Distributed Spatio-Temporal *k* Nearest Neighbors Join

Ruiyuan Li, Rubin Wang, Junwen Liu, Zisheng Yu,
Huajun He, Tianfu He, Sijie Ruan, Jie Bao,
Chao Chen, Fuqiang Gu, Liang Hong, Yu Zheng

Chongqing University, China

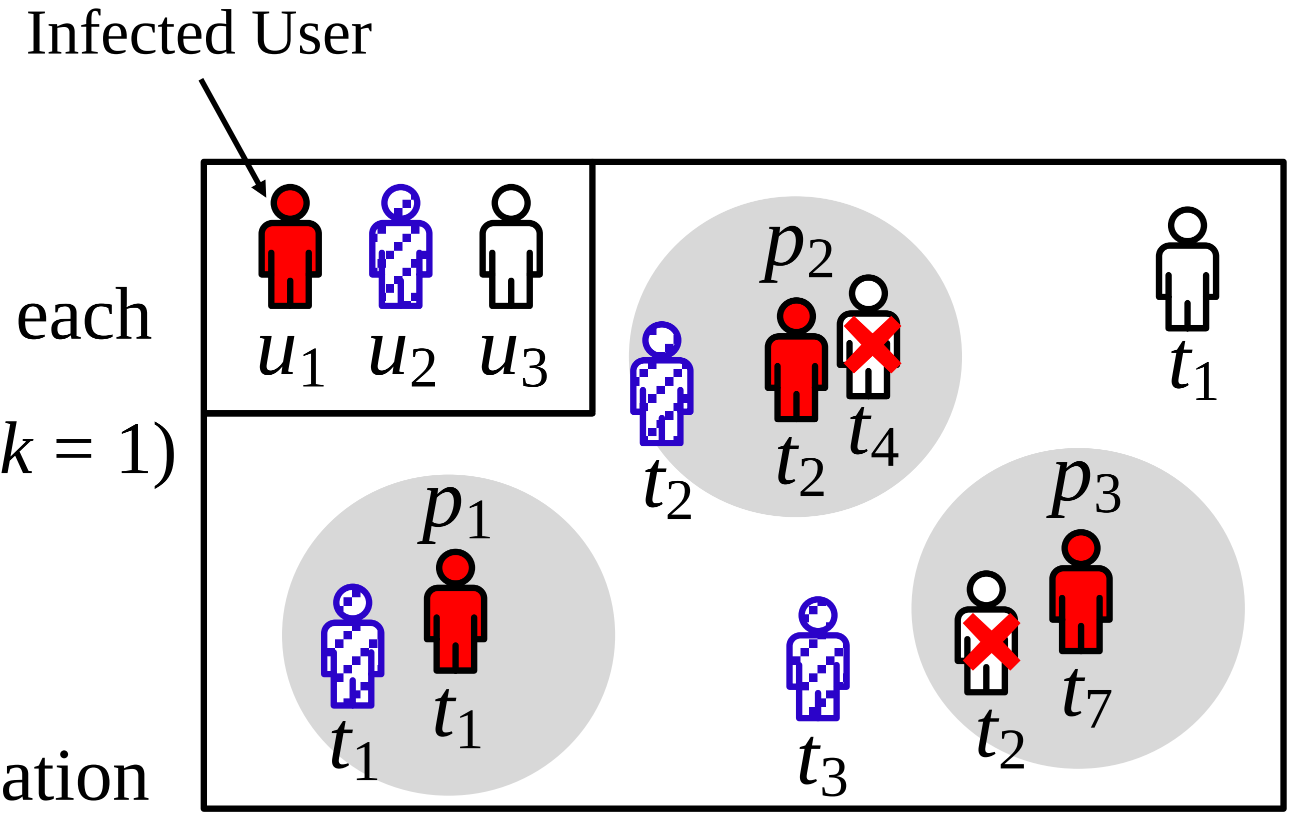
liruiyuan@cqu.edu.cn

System Demo

About Me

An Epidemic Prevention Example

- ❑ Suppose u_1 is an infected user
- ❑ Find the spatially nearest user of each check-in point of u_1 (i.e., kNN join, $k = 1$)
  - ❑ kNN(p_1) = u_2, kNN(p_2) = u_3, kNN(p_3) = u_3
  - ❑ Both u_2 and u_3 are potentially infected
- ❑ If we also consider temporal information
  - ❑ st-kNN(p_1) = u_2, st-kNN(p_2) = u_2, st-kNN(p_3) = NaN
  - ❑ Only u_2 is potentially affected ⟶ A more precise epidemic prevention!

