Article

HGST: A Hilbert-GeoSOT Spatio-Temporal Meshing and Coding Method for Efficient Spatio-Temporal Range Query on Massive Trajectory Data

Hong Liu [1,2], Jining Yan [1,2,*], Jinlin Wang [3], Bo Chen [4], Meng Chen [5] and Xiaohui Huang [1,2]

1 School of Computer Science, China University of Geosciences, Wuhan 430074, China
2 Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan 430074, China
3 Xinjiang Key Laboratory of Mineral Resources and Digital Geology, Xinjiang Institute of Ecology and Geography, Chinese Academy of Sciences, Urumqi 830011, China
4 Institute of Space Science and Applied Technology, Harbin Institute of Technology, Shenzhen 518055, China
5 Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100088, China
* Correspondence: yanjn@cug.edu.cn

Abstract: In recent years, with the widespread use of location-aware handheld devices and the development of wireless networks, trajectory data have shown a trend of rapid growth in data volume and coverage, which has led to the prosperous development of location-based services (LBS). Spatio-temporal range query, as the basis of many services, remains a challenge in supporting efficient analysis and calculation of data, especially when large volumes of trajectory data have been accumulated. We propose a Hilbert-GeoSOT spatio-temporal meshing and coding method called HGST to improve the efficiency of spatio-temporal range queries on massive trajectory data. First, the method uses Hilbert to encode the grids obtained based on the GeoSOT space division model, and then constructs a unified time division standard to generate the space–time location identification of trajectory data. Second, this paper builds a novel spatio-temporal index to organize trajectory data, and designs an adaptive spatio-temporal scaling and coding method based on HGST to improve the query performance on indexed records. Finally, we implement a prototype system based on HBase and Spark, and develop a Spark-based algorithm to accelerate the spatio-temporal range query for huge trajectory data. Extensive experiments on a real taxi trajectory dataset demonstrate that HGST improves query efficiency levels by approximately 14.77% and 34.93% compared with GeoSOT-ST and GeoMesa at various spatial scales, respectively, and has better scalability under different data volumes.

Keywords: trajectory data; spatio-temporal range query; Hilbert-GeoSOT; index; Spark

check for updates

Citation: Liu, H.; Yan, J.; Wang, J.; Chen, B.; Chen, M.; Huang, X. HGST: A Hilbert-GeoSOT Spatio-Temporal Meshing and Coding Method for Efficient Spatio-Temporal Range Query on Massive Trajectory Data. *ISPRS Int. J. Geo-Inf.* **2023**, *12*, 113. https://doi.org/10.3390/ijgi12030113

Academic Editors: Hartwig H. Hochmair and Wolfgang Kainz

Received: 3 January 2023
Revised: 25 February 2023
Accepted: 27 February 2023
Published: 7 March 2023

1. Introduction

With the widespread use of GPS-equipped mobile devices and the popularity of mapping services, massive trajectory data from various moving objects (e.g., people, vehicles, and animals) are continuously growing at high speed [1]. Generally, a trajectory is represented as a timestamped geographic location sequence, and the original trajectory data have been enriched with various semantic information [2]. Trajectory data and their analysis may benefit many real-life applications, including security [3], trajectory prediction [4], and urban computing [5], to name just a few. As a result, researchers have devoted their energy to trajectory analysis, leading to various trajectory data services. The spatio-temporal query is the basis of many services, and its processing efficiency is a key factor for application utility [6]. Therefore, providing efficient spatio-temporal query methods is one of the research hot spots for trajectory data mining.

Before mining trajectory data, different samples of trajectories or different parts of a trajectory must be accessed multiple times. This stage is called trajectory data indexing and retrieving [7], and it forms the basic step of many trajectory data mining tasks. To extract urban mobility dynamics, the mobility dynamics network is usually first generated by spatio-temporal analysis of trajectories [8]. For the study of public transit accessibility measures, it is usually necessary to account for normative geospatial constraints and temporal constraints [9]. For human trajectory forecasting, most practices usually select some track segments of adjacent spatio-temporal units to model the structure of human movement behavior [10]. Obtaining trajectories in a specific space and time period is crucial for further research in these applications. The problem is that a large amount of data leads to a long query time, which reduces the efficiency of application mining data and providing corresponding services, especially for online applications that need real-time mining trajectory data (e.g., detecting traffic anomalies) [11]. Since the establishment and use of reasonable indexes can improve query efficiency to some extent, we can design appropriate indexes to solve these problems and leave more time for subsequent data mining and analysis.

At present, the existing trajectory index construction methods are mainly divided into data-driven approaches and space-driven approaches. The data-driven approaches revolve around data-centric, dynamically generated index structures along with the data import process. Data-driven approaches usually have an R-tree-like structure, such as a 3D R-tree [12], which converts the query problem into a three-dimensional query cube by constructing spatio-temporal R-tree to build the index. However, the query efficiency of this method decreases significantly with the increase in data volume. TB-tree [13] and SEB-tree [14] alleviate this problem to some extent, but as the storage time of trajectory data increases, the overlap problem between index boxes still exists. Nidzwetzki [15] designs a secondary index, in which the global index uses a kd tree to map which node stores data, and the local index uses the R-tree to locate the partition of each data item in a single node. These approaches use the minimum bounding box (MBB) of an individual trajectory as an indicator of spatial variation. When the amount of data increases rapidly, there is lots of overlap and coverage in the MBB, and the insertion of R-tree is relatively large compared with the cost of node splitting, which all lead to excessive index maintenance overhead and result in significant performance degradation.

The main idea of space-driven approaches is to divide the geographical space into grids in advance, and then index the contents falling into each grid according to time to achieve the purpose of indexing the time and space dimensions. Grid coding generally adopts space-filling curve dimensionality reduction technology. There are two space-filling curves that can reduce the dimension of multi-dimensional data by mapping data to points in the curve that are widely used, namely the Z curve and the Hilbert curve. The open-source indexing tool GeoMesa [16] uses the Geohash implementation obtained by Base32 encoding of the Z-curve, and then intersperses Geohash and timestamp into the index alternately, which can index spatio-temporal trajectory data. Qian [17] proposes to extend the GeoSOT space subdivision to realize a space–time subdivision scheme. Li [18] proposes the TrajMesa storage engine based on GeoMesa, which extends the Z2T and XZ2T indexing methods to solve the problem of mismatch between time and space dimensions. They are basically extensions of the Z-Order curve, leading to poorer data locality preservation than the Hilbert curve [19]. Recently, some recent works have accomplished the encoding of spatial partitions based on Hilbert space-filling curves. Jiang [20] designed a new multi-level grid index structure based on Hilbert, which can improve spatial query efficiency. Lei [21] proposed the global multi-scale grid integer coding model based on the Hilbert curve, which improves the query efficiency of earth observation data. Guo [22] and Huang [23] combine the Hilbert curve with Geohash and GeoSOT, respectively, to complete the query of spatial objects. However, these approaches have not yet been extended to spatio-temporal indexing mechanisms. Wu [24] proposed a spatio-temporal indexing method using Hilbert curve coding based on the GeoHash grid. Wang [25] adopts the Hilbert curve to design a subspace partition algorithm to support multi-dimensional queries based on the idea of a secondary index. However, most of them do not consider the problem of low query efficiency when spatio-temporal range queries are imbalanced in time and space dimensions.

Inspired by these observations, in this study, we adopt a space-driven approach to achieve rapid spatio-temporal query for large-scale trajectory data. There are three ideas that we considered to design such a method. The first is to consider the Hilbert curve with superior clustering properties [26], but the Hilbert curve requires the entire space to be subdivided into $2^n \times 2^n$ subspaces. Therefore, we use the Geographical coordinates Subdividing grid with one dimension integral coding on 2^n tree (GeoSOT) to perfectly subdivide the space. In addition, we found that GeoSOT-ST selects a fixed period and performs binary recursive partition on it, which results in the time resolution obtained by the partition being limited by the initially selected period length, and the data beyond this period not being able to be indexed. Our idea is to divide time into disjoint periods and perform an undifferentiated recursive binary partition for each period, where the period length can be customized. As a result, depending on the length of the period chosen, we can obtain time scales of various resolutions. In this way, we construct a unified time division standard, which can obtain the time identification of any time under a custom time resolution. Consequently, a Hilbert-GeoSOT spatio-temporal meshing and coding method called HGST is proposed.

Second, the indexing method based on the partition idea divides space–time into spatio-temporal grids of equal size, and the query range is usually mapped to several grids when performing a spatio-temporal range query. When the grid is small, the query may map into many small grids, rendering retrieval too trivial; when the grid is large, it may involve too much data that do not intersect with the query area. To solve this problem, GeoSOT-ST provides a query approach with variable subdivision levels, which generally performs well when the temporal range scale is equal to the spatial range scale. However, we found that in some cases, there is an imbalance in time and space dimension scales for spatio-temporal range queries. For example, when querying for a long time range and a small spatial range, or the query time is short and the spatial range is large. Therefore, we design an adaptive spatio-temporal scaling and coding method to determine the optimal subdivision level according to the query range, and design a query code merging strategy to further reduce the complexity of the spatio-temporal range query.

The third is that the amount of trajectories exceeds the storage and processing capability of a single machine generally, and it calls for large-scale trajectory processing in distributed environments [27]. Note that the "filter-and-refinement" pattern usually be followed in the spatio-temporal range query of the trajectory [28]. Therefore, in the trajectory spatio-temporal range query, we introduce Spark, a distributed in-memory computing system, to first obtain the candidate results output by the filter stage and load them into the memory, and then complete the refinement stage of the query in a distributed manner. Finally, we implement our prototype system based on HBase and Spark. In our prototype system, we developed a Spark-based algorithm to speed up the spatio-temporal range query process.

The contributions of this paper are summarized as follows.

- We propose a spatio-temporal meshing and coding method called HGST. It uses Hilbert instead of default Z curves for spatial grid coding, and constructs a unified time division standard to obtain the time identification of any time under a custom time resolution. This method provides a novel spatio-temporal index for trajectory data;
- Based on HGST, we design an adaptive spatio-temporal scaling and coding method to determine the optimal subdivision level depending on the query range, and also propose a query code merging strategy to further reduce the complexity of spatio-temporal range queries;
- We implement a prototype system on top of HBase and Spark, and develop an efficient algorithm implementing the Spark paradigm to accelerate the parallel execution of spatio-temporal range queries.

The rest of this paper is organized as follows. Section 2 reviews the related works on indexing methods supporting spatio-temporal range queries on trajectory data. Section 3 introduces several technique backgrounds, including the problem statement and GeoSOT subdivision model. More details of our proposed spatio-temporal indexing method and the execution process of spatio-temporal range queries are presented in Section 4. We present the evaluation results in Section 5. Section 6 presents discussions on experimental results and points out our future works. Finally, Section 7 concludes the remarks of this article.

