ST-Hash: An Efficient Spatiotemporal Index for Massive Trajectory Data in a NoSQL Database

Xuefeng Guan [1]*, Cheng Bo[1], Zhenqiang Li[1], Yaojin Yu[1]
[1] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 129 Luoyu Road, Wuhan 430079, China
*Corresponding author, e-mail: guanxuefeng@whu.edu.cn

Abstract—**With the development of positioning technologies and the increasing popularity of location-aware devices, large volumes of trajectory data have been accumulated. However, efficient management and access to massive trajectory data remains a big challenge. The emerging NoSQL database has provided a promising solution for this challenge. But most of the current NoSQL databases do not support direct spatiotemporal indexing of massive trajectory data. This paper presents a novel trajectory indexing method to accelerate time-consuming spatiotemporal queries of massive trajectory data. This method extends the widely-used GeoHash algorithm to satisfy the requirements for both high-frequent updates and common trajectory query operations, e.g. exact point query and spatiotemporal range query. This ST-Hash index was implemented and evaluated in a NoSQL database (MongoDB). Experimental results show that this proposed ST-Hash index can greatly improve the query performance and exhibits robust performance scalability over different input data sizes.**

Keywords: ST-Hash; Spatiotemporal index; Spatiotemporal range query; Trajectory data; NoSQL

I. INTRODUCTION

Currently, most mobile devices have positioning and wireless communication capabilities. They can continuously record device trajectories and dynamically report the locations to a remote server [1, 2]. General trajectory data are the traveling history of moving objects such as a person, a vehicle, or a flight. They have typical spatiotemporal characteristics. Each trajectory record usually contains 2D/3D coordinates, a timestamp associated with it, and a sequence of features, such as speed, direction, acceleration, temperature, etc. Trajectory data have been used for complex analysis across different domains, including environmental monitoring [1], traffic management [3], satellite image analysis [4], and homeland security.

With the pervasiveness of these geo-locating devices, the volume of spatiotemporal data has increased not only in quantities but also in scale. The large volume of trajectory data exceeds the current computation capacity of traditional centralized solutions. Efficient management and access to such massive trajectory datasets remain big challenges. The emerging cloud computing now provides a promising direction to master the explosion of trajectory data. For example, researchers and industry have begun to explore various NoSQL databases that can provide high performance, availability, and scalability [5]. However, many NoSQL databases lack a specialized indexing mechanism for spatial or spatiotemporal data, so NoSQL databases cannot support typical operations such as neighborhood or range queries in both spatial and temporal dimensions. Existing spatial indexes such as the R-tree family can be easily extended as spatiotemporal indexes where time is viewed as another dimension in addition to spatial dimensions. However, R-tree family methods are too complex to support concurrent operations in the NoSQL databases, e.g. high-frequency status update of thousands of floating cars [6].

In this paper, a novel indexing method called ST-Hash is proposed and implemented in a typical NoSQL database, MongoDB, to efficiently index and query massive trajectory datasets. This indexing method extends the idea of GeoHash and augments it with the temporal dimension. It encodes latitude, longitude, and time into a short and unique string. Different spatiotemporal queries are supported by these unique hash strings. A Web-service-based query interface is also designed to speed up the range query and facilitate client usage. Experimental results show that the ST-Hash method provides high query performance and exhibits robust performance scalability over different data sizes.

The rest of the paper is organized as follows. In Section 2, an overview of current spatial or spatiotemporal indexing methods is presented. Section 3 introduces the fundamentals of the ST-Hash indexing method, and describes how to transform a spatiotemporal point (latitude, longitude and time) into a ST-Hash string. In Section 4, two query procedures using ST-Hash strings are introduced. In Section 5, the evaluation results about query performance and scalability of this indexing method are discussed. Finally, Section 6 gives conclusions and directions for future research.

II. RELATED WORK

The management of massive trajectory data presents the following typical challenges: a) mass data volume storage; b) high-frequency inserting and updating operations; and c) diverse types of trajectory queries. Generally moving objects change their locations very frequently, accordingly the index of trajectory data should be updated frequently. Thus, update efficiency must be taken into consideration. At the same time, it is quite common to index TB-size trajectory datasets, so space utilization must also be considered in the index design.