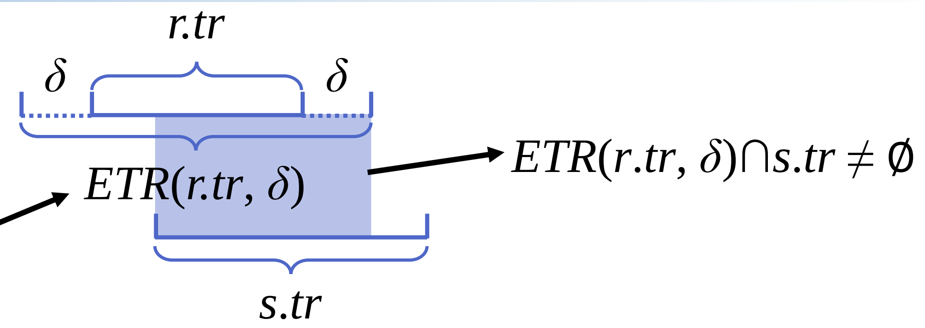


Spatial Closeness + Temporal Concurrency → ST-kNN Join

ST-kNN Join

❑ Definition of ST-*k*NN

- ❑ Given
  - ❑ Object *r* and object set ***S***
  - ❑ Integer *k* and threshold *δ*
- ❑ *r*’s ST-*k*NN in ***S***
  - ❑ Temporal Concurrency
  - ❑ Spatial Closeness: *s* ∈ ***S*** is the *k* nearest neighbors of *r* that satisfies temporal concurrency

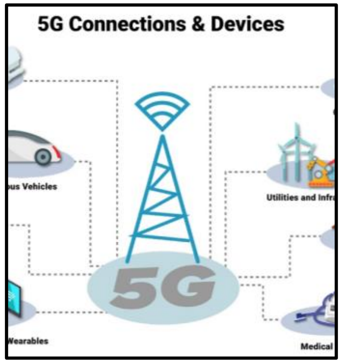


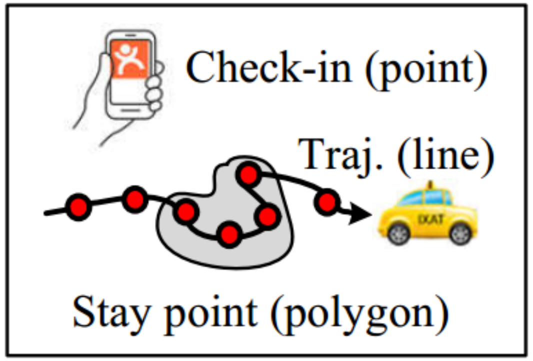
❑ Definition of ST-*k*NN Join

- ❑ $R \ltimes S = \{(r, s) | \forall r \in R, \forall s \in \text{ST-}k\text{NN}(r, k, \delta, S)\}$

❑ Challenges

- ❑ **Big Data**: Era of IoT and 5G
- ❑ **High Dimensionality**: Spatial + Temporal
- ❑ **Various Geometry Types**: Point, Line String, Polygon



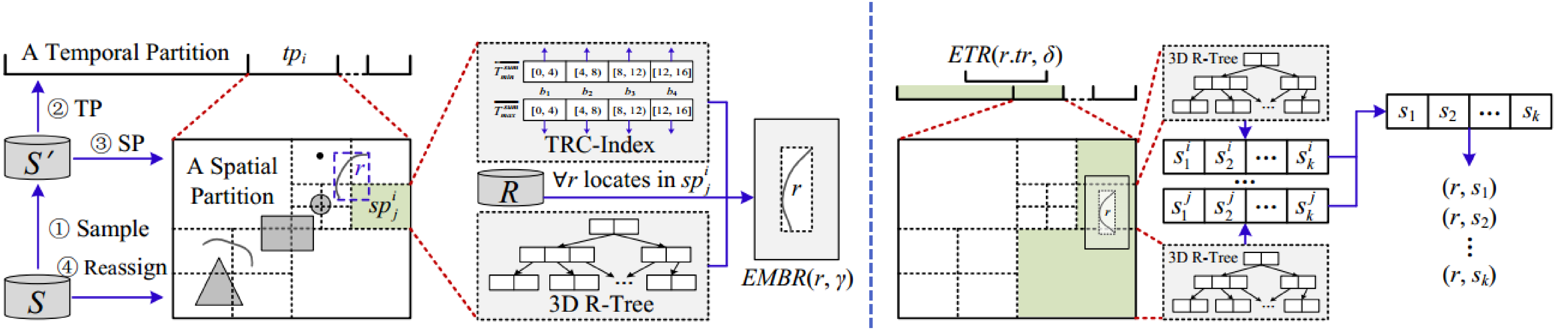


❑ Most Existing Works for *k*NN Join

- ❑ Ignore the Temporal Information
- ❑ Do Not Support Complex Geometries, e.g., Line Strings, Polygons.

