Spatio-temporal Data Index model of moving objects on fixed networks using HBase

Nianbing Du, Junfeng Zhan
School of Computer Science and Engineering
Beihang University
Beijing, China
e-mail: dunianbing@nlsde.buaa.edu.cn, buaa_zjf@163.com

Ming Zhao, Daorui Xiao, Yinchuang Xie
School of Computer Science and Engineering
Beihang University
Beijing, China
e-mail: zhaomingbeihang@163.com, xiaodaorui_bj@sina.com, yinchuangxie@163.com

Abstract—**The advent and prosperity of the GPS equipped devices and reliable location technologies has resulted in a wide growth of location based service. As a certain type of geo-spatial application, moving objects on fixed networks must sustain high update rate for millions of devices, and provide efficient real-time querying on multi-attributes such as time-period and arbitrary spatial dimension. Traditional DBMSs support complex index structures which can effectively cope with spatio-temporal data. However, current relational databases have encountered the ever-increasing scale of datasets, which make a claim for scalability of data manage system. Meanwhile, key-value store databases are designed to be scalable, available and distributed, without much support for data organization including management of spatio-temporal data. In this paper, we present a novel hybrid index structure to organize data, combining a statistical based R-tree for indexing space and applying Hilbert curve for traversing approaching space. With key-value store, which insures effective querying response time and high insert rates, we propose rules for generating target row key which take skewed data handing into account. The cluster of HBase consists 8 nodes, with data volume in a level of millions. Our implementation proves that range queries and k-NN queries sustain response time in hundreds of milliseconds.**

Keywords-spatial index structure, GPS, moving objects, road network

I. INTRODUCTION

The explosive increase of sensor-equipped devices (GPS-enabled vehicles, RFIDs, etc) contributes to prosperity of geo-spatial systems. But for applications by which users register their location updates, smart systems embed positioning sensors also generate massive amounts of data updates and require data analysis on the basis of time and space. The Vehicle equipped with GPS location devices on road is a classic instance of moving objects on fixed networks. Research on indexing moving objects has been proceeded for years, such as 3D R-tree [1] and TB-tree [2]. However, all of them merely take Euclidean space into account, in which objects move without restriction. When moving objects travel on road networks, they have specific restriction, and lots of useless space would reduce query efficiency.

Besides, the existing methods have to sustain indexes for various query requirements in memory before conducting a certain retrieval request. Data volume increases rapidly, resulting quickly rising overhead of building query plan tree, and dropping in efficiency. On the other hand, NoSQL (Not Only SQL) databases, with features such as availability and scalability, become a more feasible solution for processing massive data. However, NoSQL databases do not provides complex data organization methods which supporting dimensional access of data freely. HBase [3], as column-oriented database based on key-value store, provides a fast data retrieval by means of a primary key and a data filter, which organize data attributes in a singular way.

In case of that, we propose a multi-dimensional mapping strategy based on key-value store using HBase, which support dimensional query requirements such as a spatial range query and k-NN query.

The rest of paper is organized as follows. Section 2 provides a sketch of existing multi-dimensional index structure. In Section 3, we briefly introduce the space-time mapping method and space indexing structure we propose. Afterwards, we describe query algorithms supporting dimensional queries. In Section 4, we apply the new index structure to real data, GPS data in road networks, and give an analysis of query performance. Lastly, in Section 5, we conclude the paper and points to future work.

II. RELATED WORK

A. Basic multi-dimensional Index structure

Quad-Tree [7] is a common tree structure for indexing multi-dimensional data. In a quad tree, each node corresponds to a square region in two dimensions and would be split into four subspaces if the number of data points exceeds a threshold value. Thus, quad tree is not a balanced tree, with deeper recursive in regions containing dense data sets.

R-Tree is a widely used structure designed for multi-dimensional indexing. Since R-tree is a balance search tree by dynamically splitting nodes and restricts the number of data points by controlling the boundaries of MBRs, it is a good way for processing skewed data.

