Managing Massive Trajectories on the Cloud

JIE BAO, RUIYUAN LI, XIUWEN YI, YU ZHENG
URBAN COMPUTING GROUP
MICROSOFT RESEARCH, CHINA
What is trajectory data?

trajectory data represents the mobility of a diversity of moving objects, such as people, vehicles and animals.
Sources of trajectory data

- **Human mobility**
  - Active recording
    - Travel logs
    - Sport analysis
    - Check-ins
    - ...
  - Passive recording
    - Credit card transactions
    - Public transit records
    - Mobile phone signal, Wi-Fi...
    - ....
Sources of trajectory data

• Mobility of transportation vehicles
  – Taxis, buses, trucks,…
  – Air planes, ferries, cruise,…

• Mobility of Animals
  – Migration: Birds, zebra, tiger

• Mobility of natural phenomena
  – Hurricane, tornado,…
Motivation

Bridge the gap between massive trajectory data and urban computing applications

Enabling large scale storage & analysis real-time service providing

Cloud-based Trajectory Data Management Platform (Our Work)
Key functionalities

Conventional cloud computing platforms do not support trajectory queries well

- **ID-temporal query**: trajectory ID + time period -> trajectory segments
- **ST-Range query**: Spatio-temporal range -> partial trajectory segments
- **Map-Matching**: trajectory -> road segments
- **Reverse trajectory access**: road segments -> trajectories
Azure Preliminary

Azure Storage
- Azure Blob (Azure Files)
- Azure Table (Key-Value Storage)
  - Efficient for key-based access
  - Efficient for range access within the same partition

Azure Parallel Computing - HDinsight
- Azure Hadoop
- Azure Spark
- Azure Storm
  - Distributed streaming system
System Overview

Interface for Urban Data Management/Mining Applications

Trajectory Storage
- Azure Table
- Store
- Parse

Trajectory ST-Indexing
- Partitioned by Space & Time
- Spatio-temporal Index Building

Cached Trajectory Data
- redis

Trajectory Map-matching
- Inverted Indexing
- Map Matching
- Pre-Processing

ID-Temporal Query
Spatio-temporal Query
Map-Matched Traj.
Trajectory Storage

Step 1: Pre-Processing

Step 2: Trajectory Store

Storage Schema in Azure Table Storage

- \( p_1 \) lat, lng, t, speed, dir, ...
- \( p_2 \) lat, lng, t, speed, dir, ...
- \( p_3 \) lat, lng, t, speed, dir, ...
- \( p_4 \) lat, lng, t, speed, dir, ...

Trajectories

Parse

group->order->filter

Tables

Partitions

Table Entries
ID-query

• If the query asks for the most recent trajectory, our system answers it by retrieving the content from the Redis server. In this way, we can avoid the disk-related access and improve the response time.

  - plate ID + timestamp ➔ Redis server ➔ Result

• If the query asks for the historical trajectory data. Our system first checks the parameter of temporal range in the query. If the temporal range overlaps with multiple partitions, our system breaks the query into several small queries, to each data partitions and execute them in parallel.

  - plate ID + timestamp ➔ spatial tables ➔ Result
    - temporal partition
    - temporal partition
example:

For example: get the route of taxi NJ007 yesterday.

For example: get the route of taxi NJ007 around Edison between 20/01/2015 and 30/01/2015.
Trajectory ST-indexing

An extra spatio-temporal copy

- It may incur multiple accesses to multiple trajectory table (not efficient)
- Storage pricing is Cheap in Microsoft Azure (less than 0.1 USD per 100TB/month)

Storage Schema

- Spatial Partition -> Table
- Temporal Range -> Partition Keys

Grid-based Spatial Indexing

• Indexing
  – Partition the space into disjoint and uniform grids
  – Build inverted index between each grid and the points in the grid
Grid-based Spatial Indexing

• Range Query
  – Find the grids intersecting the range query
  – Retrieve the points from the grids and identify the points in the range
first exam the spatial range in the query to see the number of spatial grids are covered.

The system, then, retrieves the data from different spatial grids in parallel.

