GCTraj: An Grid Based Indexing for Trajectory in Cloud Environment

Tanjung Dion (201883621)
Fahriannoor (201883655)

***Abstract*—Semantic trajectory is a sequence of time-stamped places where each place has information about spatial location such as latitude and longitude. With the rapid development of location-acquisition techniques, massive trajectories are growth in large scale. Many spatial related applications rely heavily on the data mining or analysis results of massive trajectory data. Moreover, conventional cloud computing platform are not developed to deal with multi-dimensional data such trajectory that has spatial and temporal information. In this project we presents data management system for historical trajectory records based on a cloud platform, such as Microsoft Azure. The proposed system is able to efficiently support a two types of trajectory queries, called Id-Temporal Query and Path-Temporal Query.**

I. INTRODUCTION

With rapidly development of location-acquisition devices, Trajectory data is growth in large scale. This massive trajectory data offers very rich information about their locations and semantic attributes[1]. Many spatial related applications rely on the analytical results of trajectory data, such as mobility pattern discovery, and location-based recommendation. The use of cloud computing platforms is clearly the best option to efficiently manage large-scale trajectory data[2], [3].

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 query results efficiently from a large-scale trajectory. On the other hand, the conventional cloud computing platforms are not optimized to deal with the spatio-temporal datasets, like trajectories. In other words, it cannot process the spatio-temporal queries over the trajectories in an efficient way. While storing the data, we also need to consider storing semantic information that related with trajectories.

In this project, we propose distributed cloud-based storage management system for trajectory database. We develop a framework to dealing massive trajectory data and store in efficient indexing process. Our trajectory management system can: 1) use the advantage of existing cloud components, e.g., cloud data storage, and data caching; 2) preprocesses and indexes high volume trajectory data updates; and 3) supports different types of queries for many urban applications. There are two main trajectory queries that supports by the system:

1) **ID-Temporal Query** which retrieves trajectory data based on an object ID and a temporal range, e.g., finding the trajectory of taxi20000520 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.
2) **Path-Temporal Query** Query, which finds trajectories traversing a path(i.e., a sequence of connected road segments) during a temporal range [4], 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 [5].

In this project, we present our cloud-based trajectory data management system, with three main components, as shown in Fig. 1:

1) **Preprocessing** that process raw trajectory data and perform data cleaning to handle the incomplete dataset.
2) **Indexing** which organizes the massive trajectory data effectively based on different tasks in the Azure storage component, e.g.,Azure Table.
3) **Visualization** which efficiently answers different types of trajectory queries issued from users or urban applications using the indexes and parallel computing framework.

Finally, we demonstrate in our application that connect to Azure storage using our trajectory data management system.

II. PROPOSED METHOD

Fig. 1 shows framework design of the cloud based trajectory management system. We show the system process as follows: First, we clean and modify raw trajectory data. Second, we partitions trajectory data space into multi-dimensional grid cell and segments trajectory into sub-categories. Third, each cell partitions will be stored into different storage. Lastly, Our system can perform query processing.

A. Preprocessing

Preprocessing is used for parsing and cleaning noise in the trajectory dataset. The reason we do the processing is because dataset is recorded from GPS module that could generate inaccurate data or missing some important data. Dataset that are used in this project is also has some missing location points in the trajectory. First, our system will accept the recorded trajectory. We clean and serialize several data due to csv format. From this datasets, we clear the noise data by filtering some missing data. Each points will be trace according to their threshold. In this projects, we use 15 seconds as the optimal threshold.

