Ratel: Interactive Analytics for Large Scale Trajectories

Haoda Li, Guoliang Li*, Jiayang Liu, Haitao Yuan, Haiquan Wang
Department of Computer Science, Tsinghua University
lhd16,jiayang-16,yht16@mails.tsinghua.edu.cn,liguoliang@tsinghua.edu.cn

ABSTRACT

Trajectory data analytics plays an important role in many applications, such as transportation optimization, urban planning, taxi scheduling, and so on. However, trajectory data analytics has a great challenge that the time cost for processing queries is too high on big datasets. In this paper, we demonstrate a distributed in-memory framework `Ratel` base on Spark for analyzing large scale trajectories. `Ratel` groups trajectories into partitions by considering the data locality and load balance. We build R-Tree based global indexes to prune partitions when applying trajectory search and join. For each partition, `Ratel` uses a filter-refinement method to efficiently find similar trajectories. We show three kinds of scenarios - bus station planning, route recommendation, and transportation analytics. Demo attendees can interact with a web UI, pose different queries on the dataset, and navigate the query result.

ACM Reference Format:
Haoda Li, Guoliang Li, Jiayang Liu, Haitao Yuan, Haiquan Wang. 2019. Ratel: Interactive Analytics for Large Scale Trajectories. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320222

1 INTRODUCTION

The widespread use of GPS sensor equipped devices and vehicles makes it much easier to track locations of moving objects, such as cars, people and animals. The trajectories - the sequence of geo-location points - are generated continuously by the device and can be collected into the cloud for immediate or further use. For example, Didi (a Uber-like company in China) had about 2 billion car trajectories in 2016, and each trajectory contains dozens to hundreds of geo-location points which are recorded every few seconds.

Trajectory analysis can benefit many real-world applications such as taxi scheduling, transportation optimization, urban planning, trajectory recommendation, and so on. Users aim to interactively perform the trajectory analytics, where the query results should be returned instantly. However the data size of real-world trajectories is typically large, which makes the typical operations on trajectories - similarity search, similarity join - inefficient.

To solve the problem, we implement `Ratel`, a distributed in-memory trajectory analytics framework based on Spark, to efficiently perform the search and join operation. `Ratel` partitions trajectories base on data locality, load balance, and partition size. These partitions are indexed by R-Tree based indexes according to the trajectory head and tail points. Global indexes can effectively find partitions that may contain similar trajectories for a given query. In each partition, `Ratel` employs a filter-refinement method based on a new concept of *the set of Minimum Bounding Rectangles* (sMBR) to find similar trajectories.

Moreover, we build a web UI to help users easily pose queries and visualize results. Demo attendees can generate queries, submit queries and check the query results through the web UI. And the results are shown as Heat Map or drawn directly after sampling. The supported trajectory analytics in `Ratel` include:

Trajectory Range Search. Given a set of trajectories and query ranges, we can efficiently find trajectories that start or end in the query regions.

Trajectory Similarity Search. Given a set of trajectories and a query trajectory, `Ratel` finds all the trajectories matching the query - their similarity is higher than a given threshold.

Trajectory Clustering. Given a set of trajectories, `Ratel` first perform a self-join on it to get the distances between the trajectories. Then, `Ratel` apply some distance-based cluster algorithms on it to divide the trajectories into clusters.

Due to our effective indexes, efficient search and join algorithms, and cache mechanism, almost all of the queries can get the results in seconds. For more technical details, please refer to our full research paper [6, 8].

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3320222

*Guoliang Li is the corresponding author.

2 OVERVIEW

2.1 Definitions

Definition 1. A **trajectory** $T = (p_1, p_2, ..., p_n)$ is a sequence of points, where each point $p_i (i \in [1, n])$ is a d-dimensional tuple. We call the first point p_1 the $T.head$, the last point p_n the $T.tail$, and n the trajectory length (i.e., $|T| = n$).

In `Ratel`, we use the taxi trajectory dataset so that each point has two dimensions ($d = 2$), which capture the longitude and latitude of a location.

Distance Measure. There are many similarity/distance functions [1] [3] [4] to quantify the similarity between two trajectories. We use the *Dynamic Time Warping* (DTW) [2] in our demo because it has been shown to be robust and accurate from many experimental evaluations [7].

