A Cloud-Based Trajectory Data Management System*

Ruiyuan Li[1,2] Sijie Ruan[1,2] Jie Bao[2] Yu Zheng[1,2,3†]
[1]School of Computer Science and Technology, Xidian University, China
[2]Urban Computing Group, Microsoft Research, Beijing, China
[3]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
{v-ruiyli, jiebao, yuzheng}@microsoft.com sjruan94@gmail.com

ABSTRACT

With the rapid development of location-acquisition techniques, massive trajectories are continuously generated. Many urban applications rely heavily on the data mining/analysis results of massive trajectory data. This demo presents a holistic data management system for both *historical and real-time* trajectory records based on a cloud platform, such as Microsoft Azure. The proposed system is able to efficiently support a variety of trajectory queries, including *ID-Temporal query*, *Spatio-Temporal query*, and *Path-Temporal query*. With these queries, we demonstrate that different urban applications can be realized in a much easier way.

CCS CONCEPTS

• **Information systems → Spatial databases and GIS**;

KEYWORDS

Trajectory Data Management, Cloud Computing

1 INTRODUCTION

Massive trajectory data is generated continuously with the rapid development of location-acquisition devices, e.g., smart phones. This massive trajectory data offers very rich information about users and their locations [1]. Many urban computing applications rely on the analytical results of trajectory data [2], such as traffic modeling, mobility pattern discovery, and location-based recommendation. Taking advantage of cloud computing platforms is clearly the best option to efficiently manage large-scale trajectory data. Existing works [3–6] leverage distributed computing platforms, e.g. Hadoop and Spark, to support spatio-temporal queries, which cannot directly be applied to trajectories, nor can they handle massive trajectory updates. To this end, this paper demonstrates a holistic trajectory data management system for both historical and real-time trajectories, building on our previous work [7].

However, managing massive trajectory data in a real-time manner is a non-trivial task, as the system not only needs to handle the high volume updates from moving objects, but also retrieves querying results efficiently from a large-scale trajectory dataset. On the other hand, existing cloud computing platforms, e.g., Microsoft Azure, are not designed to process trajectory data. To address this challenge, we build a middle layer between the cloud computing components and urban applications, which helps urban application developers to utilize these cloud computing components in dealing with massive trajectory data more conveniently and efficiently. Figure 1 gives an overview of our proposed cloud-based trajectory data management system [8], which 1) takes advantage of existing cloud components, e.g., cloud data storage, data caching, and distributed computing; 2) preprocesses and indexes high volume trajectory data updates; and 3) supports different types of queries for many urban applications. The three main trajectory query types supported by our system are:

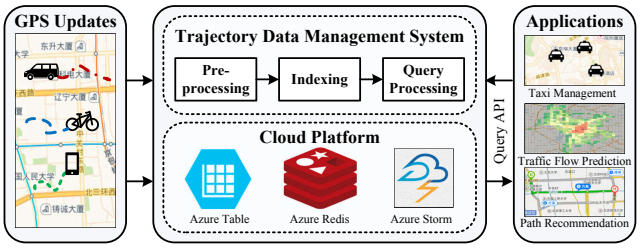


Figure 1: System Framework.

ID-Temporal Query, which retrieves trajectory data based on an object ID and a temporal range, e.g., finding the trajectory of taxi "guiaxx" from 16:00 to 18:00 on one day. This is very useful in a taxi management system, when the user wants to know the detailed taxi trajectories of a particular vehicle in the given temporal range.

Spatio-Temporal Query, which finds trajectories in a given spatio-temporal range, e.g. finding all the trajectories passing the railway station area from 11:00 to 13:00 in a day. For instance, in traffic flow prediction [9], the inflow/outflow of a region in a specific time interval can be calculated using Spatio-Temporal query.

Path-Temporal Query, which finds trajectories traversing a path (i.e., a sequence of connected road segments) during a temporal range [10], e.g., finding all taxis traveling through West 156th Street in New York from 16:00 to 18:00 today. This type of query is particularly useful in travel time estimation applications [11].

In this paper, we present our cloud-based trajectory data management system, with three main components, as shown in Figure 2:

Preprocessing, which uses the distributed streaming system, i.e., Storm, to handle massive trajectory updates and perform the map-match task in real-time (detailed in Section 2).

Indexing, which organizes the massive trajectory data effectively based on different tasks in the Azure storage component, e.g., Azure Table (detailed in Section 3).

*The work was supported by the National Natural Science Foundation of China (Grant No. 61672399, No. U1609217) and the China National Basic Research Program (973 Program, No. 2015CB352400)
†Yu Zheng is the correspondence author of this paper.