978-1-4799-6023-1/15 $31.00 © 2015 IEEE
DOI 10.1109/CICT.2015.32

B. Cloud-based Index strategy

Unlike traditional RDBMSs, HBase organizes data in a three-dimensional cube, which respectively represent the row key, the column-family name, column name and version [6]. Data cells with multi-versions distinguished by timestamps are stored in the third dimension. HBase provides fast random access of every row with the condition of a single row key.

So far, there are several proposals for the organization of geospatial data in HBase. S. Nishimura et. al[12] built a multi-dimensional index layer on top of HBase to perform spatial queries. Ya-Ting Hsu et. al [13] presented a novel key-formulation schema, based on R+-tree for spatial index in HBase. Both studies investigate how to efficiently access the multi-dimensional data with spatial indices, which is part of the problem that we are addressing in this paper. Their methods demonstrate efficient performance with the spatial indices. However, they only focus on the design of data point which is independent from each other in the spatial dimension without connection relationship. To design an appropriate index structure/data model for temporal and spatial data, we need not only take into account the design of the row key, the column name and the version, but apply linearization technology in indexing process to harness HBase's most striking features.

III. MULTI-DIMENSIONAL INDEX STRUCTURE ON FIXED NETWORKS

HBase, as one popular of NoSQL databases, provides a simple fast access of data using a single key based on the key-value data model. As a result, we developed a multi-dimensional mapping strategyfor organizing moving objects on fixed networks, which helps range query and k-nn query on HBase.

A. Mapping method

To achieve good performance on NoSQL databases such as HBase, we have to carefully take their features into account. 1) The composition of row key is a decisive factor for query efficiency, so we must focus our attention on key structure; 2) To retrieve the same amount of data, a Scan is faster than several Gets, which means target data aggregated in similar positions perform well in range query; 3) As high concurrency is a significant characteristic in such systems, data needs to be distributed evenly on data servers in case of hot spot problem.

To achieve goal 1), the space information is placed before the time information when generating a row key in case of time hot spot, and encoding of them should be as short as possible. For goal 2), different levels of data aggregation granularity and the size of query window may has a balance in query performance. As for goal 3), a hash function could be used to distribute data into different servers.

A record of a moving object consists of objectID, time, speed, altitude and longitude. To locate specific records meeting requirements, certain information should be transformed into the unique row key for each record. Among all the information, space and time compose a three-dimensional data cube. Retrieving data with temporal-spatial clues is a common demand for location based services. As a result, Three-dimensional space-time information need to be mapped to a one-dimensional expression.

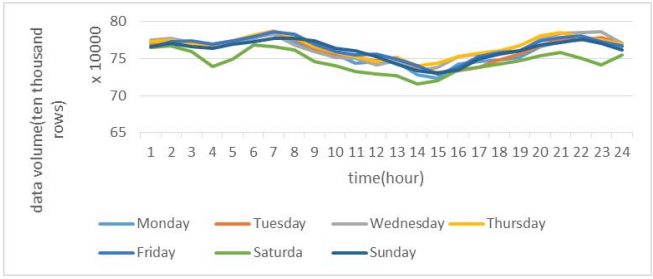

Figure 1. time sharing statistics of daily amount of taxi GPS data

First, as the statistic shows in Fig. 1, the time distribution of daily average data is presented highly historical regularity. The volatility of records varies little in one day with less ratio than 10 percent, and is basically the same among different days. Base on the analysis, data could be divided into blocks according time slice. Each block contains information of all the objects during the time period.

$$Tcode = \{T_1, \cdots, T_i, \cdots, T_k\}, 1 \le i \le k \tag{1}$$

$$Sum(T_1) = \cdots = Sum(T_i) = \cdots = Sum(T_k) \tag{2}$$

T_i is the number i time slice in each day which may be different from other time slice and we could give a code to represent it. $Sum(T_i)$ is the data volume in time slice , which is equal with data volume of other time slice. $Tcode$ records the time slice's code of daily data volume, to assure evenly distribution in time dimension.