For each retrieval process in a spatial table, the temporal range of the query is also broken into different temporal partitions and executed in parallel.

generate the route of taxi NJ007 around Edison between 20/01/2015 and 30/01/2015.
Trajectory Map-Matching

Important function for many urban applications
- Traffic inference, route recommendations, and …

Challenges:
- Huge Volume
- Complex Computation
- Real-time requirement
Trajectory Map-Matching

After the map-matching process is done for the trajectories, the trajectory data is converted into a sequence of road segment ids. Our system also builds an inverted index for each road segment to store the IDs and corresponding timestamps of the trajectories that have passed it.

In this way, we are able to answer the temporal query like, “what trajectories have passed road segment road-i yesterday afternoon?”

Table and Redis

```
Segm       traj
       Segm     traj
       Segm     traj
```
Experiments of Trajectory Storage

In this experiment, 100 one-day trajectories are inserted. With a larger partition key, the insertion performance increases significantly first, and then becomes more consistent. It is because with a large partition key, more data can be inserted in one batch. On the other hand, when the partition key gets larger, the performance bottleneck changes to the network communication.

(b) Insertion wrt. Partition Sizes.
Experiments of Trajectory Storage

the data schema with smaller partition performs better, as in the large partition the cloud storage, the system needs to scan more table entities to retrieve the query results. Also data schema with less TimeRange performs better, as less data will be retrieved to the user.

(c) Response wrt. Partition Sizes.
Experiments of Map-Matching

The figure presents the results on our map-matching module using Storm. In each experiment, we performed the map-matching task on 6000 trajectories with average length of one hour. The experiments are done with different number of bolts per worker with three different cluster sizes (i.e., 5, 10 and 15).

It is clear in the figure that the cluster has higher number of data nodes always performs better.

(a) Bolts Per Worker.
Experiments of Map-Matching

a DataNode contains a 4-core CPU;

In this experiment, we test the efficiency of the map-matching module with different number of trajectories. The figure illustrates the performance, where with more trajectories the processing time increases. One interesting insight here is that, with the total number of trajectories less than 4,000, the performance improvement of the 15-data node cluster is very limited. However, an extra five data node would cost more than 2,500 USD per month. Hence, in that scenario, a 10-node cluster is a more economical solution.
Experiments of Map-Matching

In this experiments, we present the map-matching efficiency with different length of the trajectories, where each experiment performs the map-matching for 6,000 trajectories. It is clear from the figure c that with longer trajectories, map-matching task takes more time and the bigger cluster has lower processing time.
Experiments of Map-Matching

In Figure d, we present the map-matching efficiency with different trajectory sample rates (the average temporal intervals between consecutive GPS points), where each experiment performs the map-matching for 1,000 one hour trajectories. We can see that with trajectories with less interval time take more time, as there are more points to perform the map matching.
Case Study 1

Taxi Trajectory Data Management
- Plate-temporal Query
  - ID-temporal Query
- Spatio-temporal Range Query
- OD-based Trajectory Query

http://ubigdataplatform.chinacloudsites.cn/
Urban flow

Data Sets

- Real-time trajectory feed from ~7,000 taxi cabs in City of Guiyang, China

http://ubigdataplatform.chinacloudsites.cn/
Case Study 2

Real-Time Traffic Modeling System

After that, we can infer the travel speed of the road segment without any trajectories using a matrix decomposition model [5]. Then, we can infer the traffic volume and emission levels of each road segment using a graphic model [13].
The system models real-time traffic conditions on each road segment and infers city-wide gas consumption and pollution emissions of vehicles.
Case Study 3

Trajectory-based Resource Allocation

(a) System Overview

Pre-processing
- Map-Matched Trajectories
- Inverted Trajectory Index
- Vertex-vertex Index
- Spatial Indexing

Location Set Mining
- Optimal Solution
- Approx. Solution

Interactive Process
- Users
- $k$ Location Candidates
- Mining Parameters

Mining Process
Case Study 3

Trajectory-based Resource Allocation
Case Study 3
Conclusion

- We provide a first attempt on building a holistic cloud-based system on managing massive trajectories.
- The trajectory data in the system is stored in Azure Table with a properly designed storage schema and index to answer the ID-temporal queries.
- We use Storm platform to perform the map matching service.
- We evaluate our system design with the real taxi trajectories updated continuously from Guiyang City, China.
Q&A