A GPU-friendly Universal In-memory Storage System for Big Trajectory Data

Zhang Bowen
Department of Computer Science and Engeneering
Shanghai Jiao Tong University
Shanghai, China 200240
Email: zbw0046@sjtu.edu.cn

Abstract—**This is abstract ABSTRACT**

I. Introduction

1.background

i.Introduce trajectory. Application of trajectory database. For example, analysing the traffic flow in a range, searching hotels near to a location, and research the migration of wild animals... All these applications are all based on efficient trajectory database, which can support all types of query on trajectories and perform well on large dataset.

ii.As the number of GPS-location devices grows, large amount of trajectory data are collected. For example, xxx institude collect trajectory data from 1 million users which have 1 billion sample points...

iii.The growing number of trajectory data are generated every day, it calls for solution of efficient trajectory storage and retrieval. In these years many works are proposed to solve this problem such as in-memory trajectory storage system and GPU-accelerated query algorithm.

(picture of three kinds of query)

2.problem description

Although a mass of query algorithm are designed for different proposes recently, they can be concluded in three main types: point-based query, range-based query and trajectory-based query, which can be represented by kNN query, range query and trajectory similarity query respectively.(cited by ZhengYu,ZhouXiaofang's book) This three kinds of query are all indispensable for many kinds of trajectory applications. For example, xxx need xxx. However, there is no one of them can handle all kinds of query with high throughtput and low latency. Existing works are all optimized for one or two of them, but they can not be easily modified to support the queries not included in their original optimization goal.

3.challenge of solving problem

i.To solve this problem, it is intuitive to set two database for different kind of query respectively. Unfortunately, this will cause unneccessary space comsumption because of duplicate data are stored, which is harmful for memory efficiency. Considering that less space consumption means a mass of benefits such as more data can be persisted in main memory, it's neccessary to find a space-saving solution.

4.existing work about this problem

i.SharkDB propose a column-based data structure to store trajectory data leverageing the time-aggregation of queries. But it can't support trajectory-based query, such as trajectory similarity query. Besides, the parallel query algorithm that SharkDB develope is only for thread pool, which can't easily migrate to GPU.

ii.TKSimGPU propose a query algorithm that implemented on GPU which is designed for trajectory similarity query. However, the similarity measurement which TKSimGPU use is outdated, because it doesn't take local time shifting into concern, which has been proved a significant issue for similarity measurement of trajectory(cited by DTW,EDR,...). Moreover, range query and kNN query are not supported by TKSimGPU.

5.my idea

In this work, to solve this problem, we propose an in-memory trajectory storage system, which can handle all types of the basic query on historical trajectory data. This system can get both high throughtput and low storage overhead, which dominate the performance and main memory efficiency of trajectory database. Meanwhile, our design is friendly with GPU, which means we can make use of parallel power from it to accelerate query engine.

6.challenge of idea

i.To implement this idea, there are some challenges as follow. First, it is difficult to design index for all the three types of query. In existing work, people use location-based index which is metric to handle range query and kNN query, for example, R-tree. However, trajectory similarity query calls for pruning methods which is based on inequation derived from the properties of similarity measurements, which is usually not metric.

ii.Many issues should be concerned to improve performance by answering queries with the help of GPU. For example, in traditional trajectory index such as R-tree, pruning is implemented by search every node recursively. However, recursive operation is not support by GPU efficiently.

(I'm finding more challenge...)

7.main issues

i. To address these challenges, we propose an index which have two level. At the first level, big bounding boxes are built based on location, which contains small bounding boxes at the second level. At the second level, small bounding boxes related to the bound of similarity measurement are built.

ii.We store nodes consecutively to improve GPU-accelerated query's performance. (More details about GPU) (considering better solution...)

8.contributions

- We design a in-memory trajectory storage system which support both point-based query, range-based query and trajectory-based query.
- We design GPU-friendly data arrangement and index carefully to fit with in-memory environment.
- We propose a parallel similarity search algorithm which can work well on GPU and support local time shifting.
- We deploy our system on server, evaluate the performance, and prove that our system get the 3x higher performance than state-of-the-art system.

9.organization

The rest of paper is organized as follows. In Section 2 we review the related work. After that, in section 3 we briefly introduce the architecture of our system. We propose our design of storage system in section 4. Query engine is described in detail in section 5 with three kinds of query algorithm. In section 6, we do an evaluation of our system and results are reported. Section 7 concludes the paper.

II. Background

A. Problem Definition

In this section, we propose some basic definition of trajectory and formulation about our problem. We first define some elements of trajectory.

Definition 1 (sample point): A sample point $p = (x, y, time)$ is a three-dimensional data which include spatial information represented by (x, y) and time stamp $time$. For simplicity, we assume that all sample points' coordinates have been transfered to Euclidean plane.

Definition 2 (segment): A segment $e = (p_1, p_2)$ is a line connecting two sample points, with noting that $p_1 \in e$ and $p_2 \in e$.

Definition 3 (trajectory): A trajectory of the object $t = \{e_1, e_2, \dots, e_n\}$ is a sequence of consecutive segments. It can also be represented by a sequence of sample points $t = \{p_1, p_2, \dots, p_{n+1}\}$,where $\forall i \in [1, n], e_i = (p_i, p_{i+1})$.

Given a large set of trajectories $T = t_1, t_2, \dots, t_{|T|}$, our goal is answering all kinds of queries over them. As we mentioned in Section 1, they can be classified into range query, k-nearest neighbor point query and top-k trajectory similarity query. Here we formulate this three kinds of query.