Definition 2. Given two trajectories $Q = (q_1, q_2, ..., q_n)$ and $C = (c_1, c_2, ..., c_n)$, their *DTW* distance, denoted by $D_{DTW}(Q, C)$, is defined as:

$$D_{DTW}(Q,C) = \begin{cases} 0 & \text{if } m = n = 0 \\ +\infty & \text{if } m > n = 0 \\ +\infty & \text{if } n > m = 0 \\ dist(q_1, c_1) + min\{ & \\ \quad D_{DTW}(REST(Q), REST(C)), & \\ \quad D_{DTW}(REST(Q), C), & otherwise \\ \quad D_{DTW}(Q, REST(C)) \quad \} & \end{cases} \tag{1}$$

where $REST(Q)$ is the suffix of Q without the first point, i.e., $REST(Q) = (q_2, ...; q_n)$, and $dist(q_1, c_1)$ is the distance between points q_1 and c_1. In our demo, $dist(,)$ is defined as Euclidean distance.

2.2 Base Operations

The trajectory analytics are based on the following operations:

Definition 3. Trajectory Similarity Search. Given a query trajectory Q, a collection of trajectories $\mathcal{T}$, and a threshold τ, the trajectory similarity search finds all trajectories $T \in \mathcal{T}$, such that $D_{DTW}(T, Q) \leq \tau$.

Definition 4. Trajectory Similarity Join. Given two collections of trajectories $\mathcal{T}, \mathcal{Q}$, and a threshold τ, the trajectory similarity join finds all trajectories pairs (T, Q), such that $T \in \mathcal{T}$, $Q \in \mathcal{Q}$ and $D_{DTW}(T, Q) \leq \tau$.

Definition 5. Trajectory Head-Tail Range Search. Given 2 query rectangle Q_1, Q_2 on the 2-dimensional spatial plane, and a collection of trajectories $\mathcal{T}$, the trajectory head-tail range search finds all trajectories $T \in \mathcal{T}$, such that $T.head$ is in Q_1 and $T.tail$ is in Q_2.

2.3 Architecture

We implement the `Ratel` analytic framework on Spark. Here we briefly describe the architecture of our demo. The demo mainly consists of four parts - the `Ratel` framework, a query library, a cache manager, and a web service.

Ratel Query Framwork. `Ratel` provides the basic operations - similarity search, join, and range query - based on Spark. `Ratel` distributes trajectories data into partitions by considering data locality, load balance, and partition size. To efficiently perform the operations as described in Section 2.2, global indexes and local indexes are built for the partitions and maintained in the memory.

Query Library. The basic operations provided by `Ratel` are not convenient enough for user analyzing trajectories. The query library implements some additional operations based on the operations described in Section 2.2, such as sampling, clustering, and aggregation.

Cache manager decides whether some intermediate search results should be cached for further computation. By default the latest 5 query results are cached in the Spark and old cached data is cleared by calling the Spark function *unpersist*.

The web UI provides a graphic UI for users to generate queries and submit them to the Spark platform. The web backend is built with Akka[†] and the front-end UI is developed based on AMap[‡].

3 IMPLEMENTATION

We briefly introduce how we implement the core operations - similarity search and join - in trajectory analysis. For more technical details, please refer to our full research paper [6]. The basic idea is to employ a filter-refinement strategy to find trajectories that satisfy the constraint. In the filter step, we derive several lower bounds of the distance between trajectories to build indexes and prune trajectories. In the refinement step, `Ratel` verifies each candidate trajectory and produces the final results.

3.1 Distance Lower Bounds

There are different kinds of distance lower bounds in our implementations.

First, for computing DTW of two trajectories, their heads and tails must match. So these two points can be used as a loose lower bound:

Head-Tail Lower Bound. For any two trajectories Q and C, where $|Q| = n$, $|C| = m$, $n > 1$ and $m > 1$, the following inequality holds: $dist(q_1, c_1) + dist(q_n, c_m) \leq D(Q, C)$. Denote $LB^{ht}_{QC} = dist(q_1, c_1) + dist(q_n, c_m)$ as the Head-Tail Lower Bound. This lower bound is mainly used for building global indexes.

[†]https://akka.io/

[‡]http://www.amap.com/

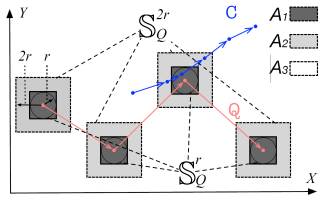


Figure 1: Progressive Lower Bound

Then, to introduce another distance lower bound, we first introduce **the set of Minimum Bounding Rectangles**.