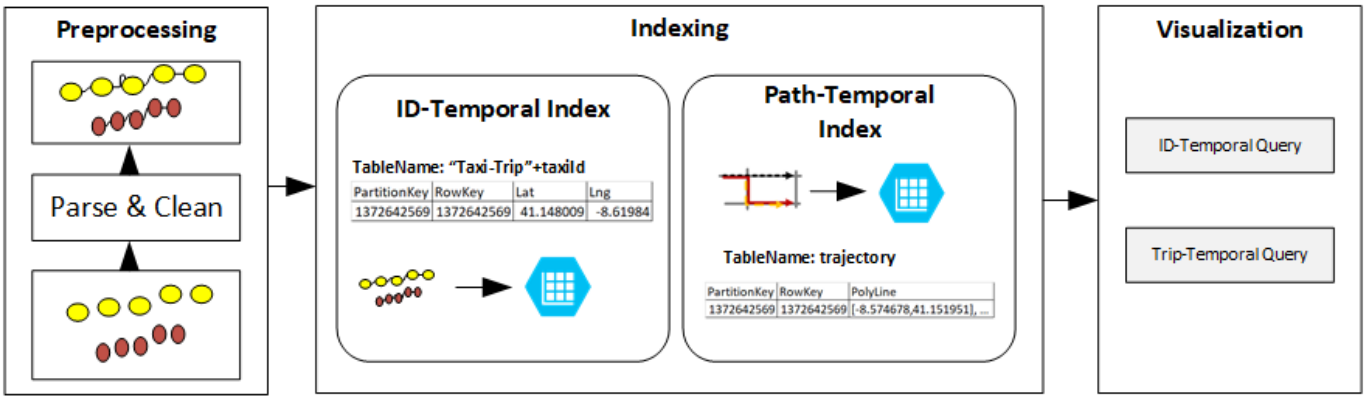

Fig. 1. Overview of the cloud based trajectory management system

B. Indexing

Before we store our data to Azure storage, we perform two types indexing as shown in Fig. 1. Most of our queries involve spatial or temporal range selections, therefore Azure Table is used extensively in our system to store indexes because it very efficient in answering range queries. 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[3]. We build trajectory indexes in Azure by grouping the data in same Azure Table partition so we can retrieve the data efficiently.

1) Id-Temporal Index: As depicted in the left portion of the indexing component in Fig. 1, we build an ID-temporal index from raw trajectories. The trajectory data is stored in Azure Table, where each table contains the trajectories of a moving object (identified by their *taxiId*), 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 recorded data. Both *PartitionKey* and *RowKey* are converted to unix timestamp. For example, if a GPS point is generated at Monday, July 1, 2013 12:00:54 AM, its *PartitionKey* is 1372636854, and RowKey is 1372636854. We design this type of indexing because GPS points that in the same partition are usually retrieved together based on their *taxiId* and *RowKey* in Azure Table can be used as identifier, therefore we can distinct several GPS points. After we create Id-Temporal Index, we store the trajectories into several table based on their taxiId.

2) Path-Temporal Index: As depicted in the right portion of the indexing component in Fig. 1, we employ the Path-Temporal index. We deploy trajectories data into one Azure Table. The *PartitionKey* of a record is the time that is accurate to the hour information, and the *RowKey* is the *taxiId*. *PartitionKey* converted to unix timestamp. For example, if a GPS point is generated at Monday, July 1, 2013 12:00:54 AM by taxi number 20000520, its *PartitionKey* is 1372636854, and RowKey is 20000520. We design this type of indexing to find the frequent path that passed in certain amount of time. For example, finding all taxis traveling through West 156th Street in New York from 16:00 to 18:00 today.

C. Visualization

1) Id-Temporal Query: When querying the GPS 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 *RowKey* ranges. For example, when searching for the trajectories of taxi 20000520 during 00:00 01200 on Jul 1st, 2013, the ID-Temporal query is transformed as retrieving data from Azure table 20000520: RowKey ge 1372636854 and RowKey lt 1372638158".

2) Path-Temporal Query: To answer Path-Temporal Query, first we find moving object during a given time period. Then, query time is converted as the *PartitionKey* ranges. For example, when searching for the trajectories tha passed during 00:00 01200 on Jul 1st, 2013, the ID-Temporal query is transformed as retrieving data from Azure table 20000520: PartitionKey ge 1372636854 and PartitionKey lt 1372638158".

III. DEMONSTRATION

A. Dataset

Dataset used in this project is [6]. This dataset record trajectories of taxi that running in the city of Porto in Portugal from July 01, 2019 until June 30, 2019. There are 442 taxis that have been collected into this dataset. We assume that each GPS points has 15 seconds differential generation, therefore we set the threshold for preprocessing as 15. Each trajectory contains a total of nine features. However, we only use taxiId, timestamps, and polylines for query processing. The details of each attributes are explains as follows:

1) TaxiId : It contains an unique identifier for the taxi driver that performed each trip
2) Timestamp : It identifies the trips start that formatted in unix timestamp
3) Polyline : It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. The last list item corresponds to the trips destination while the first one represents its start

B. Implementation Detail

We implement the visualization of our cloud database system using java programming language. Our User Interface (UI) is based on OpenGL and JavaFX rendering engine in processing sketchbook (processing.org). For the implementation of UI, we use G4P (GUI for porcessing) library (http://www.lagers.org.uk/g4p/). The purposes of choosing JavaFX instead of Swing is to improve the performance of our UI system. The other purposes is to support geographical map visualization from unfolding library (unfoldingmaps.org) that also use OpenGL rendering engines.