Existing spatiotemporal indexes can be classified into three categories. The first category uses any multi-dimensional

This work is supported by the Natural Science Foundation of China (Grant No.: 41301411) and the Natural Science Foundation of Hubei Province (Grant No.: 2015CFB399).

search tree method like R-tree indexes with augmentation in the temporal dimension, e.g. 3D R-tree [6], or STR-tree [6]. This category can adaptively adjust index structures according to the data distribution to produce better query performance, but this adjustment degrades index generation performance [7]. The second category uses multi-version structures, including MR-tree[1], HR-tree[2], HR+-tree[3], and MV3R-tree[4].This approach builds an independent R-tree for each time interval and usually indexes the temporal dimension with B-tree. This type of index adopts a strategy that first considers the temporal dimension and then deals with the spatial dimensions. It has the advantage of high efficiency in temporal range queries. The third category relies on space-partitioning, such as the B-tree based indexes [5, 6] and grid based methods [7, 8]. Typical examples of this type of spatiotemporal index include the SETI [16] and MTSB-tree [17].

Another widely-used geo-encoding algorithm should also be mentioned here, called Geohash. Geohash effectively defines an implicit, recursive quadtree over the world-wide longitude-latitude rectangle and divides this geographic rectangle into a hierarchical structure. The division continues along the longitude and latitude directions alternately until the desired resolution is achieved. During each division, if the target coordinate value is greater than the division point, a '1' bit is appended to the overall set of bits; otherwise, a'0' bit will be appended. So each node of the recursive quadtree can represent a fixed spatial bounding box. Finally, GeoHash uses a 1D string to represent a 2D rectangle from a given quadtree node. The GeoHash string is derived by interleaving bits obtained from latitude and longitude pairs and then converting the bits to a string using a Base32 character map. For example, the point with coordinates of 45.557, 18.675 falls within the GeoHash bounding box of "u2j70vx29gfu". GeoHash has been widely implemented in many geographic information systems (e.g. PostGIS), and also used as a spatial indexing method in some NoSQL databases (e.g. MongoDB). GeoHash has some remarkable characteristics:

1) Uniqueness: Each GeoHash string has a unique corresponding rectangle on the earth surface, and vice versa.

2) Recursiveness: According to the GeoHash division rule, grid cells at the lower level are split recursively from the cells at the higher level. The grid cells in the same region but at different levels have recursiveness, meaning a shorter string represents a bigger space. Adding characters to the end of a GeoHash string specifies a smaller rectangle that is contained by the original one.

The GeoHash method can be treated as one of the space-partitioning based indexes but does not consider the temporal dimension. Indexes based on space partitioning surpass the other two categories of indexing methods in two ways [18]. First, space-partitioning based indexes usually transform multiple dimensions into one dimension. Currently, the 1D indexing methods, e.g. B-tree, are very mature and have been used in all commercial DBMSs. Thus, space-partitioning based indexes can be easily integrated into an existing DBMS. No additional work is required to modify the index structure, concurrency controls or the query execution module in the underlying DBMS. Second, GeoHash string generation only needs spatial coordinates of each point and does not involve other points. So when faced with high-frequency concurrent operations, space-partitioning based indexes are far superior to other spatial indexes, e.g. the R-tree family. The concurrency controls in the R-tree family indexes are too complex and time consuming, so they are not scalable for managing large volumes of moving objects. Several research works have shown the inefficiency of R-tree family in these circumstances. Jensen, et al. [19] demonstrated that the throughput of the TPR-tree [20] does not scale up in concurrent operations because R-tree based indexes hold the locks longer during updates. Guo, et al. [4] also confirmed that the preprocessing and tree optimization strategies employed in the TPR*-tree [9] result in extra locking delays due to preprocessing during insertions and hence reduce the query performance gain.

III. THE INDEXING METHOD OF ST-HASH

A trajectory of a moving object is a polyline in the 3D space, where two dimensions refer to geographic space and the third dimension refers to time. It can be represented as a time-stamped series of location points, denoted as $\{x_1, y_1, t_1; x_2, y_2, t_2; \ldots; x_n, y_n, t_n\}$; where x and y represent the geographic coordinates of the object, t indicates the corresponding timestamp of each position, and n is the total number of elements in the series.

GeoHash has been employed as an efficient indexing solution for massive 2D location data [8, 10]. But GeoHash only encodes the information of latitude and longitude. Our proposed ST-Hash method extends the idea of GeoHash, and it includes the temporal dimension besides spatial dimensions. The core of the ST-Hash method is the encoding process that convert the items from the augmented 3D data structure into a sequence of characters, i.e. 1D string. The complete process for generating the ST-Hash string is shown in Figure 1.

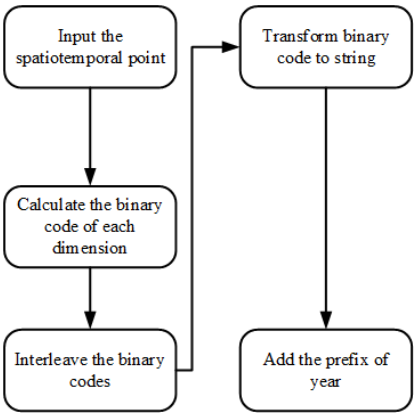

Figure 1. The flowchart of generating the ST-Hash string

Since the ST-Hash method defines a recursive octree on the temporally augmented world-wide geographic space, the maximum bounding box for this octree should be first established. The ST-Hash uses the following spatial and temporal extents:

Latitude and longitude: Since ST-Hash uses the WGS84 as the spatial references coordinate systems, the scope of

longitude is [-180°, 180°] while the scope of latitude is [-90°, 90°].

