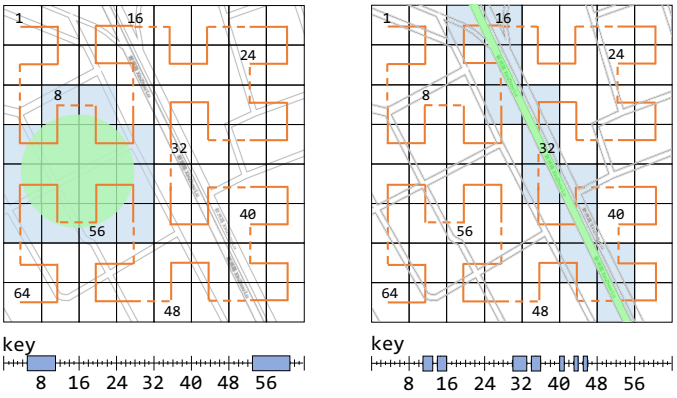
A Road-Aware Spatial Mapping for Moving Objects

Xingsheng Zhao*, Jingwen Shi†, Mingzhe Du†, Fan Ni*, Song Jiang*,Yang Wang†
*Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, USA
†Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
*xingsheng.zhao@mavs.uta.edu, †{jw.shi, mz.du}@siat.ac.cn, *fan.ni@mavs.uta.edu
*song.jiang@uta.edu, †yang.wang1@siat.ac.cn,

***Abstract*—The Internet-of-Things (IoT) attracts great attention in the past few years. With millions of devices connected to the network, data are generated at an unprecedented speed and the data must be stored efficiently in the database to serve spatial queries. In existing spatial databases that use space-filling curves to organize the data, they store spatial data without considering on-road data distribution. This will introduce unnecessary computation and I/O cost in the service of users' queries about data on the roads. In this paper, we present a Road-Aware Spatial Mapping of data to the storage, or RASM for short, which can be applied in spatial databases for highly efficient storage and query services for moving objects. Usually, a space-filling curve, such as the Hilbert curve, is used to map data in a cell of a geographical area to a segment of linear storage space. However, in a road-network system where data are most distributed and queried along the roads, using a generic square cell as a mapping unit to aggregate data is in conflict with the data use pattern. In RASM, road segment, instead of the cell, is used as the unit of space mapping and data storage so that data requested in a road query can be stored together to enable efficient I/O. Furthermore, a substantial computation may be required to identify mapping units covered in a query in a geometric space. As RASM has grouped data in the road-segment units, one can efficiently found the units covered in a road query, which is usually concerned only about data on a few segments of roads. We implemented a prototype query-serving system using RASM to map data on road segments to a linear space enabled by LevelDB, a widely-used key-value store. Experiment results with real-world traffic data show that with RASM, the road query time can be reduced by up to 43%, and the I/O traffic can be reduced by up to 70%. In the meantime, other queries about geographical regions are well supported in RASM with minimal performance impacts.**

I. Introduction

Internet-of-Things (IoT) has been widely deployed over the past few years. Over 57 percent of companies have already adopted the technology, and by 2019 that number is expected to reach 85 percent [1]. With millions even billions of GPS-enabled devices connected to the Internet [2]–[5], the total amount of spatial data produced by those devices becomes significant. For example, 15 million rides are generated by Uber every day, and there are over 5 billion records in their database [2]. Therefore, those data must be stored efficiently in the database to serve spatial queries. Space-filling curve [6]–[8] is a commonly used technique to determine what data should be stored together to serve queries efficiently. When a space-filling curve maps spatial data to a one-dimension (storage) space, its objective is to make the data in a geographic region covered in a query be stored together on the storage.



(a) Query within a circle area (b) Query on a road

Fig. 1: The 8×8 space is mapped to a one-dimensional storage space using a Hilbert curve represented by the orange polyline. In particular, a cell in the grid is mapped to a block on the line representing the disk space. An example query is concerned with a geometric region in green, and the cells overlapping the region are mapped to multiple blocks (in blue) on the line.

Among many proposed space-filling curves, such as Z curve [9] and Peano curve [10], Hilbert curve [11] is often the preferred one as it achieves the best clustering effect [12], [13] to enable efficient I/O operations during service of queries. However, the Hilbert curve shows inadequacies in serving queries on data about moving objects on the roads with compromised I/O and computation efficiency.

Weakened spatial locality of accessing data for querying moving objects on roads. A geometric surface is usually divided into $m \times m$ cells for the Hilbert curve to map and arrange the spatial data on the disk. These cells are numbered by the visiting sequence of the Hilbert curve. To serve a *region query*, which covers a geographic region (e.g., finding all yellow taxies within one mile), all the cells that intersect with the region have to be retrieved. While the curve determines the distribution of the cell data on the disk, a region of cells can be mapped to multiple discontiguous segments on the disk, and accordingly a query on a geographic leads to multiple range scans, one for a segment of data, on the disk. As illustrated in Figure 1a, a query for data within a circle is translated into

978-1-5386-6808-5/18/$31.00 ©2018 IEEE

two range scans (scans in [6 .. 11] and [53 .. 60]). For the sake of I/O efficiency, the fewer the scans on the disk, the better.