We are the **first** to address the problem of ST-kNN Join

Framework: Two Round Join with Four Steps



(a) Data Partition for S

(b) First Round Local Join

(c) Second Round Local Join

(d) Merge Result

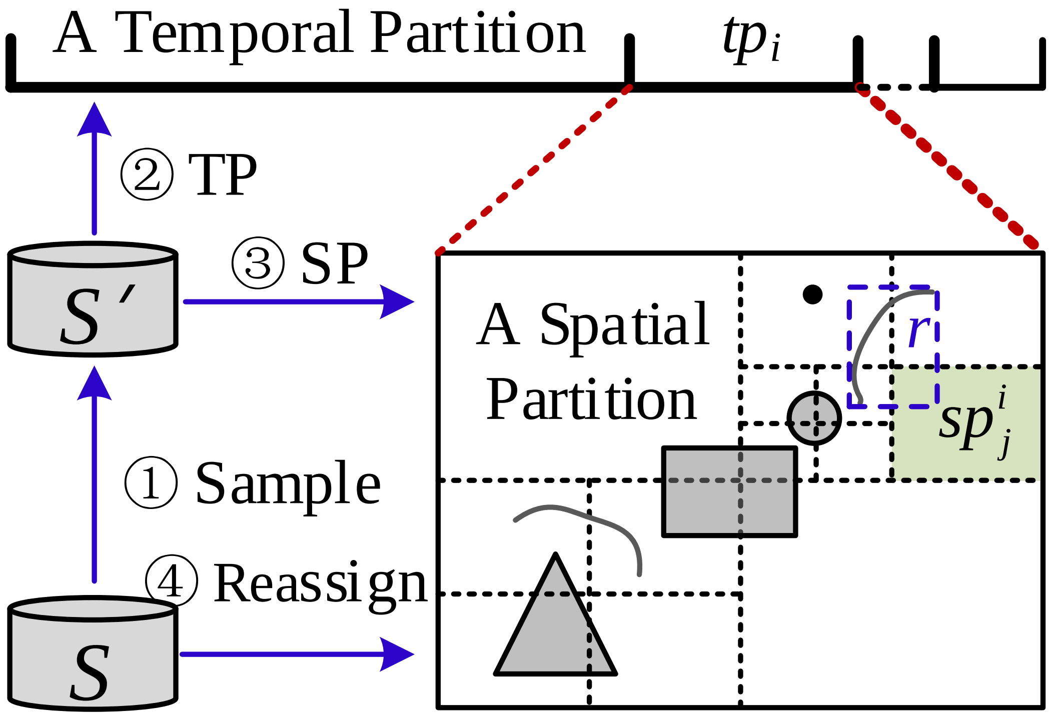
First Round Join

Second Round Join

- ❑ Process Big Spatio-Temporal Data Based on Apache Spark
- ❑ Consider Both Temporal Concurrency and Spatial Closeness
- ❑ Support All Geometry Types
  - ❑ Point, line string, polygon, or even a mixed set of them