SIGSPATIAL'17, November 7–10, 2017, Los Angeles Area, CA, USA

ACM ISBN 978-1-4503-5490-5/17/11.
https://doi.org/10.1145/3139958.3139990

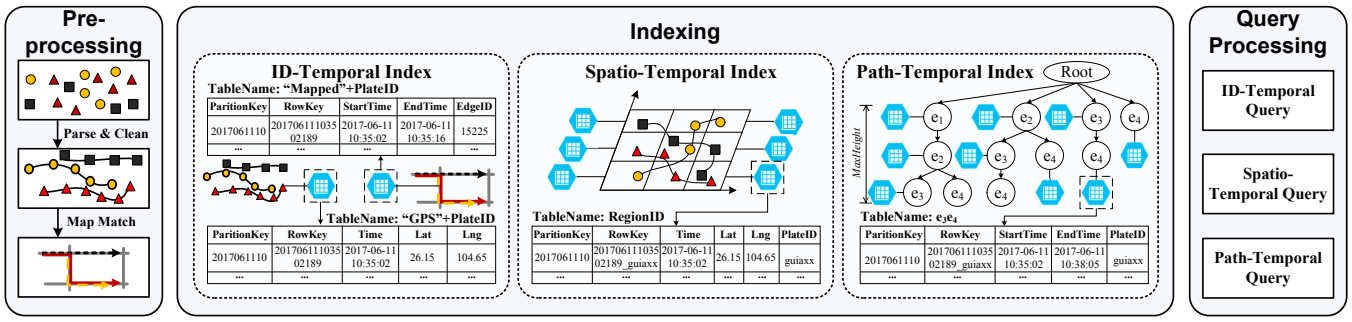


Figure 2: Overview of the cloud-based trajectory management system.

Query Processing, which efficiently answers different types of trajectory queries issued from users or urban applications using the indexes and parallel computing framework (detailed in Section 4).

Finally, we demonstrate three different real applications that are deployed on Microsoft Azure using our trajectory data management system (detailed in Section 5). Our ultimate goal is to implement the trajectory data management function as a future Azure service.

2 PREPROCESSING

The left part of Figure 2 illustrates the trajectory preprocessing component, which consists of two main tasks:

Parse & Clean. This task receives the GPS point updates from different data sources (e.g., taxis or smart phones), parsing and organizing the point data as trajectories (e.g., grouping them based on their plate numbers). Also, we remove the outlier GPS points in this step, using a heuristics-based outlier detection method [1].

Map Matching. This task maps the GPS points to the corresponding road segments. Map-matched trajectories are very useful for road network-based traffic modeling. We adopt an interactive voting-based map-matching algorithm [12] to perform the task. As the volume of GPS updates is very huge and map-matching task is time consuming, we leverage the distributed streaming computing platform in Azure (i.e., Storm) to speed up the map-matching process. The details of storm-based map-matching can be found in our previous work [7]. Finally, the preprocessed datasets are stored in an Azure Redis server (a cache service) to be accessed efficiently by different indexing algorithms.

3 TRAJECTORY INDEXING

The middle portion of Figure 2 gives the details about the trajectory indexing in our system, including ID-Temporal index, Spatio-Temporal index, and Path-Temporal index. As most of our queries involve spatial or temporal range selections and Azure Table is very efficient in answering range queries (especially within the same partition), Azure Table is used extensively in our system to store indexes. Moreover, the main difference in indexing the trajectory data in Azure storage is that we essentially store multiple copies of the trajectory data rather than maintain an index structure with pointers to the actual data. The main reason is that the storage cost is much cheaper compared to the computing cost on the cloud, e.g., it is about 10 USD per 1TB/month [1], and is more efficient to retrieve data when they are organized together in Azure Table. As a result, the main task in building trajectory indexes in Azure is to group the data that will be retrieved together within the same Azure Table partition.

3.1 ID-Temporal Index

As depicted in the left portion of the indexing component in Figure 2, we build an ID-temporal index for both raw trajectories and map-matched trajectories. The trajectory data is stored in Azure Table, where each table contains the trajectories of a moving object (identified by their plate numbers), hence we can find the trajectories of a given object directly. The `PartitionKey` of a record is the time that is accurate to the hour information, and the `RowKey` is the exact time of the record. For example, if a GPS point is generated at 10:35:02.189, Jun 11th, 2017, its `PartitionKey` is "2017061110", and `RowKey` is "20170611103502189". The reasons for designing `PartitionKey` and `RowKey` in this way are: 1) GPS points within the same partition are usually retrieved together, when answering an ID-Temporal query; 2) `RowKey` in Azure Table serves as an identifier, and there are no GPS points generated at the same time; and 3) using the hour information as the `PartitionKey` achieves a balance between indexing and query processing, as a larger `PartitionKey` is more efficient in indexing but suffers from query processing, with details found in [7].