For moving objects, or vehicles on the run, the data are mostly located on the roads, and most queries are about data on the roads, such as retrieving the trajectory of an Uber ride or collecting the traffic data at a crossroad. The type of spatial queries is named *road query*, in contrast with the generic region query. With the help of the Hilbert curve, region queries can usually receive good performance as the access locality among objects in a two-dimensional region can be retained when the corresponding data are stored on the disk. However, the roads involved in a road query usually intersects multiple cells which are not concentrated in a small region in the two-dimensional space. Accordingly, with current use of the Hilbert curve the data on the roads are not stored together on the disk. As illustrated in Figure 1b, using the Hilbert curve makes the data about the objects on a short segment of a road be scattered in the linear space, requiring many range scans (7 in the example), or many disk-seek operations on a hard disk to serve the road query. This can significantly compromise I/O efficiency and degrade the queries' service quality.

Retrieval of irrelevant data in the service of road queries. The entire geometry surface is organized as a quad-tree to facilitate the search for data covered in a query. Each node of the quad-tree is a cell and each level corresponds to a space partitioning resolution. At each level of the quad-tree, the Hilbert curve is applied to encode the cells. Accordingly, data are partitioned into different groups in the quad-tree and then mapped to the disk. All spatial queries, including region queries and road queries, are served by searching the cells that overlap with the target region or road(s) on the geometry surface at a given resolution. This searching method is named *region cover* [14].

For road queries, the region cover method can cause significant read amplification as all the data in the overlapped cells are retrieved from the disk which can include a substantial amount of irrelevant data. A cell may intersect some road segments which are not related to our query requests. To serve a read query, though only data on the relevant road segments need to be read, all data in the cells intersecting with the segments are actually read. Additionally, the storage system may require reading extra data, including that due to a disk's block access interface. Instead of directly managing data on the disks, we use LevelDB [15], a key-value (KV) store based on Log-Structured Merge Tree (LSM-Tree) [16], to store and retrieve data to minimize performance impacts of random writes. The ratio between the amount of data retrieved from the KV store and that actually used to serve a query quantifies the read amplification in the query service and represents a source of overhead.

With a lower resolution, or larger cells, a larger area will be included and more data are returned. For example, at the given resolution 50, the read amplification is about 170%. With a higher resolution to cover a road, the total area of the cells becomes smaller, and accordingly fewer irrelevant data will be retrieved to serve a query. Although increasing the resolution can reduce irrelevant data retrieved from the storage, it can compromise the I/O efficiency because skipping the non-relevant data introduces more range scans. Figure 2 shows average read amplification of serving road queries with different resolutions for covering a road. In the measurements, lookup requests (through SEEK interface) are sent to LevelDB to search for all objects on 4934 roads.

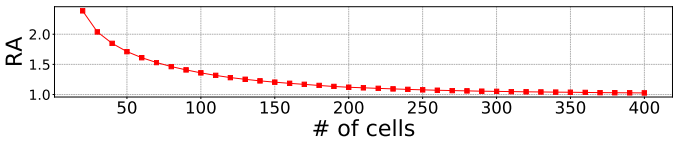


Fig. 2: Read amplification of road queries on real-world traffic dataset with different resolutions. The data are from 60 million taxi trips collected in Shenzhen, China. Here resolution is denoted by the number of cells. With more cells for covering the road, a more precise covering can be used to depict the road.

The computation cost for serving spatial queries. In addition to I/O inefficiency, existing geospatial storage systems also pay unnecessary computation cost when serving road queries for moving objects under high resolution. This cost is introduced for computing region covers, or finding the smallest set of cells to cover a query region using a recursive algorithm. The time complexity is proportional to the given resolution. The algorithm will be discussed in the next section. This road-to-cells cost is non-negligible for road queries, and we will show the results in the evaluation section. For road queries, if we can organize data based on road segments, it may not be necessary to search for cells to cover data on the roads, and the corresponding cost can be reduced.

The Road-Aware Spatial Mapping Approach. In this paper, we present a Road-Aware Spatial Mapping approach, or RASM for short, to provide highly efficient data storage and management for moving objects. To improve I/O efficiency, minimize retrieval of irrelevant data, and reduce the computation cost in the service of road queries, RASM does not use the fixed cells as the base unit for the Hilbert curve to visit for mapping. Instead, the base unit is replaced with road segment, and each segment has a unique encoding number, named segment ID. All spatial data about the moving objects are attached to their corresponding road segments, whose ID serves as the key for data storing and querying. This design is motivated by two observations. First, to serve spatial query efficiently, the data distribution should fit the data access pattern which is determined by the query pattern. Second, in addition to the region query that searches for the data in a geometric area, most queries for moving objects are road queries conforming to the road network distribution. Therefore, to improve service efficiency of road queries, we should utilize the road network information in the data storage so that access locality of data on a road segment can be retained in the linear space. Since RASM uses the Hilbert curve to encode the road segments, the locality of nearby

segments is also retained in the linear space. Meanwhile, RASM serves road queries directly by using the segment ID without using the region cover algorithms, so the computation cost is also reduced. With RASM, road queries become more efficient and the performance for region-based queries is not compromised.