2. Related Work

Recently, a number of remarkable relational database-based (e.g., HERMES [29], MobilityDB [30], etc.), Hadoop-based (e.g., Hadoop-GIS [31], HadoopTrajectory [32], etc.), Spark-based (e.g., DITA [27], Dragoon [11], etc.), and NoSQL (Not Only SQL)-based systems are proposed for trajectory management or analysis. Relational database-based systems usually extend data types and spatio-temporal operators to support the representation and analysis of moving objects. These Hadoop and Spark extensions generally use a global index in the master node to distribute the relevant data to worker nodes, thereby reducing data distribution overhead. However, the scalability of relational database-based systems is insufficient, and they always encounter bottlenecks in big data scenarios. Hadoop-based systems store the intermediate results of calculations on disk, and even a single job requires multiple disks I/Os, so there may face an efficiency problem. Spark-based systems typically load all data into memory, which requires high-performance clusters with much memory. In addition, they need to scan huge indexes for each spatio-temporal request, which is costly.

Many NoSQL-based representatives focus on building efficient spatio-temporal indexes on the highly scalable NoSQL, with millions of updates per second, to realize the purpose of discovering interesting data from massive trajectory data [33]. The main idea behind these works is to inject the spatio-temporal property of objects in 3D space (time and 2D space) into a part of the row key in the 1D space of HBase. In this way, trajectories with adjacent space–time positions are also stored adjacently in the physical positions. There are two approaches, including the data-driven approaches and the space-driven approaches, are used to accomplish this task.

2.1. Data-Driven Approaches

Data-driven spatio-temporal indexing techniques mainly focus on the deformation and extension of two-dimensional spatial index structures. They use individual trajectories' minimum bounding boxes (MBBs) as an indicator. These indexes generally adopt hierarchical, tree-like data structures (e.g., R-tree and its variants [12–14]) to recursively partition spatial objects into groups. When the amount of data increases rapidly, there is a lot of overlap and coverage between MBBs, and the index performance and query efficiency are significantly reduced [34]. Based on the R-tree, Hilbert R-tree [35] gathers adjacent data rectangles in the Hilbert curve to minimize the perimeter of the generated MBB. Although the query efficiency of Hilbert R-tree is improved compared with R-tree, it still does not simplify the complex construction process of tree index structure, and the query efficiency decreases when the data are updated frequently. These indexes' structure based on R-tree is relatively complex, and the insertion and node splitting cost of the R-tree are relatively high, so it is necessary to dynamically adjust the index structure.

2.2. Space-Driven Approaches

The space-based approaches divide the entire space into multiple grids of equal size, and use space-filling curves to transform the discrete grids in 2D space into a number of spatial codes in 1D space. Spatial objects are represented by spatial codes representing discrete grids. These indexes divide the space in advance, so that the indexes do not need to be updated when moving objects are constantly moving in the same spatial grid. In the face of high-frequency concurrent operations, space-based indexes generally perform better than R-tree family indexes [36].

Such indexes have been widely adopted in recent years due to their simplicity and mainly include quad-tree [37] and grid indexes [38]. Using the space-first strategy, the geospatial space is first constructed based on the quad-tree. The geographic hash code, i.e., the GeoHash value, is designed for spatial regions of different levels. Then, GeoMesa [16] uses the Geohash implementation obtained by Base32 encoding of the Z-curve, and then intersperses Geohash and timestamp alternately to form a spatio-temporal composite index. The ST-Hash [36] extends the widely used GeoHash to encode the longitude, latitude, and time attributes uniformly, so that the data with close geographical location and close time are physically stored close to each other. GeoSOT-ST extends the GeoSOT spatial division scheme to the time dimension to form a global spatio-temporal subdivision scheme [39,40]. Li [18] proposes the TrajMesa [41] storage engine based on GeoMesa, which extends the Z2T and XZ2T indexing methods to solve the problem of mismatch between time and space dimensions. These methods use a grid, and the Z-Order space-filling curve converts multi-dimensional data to a 1D linear key.However, the Z-Order curve frequently occurred coding mutations at certain locations, which may cause false positives in range queries [42]. GCOTraj [43] divides a large spatio-temporal data space into multi-dimensional grid cells and orders these grid cells in two different ways. The index performs well when handling large-scale spatial range queries. However, its performance is poor when the time query range is also large, because the time index is based on timestamps, and thus, only the start and end timestamps of each page are stored.

A recent work similar to ours is GeoSOT-ST. There are mainly three differences between it and our work. Firstly, in terms of spatio-temporal index, we adopt the Hilbert curve achieves better clustering than the Z-Order curve. We also construct a unified time division standard, which can obtain the time identification of any time under a custom time resolution. Secondly, we consider that in some cases, the spatio-temporal query is unbalanced in time and space dimension scales. Therefore, we designed an adaptive spatio-temporal scaling and coding method to determine the optimal subdivision level according to the query range, and designed a query code merging strategy to further reduce the complexity of the spatio-temporal query. Thirdly, we developed a Spark-based algorithm to speed up the spatio-temporal range query process on large-scale trajectory data.

3. Preliminary

This section provides the related formalized description. Then, we introduce the geographical grid division model GeoSOT in detail.

3.1. Problem Formulation

Trajectory. The definition of the trajectory was mentioned in Section 1. In a nutshell, a trajectory T is actually a finite sequence of timestamped locations, which is specifically formulated as follows.

Definition 1. *The mathematical formula for trajectory can usually be expressed as a series of trajectory points arranged in chronological order.* $T = \{P_1, P_2, P_3, \cdots, P_n\}$, $\forall P_i = \{x, y, a, t\}$; (x, y) *is the trajectory point coordinates, a is the attribute value of the trajectory point, and t is the time at which trajectory point was collected.*

Trajectory spatio-temporal range query. Given a region of time and a region of space, return all trajectories in these two regions. Spatio-temporal range query can be used in many urban applications, e.g., traffic rule identification [44].

Definition 2. *Given a trajectory dataset* $D = \{P_i | i = 1, 2, \ldots, n\}$*, which contains n trajectory points, each trajectory point has at least three dimensions, i.e., the latitude, the longitude, and the time information. Given a rectangular spatial range* $S = (lat_{min}, lon_{min}, lat_{max}, lon_{max})$*, and a temporal range* $T = (t_{start}, t_{end})$*, a spatio-temporal range query aims to find all trajectory points* $r \in D$*, where r locates in both the spatial range S and the temporal range T. Spatio-temporal range queries are of great use in many location-based services. For example, we can use a spatio-temporal range query to answer questions such as "What people or vehicles have passed by in high-risk areas of the COVID-19 outbreak in a specific temporal range?"*

3.2. GeoSOT

GeoSOT takes the intersection of the prime meridian and the equator as the central point, and recursively divides the earth's surface into four grid cells. The GeoSOT global subdivision model is a spatial indexing model based on spatial partitioning [45]. The core idea of the model is to logically expand the geographical space three times. First, the original surface space is expanded from $180^\circ \times 360^\circ$ to $512^\circ \times 512^\circ$; then, 1° is expanded from $60'$ to $64'$, and finally, $1'$ is expanded from $60''$ to $64''$. Since it ensures the division of whole degrees, minutes, and seconds, the result is a one-dimensional integral grid code on the 2^n tree [46].

Each GeoSOT code corresponds to a unique grid cell. For example, "001" represents a grid cell with a spatial range of $64^\circ \sim 128^\circ$ E and $0^\circ \sim 64^\circ$ N, the length value of the string is the level of grid cells. Note that the grid cells represented by "0010", "0011", "0012", and "0013" are all within the surface of "001" due to the upper and lower grids having inclusion relations. Meanwhile, these sub-grids also have the same characteristics.

4. Method

In this section, we first give an overview of our proposed method in Section 4.1. Then, the spatio-temporal meshing and coding method HGST is introduced in Section 4.2. Other details about the index construction and the spatio-temporal range query are presented in Sections 4.3 and 4.4, respectively.

4.1. Overview

Figure 1 depicts an overview of the proposed method. We designed a key-value-based index model for each trajectory point. The main idea of the index model is to use our proposed Hilbert-GeoSOT spatio-temporal (HGST) meshing and coding as the prefix of the keywords. In this way, the spatio-temporal proximity of trajectory data in 3D space is preserved. To solve the problem of low query efficiency caused by the imbalance of time and space dimension scales in spatio-temporal range queries. We propose an adaptive spatio-temporal scaling and coding method. This method generates the query codes at a suitable subdivision level, which is determined by the time and space scope of the query. After that, a query code merging strategy is proposed to reduce the complexity of spatio-temporal range queries.

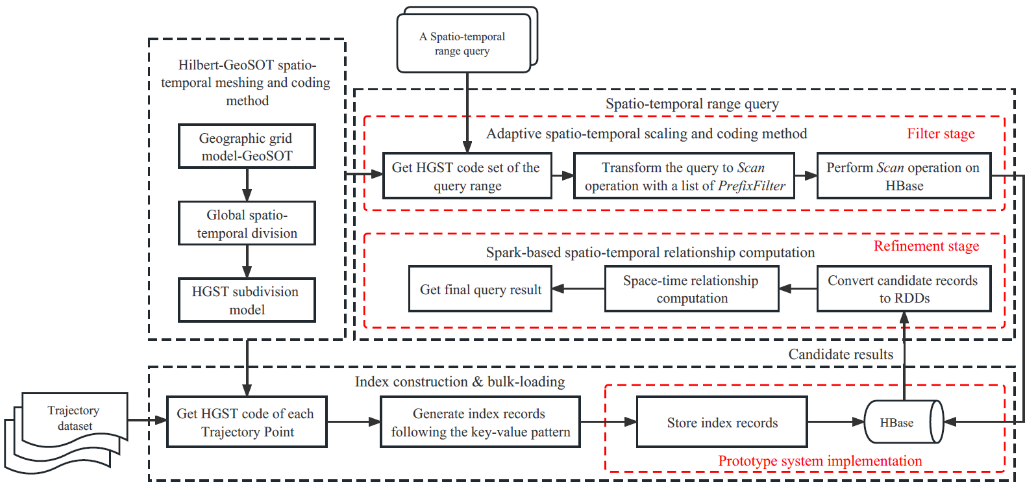

A Spatio-temporal range query
Hilbert-GeoSOT spatio-temporal meshing and coding method
Geographic grid model-GeoSOT
Global spatio-temporal division
HGST subdivision model
Spatio-temporal range query
Adaptive spatio-temporal scaling and coding method
Filter stage
Get HGST code set of the query range
Transform the query to Scan operation with a list of PrefixFilter
Perform Scan operation on HBase
Spark-based spatio-temporal relationship computation
Refinement stage
Get final query result
Space-time relationship computation
Convert candidate records to RDDs
Candidate results
Index construction & bulk-loading
Trajectory dataset
Get HGST code of each Trajectory Point
Generate index records following the key-value pattern
Store index records
HBase
Prototype system implementation

Figure 1. Overview of the proposed method (the lower part shows the process of index construction for trajectory, and the upper right part shows the process of spatio-temporal range query).