What's more, the Fig. 2 shows that the space distribution of taxis GPS data is extremely skewed. It is not difficult to explain, as taxis and buses are concentrated in prosperous area where many people working or living. Meanwhile, to realize good retrieving performance of space data, we should develop an index strategy to make space data evenly distributed. Thus, spatial index to organize altitude and longitude on skewed data is essential.

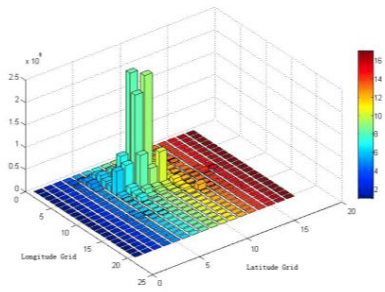

Figure 2. space distribution of daily data volume

Here we develop the Statistical grid-based R-tree index (SGR-Tree) index for space splitting and locating, and apply a mapping table to get local neighbors with Hilbert wave. With SGR-Tree, we get a space coding *Scode* for each record. Together with other information, we generate the row key.

$$rowkey = hash(time, nodeId) + hex(Scode, Tcode) + ObjecID \quad (3)$$

The row key is spliced by three parts, namely a hash prefix, a space-time coding and a detailed description such as *ObjectID*. The *hash*() function is used to allocate data to random data nodes. The *hex*() function will shorten the length of space-time code to control the length of row key. With the format of row key, data is stored into HBase and query is executed once we have the exact or part of row key.

Besides, to achieve k-NN query, we use Hilbert curve for traversal of local rectangles to maintain the locality of data. The Hilbert value is calculated from coordinates of rectangle's center point and a mapping table is maintained for neighbor finding. A record in the table is shown below.

$$MappingRecord = \langle hilbertValue(longitude,\ altitude), MBR\langle Scode\rangle\rangle \quad (4)$$

To accelerate the speed of data retrieving, we keep the mapping table in memory for fast access. Specific information about and *MappingRecord* is introduced in Section B.

B. SGR-Tree Index

The overall architecture of the SGR-Tree index is described as shown in Fig. 3. It consists of three parts. The first part is a fine-grained grid network by which data volume in certain time slices is calculated. With the result we arrange the leaf node of R tree. Meanwhile, little grids in a leaf node will make greater control of data location. The second part is an R tree indexing spatial space according each grids' data volume. A rectangle unit called MBR is used for indexing moving objects on fixed network as a leaf node of R tree. Each leaf node encircled data with the same data volume. The third part is an outer grid network which used for k-NN query by locating the relative position among the MBRs. It is divided into 2^n * 2^n tiles for traversal with a Hilbert wave. In this way, coordinates of each MBR's center point locate in the outer grid network and each MBR corresponds to a Hilbert value.

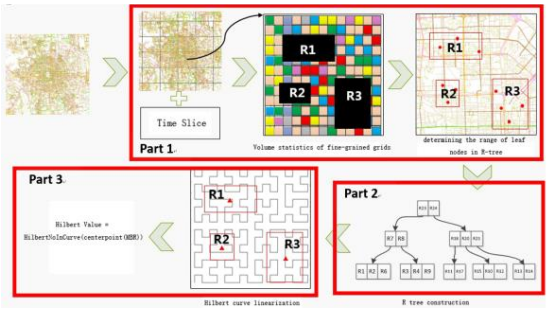

Figure 3. the overall architecture of SGR-tree index

The distribution and the size of MBR is determined by historical statistics. Base on it, the R tree is constructed. For shorter row key, the hexadecimal is used for encoding and each MBR can be represented by a one-dimensional string. That is,

$$RNode(id, list\langle RNode\rangle childs) \quad (5)$$

$$Scode = parentcode + selfcode(RNode) \quad (6)$$

The node of R-tree is represented by formula 5, and is composed of parent node's code and current node's code. With a query window, we could search the R-tree to locate specific leaf nodes. Together with other information mentioned in Section A, records in HBase could be retrieved.