In summary, we make three major contributions in this work:

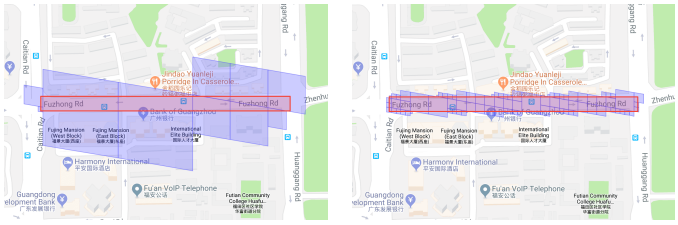
- We reveal inadequacies of existing application of the Hilbert curve in processing road queries for moving objects.
- We propose a new approach of data organization that considers the fact that data about moving objects are associated with the roads. With this road-aware local data organization, the Hilbert curve is enabled to lay out data on the storage device to support efficient spatial query processing.
- We implement a prototype of the design using LevelDB, a widely-used key-value store, as its underlying storage system, and conduct extensive experiments on real-world traffic datasets. The results show that with RASM, the road query time can be reduced by up to about 43%. Further, this improvement of road query efficiency does not come at the cost of performance and accuracy of region queries.

II. Background and Related Work

This section presents briefly background knowledge of RASM, including geospatial index structures and region query translation algorithm. Some works related to the paper are also described.

A. Indexing spatial data

The method to index spatial data can be categorized into two types, namely data partitioning and space partitioning based indices [17]. Data-partitioning-based indexes, such as R-Tree [18] and its variations R*-Tree [19] and STR-Tree [20], group spatial data with a Minimum Bounding Rectangle (MBR), where MBR acts as a node in a tree data structure. As areas covered by different MBRs may overlap, serving spatial queries can be expensive since a geometric area may intersect with many MBRs and all data in these MBRs need to be retrieved. This would cause high read amplification. The read amplification can also be excessive in R-tree because the MBRs are generated based on the data distribution, and can be very large with a low data density. Space-partitioning-based indexes, such as Quad-Tree [21] and Grid [22], partition geospatial space into non-overlap cells. Geohash and space-filling curves [10], [11] are then used to encode those cells with unique IDs. Spatial data that fall into the cell will carry the cell ID. Spatial queries for moving objects in those indexes suffer from the similar problem with data-partitioning-based indexes. Under low resolutions, the read amplification can be high, while under high resolutions too many range scans or disk-seek operations can be produced, which may become a major performance bottleneck.



(a) Max # of cells: 10 (b) Max # of cells: 60

Fig. 3: Region Cover Under Different Resolution. The red rectangle is our target region of a spatial query. The blue rectangles are the cells chosen by region cover algorithm to cover the target region. With a higher limit of cell number, a more precise covering can be found.

B. Region Cover Algorithm

For the service of spatial queries in a geospatial storage system, a region cover algorithm is used to find all the cells that overlap with the input geometric area. The Google S2 [23] library provides an implementation of the region cover algorithm [24], in which the entire earth surface is mapped to the six faces of a cube. Each face is split into cells of different sizes in the form of quad-tree with 30 levels. And the cell size varies from 7842km to 8mm [25]. To find the optimal covering of a shape, the algorithm first checks the six faces and discards ones that do not intersect with the area of interest. Then it repeatedly chooses the largest cell that intersects the area and subdivides it. A priority queue is maintained to store the cells, which is called candidates, that partially overlaps with the area. Candidates are prioritized by the cell size (larger cells first). Cells that do not intersect the area are discarded while cells fully contained within the area are added to an output buffer. The process continues by recursively subdividing the top candidates in the queue until the sum of candidates size and output buffer size reaches the limit of maximum cell size or the priority queue is empty. Then all the candidates in the priority queue will be added to the output buffer. In the end candidate cells in the buffer collectively represents the identified coverage of the area of interest. The larger the maximum cell number is, the higher the covering resolution is. Figure 3 illustrates the optimal coverage of a rectangle area under different resolution. Figure 3a shows a case where 10 cells are used to cover the rectangle region, and in Figure 3b 60 cells are used. Using fewer cells leads to including significant amount of non-relevant areas in the coverage. Though increasing number of cells can address the issue, it will make the algorithm more expensive as the computation cost is proportional to the cell count.