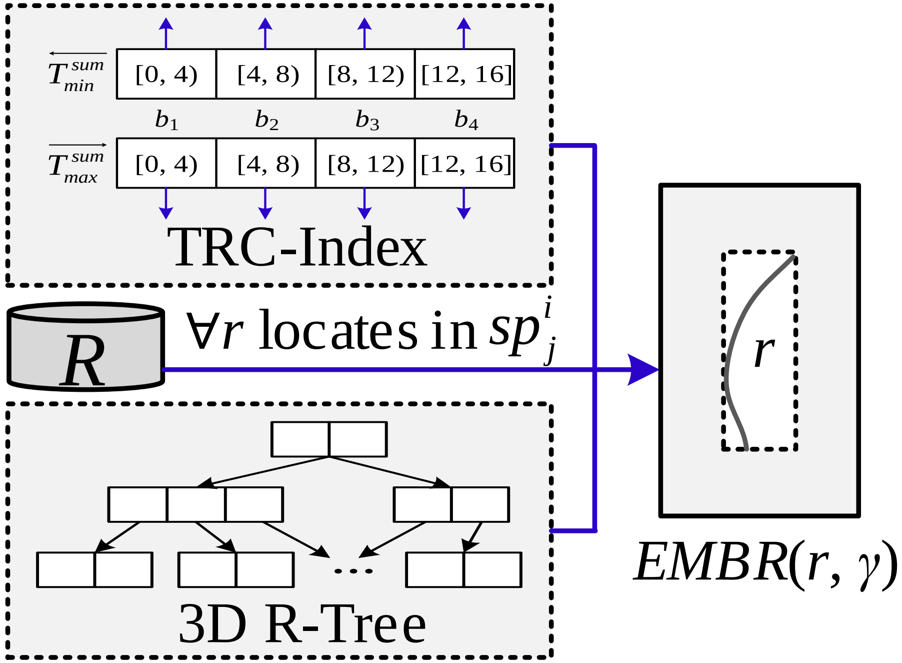
Step 1: Data Partition for *S*

- ❑ Goals
  - ❑ Spatio-Temporal Proximity
    - ❑ Each *r* find possibly its ST-*k*NNs in one partition
  - ❑ Even Distribution
    - ❑ Load balance
- ❑ Method
  - ❑ **Sample** randomly *S'* from *S*
  - ❑ **Temporal partition** using Sweep Line Alg.
    - ❑ Max temporal partition number: α (system para.)
    - ❑ Disjoint, roughly equal number of samples
  - ❑ **Spatial partition** based on Quad Tree
    - ❑ Max spatial partition number: β (system para.)
    - ❑ Disjoint, equal number of samples
  - ❑ **Reassign** $s \in S$ based on ST-partitions ⟶ Make multiple copies if *s* intersects multiple ST-partitions.



Step 2: First Round Local Join

- ❑ Goals
  - ❑ For each $r \in R$, find an area EMBR(r, γ), such that its ST-kNNs must intersect with EMBR(r, γ)
- ❑ Method
  - ❑ **Index Construction** in Each ST-partition
    - ❑ **TRC-index**: decide whether a partition has at least k objects that meet the temporal concurrency requirement
    - ❑ **3D R-Tree**[1]: support fast ST-kNN search
  - ❑ **Data Partition for R**
    - ❑ For each $r \in R$, reassign it to the nearest ST-partition that has at least k objects satisfying the temporal concurrency requirement, based on TRC-index
  - ❑ **Distance Bound Calculation**
    - ❑ Calculate γ based on the k-th nearest neighbor with 3D R-tree

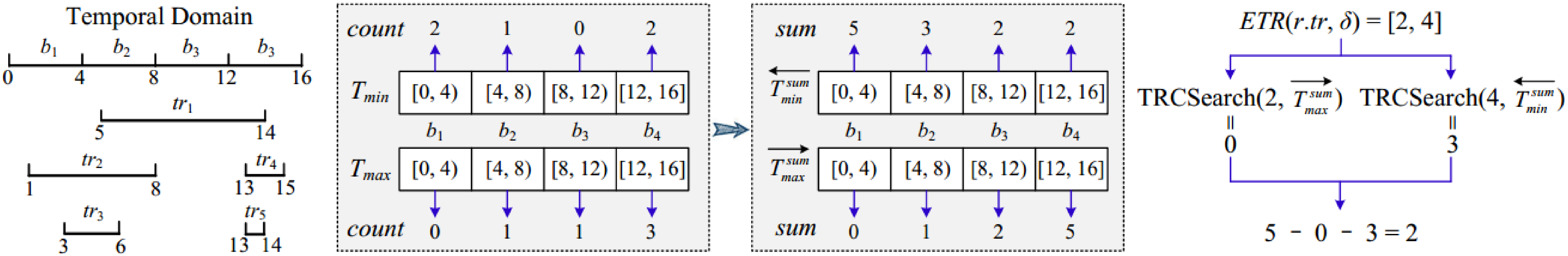


[1] Zhu Q, Gong J, Zhang Y. An efficient 3D R-tree spatial index method for virtual geographic environments[J]. ISPRS Journal of Photogrammetry and Remote Sensing, 2007, 62(3): 217-224.