Time: Time is infinite and endless, while the 2D space has definite limits. Thus, a single year is divided in the ST-Hash encoding. There are two kinds of a single year, common year and leap year. The leap year contains 527040 minutes (366×24×60) while the common year has 525600 minutes (365×24×60). So the scope of a common year is [0，525600], and the scope of a leap year is [0，527040].

A. The binary code of one trajectory point

Just like GeoHash, the proposed ST-Hash indexing method divides the latitude, longitude and time respectively until the desired spatiotemporal resolution is achieved. The derived value in each dimension is transformed to a binary code. Because the transformation principles of the latitude/longitude/time are analogous, the longitude will be used as an example to illustrate the dimensional division and code transformation in the ST-Hash method, as shown in Figure 2.

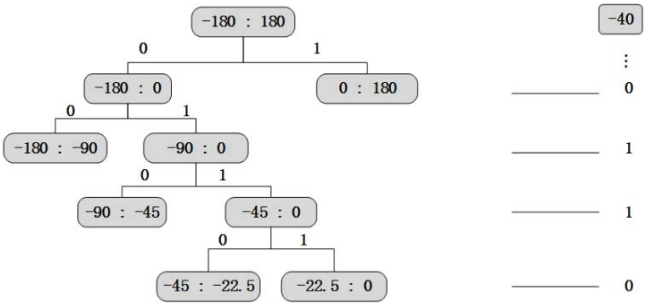

Figure 2. The binary tree built by four divisions

In Figure 2, the longitude of one input point is -40° and the division is carried out four times. As the total longitude scope is [-180°, 180°], the longitude is split into two parts ([-180°, 0°] and [0°, 180°]) in the first division. If the points belong to the left part [-180°, 0°], then the points are marked as '0', or marked as '1'. Hence, -40° belongs to [-180°, 0°], so the first binary bit is '0'. In the second division, [-180°, 0°] is split into [-180°, -90°] and [-90°, 0°]. Because -40° belongs to [-90°, 0°], the second binary bit is '1'. The third and the forth divisions continue in the same way. Thus, the final binary code is "0110".

Since each child node has an extent equal to half of the interval of the parent node, a more direct calculation is formulated as the following. Given that the total scope in one dimension is [X_{min}, X_{max}] and the total height of the final binary tree is h, the resolution of the leaf node will be r:

$$r = \frac{X_{max} - X_{min}}{2^h} \tag{1}$$

If the input value is X_i, the decimal code C_d can be derived as:

$$C_d = \left\lfloor \frac{X_i - X_{min}}{r} \right\rfloor \tag{2}$$

At last, C_d is transformed to a binary code C_b.

In the above-mentioned example, if the longitude Xi is -40°, X_{max}=180°, X_{min}=-180°, and h = 4. According to the Equation 1, r = 22.5° is obtained. Through the Equation 2, the decimal code C_d is equal to 6. Finally, C_d can be transformed to the binary code C_b = 0110.

This encoding process is applied to all three dimensions, and yields three bit sequences, where the number of bits in each corresponds to the number of levels in the tree. For example, if the input data is (-140°, 20°, 2015-6-1 00:00:00) and h is 10, then the input value of time '2015-6-1 00:00:00' will be transformed into a decimal value, 217440. The three derived bit sequences are:

longitude: 0001110001

latitude : 1001110001

time : 0110100111

Finally, the three binary codes of the input trajectory point along the longitude, latitude, time dimensions are interleaved into one long binary code. The three bit sequences in this example are interleaved, and the complete binary code of the given trajectory point is listed as the following:

010 001 001 110 111 110 000 001 001 111

B. The Base64 string of a trajectory point

The complete binary code of the given trajectory point is too long and cannot be directly stored in the target database. So the whole binary code is transformed to a Base64 string in a binary-to-text encoding schema to make it more convenient for database storage. During this encoding, six bits in the binary code are grouped into a corresponding character according to the Base64 map table. Thus the above-mentioned binary code in the Section 3.1 is transformed to a string "Re+BP", shown in Figure 3. The detailed code-to-string transformation is illustrated as follows.