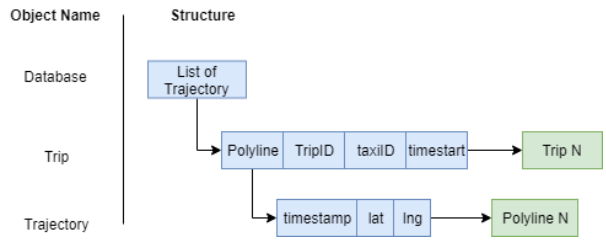

Fig. 2. Structure of temporal database in adjacency list representation

This developed system is able to read dataset from csv file and store the dataset into our storage system. First, we read the dataset file and parse the information to get the useful features that are needed in our experiment. Second, we build graph representation as our temporary database in the system. This graph is based on adjacency list that is shown in Fig. 2. In this graph representation, there are several trip that has information about tripID, taxiID, and timestart. for each trip, it has information about the movement that consist of timestamp, latitude, and longitude information. Third,we do the indexing, and partitioning using the temporal database. Finally, the result of our indexing will be stored into our storage system.

C. Query Scenarios

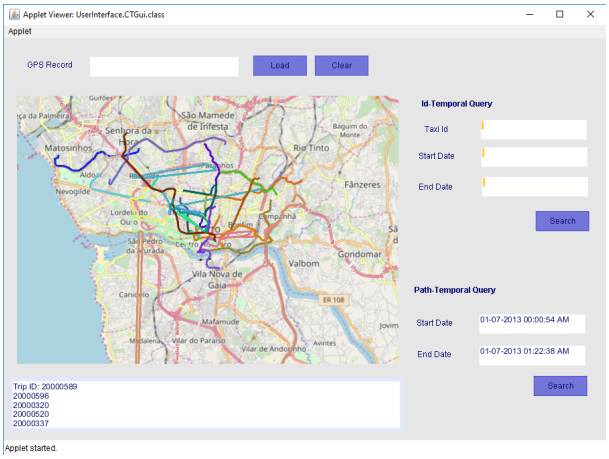

Fig. 3. ID-Temporal Query

1) ID Temporal Query Scenarios: User can submit an ID-temporal query through the GUI (Graphical User Interface) by inputting the taxiId and time period, as shown in Fig. 3. The results will be listed in the interactive area, and the corresponding trajectory is drawn on the map.

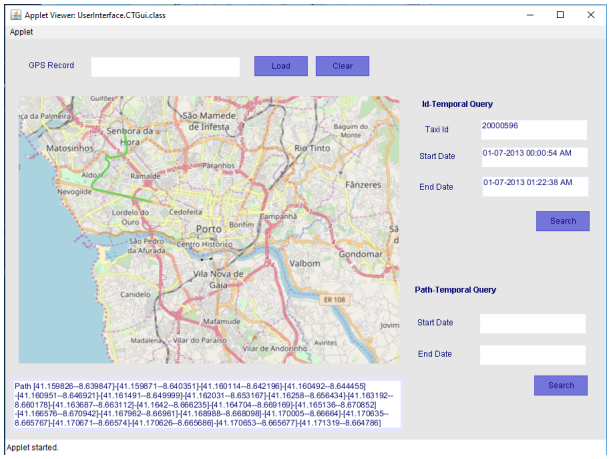

Fig. 4. Path temporal query

2) Path Temporal Query Scenarios: User can submit an Path-temporal query through the GUI (Graphical User Interface) by inputting the time period, as shown in Fig. 4. The results will be listed in the interactive area, and the corresponding trajectory is drawn on the map.

IV. Acknowledgement

This paper is submitted to fulfill term project task from Big Data Storage System class (Spring 2019), Pusan National University

References

[1] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.

[2] Jie Bao, Ruiyuan Li, Xiuwen Yi, and Yu Zheng. Managing massive trajectories on the cloud. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 41. ACM, 2016.

[3] Ruiyuan Li, Sijie Ruan, Jie Bao, and Yu Zheng. A cloud-based trajectory data management system. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 96. ACM, 2017.

[4] Ruiyuan Li, Sijie Ruan, Jie Bao, Yanhua Li, Yingcai Wu, and Yu Zheng. Querying massive trajectories by path on the cloud. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 77. ACM, 2017.

[5] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 25–34. ACM, 2014.

[6] Chris Cross. Taxi trajectory data, 2018.