TRC-Index: Time Range Count Index

- Requirements
  - Efficiency
    - Get the *minimum* number of objects whose time ranges intersecting a given time range
  - Lightweight
    - Should be small enough to be broadcast

- Intuition: **Exclusive Method**
  - If the number of objects whose time ranges will not intersect with *tr* is at most *N*, then the number of satisfied objects is at least $|S_i|$ - *N*

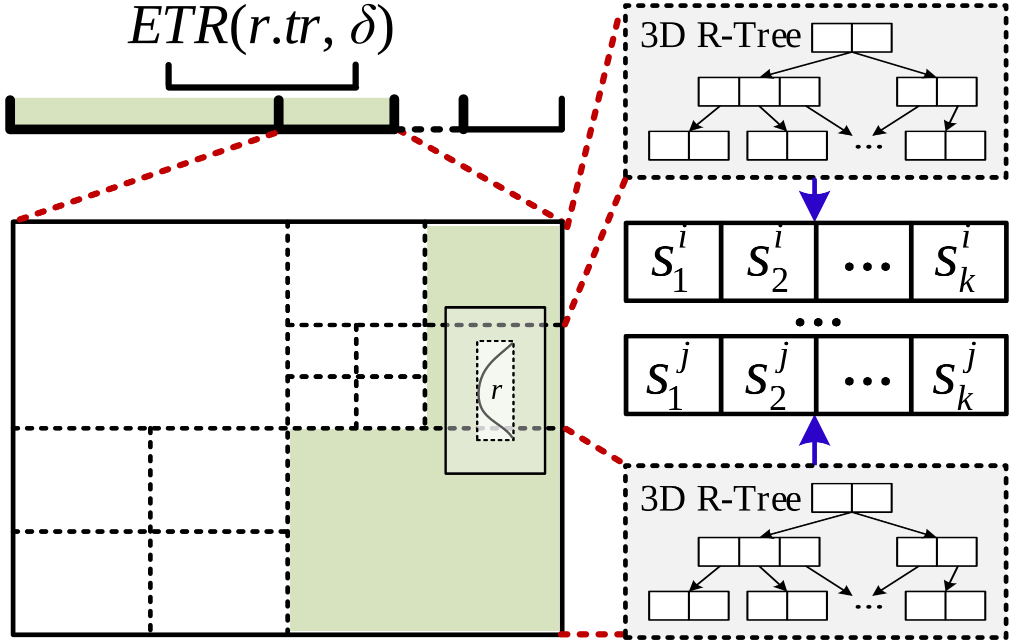

Temporal Domain
b_1 b_2 b_3 b_3
0 4 8 12 16
tr_1 5 14
tr_2 1 8
tr_4 13 15
tr_3 3 6
tr_5 13 14
count 2 1 0 2
T_{min} [0, 4) [4, 8) [8, 12) [12, 16]
b_1 b_2 b_3 b_4
T_{max} [0, 4) [4, 8) [8, 12) [12, 16]
count 0 1 1 3
sum 5 3 2 2
$\overleftarrow{T^{sum}_{min}}$ [0, 4) [4, 8) [8, 12) [12, 16]
b_1 b_2 b_3 b_4
$\overrightarrow{T^{sum}_{max}}$ [0, 4) [4, 8) [8, 12) [12, 16]
sum 0 1 2 5
$ETR(r.tr, \delta) = [2, 4]$
TRCSearch(2, $\overrightarrow{T^{sum}_{max}}$) TRCSearch(4, $\overleftarrow{T^{sum}_{min}}$)
0 3
5 - 0 - 3 = 2

(a) Time Range Database (b) Count for Each Bin (c) TRC-Index (d) TRC Search Example

An Example of TRC-Index

Step 3: Second Round Local Join

- ❑ Goals
  - ❑ For each $r \in$ R, check all possible ST-partitions, and generate local results.
- ❑ Method
  - ❑ **Reassign** $r \in$ R Based on $EMBR(r, \gamma)$ and $ETR(r.tr, \delta)$
  - ❑ Perform a **local ST-kNN search** Based on 3D R-Tree