We also implement the process of a spatio-temporal range query on our proposed indexing model following the "filter-and-refinement" pattern. In the refinement stage, we complete the spatio-temporal relation calculation based on Spark, a distributed in-memory computing technology, to accelerate the spatio-temporal range query process on large-scale trajectory data.

4.2. HGST Subdivision Model

HGST extends the GeoSOT spatial partitioning model to three-dimensional space–time. In the space dimension, Hilbert replaces the default Z-Order filling curve to ensure better spatial proximity. In the time dimension, we draw on the idea of GeoSOT space division to construct a unified time division standard, which divides infinite time into multi-scale time periods. We then integrate spatial and temporal information and use one code to identify a specific spatio-temporal scope. Finally, the characteristics of the generated code are introduced. Next, we describe these four parts in detail.

4.2.1. Spatial Encoding

GeoSOT is a global subdivision model that uses the quad-tree method to subdivide the entire earth's surface into many discrete, seamless, and hierarchical grids [23]. These grids are sequentially connected according to the Z-Order curve to obtain a one-dimensional code to achieve dimensionality reduction. We consider Hilbert to replace the default Z-Order space-filling curve. First, the Hilbert curve has superior clustering properties; that is, adjacent grids have closer codes. Second, the Hilbert curve requires subdividing the entire space into $2^n \times 2^n$ subspaces. Therefore, it can be perfectly combined with GeoSOT, the Geographical coordinates Subdividing grid with one dimension integral coding on 2^n tree, to subdivide the space.

The spatial subdivision and coding of HGST are shown in Figure 2. For example, the latitude and longitude coordinates of a trajectory point are 114°23′ E and 30°40′ N. Its spatial coding is 310 when the grid is divided at the 3rd level (if the division continues to the 9th level, the spatial coding is 310030031).

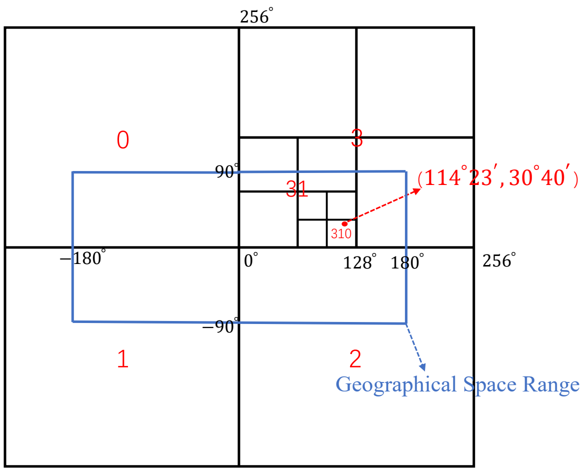


Figure 2. The diagram of spatial subdivision and coding of HGST.

4.2.2. Temporal Subdivision and Encoding

The division of HGST in the time dimension refers to the division of space by GeoSOT. However, unlike geographic space, which can be determined to be 180° × 360° in size, time is one-dimensional and infinitely long.

Our approach is to count from *RefTime* the reference time (e.g., 1970-01-01T00:00:00Z), and break time into disjoint periods. Each period is then divided recursively. We use the time period code *TPCode* to identify different subdivided periods, and *TPCode* can be obtained according to the calculation with Equation (1) (where *PeriodLen* is the time span of a period). In addition, *TimeCode* is used to distinguish different times within a subdivision period.

$$TPCode = \lfloor (time - RefTime) \div PeriodLen \rfloor \tag{1}$$

Specifically, if we set *PeriodLen* as 32 years, five recursive bisections can obtain the 1-year time slice for each subdivided period. Then, we perform four virtual extensions, and carry out strict recursive bisections for each one-year time slice. The first expansion extends the original year to 16 months, and the 16-month time slice can be divided into a 1-month time slice by four recursive bisections. The second expansion extends the original month to 32 days, and the 32-day time slice can be divided into a 1-day time slice by five recursive bisections. The third expansion extends the original day to 32 h, and the 32-h time slice can then be divided into a 1-h time slice by five recursive bisections. Finally, in a similar way, we obtain a time slice with a resolution of one minute by extending the original one hour to 64 min and performing six recursive bisections.

In this way, when *PeriodLen* is 32 years, any UTC will be divided into a minimum of one-minute time segments after four extensions and 25 bisections while ensuring that the whole month, day, hour, and minute are divided. The scales of spatio-temporal subdivision corresponding to different levels are shown in Table 1.

Table 1. The scales of spatio-temporal subdivision corresponding to different levels (if the *PeriodLen* is 32 years).

Level	Scale	Level	Scale	Level	Scale	Level	Scale
0	32 year 512°	7	4 month 4°	14	1 day 2′	21	16 min 1″
1	16 year 256°	8	2 month 2°	15	16 h 1′	22	8 min 1/2″
2	8 year 128°	9	1 month 1°	16	8 h 32″	23	4 min 1/4″
3	4 year 64°	10	16 day 32′	17	4 h 16″	24	2 min 1/8″
4	2 year 32°	11	8 day 16′	18	2 h 8″	25	1 min 1/16″
5	1 year 16°	12	4 day 8′	19	1 h 4″		
6	8 month 8°	13	2 day 4′	20	32 min 2″		

According to the above, the time dimension encoding is composed of *TPCode* and *TimeCode*. The calculation of *TimeCode* is shown in Figure 3. The root node of the tree represents the divided zone, a certain subdivision period (e.g., 2002∼2033); we recursively divide the node into two equal-sized child nodes. The binary bit is then assigned: if the position is on the left, assign 0; otherwise, assign 1. For example, at the moment of 20 August 2021 08:05:00, the time dimension code at the 9th level is 1-100111000 ("-" means a concatenation operation).

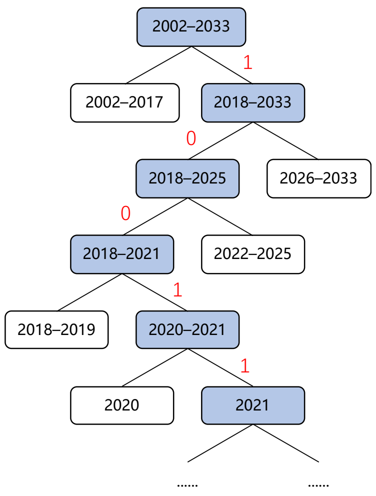


Figure 3. The diagram of time tree.

4.2.3. Calculate the HGST Spatio-Temporal Code

In this step, we integrate spatial coding with temporal coding to obtain the HGST spatio-temporal code (HGSTCode) and explain what it represents.

Now we obtain the spatial coding 310030031 and the time dimension coding 1-100111000 (*TPCode* is 1, *TimeCode* is 100111000). First, we decompose the spatial coding into two binary codes, $xCode$, and $yCode$, which are uniquely identified and determined by the spatial coordinate. For example, the code 310030031 is decomposed into $xCode$ 100010010 and $yCode$ 110010011. Second, cross bit by bit in the order of $xCode$, $yCode$, $TimeCode$, and the result is 111 010 000 001 111 001 000 110 010. Finally, convert the result to octal code 720171062, and then combine it with $TPCode$ to form the final HGSTCode 1-720171062.

To date, the trajectory point $P_i = \{114°23' \text{ E}, 30°40' \text{ N}, a, 20 \text{ August } 2021\ 08{:}05{:}00\}$ can be encoded as 1-720171062 at the 9th HGST subdivision level. Referring to Table 1, the spatial resolution of the HGSTCode is 1°, and the temporal resolution is one month.

4.2.4. Characteristics of the HGSTCode

The data structure represented by each HGSTCode can be viewed as a three-dimensional space–cube, as shown in Figure 4, where the x and y dimensions represent space, and the t dimension represents time.

Each HGSTCode corresponds to a unique space–time cube, representing a specific space–time range. Moreover, any space–time cube can be further divided to obtain eight child space–time cubes, which corresponding HGSTCode order is shown in Figure 5.

In addition, the HGSTCode corresponding to the parent space–time cube is the prefix of the HGSTCode corresponding to the child space–time cube. Therefore, the HGSTCode can directly support spatio-temporal range queries by performing simple string operations, and most candidates are filtered out before performing delicate spatial or temporal calculations.

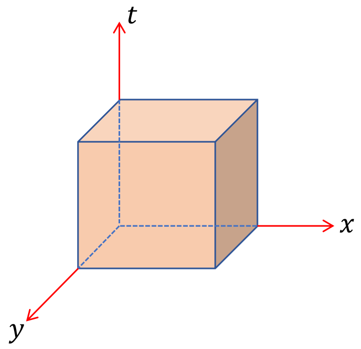


Figure 4. The data structure represented by HGSTCode.

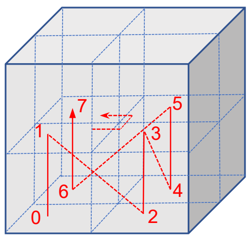


Figure 5. The corresponding coding order of child space–time cubes.

4.3. Index Construction

The tremendously increasing volume of spatio-temporal data presents challenges in storing, managing, processing, and analyzing data [47]. Building efficient spatio-temporal indexes on trajectories can significantly facilitate query processing in trajectory databases [28]. If we index the entire track, we may need to update the index frequently, because the track is dynamically updated in real time. Therefore, we consider indexing track points based on the HGST subdivision model.

Firstly, note that the HGSTCode represents a spatio-temporal cube, which can identify a specific space–time scope $stScope$. We then maintain an index for the track points in $stScope$ that belong to the same track; when the tracked moving object generates a new track point that "exceeds" $stScope$, a new index is added for the point. The size of $stScope$ can be customized according to the required resolution, and if $stScope$ is small enough, it can accurately describe the motion state of a trajectory. Figure 6 is a schematic of our indexing method. With this approach, we first create the 3D grid-based spatio-temporal index, and then incrementally add data without causing any change to the index structure.