Binary code :	**010001**	**001110**	**111110**	**000001**	**001111**
Value :	**17**	**30**	**62**	**1**	**15**
String :	**R**	**e**	**+**	**B**	**P**

Figure 3. The binary-code-to-string transformation process

If the length of the transformed string is l and the height of binary tree is h, then the relationship between the l and h will be

$$h = 2l \tag{3}$$

The Equation 3 indicates that each character in the ST-Hash string represents two levels in the octree. The resolution r of the octree leaf node in one dimension is

$$r == \frac{X_{max} - X_{min}}{2^{2l}} \tag{4}$$

If the spatiotemporal resolution r is known, the height h of the octree is obtained through the Equation 5:

$$h = \left\lceil \frac{ln((X_{max} - X_{min}) / (r / 2))}{ln(4)} \right\rceil \quad (5)$$

The string does not yet include the year information, so a prefix representing the year is appended to the string. Since the input data is (-140°, 20°, 2015-6-1 00:00:00), the prefix is "2015-". Finally, the complete ST-Hash string is "2015-Re+BP"

IV. THE QUERY INTERFACE FOR THE ST-HASH INDEX

The trajectory data and the generated ST-Hash strings are stored in a target database, and a B-tree index is created on the ST-Hash string field in advance. Both spatiotemporal point query and range query are developed based on these ST-Hash strings.

A. Spatiotemporal point query

The objective in a spatiotemporal point query is to find out a trajectory point whose 3D coordinates match exactly the input values along the longitude, latitude, and time dimensions. Because each point record is tagged with an ST-Hash string, the ST-Hash string is used to match point records. However, different points might have the same ST-Hash string, so the unrelated points should be filtered.

This point query process is described in Figure 4. If the tuple values of the query point (longitude, latitude, time) are given, the ST-Hash string of this point is generated. The database can be queried to find out which point records in the database match the generated ST-Hash string. Finally, the qualified records are filtered to obtain points whose values are exactly the same as the query point.

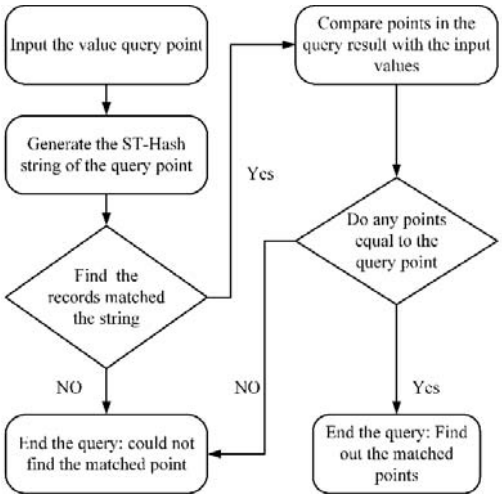

Figure 4. The flowchart of spatiotemporal point query

B. Spatiotemporal range query

The objective of a spatiotemporal range query can be stated as: Given a spatial extent E_s and a temporal extent E_t, the query Q (E_s, E_t) returns all those trajectory points S_k fully contained by the spatiotemporal cube defined by E_s and E_t.

$$Q(E_s, E_t) \rightarrow \{S_k = (p_1, p_2, \cdots, p_k)\}$$

Just like GeoHash, it is convenient to execute exact point query but inconvenient to execute spatiotemporal range queries with an ST-Hash strings. Hence, an additional algorithm was designed to support complex spatiotemporal range queries. Since the ST-Hash indexing method defines a multi-level octree, each node in this octree represents a cube in the spatiotemporal space and can be labeled by a unique ST-Hash string. The parent node at the upper level fully contains the child nodes at the lower levels, so the ST-Hash string of the parent node is derived by simply omitting the last character of a given string. When conducting a spatiotemporal range query, the client first inputs a spatiotemporal query filter defined by $\{x_1, y_1, t_1, x_2, y_2, t_2\}$. This input filter also can be represented as a cube in augmented 3D space; the filter cube is positioned by the left lower corner P_1 (x_1, y_1, t_1), and the right upper corner is P_2 (x_2, y_2, t_2), illustrated in Figure 5.

As shown in Figure 5, the core of the spatiotemporal range query can be simply depicted as the process to find a collection of octree nodes whose combination fully contains the input filter cube and then uses the ST-Hash strings of the found octree nodes to filter the point records. The node search should first determine the target octree level by matching the edge length of input filter cube with the node resolution at the different octree levels. Then the minimum containing node cube can be found at the target octree level. The found minimum containing node cube, however, might be much bigger than the input filter cube; therefore, the query process must be refined by intersecting its child nodes at the lower second level with the input filter cube. The ST-Hash string length at the lower second level node is exactly one character longer than that of the found minimum containing node cube. The number of the child nodes at the lower second level is 64, so the intersection results could be two or more cubes, as shown in Figure 5. Therefore this refinement step can narrow down the number of query strings, filter out much unrelated data more quickly and therefore accelerates query performance.