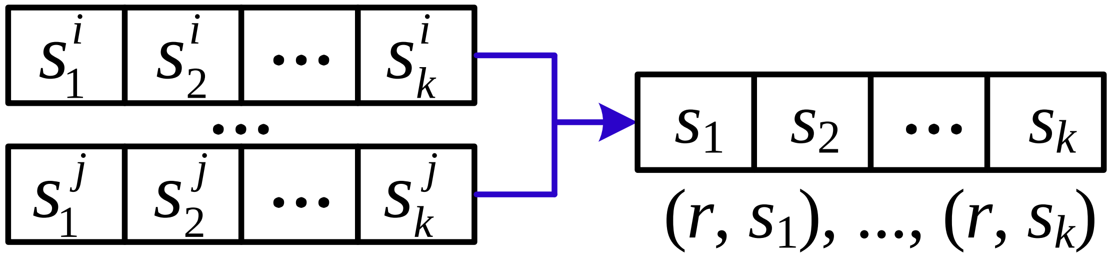


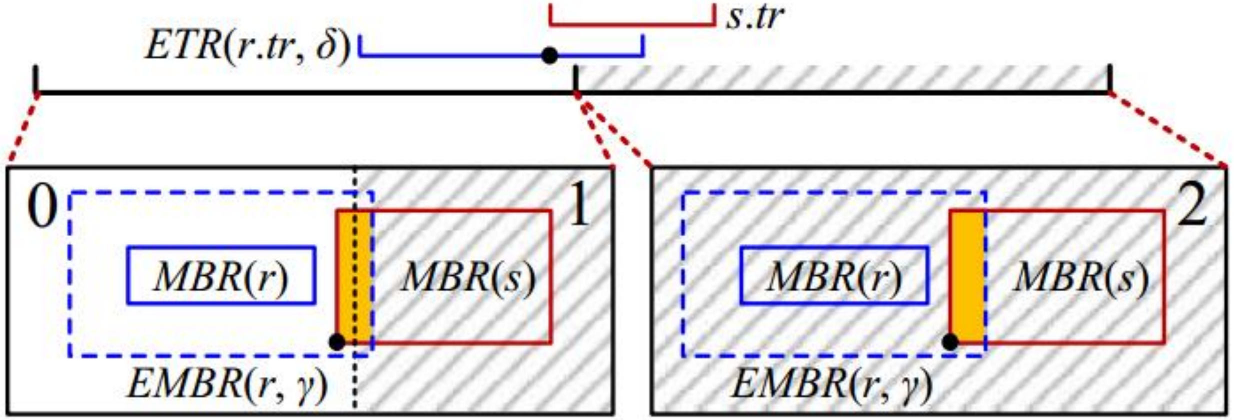
Step 4: Merge Result

- ❑ Goals
    - ❑ Combine multiple local results, and produce a global one
- ❑ A Straightforward Method
    - ❑ Shuffle Local Results by *r*
    - ❑ Combine Them into a Global Result using Multiway Merge Algorithm
    - ❑ Remove Duplicates
    - ❑ Take the First *k* Combinations

☹ Too Heavy Network Transmission!

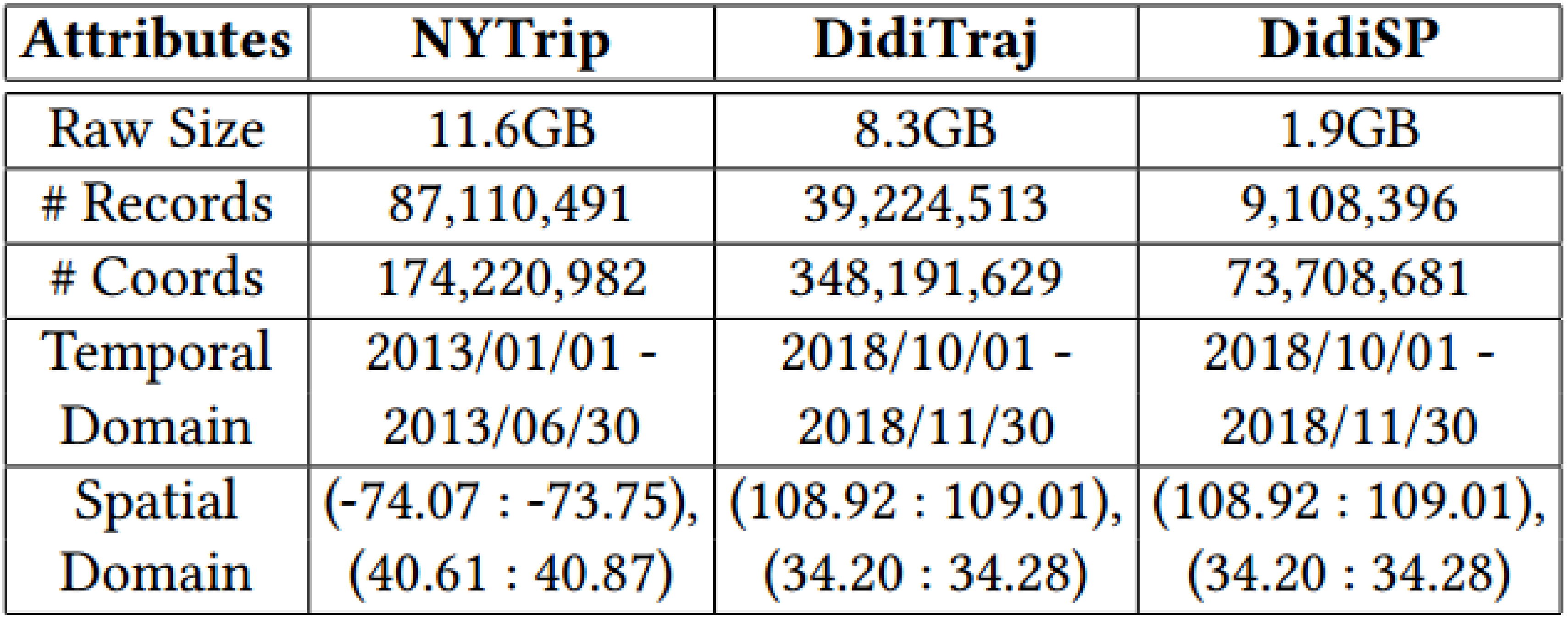
- ❑ Our Method
    - ❑ Remove Duplicates before Shuffling Local Results Based on *Spatio-Temporal Reference Points*