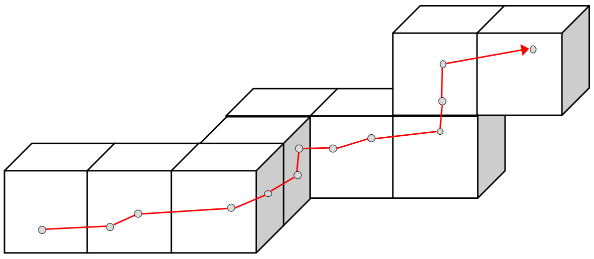


Figure 6. Index trajectory based on HGST subsection model (the dots are the collected trajectory points, and the red line is the formed trajectory, which can be enveloped by several spatio-temporal cubes).

The indexing records are organized following the key-value model. The key is a combination of the HGSTCode of the trajectory point and the unique identifier UUID. The value is other attribute information of the trajectory point (e.g., TrajectoryID, longitude, latitude, timestamp, speed, and direction). The HGSTCode is used as the prefix to preserve the spatio-temporal proximity. That is, adjacent trajectory points in 3D space are stored adjacently in the physical location. Thus, the spatio-temporal range query performance is improved due to the number of disks seeking operations reduced. In this paper, we choose HBase, a famous key-value-based distributed storage system based on HDFS (Hadoop Distributed File System) [48], as the prototype of our storage system for spatio-temporal indexing records.

The key of the above key-value model is used as the row key of the HBase table, as shown in Figure 7.

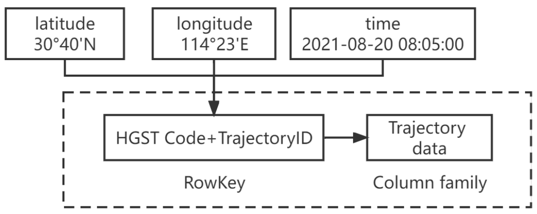


Figure 7. The diagram of key-value store model.

4.4. Spatio-Temporal Range Query

The trajectory spatio-temporal range query takes a spatial range and a time interval as input, and the goal is to find all trajectory objects within the scope. Figure 8 shows the general framework of the "filter-and-refinement" pattern to the spatio-temporal range query processing. The filtering stage uses relatively low computation cost to find a set of candidate trajectories that are likely to be the results; the refinement stage then identifies the final query result from the small set of candidates.

4.4.1. Filter Stage

In the filtering stage, we abstract the time range and space range input by the user into a space–time cube $Query^{st}$. The problem is how to find all the trajectory data in the $Query^{st}$. Based on the HGST subdivision model, we know that $Query^{st}$ can be completely covered by several space–time cubes at a certain subdivision level. Therefore, we only need to focus on querying these space–time cubes to filter out most of the data that are not in $Query^{st}$. Note that each space–time cube corresponds to a unique HGSTCode, which realizes the transformation of the 3D space range query of trajectory data into a 1D code query in the HGST index record table.

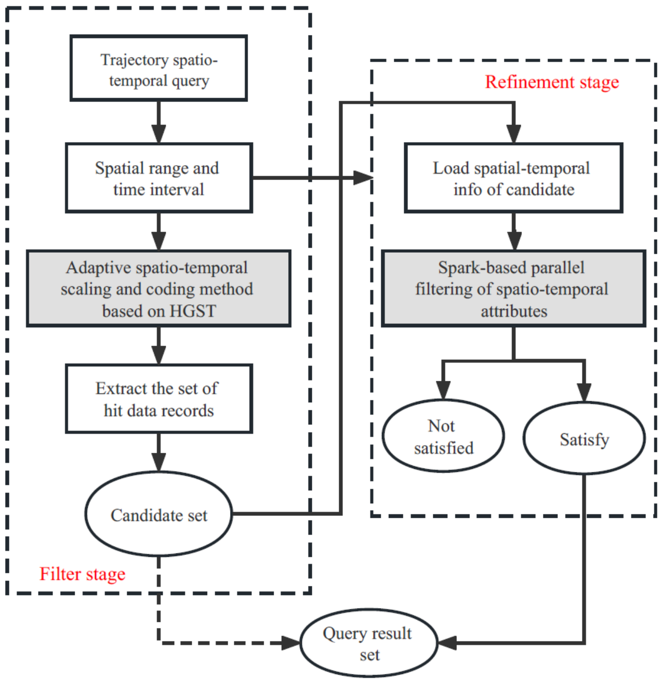

Trajectory spatio-temporal query
Spatial range and time interval
Adaptive spatio-temporal scaling and coding method based on HGST
Extract the set of hit data records
Candidate set
Filter stage
Refinement stage
Load spatial-temporal info of candidate
Spark-based parallel filtering of spatio-temporal attributes
Not satisfied
Satisfy
Query result set

Figure 8. The general framework of spatio-temporal range query.

The purpose of the filtering stage is to convert the query conditions into HGSTCode; the key is to determine the subdivision level. As shown in Figure 9a, if the level is too high, the query scope will be mapped to many small grids, making the filtering conditions too complicated and reducing the query efficiency. When the level is too low, the query process will return abundant "fake data" that are not within the query scope (Figure 9b), which leads to more tasks in the subsequent refinement stage.

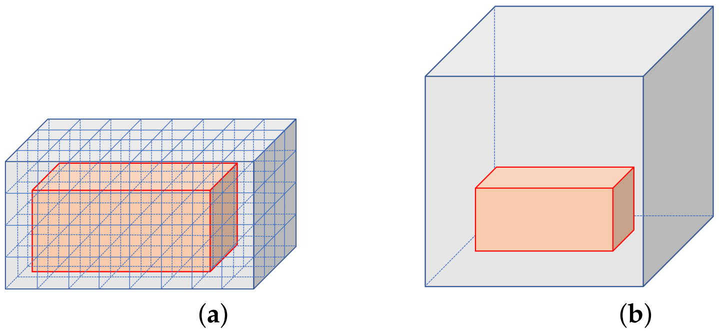


Figure 9. Space–time cubes covered by grid of different sizes. (**a**) Schematic diagram of a query at a high subdivision level. (**b**) Schematic diagram of a query at a low subdivision level.

To solve the problem, we designed an adaptive spatio-temporal scaling and coding method to determine the optimal subdivision level according to the query scope. It contains the following three steps. (1) Calculate the optimal subdivision level $optimalLevel$; (2) Find several space–time cubes at the $optimalLevel$th level that intersect, contain, or be contained by the $Query^{st}$; (3) Merge query codes.

Referring to Table 1, we assume that the temporal resolution is $tScale_N$, and the spatial resolution is $sScale_N$ when the subdivision level is N. $optimalLevel$ is obtained according to the calculation with Equation (2). Dis_S is the larger in the length and width of the Minimum Bounding Rectangle (MBR) corresponding to the query space range, and Dis_T is the query time interval value. For example, if the spatial range of the spatio-temporal query is $3' \times 3'$, and the time span is 3 h. When the subdivision level is not greater than 13, the spatial scale corresponding to the space–time cube is greater than $3'$; when the subdivision level is not greater than 17, the temporal scale corresponding to the space–time cube is greater than 3 h. Finally, the optimal subdivision level is 17.

After the $optimalLevel$ is determined, the next step is to find several HGST space–time cubes that can cover $Query^{st}$ at the $optimalLevel$th level. For its implementation, refer to Algorithm 1. It is mainly divided into three cases: if $sLevel$ is equal to $tLevel$ ($sLevel$ and $tLevel$ are calculated by Equation (2)), similar to the query strategy of GeoSOT-ST, it only needs to calculate the HGSTCodes of the 8 vertices in the query range, and then 1, 2, 4 or 8 space–time cubes can be found. If $sLevel$ is greater than $tLevel$, it indicates that it is a long time and small spatial range query; we then segment the query time according to $tScale_{sLevel}$ (Figure 10a). If $sLevel$ is less than $tLevel$, it means that it is a short time and large spatial range query; we then partition the query space according to $sScale_{tLevel}$ (Figure 10b). To date, we can obtain several space–time cubes that completely cover $Query^{st}$, and these space–time cubes are at the $optimalLevel$th level of HGST.

$$\begin{cases} sScale_{sLevel+1} < Dis_S \leq sScale_{sLevel} \\ tScale_{tLevel+1} < Dis_T \leq tScale_{tLevel} \\ optimalLevel = Max(sLevel, tLevel) \end{cases} \tag{2}$$

As discussed in Section 4.2.4, any HGST space–time cube can be divided into 8 child space–time cubes by one octree division, and the HGSTCode corresponding to the parent space–time cube is the prefix of the HGSTCode corresponding to the child space–time cube. Based on this, we designed a query code merging strategy. The idea is to replace the eight HGSTCodes with one query code, provided that the space–time cubes corresponding to these eight HGSTCodes are obtained by the same space–time cube after one subdivision. For example, when the $Query^{st}$ is completely covered by the space–time cubes corresponding to the eight HGSTCodes of {1-62030, 1-62031, 1-62032, 1-62033, 1-62034, 1-62035, 1-62036, and 1-62037}, we only need to query the data in the space–time cube 1-6203 to obtain the same result. The query unit grid merge method can reduce the scanning times of the index record table and improve query efficiency.

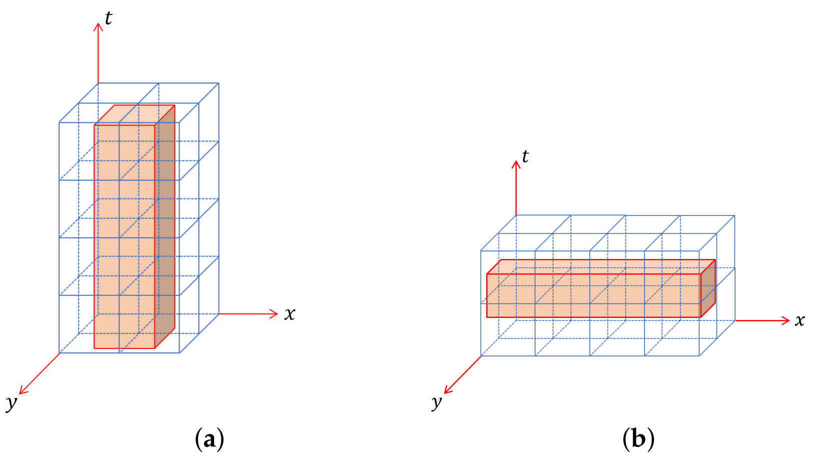


Figure 10. Spatio-temporal range query with unbalanced time and space dimension scales. (**a**) The query for a long time range and a small spatial range. (**b**) The query for a short time range and a large spatial range.

We calculated a set of HGSTCode according to the spatio-temporal range query conditions input by the user. Note that all the row keys prefixed with each element of the set in the HBase index table are candidate records. Therefore, we generated a set of HBase prefix filter lists based on the HGSTCode set, and obtained candidate results through the $SCAN$ operation.