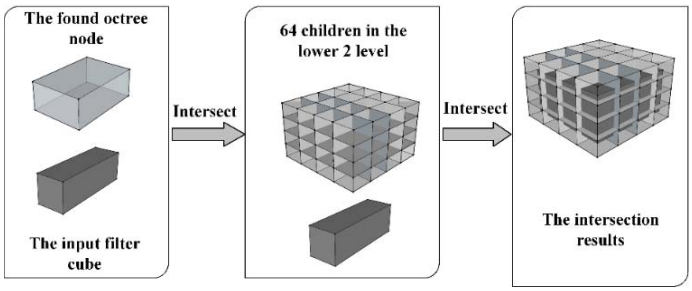

Figure 5. The intersection between the input filter cube and the 64 children nodes

The detailed range query process is listed as follows.

Step 1: Find the minimum octree node fully containing the input filter cube.

a) Obtain the three edge lengths of the input filter cube;

b) Calculate the node resolution r in each dimension that equals to the edge length, yielding r_{lon}, r_{lat} and r_{time};

c) Put r_{lon}, r_{lat} and r_{time} into the Equation 5, to get the minimum octree level for each dimension (h_{lon}, h_{lat}, h_{time}).

d) Compare h_{lon} , h_{lat} , and h_{time} to get the minimum value hmin ;

e) According to the eight corners of the filter cube, determine the required minimum containing node.

f) From this minimum containing node, the 64 child nodes in the lower 2 level are determined;

Step 2：Refine the collection of input query ST-Hash strings.

a) Check the intersections between 64 child cubes and the input filter cube to obtain the intersection results;

b) According to the intersection results, if the intersection number is less than 64, generate all the ST-Hash strings for the intersected child cubes; otherwise, if the intersection number equals to 64, go up to the parent node to generate the ST-Hash string for the parent node.

c) These generated ST-Hash strings in the step 2b are used to query the database to obtain the coarse result set.

Step 3：Filter the unrelated points from the coarse result set.

Based on the coordinates of the lower-left corner and the upper-right corner, the intermediate coarse results are filtered to obtain the final result set. All the points in the final result set must satisfy the filtering conditions.

C. *Spatiotemporal circle query*

The spatiotemporal circle query is extended on spatiotemporal range query in Section 4.2. The key point of one spatiotemporal circle query is to convert the input circle in the spatial dimension and the temporal range into an ST-Hash string set. The given circle spatial region is (x，y，R) and the given time range is (t_1, t_2), that x and y mean the latitude and longitude of the center of the circle, while the R is the radius of the circle. The detailed spatiotemporal circle query process is listed as follows.

Step 1: The (x，y，R) is used to generate the MBR of the spatial circle region. The lower left corner of the MBR is (x_1, y_1) and the upper right corner is (x_2, y_2). x_1 is got from $x_1=x - R$ and y_1 is got $y_1=y - R$, while x_2 is got from $x_1=x + R$ and y_1 is got $y_1=y + R$.

Step 2: The MBR, the given time range (t_1, t_2) can define the filter cube described in Section 4.2. The filter cube is positioned by the left lower corner P1 (x_1, y_1, t_1), and the right upper corner is P2 (x_2, y_2, t_2), illustrated in Figure 5. This filter cube will be used to conduct the spatio-temporal range query. This step will generate the raw result.

Step 3: Filter the raw result. Each point in raw result, will be calculate its distance d to the center of the circle. By the time range and comparing d with R, the final result set can be obtained.

D. *The web-service-based query interface*

During these three steps as described in Section 4.2, Step 2 and Step 3 are time consuming and I/O intensive. Since Web services are becoming a standard method for sharing data and functionality, a query web service is also implemented for clients to execute the different queries in order to reduce the query time, as illustrated in Figure 7. The user can use a development tool such as Microsoft Visual Studio to create a client application to access the query service.

First, the intersection results in the Step 2 can be a collection of discrete ST-Hash strings derived from the refined child octree nodes. The query service uses multi threads to accelerate the query. This collection of ST-Hash strings is spread among these independent threads to query the target database concurrently. Second, there are many unrelated points in the coarse results; these points must be removed. The query service resides in the same server with the target database. Therefore, shuffling the coarse results between the server and clients can be avoided. This greatly reduces the data transmission time and accelerates filtering.