3.2 Spatio-Temporal Index

We construct a Spatio-Temporal index to accelerate Spatio-Temporal query that finds the sub-trajectories within a given spatio-temporal range. As shown in the center portion of the indexing component in Figure 2, we build a static spatial index by partitioning the space into disjoint and uniform grids, where each grid represents a region with an identity. Any other static partition method, e.g. R-tree or quad-tree, can also be applied. We then create an Azure table for each region based on its identity. The trajectory records that belong to the same region are grouped together and inserted into the corresponding table. Thus, we can retrieve the trajectories in a specified region by directly finding the corresponding table. The `PartitionKey` of a GPS point is the time that is accurate to the hour, e.g., "2017061110", and the `RowKey` is the combination of exact time and moving object ID, e.g. "20170611103502189_guiaxx". The reasons for designing the `RowKey` in that way are: 1) we need

[1] https://azure.microsoft.com/en-us/pricing/details/storage/

to identify the GPS points based on RowKey, adding the moving object ID at the end to avoid the conflicts of two GPS points generated at the same time by different objects; 2) putting the accurate time in the front of the RowKey enables us to take advantage of the efficient range queries of the RowKeys to serve the temporal range selections.

3.3 Path-Temporal Index

As depicted in the right portion of the indexing component in Figure 2, we employ a *table-based suffix tree*, as the Path-Temporal index. The table-based suffix tree treats the road segments of map-matched trajectories as a sequence of characters. To overcome the issue of generating a very large suffix index with massive trajectories, the *table-based suffix tree* is designed with two main parts: 1) *suffix tree structure*, which has a limited height and a pointer to an Azure table, and 2) *suffix tree records*, which are stored in Azure tables and organized based on trajectories with the same suffix, i.e., passing the same road segments.

There are three main steps to construct the Path-Temporal index: 1) Suffix Generation, which generates the sub-trajectories whose lengths are no more than H; 2) Suffix Grouping, which groups the generated sub-trajectories based on the common suffixes; 3) Record Insertion, which inserts the grouped sub-trajectories information into the corresponding Azure table. Suppose we set the max height H as two, the map-matched trajectory $e_1 \rightarrow e_2 \rightarrow e_3$ will be broken into five sub-trajectories: e_1, e_2, e_3, $e_1 \rightarrow e_2$, and $e_2 \rightarrow e_3$. These sub-trajectories will be stored in their corresponding tables, among which an entity records the trajectory information including the moving object ID, and the enter and leave times of the moving object. The PartitionKey of an entity is the enter time that is accurate to the hour, e.g. "2017061110", while the RowKey is the combination of exact enter time and moving object ID, e.g. "20170611103502189_guiaxx". The reason for the key design is the same as Spatio-Temporal index's.

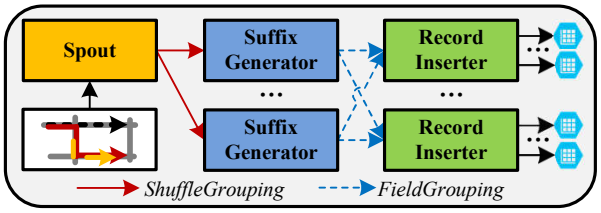


Figure 3: Storm-Based Path-Temporal Index.

As there are numerous sub-trajectories generated, a large number of trajectory records need to be inserted into many tables, which may cause CPU and I/O bottlenecks in the standalone environment. Therefore, we develop a distributed framework on the Azure distributed processing platform, i.e. Storm, for the index building process. As depicted in Figure 3, the *Spout* reads map-matched trajectories and dispatches them to the *Suffix Generator* bolts using the *ShuffleGrouping* mechanism (i.e., random assignment, to achieve the workload balance). The *Suffix Generator* receives a map-matched trajectory, and breaks it into sub-trajectories. These sub-trajectories are finally emitted to the *Record Inserter* bolts using the *FieldGrouping* mechanism grouped by the tables, which guarantees that the records to be inserted in the same table flow into the same bolt. The *Record Inserter* bolt aggregates the same suffixes and inserts them into an Azure table in batches to reduce high I/O overhead of frequent small writings.