To find trajectories whose distances to a query trajectory Q are no larger than a threshold ϵ. Let $r = \epsilon/n$, where $n = |Q|$. Using r as the radius, we plot n circles centered at each point of Q. To facilitate subsequent computations, we substitute these n circles with n squares of length $2r$. Each square corresponds to the *Minimum Bounding Rectangle* (MBR) of a circle. These n squares (i.e., MBRs) form **the set of Minimum Bounding Rectangles** (sMBR) for query Q with radius r (e.g., the 4 dark gray rectangles in Fig.1). We write $\mathcal{S}_Q^r$ as the sMBR of Q relative to r.

We enlarge each square in $\mathcal{S}_Q^r$ to construct $\mathcal{S}_Q^{\alpha r}$ (with αr as the half length of a square where $\alpha > 1$. Consequently, the points on the plane are divided into three different areas A_1 (The same area as $\mathcal{S}_Q^r$), A_2 (The area outside of $\mathcal{S}_Q^r$ but inside of $\mathcal{S}_Q^{\alpha r}$) and A_3 (The area outside of $\mathcal{S}_Q^{\alpha r}$). For each area, we estimate the distance from a point in a trajectory C to the matching point in Q. If a point in C falls in area A1, the lower bound is 0. For areas A2 and A3, the lower bounds are r and αr, respectively.

By counting the points of any trajectory C in each of the areas, we get a **Progressive Lower Bound**:

$$LB_{QC}^{pg} = N_{A_1} \times 0 + N_{A_2} \times r + N_{A_3} \times \alpha r$$

where N_{A_1}, N_{A_2} and N_{A_3} represents the number of points of C falls in A_1, A_2 and A_3.

The **Progressive Lower Bound** is mainly used to build local indexes.

3.2 Indexing and Query Implementation

3.2.1 Indexing. The indexes consist of global indexes and local indexes.

Global Indexes. For each partition, `Ratel` first extracts the *head* of each trajectory and computes the MBR enclosing all of these *heads*, yielding R_{head}. All partitions' R_{head} form the set $\mathcal{R}_{head}$. Similarly, we get R_{tail} that encloses all tails in a partition and $\mathcal{R}_{tail}$ contains all partitions' R_{tail}. `Ratel` then constructs two R-Tree indexes for the set $\mathcal{R}_{head}$ and the set $\mathcal{R}_{tail}$, respectively to speed up the global pruning. Then the global indexes can be used to find partitions that might contain results for a given query.

Local Indexes. For a given query, to compute the *Progressive Lower Bound* LB_{QC}^{pg} we described in Section 3.1, we need to count the points of a trajectory falling in $\mathcal{S}_Q^r$ and $\mathcal{S}_Q^{\alpha r}$. So we construct an mesh index for all points of each trajectory in the partition that each point of each trajectory is assigned to a certain grid.

3.2.2 Search Implementation. The search queries are processed in three steps: (1) Base on the *Head-Tail Lower Bound* LB^{ht}, the driver uses the global indexes to find out all the partitions that might contain trajectories similar to the query Q. Then it sends Q to each worker of the partitions. (2) In each partition, the worker applies the filter-refinement strategy to find all the trajectories that satisfy the constraint based on the *Progressive Lower Bound*. (3) The driver collects the final results.

3.2.3 Join Implementation. Similar to the search query, the join query process consists of three steps: (1) Find all the partition pairs that might contain similar trajectories. (2) For each partition pair, apply a local join. (3) Collect the results.

3.2.4 Query Optimization on Complex Queries. For a complex query with multiple operations, we propose two optimization techniques. First, for a set of similar queries that can be optimized with indexes, we can follow the filter-refinement strategy to execute them at the same time, which is more efficient than execute them one by one. Second, for queries with different types of operations, such as range search, join and sampling, a cost-based optimization strategy similar to the cost model in [5] can be used to adjust the query order, filter strategy, and execution plan.

4 DEMONSTRATION

We demonstrate three use cases of the `Ratel` analytic framework: (1) bus station planning based on trajectory range search, (2) route recommendation base on range query and clustering, (3) transportation analysis based on trajectory similarity search. Demo attendees can generate queries and visualize results through the web UI we built based on AMap.

The dataset we used is taxi trajectories in Beijing, which are collected from the taxi GPS data and contains about 10 million trajectories.