C. Efforts on improving I/O efficiency

Some studies have been focusing on improving I/O efficiency of the geospatial index. QUILTS [26] proposes a cost model on a specific query pattern, where the optimal space-filling-curve-encoding method is proposed to minimize the number of page accesses for that pattern. However, the

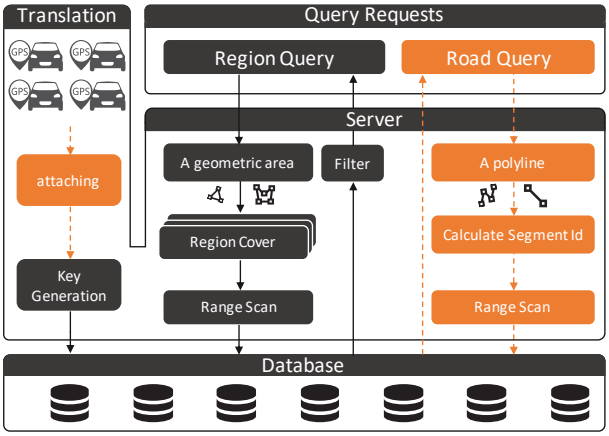
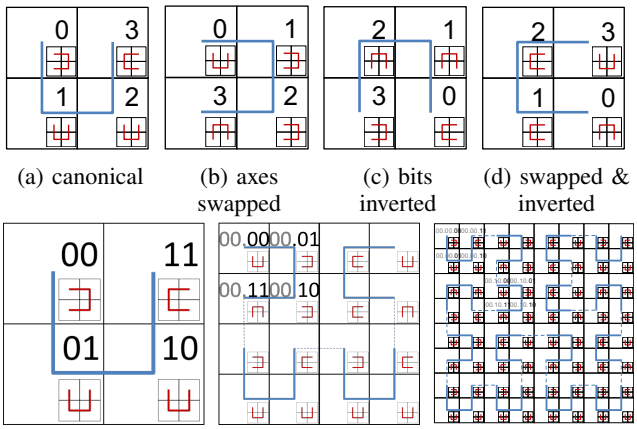


Fig. 4: Overall structure of the RASM-based storage system. Raw GPS coordinates are attached to road segments and then translated to one-dimension keys based on the Hilbert curve. User issues geospatial query requests by specifying road names or geo-regions. Those requests are translated to geometrical shape (polygon or polyline).



(e) The first, second, third order Hilbert curves

Fig. 5: Four unit in Hilbert encoding

method is less practical for real uses because it is difficult to identify a fixed query pattern or a fixed rectangle region. This is especially the case for spatial queries for moving objects, where the queries are based on complicated road networks.

Pyro [27] proposes an adaptive aggregation algorithm to apply interpolation to fill in the gaps between two position reads from a disk to minimize disk seek cost. Pyro reduces the read latency at the block level by allowing higher read amplification. However, filling the gaps between continous reads is equivalent to using large cells in region cover, which can cause high read amplification and increase query time due to retrieval of a large amount of non-relevant data. Pyro only considers the query in a square geometric space, so their method is not effective for serving road query.

D. Leveraging of Road Network

There are some studies on leveraging road network information in the geospatial index. [28] proposes an efficient k-nearest-neighbor (KNN) query algorithm on moving objects by storing data in the road network graph. Luo et al. [29] build a distributed index to conduct spatial-keyword queries on road networks. G-Tree [30] is proposed as a consistent framework to support single-pair shortest path query, KNN and keyword-based KNN query in the road network. However, none of these methods consider efficiency of retrieving data from a storage device, especially in IoT applications where a large amount of data are involved and rapid data processing is required.

III. Design

Our primary design goal of RASM is to avoid retrieval of irrelevant data and improve data locality for better I/O performance in the service of road queries. To achieve the goal, it is required that spatial data can be stored based on their coordinate information for preserving the locality and in the disk space be separated from other data that are not associated to roads for their efficient retrieval. In this section we will first present an overview of the system using RASM. Then Hilbert curve encoding and RASM encoding are introduced in Sections III-B and III-C.

A. Overall structure of a RASM-based spatial database

Figure 4 shows the framework of a spatial database based on RASM. It has two unique features. First, a new translation layer which translates physical coordinates to road-related locations is introduced. For each piece of data to be stored in the database, we associate it with a road segment its corresponding vehicle runs on. With the translation, or attaching vehicles to the roads, locality of data about moving objects on the same road segment is retained in the database. Second, road queries and region queries are served separately for better performance. For road queries, the middle points of road segments are found and the Cell-ID of the middle points are used as the prefix to generate key ranges. For region query, region cover algorithm generates appropriate cells to cover the area and covert the cells to key ranges. By separating road queries from region queries, the computation cost of region-cover for road queries can be removed.

B. The Hilbert Encoding

RASM is based on the Hilbert curve encoding. In this section, we present design of the Hilbert curve. A geometric surface is indexed by a quad-tree. In each level of the quad-tree, space is divided into $2^R \times 2^R$ cells (R denotes the resolution). Cells in each level are numbered by the Hilbert curve of the corresponding order. Using the Hilbert curve to generate keys for the spatial objects is a recursive procedure. There are four first-order units, which are ① canonical, ② axes swapped, ③ bits inverted and ④ swapped & inverted as shown in Figure 5. The number inside each cell in the unit reflects the order in which the sequence of the cells appear on the Hilbert curve. The red line in each cell represents the

subdivision rule. Hilbert curve of any order in the corresponding level of the quad-tree can be generated from those four first-order Hilbert curve unit. For example, after the ①-0 cell is subdivided into four cells, they should follow the rule of ②. Similarly, ①-1 should follow ① and ①-3 should follow ④. Every next order repeats the process by replacing each cell with four smaller cells. The subdivision rules are:

- ①: ② ↓ ① → ① ↑ ④
- ②: ① → ② ↓ ② ← ③
- ③: ④ ↑ ③ ← ③ ↓ ②
- ④: ③ ← ④ ↑ ④ → ①

Following the rules above, each cell in the quad-tree will have a unique *Cell-ID*. An example of generating third order Hilbert curve is illustrated in Figure 5e. The Cell-ID, which is the sequence of visiting of the cells in each level of the quad-tree, is denoted in sequence of bits. And Cell-ID in one level is the prefix of the ID of the cells in the next level. For example, in the first order curve, cell 00's third child has Cell-ID of 00.10, whose second child has Cell-ID 0010.01. The data about moving objects will reach to the last level of the quad-tree, and carry the Cell-ID as the key and the data is stored as a key-value pair in the database.

C. RASM encoding

A road is stored as a set of segments, each of which includes two two-dimensional coordinates indicating the start and end points. A road segment starts from a crossroad and ends at an adjacent intersection. One big challenge of designing RASM is how to gather spatial data based on their corresponding road segments without compromising spatial locality. An ideal solution would allow one range scan on the disk to retrieve all the data within the geometric area given a spatial query. This requires the data distribution follows the query pattern, or the data should have keys close to each other in the key space so they can be stored together in the database. To provide efficient road query, one straightforward way is to use a road name as the key to store the data. This method is efficient for queries on a single road. But it can be problematic when we want to search for the objects at a crossroad or within a geometric area because the locality cannot maintain in one-dimensional space with road name as the key. Hence the data scatter in the disk, and random reads still occur. It will significantly compromise I/O efficiency when serving region query.

To maintain the data locality within the road segments and the locality between the segments in the same time, RASM aggregates the data on the same road segment and uses Hilbert curve to generate IDs for them. The segment ID is defined as the Cell-ID of the middle point of the road segment. This ID serves as the prefix of the key of the data, which ensures spatial data collected on the same road segment can be kept in a small range and accordingly stored close to each other on the disk. Because segment-IDs are named to preserve locality in the Hilbert curve, nearby road segments will have segment IDs close to each other. After we sort the segment IDs with ascending order and encode them from 1 to n (n is the total number of segments), space-filling curve under RASM encoding is generated, and the locality is well preserved, as illustrated in Figure 6.

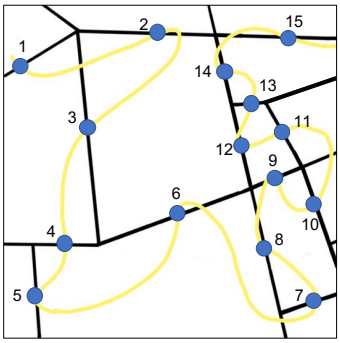


Fig. 6: Illustration of the space-filling curve under RASM encoding after replacing the unit of fixed cells with road segments, which are depicted in black line. Blue dots are the middle points of the segments. Digital numbers are the segment IDs. The yellow line represents the curve. Most nearby segments have IDs close to each other.

Another issue in designing RASM is how to determine the maximum length of road segments. The length of road segments varies considerably in the road network. If the segment length is short, the localities between segments are well maintained by the Hilbert curve. However, if the length is long, it causes high read amplification while a small portion of the data on the segment is involved in a query. In the meantime, long segment causes loss of locality because the middle point of the segment is far from its adjacent one. Then the spatial data about neighboring objects will scatter in linear space. So a limit on the segment length is necessary. Since the segment length varies between cities, the limit should be decided accordingly to local road network. In RASM, we analyze the median value of the road segment length and use it as the limit. If a road segment reaches the limit, we split it recursively until the remaining part is smaller than the limit.

When the GPS-enabled devices, such as smartphones, generate spatial data, the only location information is the coordinates, which need to be translated and associated with road segments for efficient service of road queries. The translation can be conducted either on the GPS device or in the centralized server, which is in charge of collecting traffic data from a number of devices and storing them to the back-end storage system and answering users' requests. Since the devices are powerful enough to do the translation, they can carry out the task. This distributes the translation on GPS devices, which can provide maximized concurrency and help to keep the server from becoming a performance bottleneck.

For the data that are uploaded and about to be stored in the database, RASM translates the raw location information of the data to road-related one. For the translation, RASM needs to store the road information, which is represented as a set of road segments. Since the road information is almost static and of of limited size, it is affordable to keep it in the memory and keep it up-to-date. For example, the size of the

data about road network is about 20MB (including 399178 road segments with attributes like road name and road type) for Shenzhen, which is a big city in China. In RASM, the road is organized in a quad-tree-based on roads' spatial locations in the space. Some moving objects may appear on a location without any road networks. To avoid mapping those data to wrong roads, we define a distance threshold in RASM which determines whether a piece of data should be attached to a road. For data whose associated coordinates indicate a distance larger than the threshold from any nearby road, it is stored to database directly using Hilbert encoding. In our evaluation, the threshold is set to be 10 meters because U.S. government commits to an average error of 7.8 m (25.6 ft.), with 95% probability [31] of the GPS signal.