4 QUERY PROCESSING

4.1 ID-Temporal Query Processing

When querying the GPS/map-matched trajectories of a moving object during a given time period, we first find the corresponding Azure table with the object ID. Then, query time is converted as the PartitionKey and RowKey ranges. For example, when searching for the trajectories of moving object "guiaxx" during 10:30~11:00 on Jun 11th, 2017, the ID-Temporal query is transformed as retrieving data from Azure table "guiaxx": "PartitionKey ge '2017061110' and PartitionKey lt '2017061111' and RowKey ge '20170611103000000' and RowKey lt '20170611110000000'".

4.2 Spatio-Temporal Query Processing

To answer the Spatio-Temporal query, we first find the regions that overlap with the given spatial range, which gives us the Azure Table names. After that, we can execute the temporal range queries in parallel over the corresponding tables in light of the specified time period. Finally, the results obtained from different tables are aggregated and returned.

4.3 Path-Temporal Query Processing

There are three main steps to answer the Path-Temporal query. 1) Query Path Decomposition. We decompose the query path whose length is longer than H into several overlapped H-length sub-paths which can map to corresponding table names. For example, suppose that H is set as three, we can decompose the query path $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4$ into $e_1 \rightarrow e_2 \rightarrow e_3$ and $e_2 \rightarrow e_3 \rightarrow e_4$, where $e_1e_2e_3$ and $e_2e_3e_4$ are tables generated in the index building process; 2) Record Retrieval. We retrieve the index records on corresponding tables in parallel in light of the query time period; and 3) Sub-Trajectory Reconstruction. Based on the order of the decomposed sub-paths, we check if the time ranges of the same moving object's records retrieved from the corresponding neighbor tables have overlaps. If so, the corresponding moving object is qualified. For example, if Tr_1 passes $e_1e_2e_3$ during [10:21, 10:23] and passes $e_2e_3e_4$ during [10:22, 10:24], then Tr_1 passes $e_1e_2e_3e_4$ during [10:21,10:24] due to temporal ranges having overlaps.

We can see that there is a trade-off in choosing a suitable H between index construction and query processing. While smaller H makes index construction take less computation and smaller records sizes, the query path needs to be decomposed to more sub-paths and take more table access.

5 DEMO SCENARIOS

The proposed cloud-based trajectory data management system [8] will be demonstrated to the conference audience using real-time taxi trajectories from Guiyang (the capital of Guizhou Province, China) starting from Sept. 30th, 2016. The dataset contains around 5,000 daily active taxis, with each taxi generating one GPS

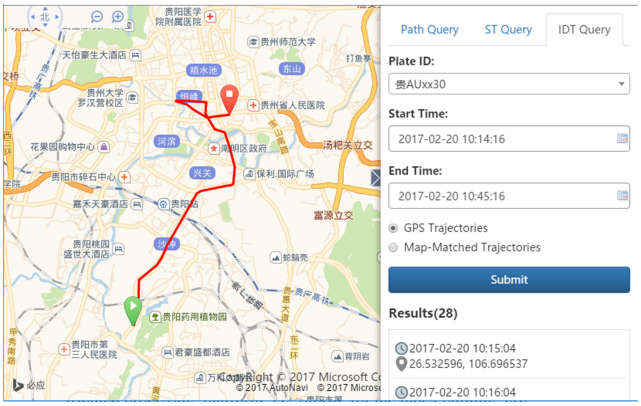
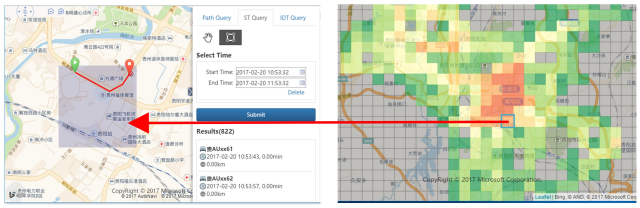


Figure 4: ID-Temporal Query Interactive Interface.



(a) Spatio-Temporal Trajectory Retrieval. (b) Traffic Inflow Heatmap.

Figure 5: Spatio-Temporal Query Interactive Interface.

record every 1 minute. The raw trajectory data and map-matched trajectory data are about 7GB and 60.86GB in disk space per month respectively. As shown in Figure 4, the map view is displayed on the left, and the interactive area is on the right. Attendees can switch the top-right tabs among three types of query.