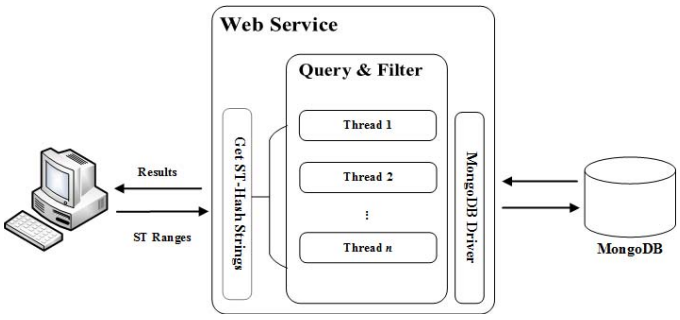

Figure 6. The architecture of the web-service-based query interface

V. PERFORMANCE RESULTS AND DISCUSSIONS

To evaluate the effectiveness and performance of the ST-Hash indexing method, two independent experiments were carried out on massive floating car data stored in a NoSQL database. MongoDB was chosen for the experiment because of the availability of GeoHash spatial index in it (i.e. 2D location index). One experiment was a query performance comparison between different indexing methods. The other was a scalability experiment to evaluate ST-Hash scalability over different data sizes.

The dataset used in our experiments was a GPS trajectory dataset collected for about two months from approximately 6,000+ electricity maintenance vehicles in the Fujian Province of China. There were about 60 million trajectory records in total; the data size was around 10GB. The data covered a swath of 115.8494~120.6702 in longitude and 23.5161~28.3702 in latitude. For the scalability experiment, the data was replicated from 10 GB to 80 GB, the three coordinates of the replicated records are slightly adjusted. The MongoDB database was deployed in the single-node mode on a physical Linux server. The server is equipped with one CPU (Intel Core I7-3770M 3.40 G), 8 GB RAM, and 1 TB hard disk (7200 rpm, 64MB cache). The server's operating system is CentOS 6.5 x86-64.

A. *The storage schema of trajectory data in the MongoDB database*

The MongoDB is a cross-platform document-oriented NoSQL database. In the MongoDB, a database holds a set of collections, a collection holds a set of documents, and then a document is a set of key-value pairs. In the following

experiments, the trajectory data were stored using only one MongoDB collection. Each trajectory point is encapsulated into one MongoDB document consisting of State-ID, latitude, time, longitude, and other attributes (e.g., such as direction, speed). The storage structure for trajectory data in the MongoDB database was listed in Table 1. Shown from the Table 1, State-ID is the primary key, which is composed of Taxi-ID and timestamp. Furthermore, a B-tree index on the ST-Hash field was created to accelerate the spatiotemporal query.

TABLE I. THE TABLE STRUCTURE OF TRAJECTORY DATA

Column	Type	Value
State-ID	String	Taxi-ID + Timestamp
TaxiID	String	Taxi-ID
STHash	String	ST-Hash string
Longitude	Double	Longitude
Latitude	Double	Latitude
Time	Date	Time
Attributes	JSON	Speed, Direction, …

B. The performance comparison on the spatiotemporal range queries

This experiment is a comparison of the spatiotemporal range query performance of the ST-Hash index and a composite index method. The composite method was realized with two independent indexes: a 2D location index on the Longitude and Latitude fields, and a B-tree index on the Time field. The following four query cases were tested: the spatial ranges were 0.01×0.01, 0.02×0.02, and 0.04×0.04, 0.08×0.08, respectively, and the time range were raised from 10 minutes to 120 minutes in each case. The experimental results for different range queries in different scales are illustrated in Figure 8.

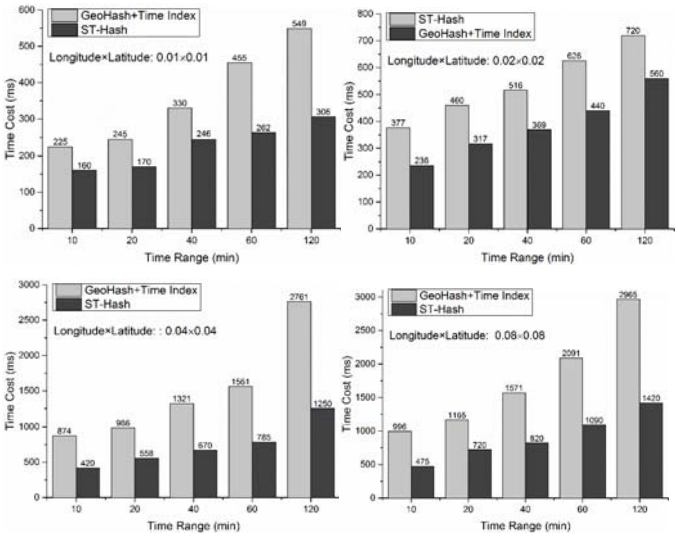

Figure 7. Time performance of different spatiotemporal range queries

Shown from Figure 8, the ST-Hash method performs much better than the composite index method. When the query ranges became larger, the returned query results also became larger, but the ST-Hash query time rose very slowly. In each case however, the query time of the composite index grew very quickly. This phenomenon occurred because the composite index could only filter one dimension at a time, i.e. spatial or temporal, and then filtered the other dimension; when the query ranges became larger, 1D query filtered much fewer records than those by the ST-Hash method. In contrast, the ST-Hash method queried all the records in both spatial and temporal dimensions at the same time and therefore filtered out unrelated data much more quickly.

