6, 13, 2011.

[19] W. Zhu, G.-Q. Zhang, S. Tao, M. Sun, and L. Cui, "NEO: systematic non-lattice embedding of ontologies for comparing the subsumption relationship in SNOMED CT and in FMA using MapReduce," *AMIA Summits on Translational Science Proceedings*, vol. 2015, 216, 2015.

[20] R. Sinha, R. K. Pal, and R. K. De, "GenSeg and MR-GenSeg: A Novel Segmentation Algorithm and its Parallel MapReduce Based Approach for Identifying Genomic Regions With Copy Number Variations," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 1, pp. 443-454, 2022.

[21] E. Bāders, K. Jõgiste, D. Elferts, F. Vodde, A. Kiviste, S. Luguza, and Ā. Jansons, "Storm legacies shaping post-windthrow forest regeneration: learnings from spatial indices in unmanaged Norway spruce stands," *European Journal of Forest Research*, vol. 140, no. 4, pp. 819-833, 2021.

[22] A. Guo, J. Yang, X. Xiao, J. Xia, C. Jin, and X. Li, "Influences of urban spatial form on urban heat island effects at the community level in China," *Sustainable Cities and Society*, vol. 53, 101972, 2020.

[23] M. Hemati, M. Hasanlou, M. Mahdianpari, and F. Mohammadimanesh, "Iranian wetland inventory map at a spatial resolution of 10 m using Sentinel-1 and Sentinel-2 data on the Google Earth Engine cloud computing platform," *Environmental Monitoring and Assessment*, vol. 195, no. 5, 558, 2023.

[24] S. Ma, X. Fang, Y. Yao, J. Li, D. C. Morgan, Y. Xia, C. S. M. Kwok, M. C. K. Lo, D. M. D. Siu, K. K. Tsia, A. Yang, and J. W. K. Ho, "StarmapVis: An interactive and narrative visualization tool for single-cell and spatial data," *Computational and Structural Biotechnology Journal*, vol. 21, pp. 1598-1605, 2023.

[25] P. Du, X. Bai, K. Tan, Z. Xue, A. Samat, J. Xia, E. Li, H. Su, and W. Liu, "Advances of four machine learning methods for spatial data handling: A review," *Journal of Geovisualization and Spatial Analysis*, vol. 4, pp. 1-25, 2020.

[26] R. Bivand, "R packages for analyzing spatial data: a comparative case study with areal data," *Geographical Analysis*, vol. 54, no. 3, pp. 488-518, 2022.