Algorithm 1: Adaptive spatio-temporal scaling and coding method based on HGST

```
   /* The scope of a spatio-temporal range query */
1  the spatial range $S$, the temporal range $T$ /* Generate the query code set */
2  $codeSet$: Each element in $codeSet$ corresponds to a space–time cube, and all cubes
   identified by $codeSet$ can cover the $Query^{st}$
3  $codeSet \leftarrow \phi$
4  Regard the MBR of $S$ as $S_{MBR}$
   // $sLevel, tLevel$ and $optimalLevel$ are calculated by Equation (2)
5  if $sLevel = tLevel$ then
       /* See Algorithm 2 for details */
6      $codeSet$ = sEqualst($S_{MBR}, T, optimalLevel$)
7  if $sLevel > tLevel$ then
       /* See Algorithm 3 for details */
8      $codeSet$ = sGreatert($S_{MBR}, T, optimalLevel$)
9  else
       /* See Algorithm 4 for details */
10     $codeSet$ = sLesst($S_{MBR}, T, optimalLevel$)
```

Algorithm 2: The query on the same temporal and spatial scale

```
1  Function sEqualst($S_{MBR}$, $T$, $level$):
2      $codeSet \leftarrow \phi$
3      initialize the space–time cube $Query^{st} \leftarrow (S_{MBR}, T)$ corresponding to the query
        range
4      for each vertex coordinate $O \in Query^{st}$ do
           /* encode is a function to calculate HGSTCode based on the
              HGST subdivision model */
5          $codeSet \leftarrow$ encode($level, O_{lon}, O_{lat}, O_{time}$)
           /* $O_{lon}$, $O_{lat}$ and $O_{time}$ are the longitude, latitude and time of
              the 3D coordinate point $O$ */
6      return $codeSet$
```

Algorithm 3: The query for a long time range and a small spatial range

```
1  Function sGreatert($S_{MBR}$, $T$, $level$):
2      $codeSet \leftarrow \phi$
3      Get the temporal range $(t_{start}, t_{end})$ from $T$
4      $verT \leftarrow t_{start}$
       /* $tScale_{level}$ is the temporal resolution when the subdivision level
          is $level$ according to Table 1 */
5      $gradient \leftarrow tScale_{level}$
6      while $verT <= t_{end}$ do
7          for $P \in four\ Coordinates\ Of\ S_{MBR}$ do
8              $codeSet \leftarrow$ encode($level, P_{lon}, P_{lat}, verT$)
               /* $P_{lon}$ and $P_{lat}$ are the longitude and latitude of the
                  spatial coordinate point $P$ */
9          $verT$+=$gradient$
10     return $codeSet$
```

Algorithm 4: The query for a short time range and a large spatial range

```
1  Function sLesst($S_{MBR}$, $T$, $level$):
2      $codeSet \leftarrow \phi$
3      Get the rectangular range $(lon_{min}, lat_{max}, lon_{max}, lat_{min})$ from $S_{MBR}$
4      Get the temporal range $(t_{start}, t_{end})$ from $T$
5      $verLon \leftarrow lon_{min}$
6      $verLat \leftarrow lat_{min}$
       /* $sScale_{level}$ is the spatial resolution when the subdivision level
          is $level$ according to Table 1                                          */
7      $gradient \leftarrow sScale_{level}$
8      while $verLon <= lon_{max}$ do
9          while $verLat <= lat_{max}$ do
10             $codeSet \leftarrow$ encode($level, verLon, verLat, t_{start}$)
11             $codeSet \leftarrow$ encode($level, verLon, verLat, t_{end}$)
12             $verLat$+=$gradient$
13         $verLon$+=$gradient$
14     return $codeSet$
```

4.4.2. Refinement Stage

The refinement stage takes candidate results as input and computes the precise spatio-temporal relationship between each trajectory point and the query scope. Loading, indexing, and processing these data in a spatial DBMS is often too time-consuming and requires a lot of preprocessing [49]. We adopt the memory-based computing framework Spark to obtain candidate results and finish the spatio-temporal relationship computation in a distributed manner. In Spark, the core abstraction is RDD (Resilient Distributed Dataset), which is a read-only recoverable dataset distributed across a cluster [50].

Spark-based spatio-temporal relationship computation uses the RDD programming paradigm to customize the $filter$ operator to distribute spatio-temporal relationship computation tasks to multiple nodes for parallel execution. Each node judges whether the location of trajectory data is within the query area according to the spatio-temporal containment relation. Afterward, the trajectory data satisfying the query condition are mapped to new records that are easy to manipulate or save through the map operator. These steps run in memory, and the intermediate results of the job do not need to be written to disk, which can significantly improve the execution efficiency of the refinement stage in the spatio-temporal range query.

In summary, we designed a Spark-based method for the refinement stage of trajectory spatio-temporal range query. The implementation details of the method refer to Algorithm 5. First, we take the spatio-temporal range as input and obtain a set of query codes based on the adaptive spatio-temporal scaling and coding method. From the characteristics of HGSTCode, we know that a space–time cube corresponding to a $hgstCode$ must contain the space–time cube corresponding to any code prefixed with $hgstCode$. Therefore, we use $FilterList$, a list of $PrefixFilter$, to perform server-side data filtering on the row key when accessing data through the $SCAN$ operation. The operator of $FilterList$ is set to be “MUSS_PASS_ONE,” i.e., index records whose row keys contain any prefix defined in the $PrefixFilter$ of the $FilterList$ will be returned. We then use Spark’s $newAPIHadoopRDD$ operator to retrieve the candidate result set from HBase according to the generated $FilterList$ and load the data into memory. Finally, the $filter$ operator is used to filter the trajectory objects within the given spatio-temporal range, and the final result is saved.

Through the above method, once the $SCAN$ needed to scan HBase is built, we can directly read the data stored in HBase into the memory through Spark, and then complete subsequent time-consuming tasks based on the rich programming paradigm of RDD.

After reading the data into memory, the rest of the process is performed in parallel on multiple machines. This parallel computing in a distributed environment is suitable for the analysis of large-scale trajectory data, and this method avoids frequent disk IO and has good scalability.

Algorithm 5: Spark-based method for the refinement stage of trajectory spatio-temporal range query

```
   /* The scope of a spatio-temporal range query                              */
1  the spatial range $S$, the temporal range $T$ Query result set
2  Get the rectangular range $(lon_{min}, lat_{max}, lon_{max}, lat_{min})$ from the MBR of $S$
   /* Refer to Algorithm 1                                                    */
3  Get the query code set $codeSet$
4  $FilterList \leftarrow \phi$
5  for $hgstCode$ in $codeSet$ do
6      $FilterList \leftarrow PrefixFilter(hgstCode)$
7  Initialize the HBase configuration $stConf$
   /* Set the HBase client scan $stScan$                                      */
8  $stScan.setFilter(FilterList)$
9  $stConf \leftarrow stScan$
10 Initialize the Spark context $sc$
   /* Get candidate results from HBase according to $stScan$                  */
11 $candidateRdds \leftarrow$ Get RDDs with $stConf$ as a parameter in the
     $newAPIHadoopRDD$ operator of $sc$
   /* Refinement stage: Computes the spatio-temporal relationship             */
12 $finalRdds \leftarrow$ Get final results with $S$ and $T$ as parameters in the $filter$ operator of
     $sc$
13 save the query result $finalRdds$
   /* Submit the task to the Spark standalone cluster                         */
14 $sc$.stop() // The task finished
```

5. Experiments and Results

In this section, we first describe the experimental setup and approach to evaluate the performance of the proposed indexing method, and then present the evaluation results, which are also discussed and interpreted.

5.1. Experimental Setup and Methodology

Hardware and Software Configuration. Experiments are performed on a three-node virtual machine cluster running Apache Hadoop 2.7.3, Apache Zookeeper 3.4.10, Apache HBase 1.2.6, and Apache Spark 3.1.3 over CentOS 7.6. All nodes play the role of the Datanode of Hadoop and the Regionserver of HBase. One of the nodes acts as both the Namenode of Hadoop and the HMaster of HBase. Each node in the cluster is equipped with a 2-core CPU and 500GB disk; of particular note, we allocated 10GB of memory for the master node and 6GB of memory for the other two nodes.

Datasets. To evaluate the performance of HGST, we use a real spatio-temporal dataset T-Drive that contains the GPS trajectories of 10,357 taxis during the period of 2 February to 8 February 2008 within Beijing [51,52]. The total number of points in this dataset is about 15 million, and the total distance of the trajectories reaches 9 million kilometers.

Baselines and Settings. We compare HGST with three other indexing methods that support spatio-temporal range query, including GeoSOT-ST, GeoMesa Z3, and JUST Z2T. Both the first two and HGST divide the space–time recursively to obtain 3D coding, and then perform bitwise crossover to filter time and space at the same time; we set their initial subdivided period to one year for fairness of the experiment. However, Z2T is different in that it indexes time and space separately; we adopt the same parameters as the JUST experiment [18] and set the period of Z2T as a day. We test the efficiency of HGST for spatio-temporal range queries. Table 2 summarizes the query parameters. For each query, we perform this three times and calculate the average as the query time-consuming result.

Table 2. Query settings (the red font is the default value).

Parameters	Setting
Data Size (millions)	1, 3, 5, 10, 15
Time Window	1 h, 4 h, 12 h, 1 day, 3 day
Spatial Window (0.009° × 0.009°)	3 × 3, 5 × 5, 10 × 10, 20 × 20, 30 × 30

5.2. Evaluation of the Efficiency of the Index Construction

In this section, we study the performance of the index construction for a trajectory point. Figure 11 illustrates the time spent on index construction of different methods; the time includes both the cost of encoding and storing. The first observation is that as the number of data increases, index construction takes longer, because of more index encoding and storage overhead. Secondly, Z3 and Z2T consume relatively less time. They are both variants of GeoHash and are encoded in the same way in the spatial dimension. In particular, Z3 divides the time dimension at the granularity of days, while Z2T divides it at the granularity of minutes. Thirdly, HGST takes longer time than GeoSOT-ST. The reason is that HGST uses Hilbert filling curves to concatenate the divided spatial grids. The computational complexity of Hilbert is higher compared to the Z-curve used by GeoSOT-ST, so it leads to longer index construction time. However, Hilbert maintains better spatial aggregation, making spatially adjacent objects closer in storage, which will reduce the random disk IO that occurs during data query reads. We evaluate the benefits of using Hilbert curves in Section 5.4.1.