C. The performance comparison on the spatiotemporal circle queries

This experiment was a spatiotemporal circle query performance comparison of the ST-Hash index and a composite index method. Spatiotemporal circle query also had four query cases：0.01, 0.02, 0.04 and 0.08, which also mean the MBR of query circle was 0.01×0.01, 0.02×0.02, and 0.04×0.04, 0.08×0.08. Just like the spatiotemporal range query comparison experiment, the time range was raised from 10 minutes to 120 minutes in each case. The experimental results for different range queries in different scales are illustrated in Figure 9.

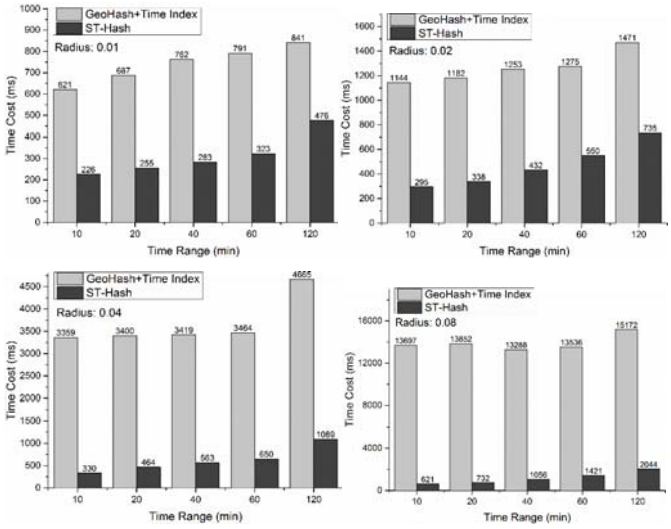

Figure 8. Time performance of different spatiotemporal circle queries

From the Figure 8, it is obviously that the time cost of composite index is far greater than that of ST-Hash. By comparing each case in Figure 7 with the corresponding case in Figure 8, the composite index cost more time to derive the final result; however, the time cost of ST-Hash was almost the same. Because spatiotemporal circle query was based on spatiotemporal range query, the time cost of ST-Hash would not change dramatically.

D. The scalability of the ST-Hash indexing method

This experiment was carried out to evaluate the ST-Hash scalability and demonstrated the relationship between query time and data size. In this experiment, all of the spatiotemporal range queries were executed with different data sizes. The data size rose from 10 GB to 80 GB. The query ranges used were the same as section 5.1. The experimental results for different

range queries on different data sizes were illustrated in Figure 8. Figure 8 shows that query time has little dependence on data size. Even when the data size was increased eight times, the query time remained stable. This stability could be explained by that the spatiotemporal range query was transformed into a search in the B-tree while the time complexity of the search in the B-tree was O (log n). Thus, these experimental results illustrate that our proposed ST-Hash indexing method has robust scalability over the input data size.

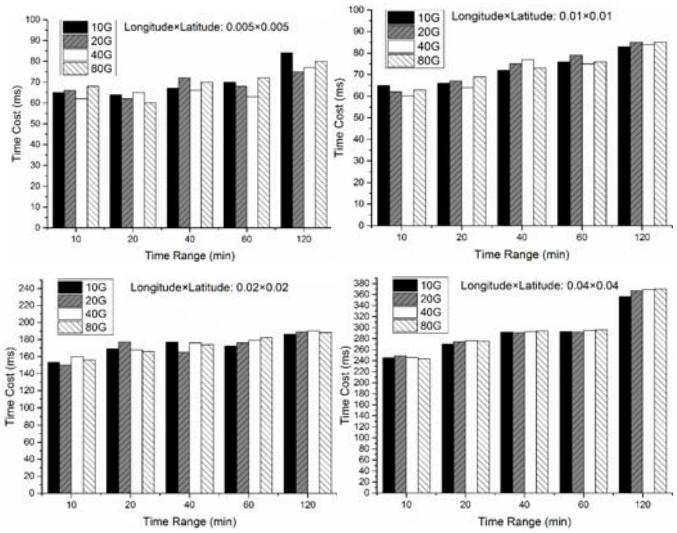

Figure 9. Time performance of difference input data sizes

VI. CONCLUSIONS

Management of large volume trajectory data is a challenging task in many fields. The emerging NoSQL database technology promises a solution to efficiently manage these massive trajectory data, but most of current NoSQL databases do not support the direct indexing method on spatiotemporal data. To address this, this paper presents a novel trajectory indexing method for accelerating the processing of spatiotemporal queries. This method extends the widely-used GeoHash algorithm to satisfy the requirements for high-frequency update and common trajectory query operations. The ST-Hash indexing solution was implemented in the MongoDB, a typical NoSQL database. Experimental results show that the proposed indexing method can greatly improve the query performance. At the same time, this method exhibits robust scalability over different input data sizes. When the size of target data increases from 10GB to 80GB, the query time remains steady. Future work will focus on more complex spatiotemporal query support, and evaluate the feasibility of the ST-Hash method in more complex scenarios, e.g. pattern mining, traffic simulation, etc.