s_1^i s_2^i … s_k^i
…
s_1^j s_2^j … s_k^j
s_1 s_2 … s_k
(r, s_1), ..., (r, s_k)



Evaluation

- ❑ Datasets

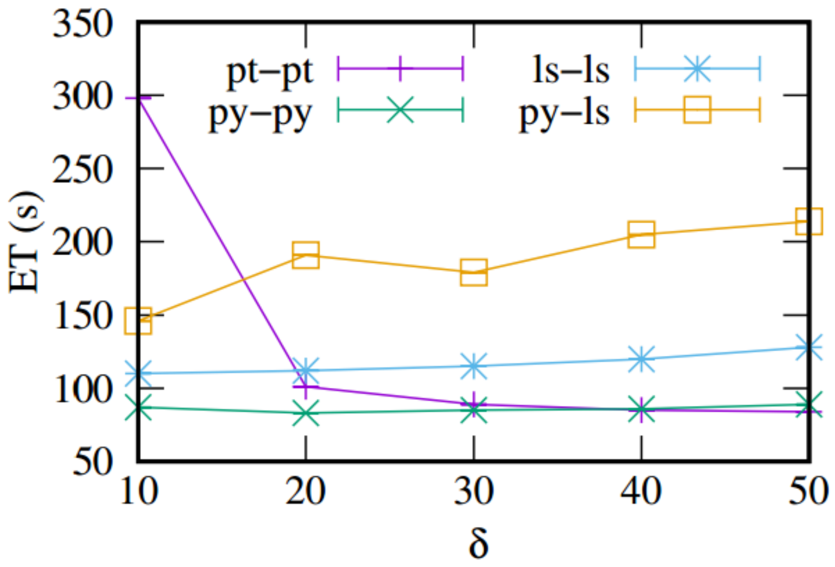


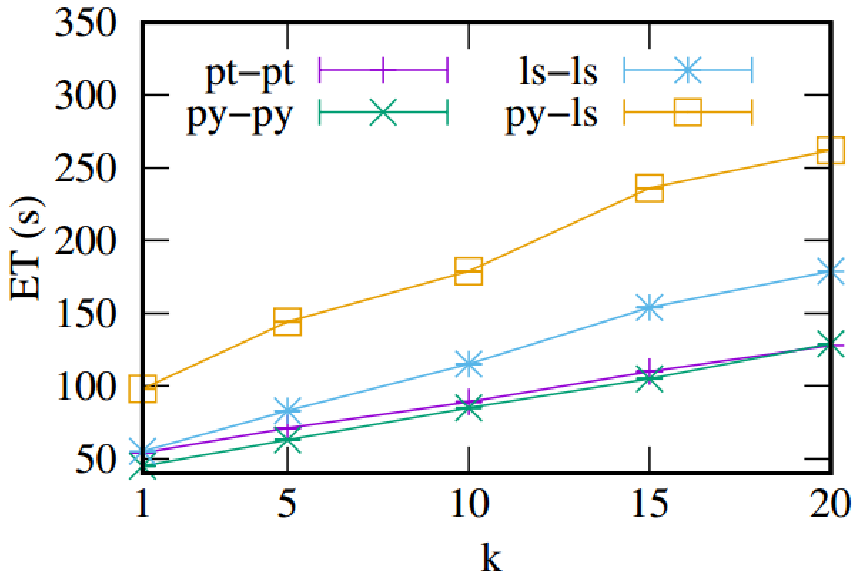
Attributes	NYTrip	DidiTraj	DidiSP
Raw Size	11.6GB	8.3GB	1.9GB
# Records	87,110,491	39,224,513	9,108,396
# Coords	174,220,982	348,191,629	73,708,681
Temporal Domain	2013/01/01 - 2013/06/30	2018/10/01 - 2018/11/30	2018/10/01 - 2018/11/30
Spatial Domain	(-74.07 : -73.75), (40.61 : 40.87)	(108.92 : 109.01), (34.20 : 34.28)	(108.92 : 109.01), (34.20 : 34.28)