Meanwhile, a mapping table for grid and MBR is maintained for data access in k-NN query. As seen in Fig. 3, the Hilbert value of outer grids correspond to the encoding of MBR. What's more, the finer division information is recorded in the MBR object, supporting precise positioning.

C. Query Processing

In this section, we introduced exhaustive description of query algorithm, including insertion, range query and k-NN query.

1) Range Query

Range query is a common application in multi-dimensional data processing. Given the coordinates of a scope of circle or rectangle, a data set of trajectories located within the scope is returned. Here we take query over rectangle scope as a case for descripting. Algorithm 2 is the pseudo code for range query in HBase. (pl, pu) is the range for the query that pl is the lower bound and pu is the upper bound. Firstly, the algorithm searches the R tree to get all the overlap MBRs with the target scope. And after that, we get a series of row keys for further data scanning. Then, finer scope within a MBR overlap with target scope is calculated for precise positioning. Last, the function GetCertainedData () gets data through a row key containing space and time information by accessing HBase. In this progress, Coprocessor is used for data filtering in the region server side, avoiding data transportation to the client.

Algorithm 2 Range Query

Input: pl, pu, time: the rectangle scope of the query
Output: data contained in the range

```
1: Set<Record> result ← NULL;
2: Set<RowKey> srk ← NULL;
3: Set<MBR> smbr ← NULL;
4: Stack sta ← NULL;
5: TimeEncode← GetTimeEncode(k);
6: sta.push(rt);
7: while(!rt)
8:     Node tn← sta.pop();
9:     for Node child in Node tn do
```

```
10:      sta.push(child);
11:    end for
12:    if(tn.constain(altitude,longitude)&&!tn.hasChild)
13:       smbr ← tn.MBR;
14:       Break;
15:    end if
16: end while
17: for each MBR mbr in smbr do
18:    FinerCode = GetOverlapwithMBR(pl,pu,mbr);
19:    SpaceEncode ← GetSpaceEncode(smbr, FinerCode);
20:    srk ← HashFunc()+SpaceEncode+TimeEncode;
21: end for
22: for each RowKey rk in srk do
23:    result ← GetCertainedData(rk);
24: end for
25: return Results;
```

2) kNN Query

The K-Nearest Neighbor (kNN) queries aim to fetch a number (k) of data units near to a given location. Algorithm 3 shows the k-NN query algorithm in HBase. Point p is the target center in our query. SetK stores the result k nearest neighbors, distance is the range for MBR search, MBRAll is a queue that stores the MBRs would be scanned, MBRScanned stores the MBR that had been scanned. Procedures of kNN query is that: firstly, we find the MBR overlap with point p, get the records belongs to this MBR and store them into a set sorted by distance between each record point and point p. In this time, if the number of records founded is beyond k, we terminate the query and return the records with minimum k distance; if not, we find the maximum distance and set it to variable distance. With the scope surrounded by this distance and point p, we recalculate the MBRs overlap with it. Further query will be continued on the newer MBR found until we found the target k records. The function MBRLocate() determine all the MBRs overlap with target scope according the Hilbert wave, for better locally query.

Algorithm 3 k-NN Query

Input: k: parameter k for nearest data points; p = (x, y): coordinates of a random point within a data MBR

Output: k nearest neighbors of (x, y)

```
1: SetK ← null;
2: distance ← 0;
3: MBRAll ← null;
4: MBRScanned ←null;
5: loop
6:    If MBRAll == null then
7:       MBRNext ← MBRLocate(p, distance)–MBRScanned
8:       For each MBR mbr in MBRNext do
9:          Push (mbr, MinDist(p, mbr), MBRAll);
10:      End for
11:   End if
12:   MBR ← Pop (MBRAll);
13:   For each Point m in MBR do
14:      SetK←SetK U<m,Dist(p,m)>and sort SetK by distance;
15:   End for
16:   If dist( k-th point in SetK, p) <= distance then
17:      Break;
18:   End if
19:   MBRScanned ← MBRScanned U MBR;
20:   distance ← Max(dist, MaxDist(p, R));
21: End loop
22: Return SetK;
```

Algorithm 4 MBRLocate(p, distance)

Input: p = (x, y): query point as the centroid of scan scope; distance: the radius of target scan scope; list: the list of sorted Hilbert values for MBRs.