REFERENCES

[1] L. Wang and Q. Xu, "GPS-free localization algorithm for wireless sensor networks," Sensors, vol.10(6), 2010, pp.5899-926.

[2] S. van der Spek, J. van Schaick, P. de Bois, and R. de Haan, "Sensing Human Activity: GPS Tracking," Sensors, vol. 9(4), 2009, pp.3033-55.

[3] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, "Spatio-temporal indexing in non-relational distributed databases," Proc. IEEE Conf. of Big Data, 2013.

[4] S. Guo, Z. Huang, H.V. Jagadish, B.C. Ooi, and Z.Zhang, "Relaxed space bounding for moving objects: a case for the buddy tree," Sigmod Record, vol. 35(4), 2006, pp.24-9.

[5] S. Ke, J. Gong, S. Li, Q. Zhu, X. Liu, and Y. Zhang, "A hybrid spatio-temporal data indexing method for trajectory databases," Sensors, vol.14(7), 2014, pp.12990-3005.

[6] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," Proc. of VLDB, pp.395-406, 2000.

[7] Long-Van Nguyen-Dinh and F. Mokbel, "Spatio-Temporal Access Methods: Part 2 (2003 - 2010)," Bulletin of the Technical Committee on Data Engineering, 2010.

[8] K.C. Kim and S.W. Yun, "MR-Tree: A Cache-Conscious Main Memory Spatial Index Structure for Mobile GIS," International Conference on Web & Wireless Geographical Information Systems, 2005.

[9] M.A. Nascimento and J. Silva, "Towards historical R-trees," ACM Symposium on Applied Computing, 1998.

[10] Y. Tao and D. Papadias, "Efficient historical R-trees," International Conference on Scientific and Statistical Database Management, 2001.

[11] Y. Tao and D. Papadias, "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," Proceedings of the 27th International Conference on Very Large Data Bases, 2001.

[12] D. Sacharidis and V. Kantere, "On-line discovery of hot motion paths," International Conference on Extending Database Technology, pp.392-403, 2008.

[13] S. Shekhar and S. Jin, "Processing in-route nearest neighbor queries: a comparison of alternative approaches," Proceedings of ACM International Symposium on Advances in Geographic Information Systems, pp.9 – 16, 2003.

[14] Z. Song and N.Roussopoulos, "SEB-tree: An Approach to Index Continuously Moving Objects," Lecture Notes in Computer Science, 2003.

[15] Y. Zheng, L. Zhang, X. Xie, and W. Ma, "Mining interesting locations and travel sequences from gps trajectories," Proc. of 2009 International World Wide Web Conf., pp. 791-800, 2009.

[16] V.P. Chakka, V. Prasad, C. Adam, A.C. Everspaugh, and J.M. Patel, "Indexing Large Trajectory Data Sets With SETI," Proc. Of International Conference on Innovative Data Systems Research, 2003.

[17] D. Zhang, P. Zhou, B. Salzberg, G. Cooperman and G. Kollios, "Close pair queries in moving object databases,". Proc. of the 13th Annual ACM International Workshop on Geographic Information Systems, 2005.

[18] S. Chen, B.C. Ooi, K.L. Tan, and M.A. Nascimento, "ST2B-tree: a self-tunable spatio-temporal b+-tree index for moving objects," Proc. International Conference on Management of Data, 2008.

[19] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and Update Efficient B-Tree Based Indexing of Moving Objects," VLDB. pp.768-79, 2004.

[20] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects," ACM Sigmod Record, vol. 29(2), 2000, pp. 331-42.

[21] Y. Tao, D. Papadias, and J.Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," VLDB, pp.790-801, 2003.

[22] Z. Balki and G. Horvat, "GeoHash and UUID Identifier for Multi-Agent Systems," Lecture Notes in Computer Science, vol.(7327), 2012, pp.290-8.

[23] J. Ježek and I. Kolingerová, "STCode: The Text Encoding Algorithm for Latitude/Longitude/Time," Connecting a Digital Europe Through Location and Place, pp.163-77, 2014.