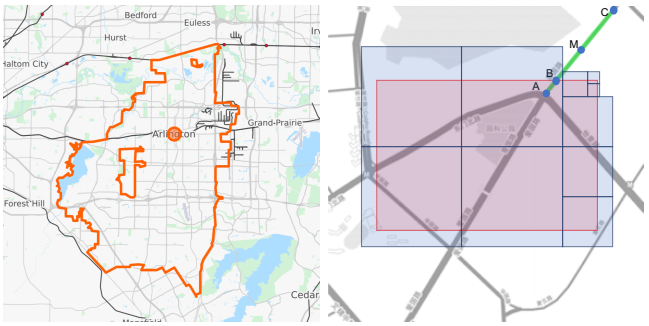
D. Storing traffic data

In RASM, the server generates a key for each piece of data uploaded from a GPS device based on the road segment. For data that is attached to a road segment, coordinates of the middle point of the segment are chosen and fed into the Hilbert curve generation function to get a number which indicates the point's Cell-ID in the space. For data that have no valid road information, the Cell-ID is generated directly by feeding its original coordinates into the function. In our current design of RASM, we use the Hilbert encoding [26], [32] used in the Google S2 library to generate a 64-bit Cell-ID for each piece of data. In the S2 library implementation, the earth surface is mapped to six faces of a cube and each face is organized as a quad-tree with 30 levels. In each level, Hilbert curve is used to number the cells, and the ID in the lower level can be generated recursively following Hilbert curve generation rules.

The generated Cell-ID, together with a 64-bit timestamp and a 32-bit device ID, serves as the key of the data. In this way, data mapped to a single road segment are naturally organized together. As long as the data are sorted according to their keys and stored on the disk sequentially like that in LevelDB, any queries on the road segment can be efficiently served by a single range scan. Further, any query to a road can be answered by synthesizing the results of queries of each segment of the road.

E. Serving region query

With RASM, road queries can be served efficiently as the data on a road segment are grouped together for storage and can be retrieved with a single range scan. To also provide efficient service to region queries, we first classify region query into two types according to the region shape. One is *district query*, whose region is bounded by the road segments as shown in Figure 7a. The other is *arbitrary query*, whose region can be bordered with any lines, as shown in Figure 7b. In the two types of queries, district query is more common because most borders of region queries by IoT applications are defined by roads. For district queries, the RASM encoding has the same accuracy as the Hilbert encoding because it divides geographic area with road segments so that all the midpoints of the road segments within a district can be retrieved. However,



(a) district query (b) arbitrary query

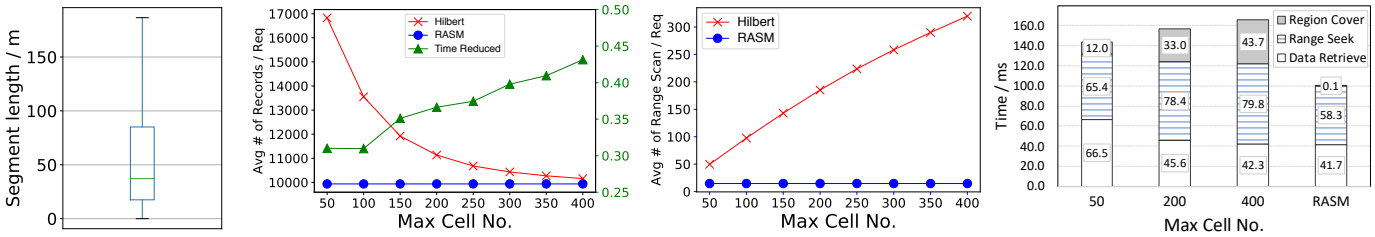
Fig. 7: (a) depicts the outline of Arlington, Texas. (b) illustrates the error introduced by mapping data to road segments in arbitrary query. The geometric area is the rectangle in red, and the cells used to cover the area is in blue. In this query, the data on segment AB will be lost since the middle point M of the road segment AC is out of the range of the cover.

for arbitrary queries RASM needs extra efforts to maintain query accuracy. When serving an arbitrary query, all the cells intersecting with the region must be extracted, and all the data in the cells are retrieved. Since RASM associates data to road segments for the convenience of serving road queries, it may risk missing some data on the boundaries in serving arbitrary query. For the data on a road segment, the middle point of the segment represents the data's geometry location and is used to generate a key for the data to be stored in the storage system. There is a chance that the original data belongs to the region covered by a query but the middle point of the segment is out of the region, as shown in Figure 7b. So some data covered in the query may not be included in its service.

For an accurate response of an arbitrary query, a correction is made in RASM to retrieve the data on the segments whose midpoints are out of range. To this end we use *get-crossing-edges* operation in the Google's S2 Geometry Library to retrieve all the road segments intersecting with the borders of the region, on the segment index built with S2's *MutableS2ShapeIndex* class. We then merge the segment IDs with the Cell-IDs to serve this query. In this way, all the road segments that contain the query data are included.