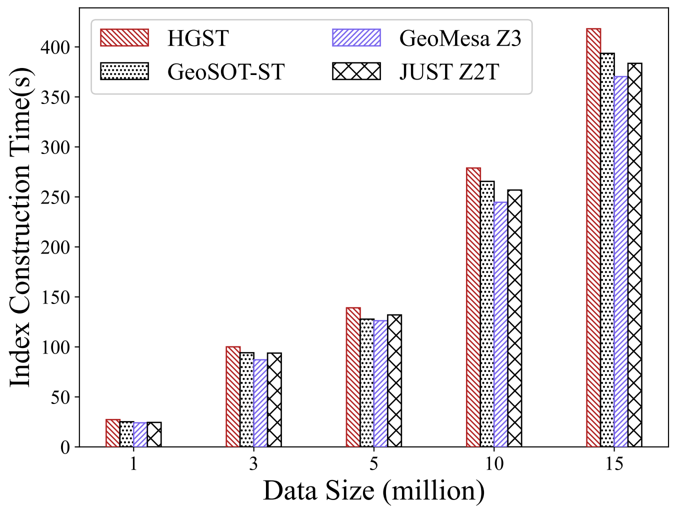
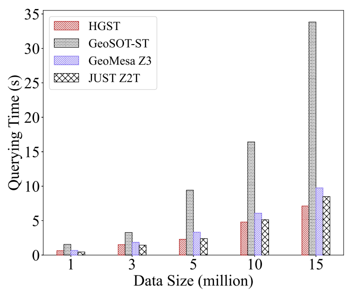


Figure 11. Time of index construction.

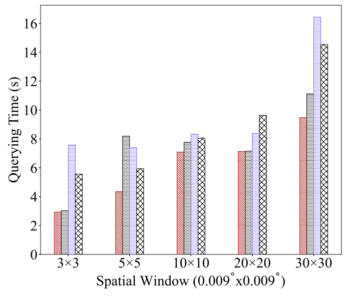
5.3. Performance of Spatio-Temporal Query

To test the query performance of HGST under different data volumes and different spatial and temporal dimension scales, we conducted the following three experiments.

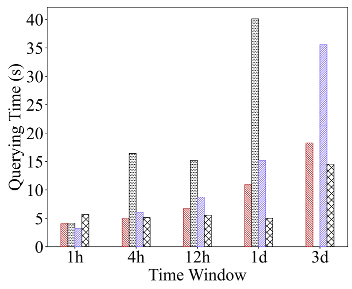
Different Data Sizes. Figure 12a shows the query efficiency varies with different data sizes of the T-Drive dataset. There are three observations: (1) with more data, the time of spatio-temporal query becomes longer for all methods, because it scans and returns more data; (2) with the increase in data volume, the query time consumption of GeoSOT-ST grows rapidly. The reason is that GeoSOT-ST, with the parameters set in this experiment (the time window is 4 h, and the spatial window is 0.045° × 0.045°), is a query with unbalanced spatial and temporal scales (i.e., a long time and small area query). As shown in Figure 13, GeoSOT-ST determines the index level according to the time range of the query, which results in returning too much "fake data", so a lot of spatio-temporal relationship calculations need to be performed in the refinement stage; (3) HGST is faster than other methods, because the adaptive spatio-temporal scaling and coding method based on HGST avoids the invalid spatial filtering caused by the difference in scale between spatial and temporal dimensions; although Z2T indexes the time dimension and the spatial dimension separately to solve this problem, it is not good at spatio-temporal range queries with a smaller time period; (4) the time consumption of HGST increases slower, and when the amount of data increases by one order of magnitude, the query time remains stable compared with other methods, which demonstrates that HGST has better scalability under different data amounts.



(**a**)



(**b**)



(**c**)

Figure 12. Performance of spatio-temporal range query. (**a**) Spatio-temporal range query under different data sizes. (**b**) Spatio-temporal range query under different spatial windows. (**c**) Spatio-temporal range query under different time windows.

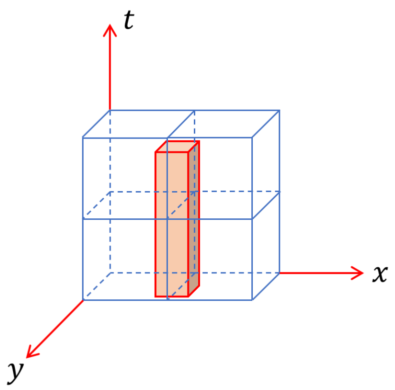


Figure 13. Schematic diagram of GeoSOT-ST under unbalanced spatial and temporal scales.

Different Spatial Windows. Figure 12b presents the query time with different spatial windows for the real dataset. It is observed that (1) with a larger spatial window, most methods need more time to answer a spatio-temporal range query, as we retrieve and return more data; (2) as the scope of the query space increases, the performance of Z3 and

Z2T is poor. Z3 performs interval scanning according to the start and end space–time codes, and the order of space–time codes is time, longitude, and latitude (Figure 14). Therefore, as the spatial query range increases, the latitude and longitude encoding of the high bit will invalidate the filtering effect of the time encoding of the low bit in the temporal dimension. Z2T divides the time dimension at a granularity of one day, and each query returns an integer number of days of data, which is not friendly to queries with a small time range (e.g., the time window is 4 h), and especially returns more "fake data" when the spatial range expands; (3) HGST is usually optimal at various spatial scales, and on average, the query time consumption of HGST is approximately 14.77% and 34.93% lower than that of GeoSOT-ST and GeoMesa, respectively. This indicates that the HGST indexing strategy proposed by this paper can better cope with the imbalance of spatial and temporal scales.

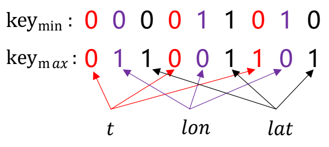

key$_{min}$: 0 0 0 0 1 1 0 1 0
key$_{max}$: 0 1 1 0 0 1 1 0 1
t
lon
lat

Figure 14. Example of key range in Z3.

Different Time Windows. Figure 12c compares the spatio-temporal range query time with different time windows for T-Drive data. With a bigger time window, most methods need more time, because a bigger time window means more qualified records. GeoSOT-ST fails when querying data within 3 days (the spatial window is 0.045° × 0.045°), which is due to the imbalance of spatial and temporal scales, resulting in returning too much data that require computing spatio-temporal relationships in memory. When the memory is about to run out, they incur disk thrashing. Z2T is better than our proposed HGST when the query time range size is an integer multiple of days, such as 1 day or 3 days. This is because Z2T filters by day in the time dimension, which is advantageous for the query with integer days, but the query performance is slightly degraded for small time ranges or non-integer days.

5.4. Effects of Method Optimization

5.4.1. Effect of Hilbert Filling Curve

The proposed HGST abandons the default Z-Order curve of the GeoSOT grid model for the encoding of spatial dimensions but adopts the Hilbert curve, which has better spatial aggregation in theory. To discuss the effect of introducing Hilbert space-filling curves on the performance of spatio-temporal range queries, we conduct ablation experiments to compare the performance of the following two variants: (1) HGST and (2) HGST$_z$, which differs from HGST in that it encodes based on Z-curves in the spatial dimension. To avoid accidental differences in space, we performed spatio-temporal range queries with two landmarks as the center points of the spatial range; one is the Forbidden City, and the other is Beijing West Railway Station. We set the data size to 15 million, and performed two experiments for each site, setting the time window to 1 h and 3 days, respectively; refer to Table 2.

Figure 15 shows the effect of Hilbert filling curves on spatio-temporal range queries. We observe that (1) HGST mostly takes less time as the spatial extent increases. The reason is that the Hilbert curve has better aggregation than the Z-Order curve, which can usually make the trajectory points that are close in space have good proximity in physical storage. This is likely to go through less random disk IO when reading data; therefore, the more data returned, the more time will be saved. (2) The increase in the time window will enlarge the gap between HGST and HGST$_z$, and HGST is better in most cases. This demonstrates that our adoption of the Hilbert curve on spatial subdivision encoding ensures better spatial proximity while not destroying the original temporal proximity feature, which is beneficial for large-scale spatio-temporal range queries.

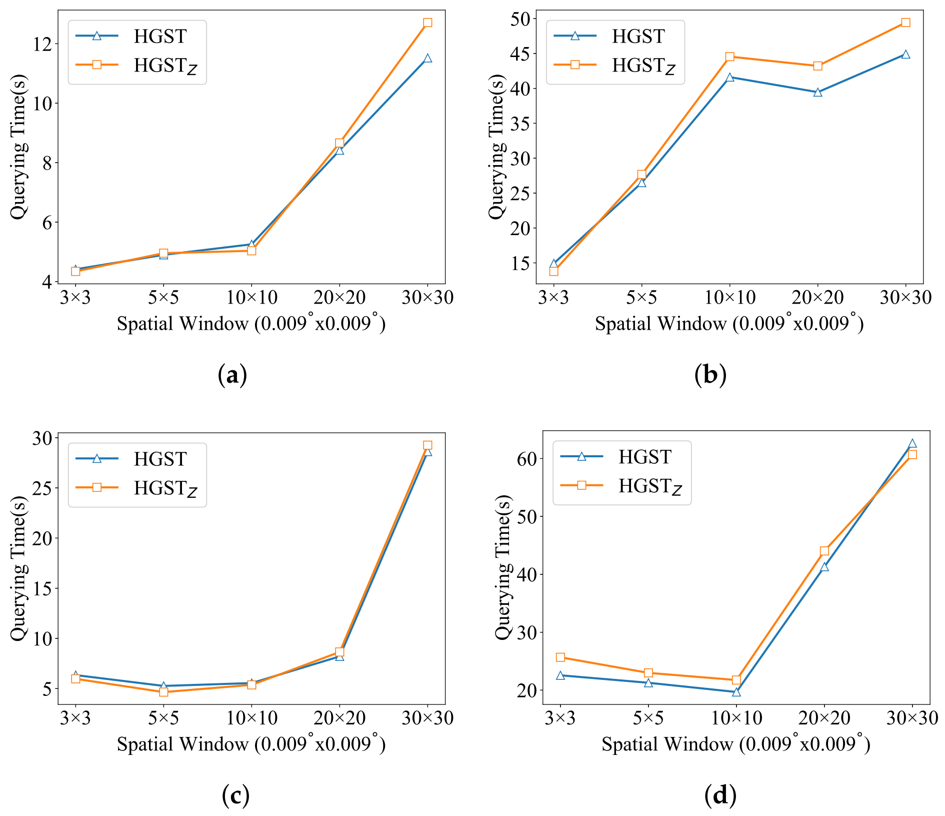


Figure 15. The effect of Hilbert filling curves on spatio-temporal range query. (**a**) Forbidden City (time window = 1 h); (**b**) Forbidden City (time window = 3 days); (**c**) Beijing West Railway Station (time window = 1 h); (**d**) Beijing West Railway Station (time window = 3 days).