Output: the MBRIdentifiers of MBRs overlap with the target scope.

```
1: MBRId ← null;
2: xl ← (x-distance/2) mod list;
3: xu ← (x+distance/2) mod list;
4: yl ← (y – distance/2) mod list;
5: yu ← (y + distance/2) mod list;
6: for i=xl → xu do
7:    for j=yl → yu do
8:       OuterGridID← OuterGridID U Hilbert(i, j);
9:    end for
10: end for
11: return MBRId ← KeyTable(OuterGridID);
```

IV. EXPERIMENTAL RESULTS

Our experiments were performed on an 8-node cluster, with one name node and 7 data nodes composing a YARN platform. The servers run 64bit CentOS 6.2.0 and have 4 cores, 32GB of RAM and a 200GB disk for each. The YARN version is 2.2.0, and the HBase version is 0.94.12. Each node of the cluster is allocated 2GB of heap size for data processing. Replication factor of HDFS is set to 3. An independent Zookepper cluster is configured for HBase communication and synchronization.

In the experiment, we adopted gzip compression to reduce data transporting time, and column filter was applied for data pruning. The cache size of scan was set to 5000 and the block cache was enabled after considering the size of each row and the heap size of each nodes. Besides, minor and major compaction were done before conducting queries to avoid occupancy of resources.

For all test cases, we repeated for 5 times and recorded the mean value. We implemented the range query with the Coprocessor framework and k-NN with Scan.

In our experiments, we used two kinds of data sets, a uniform one generated by program and a skewed one came from a real dataset, which records the points of taxis moving on Beijing road network. The number of records is 1 million.

TABLE I. RANGE QUERY(SECONDS)

Km^2	0.1*0.1	0.5*0.5	1*1	1.5*1.5	2*2
Data volume of each MBR:10000	1.865	6.487	14.553	16.360	18.916
Data volume of each MBR:50000	1.503	4.594	11.726	12.466	14.831

Data volume of each MBR:100000	2.312	6.858	15.647	17.904	20.544
ScanDB	198.45	197.30	198.22	197.71	197.97

TABLE II. K-NN QUERY(SECONDS)

km2	0.1*0.1	0.5*0.5	1*1
Data volume of each MBR:10000	3.712	9.148	15.831
Data volume of each MBR:50000	2.443	7.472	14.263
Data volume of each MBR:100000	4.773	9.836	16.479
ScanDB	198.76	198.33	197.42