IV. Evaluation

In this section, we show some experimental results of RASM and compare them with those collected on a system directly applying the Hilbert curve for key generation, which is a commonly-used approach in existing geospatial storage systems. Our evaluation will answer three questions: (1) can RASM improve data access locality for road queries? (2) To what degree can RASM reduce computation cost and improve the I/O efficiency by reducing retrieval of irrelevant data for serving road queries? and (3) How about RASM's service performance for arbitrary query when correction is applied?



(a) road segments (b) Records and Service time (c) Query Request (d) Service time breakdown

Fig. 8: The service of road query with/without RASM

A. Experiment Setup

The evaluation experiment is conducted on a Dell R630 server running Arch Linux with Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz, 64GB RAM, and WD 1TB 7200 RPM SATA hard disk. We implement RASM on top of Google S2 library and use LevelDB for storing the data. The dataset used in the experiment is from a real-world traffic analysis system in Shenzhen, China, which contains 61,924,348 GPS records collected with taxis on November 1st, 2016.

Compared to the amount of data processed by an extensive spatial database, the dataset used in our evaluation is small. To avoid the impact of data caching by the large memory of the server on the I/O performance evaluation, we use cgroups tool [33] to limit the amount of memory space used in the experiments. The test environment ensures that the dataset is 10X larger than the memory space, which is a reasonable setting considering the dataset size stored on the large disks and usable memory size in a real system.

B. Experiment Results

To answer the first two questions, we randomly load data about 500 roads from the OpenStreetMap [34], and conduct road queries separately on both systems using Hilbert and RASM encoding. Those roads consist of segments whose length distribution is shown in Figure 8a. The median is 37 meters, and for the segments whose length is larger than this value, we recursively split them.

The results are illustrated in Figure 8b-8d. They show speedup and amount of data retrieved for serving each road request. In the Hilbert encoding, spatial data are stored without being attached to any road segments, and road queries are served similarly as region queries. To answer a road query, the system employs region-cover algorithm to generate all cells that overlap the roads and uses the Cell-ID to generate a key prefix and send a range scan request to the LevelDB to retrieve all data whose keys share the key prefix. We set a limit of the maximum number of cells allowed to cover the road (different resolutions) in the region cover algorithm and change it in different runs and compare the experimental results with RASM encoding. The results show that with RASM the average time of serving each road request can be reduced by up to 43%. Compared to the Hilbert encoding with small cell number, the amount of data retrieved can be reduced by up to 70%. A larger cell number limit can help to reduce the amount of irrelevant data retrieved when serving road queries for Hilbert encoding. However, the performance becomes even worse as shown in Figure 8b. The green line depicts the time reduced by RASM compared to Hilbert. With maximum cell limit of 400, Hilbert encoding is 43% slower, while with the limit of 50 it is 30% slower. With a larger cell number limit, the number of range scans for retrieving the data increases as shown in Figure 8c, which results in more disk seek operations. The performance of the hard disk can be significantly compromised due to frequent disk head seek.

To understand the cause of the time reduction, we measure the time spent on each phase when serving a road request. As shown in Figure 8d, with Hilbert encoding, the service time is spent on three phases: region-cover, range-seek, and data-retrieval. In the region-cover phase RASM calculates all regions overlapping with the road. In the range-seek phase RASM locates the data on the disk for the regions produced by the region-cover phase. In the data-retrieval phase RASM reads all data from the location returned in the range-seek phase and performs data filtering. For RASM, the computation cost is negligible because the region-cover phase is removed and RASM only calculates the cell IDs for middle points of the road segments to translate road query from two-dimensional space to one-dimensional space. As shown in Figure 8d, with Hilbert encoding using a small cell number limit introduces a longer data retrieval time as more irrelevant data are read, while with a larger cell number limit the range-seek time increases as more range scans occur in the storage system. With RASM encoding, the computational cost is 0.1ms on average, and the range-seek cost reduces up to 27%. RASM retains data access locality, reduces computation cost, and achieves high I/O efficiency.

With RASM, the performance of road queries can be considerably improved, while we need to ensure it imposes only a negligible impact on arbitrary queries. To answer the third question, we randomly choose 50 rectangle regions, each of size $1km \times 1km$, in Shenzhen. The average service time and the amount of data are measured, as shown in Figure 9. If no corrections are applied for returning 100% accurate query results, the service time is reduced because fewer data are returned. With corrections, the average service time increases by 3.3%. The extra time cost comes from two sources. One is the cost of get-crossing-edges operation, which is negligible and stays blow 2ms. Another is the cost to retrieve additional

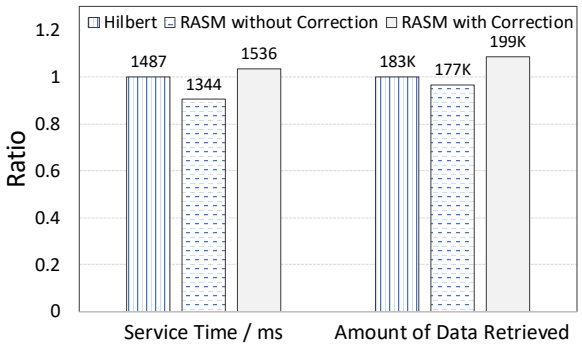


Fig. 9: The service time and amount of data retrieved in the servce of region query with/without RASM

data on the road segments that intersect the outline.