5.4.2. Effect of Merging Query Codes

In the filtering stage of the trajectory spatio-temporal range query, we designed a query code merging strategy to reduce the query complexity. Note that code merging only works on specified conditions, which are related to the spatio-temporal location of the query. To discuss the effect of the merge of query codes on the performance of Spatio-temporal range queries, we set up the following experiment. We randomized several 3D points within the spatio-temporal range covered by the real dataset T-Driver. Then, spatio-temporal range queries are conducted under multiple sets of spatio-temporal range sizes, with each 3D point as the center. Each query is performed on both variants: (1) HGST; (2) HGST$_{unmerge}$, which differs from HGST in that it does not use the query encoding merge strategy.

Table 3 shows the effect of merging query codes. *merge quota* is the number of original query codes divided by the number of merged codes. *time ratio* is the ratio of query time consumption before and after merging. There are two observations: (1) When *merge quota* is equal to 1, *time ratio* is also close to 1. This is because there is no code merging, so the query time of HGST and HGST$_{unmerge}$ is comparable. (2) When *merge quota* is bigger, *time ratio* is usually larger. The reason is that the query codes are merged, resulting in a smaller filter list being constructed, thus reducing the number of scanning HBase tables. The larger *merge quota* is, the better the code merge effect, and the more obvious the improvement. This demonstrates that our designed query code merging strategy can improve the efficiency of spatio-temporal range queries.

Table 3. The effect of merging query codes on spatio-temporal range query with different time windows.

	***P*$_1$ (116.41961, 39.95879, 6 February 2008 18:18:50)**		***P*$_2$ (116.31314, 39.95514, 6 February 2008 07:06:19)**		***P*$_3$ (116.44057, 39.91701, 5 February 2008 05:45:46)**		***P*$_4$ (116.40911, 39.95973, 2 February 2008 09:33:17)**	
time window	***merge quota***	***time ratio***	***merge quota***	***time ratio***	***merge quota***	***time ratio***	***merge quota***	***time ratio***
1 h	18/18	7.409/7.212	12/12	5.663/5.631	12/12	4.425/4.364	12/12	0.264/0.261
4 h	12/5	7.057/6.409	12/5	6.184/5.345	12/5	4.816/4.216	6/6	0.24/0.236
12 h	28/7	9.754/7.715	28/7	8.515/6.223	28/7	7.394/5.718	14/14	0.68/0.598
1 day	52/10	12.943/8.606	52/10	12.492/8.153	52/10	12.868/9.34	26/26	1.918/1.862
3 days	148/22	31.381/17.772	148/22	28.438/15.605	148/22	30.33/19.214	74/74	6.904/6.75

5.4.3. Effect of Using Spark

In the refinement stage of the trajectory spatio-temporal range query, based on Spark's RDD programming paradigm, we take the results scanned from HBase as input to filter and save the results. To discuss the effect of using Spark to complete the spatio-temporal range query, we conduct an ablation experiment to compare the performance of the following three variants. (1) HGST$_{Spark}$; (2) HGST$_{MapReduce}$, which differs from HGST$_{Spark}$ in that it uses MapReduce paradigm to accomplish functions similar to Spark; and (3) HGST$_{Standalone}$, which only uses the client to collect the results from HBase and perform subsequent spatio-temporal relationship filtering. We set the time window to 3D and the spatial window to $0.27^\circ \times 0.27^\circ$, and then carried out spatio-temporal range queries of different data sizes.

Figure 16 shows the effect of using Spark to complete the spatio-temporal range query. We observed that HGST$_{Standalone}$ has good performance at the beginning, but when the data size increases to 15 million, it is much slower than the other two, although the time window is only 70% of the original. This is because the returned data are too large, which may lead to the single machine bottleneck in the collection of query results and the calculation of spatio-temporal relationships. Secondly, in large-scale data scenarios, HGST$_{Spark}$ has better performance than HGST$_{MapReduce}$. The reason is that MapReduce writes intermediate results to disk, which incurs more IO overhead. Therefore, our method can utilize the memory resources of the cluster to complete efficient spatio-temporal range queries on massive trajectory data.

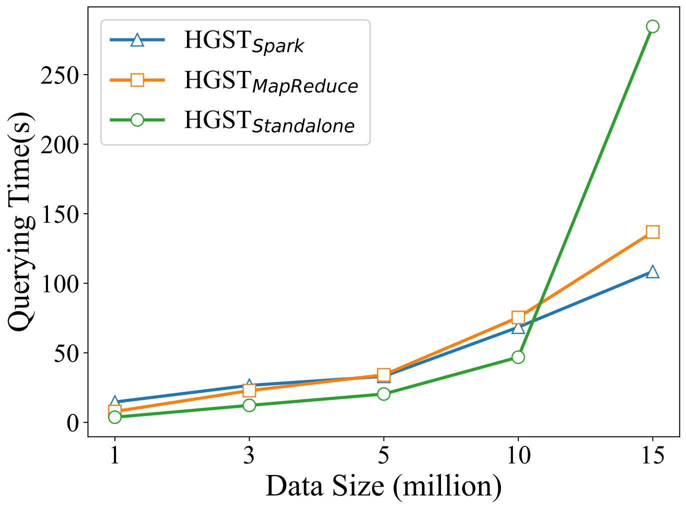


Figure 16. The effect of using Spark to complete the spatio-temporal range query (for HGST$_{Standalone}$, the time window is reduced to 70% of the original when data size is 15 million).

6. Discussion

In this study, we propose a spatio-temporal partitioning-based indexing method for trajectory data. In theory, this method can construct a multi-scale spatio-temporal index for trajectory data at any time and in any geographical location. The cost of the index update is low. When the data are updated, the data table records are simply added or deleted. In this paper, we apply the index model to the NoSQL database HBase by combining the HGST index with big data technology-distributed storage. Compared with some works, such as GeoOST-ST that supports spatio-temporal data discovery of trajectories generated by GNSS-equipped mobile devices, we found that HGST exhibits relatively stable and competitive query capabilities at different spatial and temporal dimension scales. In addition, the accuracy of spatio-temporal information represented by encoding in the HGST index model shows advantages in query efficiency that are platform-independent, so the index can be integrated with existing database systems. In conclusion, HGST helps to reveal some valuable information in a very short time, which has certain significance for data mining of large-scale trajectory datasets. However, this work still has some parts that can be further improved in our future works.

- The current work lacks scan optimization for filtering operations. If too many HGST query codes are generated, it may need to scan the database too many times, resulting in reduced efficiency. In our future works, we will consider using multi-threads to trigger $SCAN$ operations over the underlying key-value data storage in parallel, or remotely execute coded scan filtering in RegionServers based on the HBase endpoint coprocessor to achieve parallel queries at the Region level of Table. The above may provide researchers with more efficient data discovery capabilities;
- In the current work, HGST cannot intelligently select a standalone version or Spark mode to execute jobs according to the size of the data. To overcome this limitation, in our future works, we will consider using data sampling technology to estimate the amount of data requested, and then choose a single-machine version for a small data request, which will save the overhead of cluster resource scheduling to a certain extent, making it more suitable for practical applications.

7. Conclusions

This paper presents HGST, a Hilbert-GeoSOT spatio-temporal meshing and coding method for efficient spatio-temporal range query on massive trajectory data. HGST offers a novel spatio-temporal index structure, which realizes the transformation of the spatio-temporal information of the trajectory data into key-value-based index records while ensuring temporal and spatial proximity. We designed an adaptive spatio-temporal scaling and coding method based on HGST, and then developed a Spark-based algorithm to improve spatio-temporal range query performance on huge trajectory data. Experiments based on a real dataset verify the superior query efficiency and scalability of HGST. The insights gained from this study may help to realize efficient spatio-temporal range queries of real-time and massive trajectory data.

HGST allows different bodies with spatio-temporal information from multiple sources to use the same set of spatio-temporal identification codes, thereby achieving coding uniformity. In theory, by using similar codes in navigation services, HGST can provide support for rapid sharing, efficient retrieval, and other operational services for various urban spatio-temporal applications. Future work will focus on more complex spatio-temporal query support, and evaluate the feasibility of HGST in more complex scenarios, e.g., trajectory similarity search, traffic simulation, etc.