Bus Station Planning. The most important points of a trajectory are the starting point and the ending points. Analyzing these points can benefit many applications. For example, if we want to design new bus lines starting from the airport. We need to decide where to set the bus stations. As shown in Fig. 2, we can first draw a range query around the airport area to find all the trajectories starting from the airport. By drawing a heat map of their ending points, we can find out where the trajectories mostly go. It shows that most of the passengers starting from the airport do not go too far from

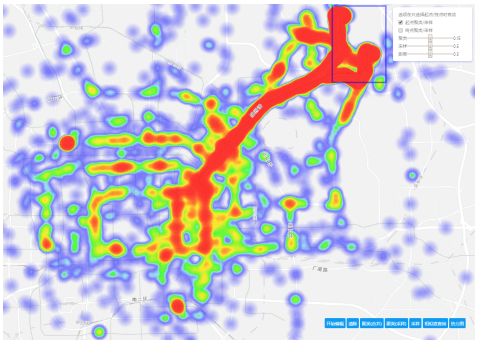


Figure 2: Bus Station Planning base on Heat Map

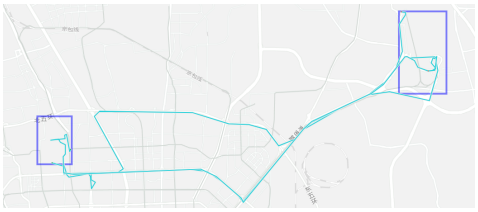


Figure 3: Route Recommendation

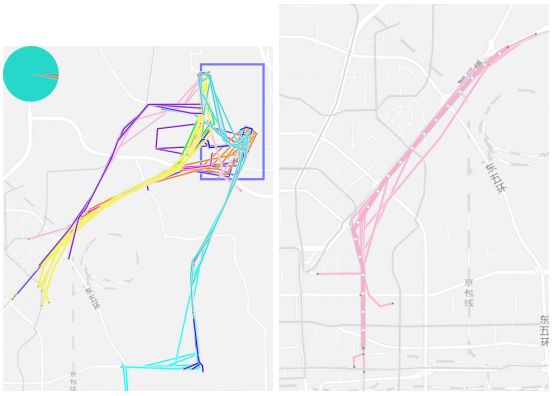
the airport because the subway would be a better choice for farther passengers, which means the bus should stop more frequently at places close to the airport.

Route Recommendation. One of the applications about trajectory is route recommendation given a starting area and a destination area. We can easily implement this by range query. Let us continue our new bus lines planning. After deciding where to place the bus stations, we need to decide how the bus get to these stations from the airport. As we can see from Fig. 3, if we want to know the possible routes from the airport to a certain area, we draw 2 rectangles to represent the starting area and the destination area and apply a range query. The range query results show that there mainly exists 2 types of routes from the airport to the destination area. We can use the bottom one as our bus route.

Transportation Analysis. One of transportation analytics is to find out how trajectories get out from a certain area. But the number of trajectories is usually too large to easily extract valuable information. So we need to apply clustering to the query results. Fig. 4a shows our results - we draw the trajectories in the same cluster with the same color. We can observe that there are mainly three types of trajectories that get out of the airport area. Then for a typical trajectory shown in Fig. 4b, we can apply the trajectory similarity search directly to get similar trajectories passing through a certain road.

5 RELATED WORK

The most recent work similar to our work is DITA[5]. DITA implements distributed trajectory search and join operations on Spark and extends Spark SQL to support these operations. There are mainly three differences between DITA and our



(a) Clustering **(b) Similarity Search**

Figure 4: Transportation Analysis

work. First, DITA provides a SQL interface, which is not convenient enough for interactive analytics Second, we provide a library that consists of a set of additional operations to optimize the interactive analytic experience, such as sampling and clustering. But DITA only provides the basic operations. Third, the definition of DITA's range query is different from ours - DITA searches for trajectories with all points in the given area, while we search for trajectories with head or tail in it. Moreover, DITA's implementation is not as efficient as ours due to the time-consuming pre-processing [6].

ACKNOWLEDGEMENT

This work was supported by the 973 Program of China (2015CB358700), NSF of China (61632016, 61521002, 61661166012), Huawei, and TAL education.

REFERENCES

[1] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications* 5, 01n02 (1995), 75–91.
[2] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *KDD workshop*, Vol. 10. 359–370.
[3] Lei Chen and Raymond Ng. 2004. On the marriage of lp-norms and edit distance. In *VLDB*. VLDB, 792–803.
[4] Jean-Francois Hangouet. 1995. Computation of the Hausdorff distance between plane vector polylines. In *AUTOCARTO-CONFERENCE-*. 1–10.
[5] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. Dita: Distributed in-memory trajectory analytics. In *SIGMOD*. ACM, 725–740.
[6] Haiquan Wang, Guoliang Li, Nan Tang, and Jianhua Feng. 2019. Distributed Trajectory Similarity Search and Join. In *VLDB*.
[7] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *DMKD* (2013).
[8] Haitao Yuan and Guoliang Li. 2019. Distributed In-Memory Trajectory Similarity Search and Join on Road Network. In *ICDE*.