V. Conclusion

In this paper, we present RASM, a road-aware spatial mapping for moving objects. In RASM, spatial data are associated to road segments and stored in the storage systems based on their segment IDs, which accelerates service of road queries due to improved spatial locality and reduces the cost of computing region cover. Meanwhile, the performance for district query is retained. The negative performance impact on arbitrary queries is very limited. Our experiment results on real-world traffic dataset show that RASM reduces the road query time by up to 43% and the I/O traffic can be reduced by up to 70%.

Acknowledgments

We are grateful to the paper's reviewers who helped to improve the papers quality. This work was supported by a new faculty startup fund from UT Arlington and National Natural Science Foundation of China under Grants No.61572487 and No.61472323.

References

[1] The internet of things: Today and tomorrow. https://www.arubanetworks.com/assets/eo/HPE_Aruba_IoT_Research_Report.pdf. (Accessed on 07/29/2018).

[2] 97 amazing uber statistics, demographics and facts (june 2018). https://expandedramblings.com/index.php/uber-statistics/. (Accessed on 07/29/2018).

[3] The us mobile app report - google maps app 64.5m users, apple maps 42m - geoawesomeness. https://goo.gl/iZye3q. (Accessed on 04/24/2018).

[4] Iot: number of connected devices worldwide 2012 - 2025. https://goo.gl/jpDjiw. (Accessed on 08/02/2018).

[5] Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. https://www.gartner.com/newsroom/id/3598917. (Accessed on 08/02/2018).

[6] Spatial database - wikipedia. https://en.wikipedia.org/wiki/Spatial_database. (Accessed on 07/29/2018).

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

[8] Jonathan K. Lawder and Peter J. H. King. Querying multi-dimensional data indexed using the hilbert space-filling curve. *ACM Sigmod Record*, 30(1):19–24, 2001.

[9] Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

[10] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.

[11] David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[12] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on knowledge and data engineering*, 13(1):124–141, 2001.

[13] David J Abel and David M Mark. A comparative analysis of some two-dimensional orderings. *International Journal of Geographical Information Systems*, 4(1):21–31, 1990.

[14] Region coverer. https://s2.sidewalklabs.com/regioncoverer/. (Accessed on 07/30/2018).

[15] google/leveldb: Leveldb is a fast key-value storage library written at google that provides an ordered mapping from string keys to string values. https://github.com/google/leveldb. (Accessed on 04/24/2018).

[16] Patrick ONeil, Edward Cheng, Dieter Gawlick, and Elizabeth ONeil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[17] Anand Padmanabha Iyer and Ion Stoica. A scalable distributed spatial index for the internet-of-things. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 548–560. ACM, 2017.

[18] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[19] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, volume 19, pages 322–331. Acm, 1990.

[20] Dieter Pfoser, Christian S Jensen, Yannis Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, 2000.

[21] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[22] Kevin Sahr, Denis White, and A Jon Kimerling. Geodesic discrete global grid systems. *Cartography and Geographic Information Science*, 30(2):121–134, 2003.

[23] S2 geometry. http://s2geometry.io/. (Accessed on 04/26/2018).

[24] S2 region cover. https://github.com/google/s2geometry/blob/master/src/s2/s2region_coverer.cc. (Accessed on 04/26/2018).

[25] S2 cell statistics. http://s2geometry.io/resources/s2cell_statistics.html. (Accessed on 04/24/2018).

[26] Shoji Nishimura and Haruo Yokota. Quilts: Multidimensional data partitioning framework based on query-aware and skew-tolerant space-filling curves. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1525–1537. ACM, 2017.

[27] Shen Li and Shaohan Hu. Pyro: A Spatial-Temporal Big-Data Storage System. *2015 USENIX Annual Technical Conference*, pages 97–109, 2015.

[28] Bilong Shen, Ying Zhao, Guoliang Li, Weimin Zheng, Yue Qin, Bo Yuan, and Yongming Rao. V-tree: Efficient knn search on moving objects with road-network constraints. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 609–620. IEEE, 2017.

[29] Siqiang Luo, Yifeng Luo, Shuigeng Zhou, Gao Cong, Jihong Guan, and Zheng Yong. Distributed spatial keyword querying on road networks. In *EDBT*, pages 235–246. Citeseer, 2014.

[30] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, and Zhiguo Gong. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(8):2175–2189, 2015.

[31] Gps will be accurate within one foot in some phones next year - the verge. https://www.theverge.com/circuitbreaker/2017/9/25/16362296/gps-accuracy-improving-one-foot-broadcom. (Accessed on 08/01/2018).

[32] Jonathan K Lawder. *The application of space-filling curves to the storage and retrieval of multi-dimensional data.* PhD thesis, Citeseer, 2000.

[33] cgroups - archwiki. https://wiki.archlinux.org/index.php/cgroups. (Accessed on 04/24/2018).

[34] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.