Author Contributions: All authors contributed to this study's conception and design. Hong Liu wrote the manuscript and conceived the experiment. Jining Yan provided algorithm ideas and provided guidance on the writing of the entire manuscript. Jinlin Wang provided guidance on the actual algorithm process and corrections. Bo Chen provided data curation and investigation. Meng Chen provided verification, and Xiaohui Huang provided review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Third Xinjiang Scientific Expedition Program (2022xjkk1306), the Hubei Key Laboratory of Yangtze Catchment Environmental Aquatic Science, China University of Geosciences, Wuhan, 430078, China, and the Hubei Province Department of Science and Technology (ZRZY2023KJ15).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are openly available at https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/ (accessed on 22 March 2022) [51,52].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xie, X.; Mei, B.; Chen, J.; Du, X.; Jensen, C.S. Elite: An elastic infrastructure for big spatiotemporal trajectories. *VLDB J.* **2016**, *25*, 473–493. [CrossRef]
2. Gao, C.; Zhang, Z.; Huang, C.; Yin, H.; Yang, Q.; Shao, J. Semantic trajectory representation and retrieval via hierarchical embedding. *Inf. Sci.* **2020**, *538*, 176–192. [CrossRef]
3. Dodge, S.; Gao, S.; Tomko, M.; Weibel, R. Progress in computational movement analysis—Towards movement data science. *Int. J. Geogr. Inf. Sci.* **2020**, *34*, 2395–2400. [CrossRef]
4. Burger, C.N.; Kleynhans, W.; Grobler, T.L. Extended linear regression model for vessel trajectory prediction with a priori AIS information. *Geo-Spat. Inf. Sci.* **2022**, 1–19. [CrossRef]
5. Zheng, Y.; Capra, L.; Wolfson, O.; Yang, H. Urban Computing: Concepts, Methodologies, and Applications. *ACM Trans. Intell. Syst. Technol.* **2014**, *5*, 38:1–38:55. [CrossRef]
6. Bakli, M.S.; Sakr, M.A.; Zimányi, E. Distributed Spatiotemporal Trajectory Query Processing in SQL. In Proceedings of the 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, 3–6 November 2020.
7. Deng, K.; Xie, K.; Zheng, K.; Zhou, X. Trajectory Indexing and Retrieval. In *Computing with Spatial Trajectories*; Springer: New York, NY, USA, 2011.
8. Ghosh, S.; Ghosh, S.K.; Buyya, R. MARIO: A spatio-temporal data mining framework on Google Cloud to explore mobility dynamics from taxi trajectories. *J. Netw. Comput. Appl.* **2020**, *164*, 102692. [CrossRef]
9. Zhang, T.; Zhang, W.; He, Z. Measuring positive public transit accessibility using big transit data. *Geo-Spat. Inf. Sci.* **2021**, *24*, 722–741. [CrossRef]
10. Kothari, P.; Kreiss, S.; Alahi, A. Human Trajectory Forecasting in Crowds: A Deep Learning Perspective. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 7386–7400. [CrossRef]
11. Fang, Z.; Chen, L.; Gao, Y.; Pan, L.; Jensen, C.S. Dragoon: A hybrid and efficient big trajectory management system for offline and online analytics. *VLDB J.* **2021**, *30*, 287–310. [CrossRef]
12. Zhu, Q.; Gong, J.; Zhang, Y. An efficient 3D R-tree spatial index method for virtual geographic environments. *ISPRS J. Photogramm. Remote. Sens.* **2007**, *62*, 217–224. [CrossRef]
13. Pfoser, D.; Jensen, C.S.; Theodoridis, Y. Novel Approaches to the Indexing of Moving Object Trajectories. *Proc. VLDB* **2000**, 2000, 395–406.
14. Song, Z.; Roussopoulos, N. SEB-tree: An Approach to Index Continuously Moving Objects. In Proceedings of the Mobile Data Management, Melbourne, Australia, 21–24 January 2003.
15. Nidzwetzki, J.K.; Güting, R.H. BBoxDB—A Scalable Data Store for Multi-Dimensional Big Data. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino, Italy, 22 October 2018.
16. Fox, A.D.; Eichelberger, C.N.; Hughes, J.N.; Lyon, S. Spatio-temporal indexing in non-relational distributed databases. In Proceedings of the 2013 IEEE International Conference on Big Data, Silicon Valley, CA, USA, 6–9 October 2013; pp. 291–299.
17. Qian, C.; Yi, C.; Cheng, C.; Pu, G.; Wei, X.; Zhang, H. GeoSOT-Based Spatiotemporal Index of Massive Trajectory Data. *ISPRS Int. J. Geo Inf.* **2019**, *8*, 284. [CrossRef]
18. Li, R.; He, H.; Wang, R.; Huang, Y.; Liu, J.; Ruan, S.; He, T.; Bao, J.; Zheng, Y.X. JUST: JD Urban Spatio-Temporal Data Engine. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 1558–1569.

19. Xu, P.; Nguyen, C.; Tirthapura, S. Onion Curve: A Space Filling Curve with Near-Optimal Clustering. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 1236–1239.
20. Jiang, H.; Kang, J.; Du, Z.; Zhang, F.; Huang, X.; Liu, R.; Zhang, X. Vector Spatial Big Data Storage and Optimized Query Based on the Multi-Level Hilbert Grid Index in HBase. *Information* **2018**, *9*, 116. [CrossRef]
21. Lei, Y.; Tong, X.; Zhang, Y.; Qiu, C.; Wu, X.S.; Lai, G.; Li, H.; Guo, C.; Zhang, Y. Global multi-scale grid integer coding and spatial indexing: A novel approach for big earth observation data. *ISPRS J. Photogramm. Remote. Sens.* **2020**, *163*, 202–213. [CrossRef]
22. Guo, N.; Xiong, W.; Wu, Y.; Chen, L.; Jing, N. A Geographic Meshing and Coding Method Based on Adaptive Hilbert-Geohash. *IEEE Access* **2019**, *7*, 39815–39825. [CrossRef]
23. Huang, X.; Deng, Z.; Yan, J.; Li, J.; Chen, Y.; Wang, L. A High-Performance Spatial Range Query-Based Data Discovery Method on Massive Remote Sensing Data via Adaptive Geographic Meshing and Coding. *IEEE J. Miniaturizat. Air Space Syst.* **2021**, *2*, 117–128. [CrossRef]
24. Wu, Y.; Cao, X.; An, Z. A Spatiotemporal Trajectory Data Index Based on the Hilbert Curve Code. *IOP Conf. Ser. Earth Environ. Sci.* **2020**, *502*, 012005. [CrossRef]
25. Wang, X.; Sun, Y.; Sun, Q.; Lin, W.; Wang, J.Z.; Li, W. HCIndex: A Hilbert-Curve-based clustering index for efficient multi-dimensional queries for cloud storage systems. *Clust. Comput.* **2022**, 1–15. [CrossRef]
26. Moon, B.; Jagadish, H.V.; Faloutsos, C.; Saltz, J. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Trans. Knowl. Data Eng.* **2001**, *13*, 124–141. [CrossRef]
27. Shang, Z.; Li, G.; Bao, Z. DITA: Distributed In-Memory Trajectory Analytics. In Proceedings of the 2018 International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018.
28. Zheng, K.; Zhao, Y.; Lian, D.; Zheng, B.; Liu, G.; Zhou, X. Reference-Based Framework for Spatio-Temporal Trajectory Compression and Query Processing. *IEEE Trans. Knowl. Data Eng.* **2020**, *32*, 2227–2240. [CrossRef]
29. Pelekis, N.; Frentzos, E.; Giatrakos, N.; Theodoridis, Y. HERMES: A Trajectory DB Engine for Mobility-Centric Applications. *Int. J. Knowl. Based Organ.* **2015**, *5*, 19–41. [CrossRef]
30. Zimányi, E.; Sakr, M.A.; Lesuisse, A. MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS. *ACM Trans. Database Syst.* **2020**, *45*, 19:1–19:42. [CrossRef]
31. Aji, A.; Wang, F.; Vo, H.; Lee, R.; Liu, Q.; Zhang, X.; Saltz, J. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB Endow. Int. Conf. Very Large Data Bases* **2013**, *6*, 11. [CrossRef]
32. Bakli, M.S.; Sakr, M.A.; Soliman, T.H.A. HadoopTrajectory: A Hadoop spatiotemporal data processing extension. *J. Geogr. Syst.* **2019**, *21*, 211–235. [CrossRef]
33. Tian, R.; Zhai, H.; Zhang, W.; Wang, F.; Guan, Y. A Survey of Spatio-Temporal Big Data Indexing Methods in Distributed Environment. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.* **2022**, *15*, 4132–4155. [CrossRef]
34. Li, G.; Tang, J. A New R-tree Spatial Index Based on Space Grid Coordinate Division. In Proceedings of the 2011 International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011), Melbourne, Australia, 19–20 November 2011.
35. Qi, J. Packing R-trees with Space-Filling Curves: Theoretical Optimality, Empirical Efficiency, and Bulk-loading Parallelizability. *ACM Trans. Database Syst.* **2020**, *45*, 1–47.
36. Guan, X.; Bo, C.; Li, Z.; Yu, Y. ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database. In Proceedings of the 2017 25th International Conference on Geoinformatics, Buffalo, NY, USA, 2–4 August 2017; pp. 1–7.
37. Ding, R.; Meng, X. A quadtree based dynamic attribute index structure and query process. In Proceedings of the 2001 International Conference on Computer Networks and Mobile Computing, Beijing, China, 16–19 October 2001; pp. 446–451.
38. Huang, M.; Hu, P.; Xia, L. A grid based trajectory indexing method for moving objects on fixed network. In Proceedings of the 2010 18th International Conference on Geoinformatics, Beijing, China, 18–20 June 2010; pp. 1–4.
39. Qu, T.; Wang, L.; Yu, J.; Yan, J.; Xu, G.; Li, M.H.; Cheng, C.; Hou, K.; Chen, B. STGI: A spatio-temporal grid index model for marine big data. *Big Earth Data* **2020**, *4*, 435–450. [CrossRef]
40. Liu, H.; Yan, J.; Huang, X. HBase-based spatial-temporal index model for trajectory data. *Iop Conf. Ser. Earth Environ. Sci.* **2022**, *1004*, 012007. [CrossRef]
41. Li, R.; He, H.; Wang, R.; Ruan, S.; Sui, Y.; Bao, J.; Zheng, Y. TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 2002–2005.
42. Lê, H.V.; Takasu, A. G-HBase: A High Performance Geographical Database Based on HBase. *IEICE Trans. Inf. Syst.* **2018**, *101-D*, 1053–1065. [CrossRef]
43. Yang, S.; He, Z.; Chen, Y.P.P. GCOTraj: A storage approach for historical trajectory data sets using grid cells ordering. *Inf. Sci.* **2018**, *459*, 1–19. [CrossRef]
44. Wang, C.; Zourlidou, S.; Golze, J.; Sester, M. Trajectory analysis at intersections for traffic rule identification. *Geo-Spat. Inf. Sci.* **2021**, *24*, 75–84. [CrossRef]
45. Cheng, C.; Tong, X.; Chen, B.; Zhai, W. A Subdivision Method to Unify the Existing Latitude and Longitude Grids. *ISPRS Int. J. Geo Inf.* **2016**, *5*, 161. [CrossRef]
46. Li, S.; Pu, G.; Cheng, C.; Chen, B. Method for managing and querying geo-spatial data using a grid-code-array spatial index. *Earth Sci. Inform.* **2018**, *12*, 173–181. [CrossRef]

47. Bakli, M.S.; Sakr, M.A.; Soliman, T.H.A. A spatiotemporal algebra in Hadoop for moving objects. *Geo-Spat. Inf. Sci.* **2018**, *21*, 102–114. [CrossRef]
48. Borthakur, D. HDFS architecture guide. *Hadoop Apache Proj.* **2008**, *53*, 2.
49. Hagedorn, S.; Goötze, P.; Sattler, K.U. The STARK Framework for Spatio-Temporal Data Analytics on Spark. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*; Mitschang, B., Nicklas, D., Leymann, F., Schöning, H., Herschel, M., Teubner, J., Härder, T., Kopp, O., Wieland, M., Eds.; Gesellschaft für Informatik: Bonn, Germany, 2017; pp. 123–142.
50. Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauly, M.; Franklin, M.; Shenker, S.; Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In Proceedings of the NSDI, San Jose, CA, USA, 25–27 April 2012.
51. Yuan, J.; Zheng, Y.; Zhang, C.; Xie, W.; Xie, X.; Sun, G.; Huang, Y. T-drive: Driving directions based on taxi trajectories. In Proceedings of the ACM SIGSPATIAL International Workshop on Advances in Geographic Information Systems, San Jose, CA, USA, 2–5 November 2010.
52. Yuan, J.; Zheng, Y.; Xie, X.; Sun, G. Driving with knowledge from the physical world. In Proceedings of the Knowledge Discovery and Data Mining, San Diego, CA, USA, 21–24 August 2011.