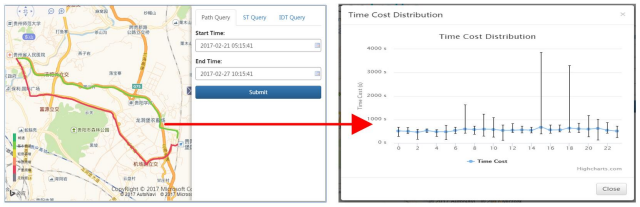
5.1 Scenarios for ID-Temporal Query

Attendees can submit an ID-temporal query through the GUI (Graphical User Interface) by selecting a taxi, inputting a time period, and choosing the trajectory type (GPS trajectory or map-matched trajectory), as shown in Figure 4. The results will be listed in the interactive area, and the corresponding trajectory is drawn on the map, among which the green mark and red mark denote the start position and end position respectively.

ID-Temporal query can be widely used in the taxi management system, in which the manager can have full knowledge of driving positions and behaviors of taxi drivers at any time. In package tracing and carry-out services, clients can see their delivery progress through ID-Temporal query.

5.2 Scenarios for Spatio-Temporal Query

The GUI for the Spatio-Temporal query is depicted in Figure 5. Attendees can drag the map to adjust the view by selecting the hand icon and drawing one or more rectangle regions on the map after selecting the rectangle icon. Each rectangle region corresponds to two text fields for its temporal period constraint input. After attendees click the *Submit* button, the qualified records that satisfy all spatio-temporal constraints are listed. Attendees can click each record, and its corresponding trajectory is drawn on the map.



(a) Path-Temporal Moving Objects Retrieval. (b) Path Time Cost Distribution.

Figure 6: Path-Temporal Query Interactive Interface.

The Spatio-Temporal query can be used in many urban computing applications, such as air quality inference and traffic flow prediction. In the air quality inference scenario, Zheng et al. [9] adopt Spatio-Temporal query to model the spatio-temporal dependencies between air quality and traffic. In the traffic flow prediction scenario, Zhang et al. [13, 14] use Spatio-Temporal query to count the inflow and outflow of every region in a city for a specific time interval. The traffic inflow heat map is shown in Figure 5(b).

5.3 Scenarios for Path-Temporal Query

Attendees can view the traffic conditions of a path through the GUI by Path-Temporal query. As shown in Figure 6(a), attendees select two paths from Guizhou Provincial People's Hospital to Guiyang Longdongbao International Airport. After clicking the *Submit* button, the selected paths will be filled with color, where red denotes heavy traffic and green represents smooth traffic. Attendees can click each path, and its travel time cost distribution w.r.t. hour will come up, which is helpful for traffic analysis, as Figure 6(b) shows.

Path-Temporal query can be adopted in many traffic-related applications. Wang et al. [11] estimate the travel time of each path to find an optimal path concatenation solution using Path-Temporal query. Knowing several candidate paths' travel time, path recommendation can also be performed. Mining the frequent paths and discovering the bottlenecks of a road network using Path-Temporal query can help decision makers plan road networks.

REFERENCES

[1] Yu Zheng. Trajectory data mining: an overview. *TIST*, 6(3):29, 2015.
[2] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *TIST*, 5(3):38, 2014.
[3] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*, page 70, 2015.
[4] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363. IEEE, 2015.
[5] Louai Alarabi. St-hadoop: A mapreduce framework for big spatio-temporal data. In *ACM International Conference on Management of Data*, pages 40–42, 2017.
[6] Qiang Ma, Bin Yang, Weining Qian, and Aoying Zhou. Query processing of massive trajectory data based on mapreduce. In *Proceedings of the first international workshop on Cloud data management*, pages 9–16. ACM, 2009.
[7] Jie Bao, Ruiyuan Li, Xiuwen Yi, and Yu Zheng. Managing massive trajectories on the cloud. In *SIGSPATIAL*, page 41. ACM, 2016.
[8] Urbantraffic system. http://urbantraffic.chinacloudsites.cn/, 2017.
[9] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. U-air: When urban air quality inference meets big data. In *SIGKDD*, pages 1436–1444. ACM, 2013.
[10] Ruiyuan Li, Sijie Ruan, Jie Bao, Yanhua Li, Yingcai Wu, and Yu Zheng. Querying massive trajectories by path on the cloud. In *SIGSPATIAL*. ACM, 2017.
[11] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *SIGKDD*, pages 25–34. ACM, 2014.
[12] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guang-Zhong Sun. An interactive-voting based map matching algorithm. In *MDM*, pages 43–52, 2010.
[13] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. AAAI 2017, November 2016.
[14] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, and Xiuwen Yi. Dnn-based prediction model for spatio-temporal data. In *SIGSPATIAL*, page 92. ACM, 2016.