Definition 4 (query): A query $q = (kind, cond, T)$ aims to find a set of trajectories or segments as results S from trajectory dataset T which satisfy query conditions $cond$, where $kind$ is the notation of the kind of the query.

For range query, the condition usually includes a bounding box $B = [xmin, xmax, ymin, ymax]$ and a time range $[t_s, t_e]$. A segment e is said to satisfy range query if the two sample points $e.p_1$,$e.p_2$ are both included in the bounding box and their time stamps are all within time range, which can be represented as $\forall p \in e, (xmin \leq p.x \leq xmax) \wedge (ymin \leq p.y \leq ymax) \wedge (t_s \leq p.time \leq t_e)$.

K-nearest neighbor point query retrieves a set of k trajectories $R_{kNN} = \{t_1, t_2, \dots, t_k\}$ which are top-k nearest to given coordinate (q_x, q_y) in query condition. We use euclidean distance between the coordinate and trajectory as measurement of nearness.

Top-k trajectory similarity query retrieves a set of k trajectories $R_{sim} = \{t_1, t_2, \dots, t_k\}$ which are most similar with given trajectory t_q. There are many metric about trajectory similarity, and it's widely accepted that EDR distance(cited by EDR) which can handle local time shifting may be a good choice. So we use EDR distance as our similairity metric.

Definition 5 (EDR distance): (cited by EDR) Given two trajectories $t_1 = \{p_1, p_2, \dots, p_{n1+1}\}, t_2 = \{p_1, p_2, \dots, p_{n2+1}\}$, the EDR distance between them is calculated by:

$$EDR(t_1, t_2) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ \min\{EDR(Rest(t_1), & \\ Rest(t_2) + subcost), & \\ EDR(Rest(t_1), t_2) + 1, & \\ EDR(t_1, Rest(t_2)) + 1\} & \text{otherwise} \end{cases}$$

where $subcost = 0$ if $dist(t_1.p_1, t_2.p_1) \leqslant \varepsilon$ and $subcost = 1$ otherwise, and $Rest(t) = \{t.p_2, \dots, t.p_{n+1}\}$, noting that ε is a threshold set by users.

B. more?

III. System Architecture

(architecture picture)

IV. GPU-friendly In-memory Storage

A. Data Arrangement

B. GPU-friendly Index

(waiting for designing)

V. GPU-accelerated Query Engine

A. Formulation

formulation of three queries

B. Range Query

C. kNN Query

D. Trajectory Similarity Query

VI. Evaluation

A. Environment

1) Dataset:

2) Metric: throughtput and latency memory consumption

3) Influence of Parameters Setting:

B. Scalability

C. Query Performance

compare with state-of-art method including TrajStore, SharkDB(Non-GPU method), ST2I+consistent storage(GPU method, intuitive approach, use s-t index to implement trajectory index) show that our system is more accurate than TKSimGPU(GPU method)

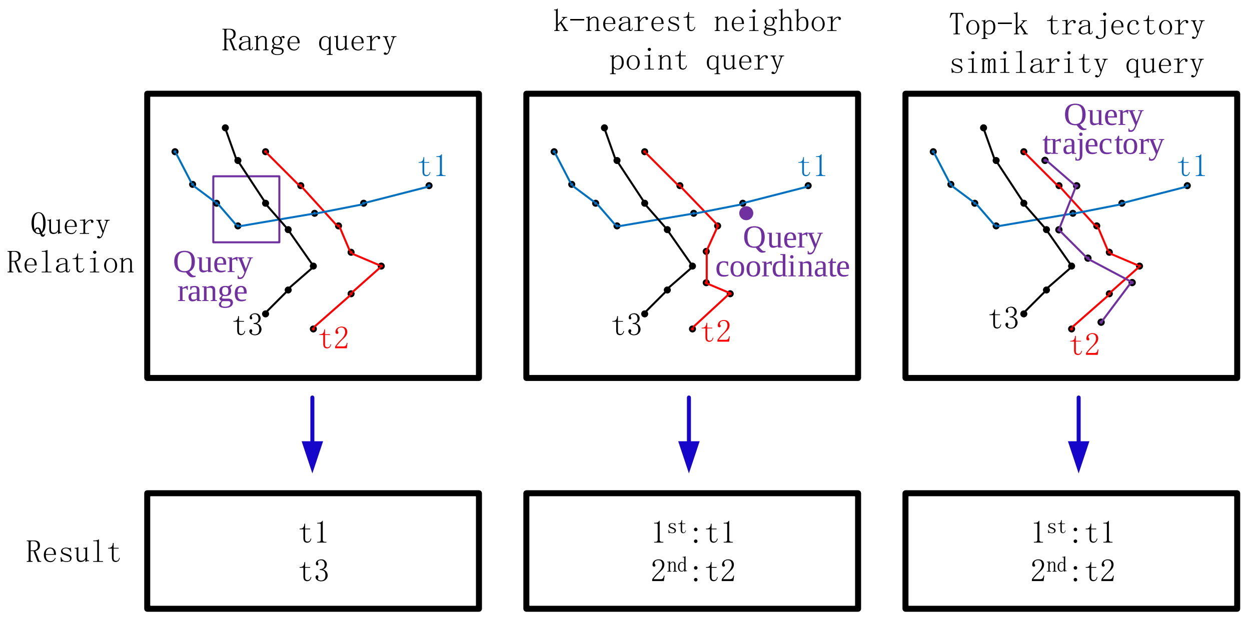

Fig. 1. three kind of query

D. *Impact of Parameters*

E. *Other Details Should Be Evaluated*

VII. Related Work

A. *Trajectory Storage and Indexing*

B. *GPU-accelerated Storage System*

VIII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.