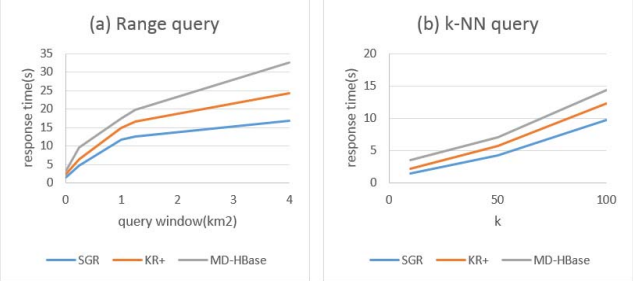

Figure 4. Uniform data distribution

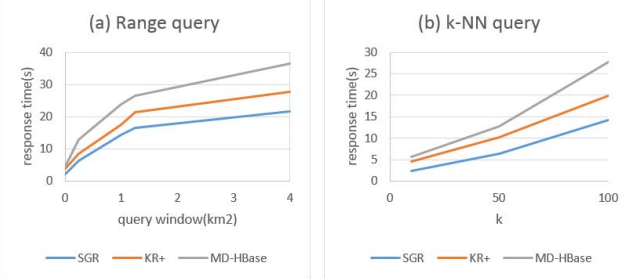

Figure 5. Skewing data distribution

Performance analysis of range query and k-NN query over parameter on real data set is shown in Table I and II. The statistical grid size is 0.1km×0.1km. Comparison about SGR on parameter is conducted with different data volume in each MBR. Besides, the time efficiency of scanning the database is put for a reference. It needs to scan all the data and filter out the mismatched ones. As we have seen, the SGR is much faster and perform best when the parameter is 50 thousand. The SGR with 50 thousand records in each MBR of range query for 0.1km×0.1km is about 1.5s and k-NN query for k = 10 is about 2.4s.

We compare our method SGR with KR+ and MD-HBase for skewing data and uniform data. Figure 3(a) and Figure 3(b) are range query for uniform and skewed data. SGR improves a little efficiency comparing with KR+ and MD-HBase for uniform data. SGR with 50,000 points in one MBR for 0.1km×0.1km range query is about 1.5s and k-NN is about 2.4s as k = 10. In addition, with skewing data, in Figure 4(a) and Figure 4(b) show the SQR is relatively faster than KR+ and MD-HBase. As SGR balance the number of false-positive (Scan) and the number of sub-queries (Get) better, it improves the efficiency of range query and k-NN query in a certain extent. SGR for range query 0.1km×0.1km is about 2.1s and k-NN query as k = 10 is about 2.4s. However, KR+ for range query 0.1km×0.1km is about 3.8s and k-NN query as k = 10 is about 4.5s. The difference between the methods becomes more significant when the query window or the k value is bigger. The result of evaluation shows our SGR has much better performance for range query and k-NN query. Besides, SGR is based on historical statistics. As a result, it has a better adaptive capability than the others, especially for skewing data

V. CONCLUSION

We proposed a new multi-dimensional index for moving objects on fixed network, based on a now existing NoSQL database, HBase. It supports efficiently multi-dimensional range queries and k nearest neighbor queries. We used R-tree to construct spatial network and applied Hilbert linearization for the mapping to the target leaf node of R-tree for formulating row key of query. The experiments showed that our index structure performed effectively compared to other methods, especially when using skewing data.

[1] M. Vazirgiannis, Y. Theodoridis, and T. Sellis, "Spatio-temporalIndexing for Large Multimedia Applications.", Proceedings of the IEEE Conference on Multimedia Computing and Systems6(4), pp. 284-298 ,1998.

[2] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approach to the Indexing of Moving Object Trajectories.", Proceedings of International Conference on Very Large Data Base, pp 395-406, 2000.

[3] A. S. Foundation, "Apache HBase Reference Guide," April 2012. [Online]. Available: http://hbase.apache.org/book/book.html

[4] R.A. Finkel and J.L. Bentley. Quad trees a data structure for retrieval on composite keys. Acta informatica, 4(1):1–9, 1974.

[5] Alshaer J. J. and Gubarev V. V., "Indexing Moving Objects on Transportation Networks," The Third International Forum on Strategic Technologies, pp. 249-252, 2008.

[6] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4, 2008.

[7] S. Nishimura, S. Das, D. Agrawal, and A. Abbadi, "MD-HBase: A scalable multi-dimensional data infrastructure for location aware services," in Mobile Data Management (MDM), 2011 12th IEEE International Conference on, vol. 1. IEEE, 2011, pp. 7–16.

[8] Y.-T. Hsu, Y.-C. Pan, L.-Y. Wei, W.-C. Peng, and W.-C. Lee, "Key formulation schemes for spatial index in cloud data managements," in Mobile Data Management (MDM), 2012 IEEE 13th International Conference on, 2012, pp. 21–26.

[9] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. Information Theory, IEEE Transactions on, 15(6):658–664, 1969.

[10] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. IBM, Ottawa, Canada, 1966.

[11] A.R. Butz. Convergence with hilbert's space filling curve*. Journal of Computer and System Sciences, 3(2):128–146, 1969.