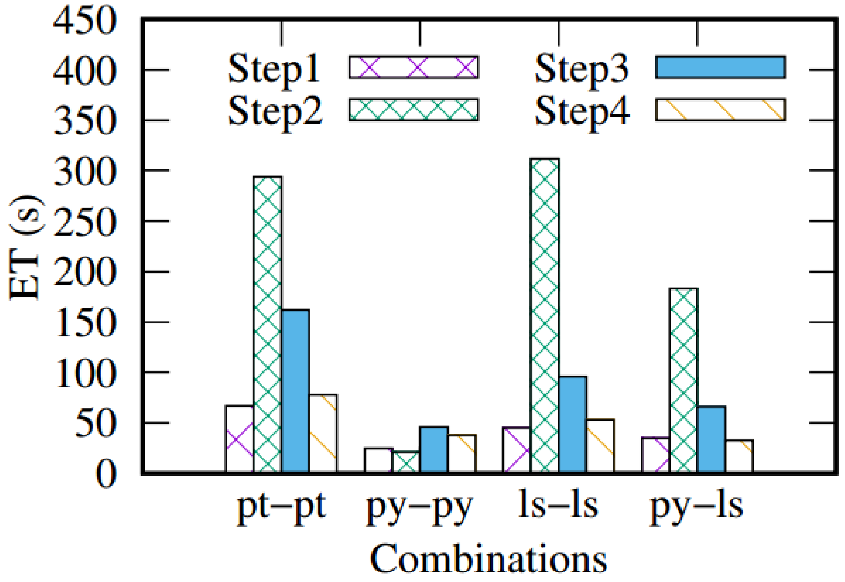
- ❑ Settings
  - ❑ 5 Nodes, 24-core CPU, 128GB RAM
  - ❑ Hadoop 2.7.6, Spark 2.3.3
  - ❑ 30 Executors, 5 Cores and 16 GB RAM

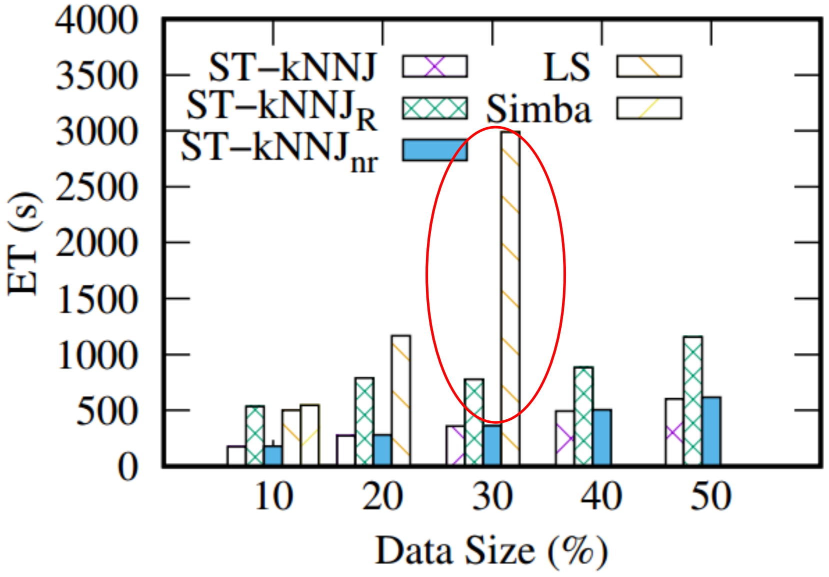
- ❑ Metrics
  - ❑ **Execution Time (ET)**
  - ❑ Copy Amplification (CA)
  - ❑ Hit Rate (HR)

Evaluation





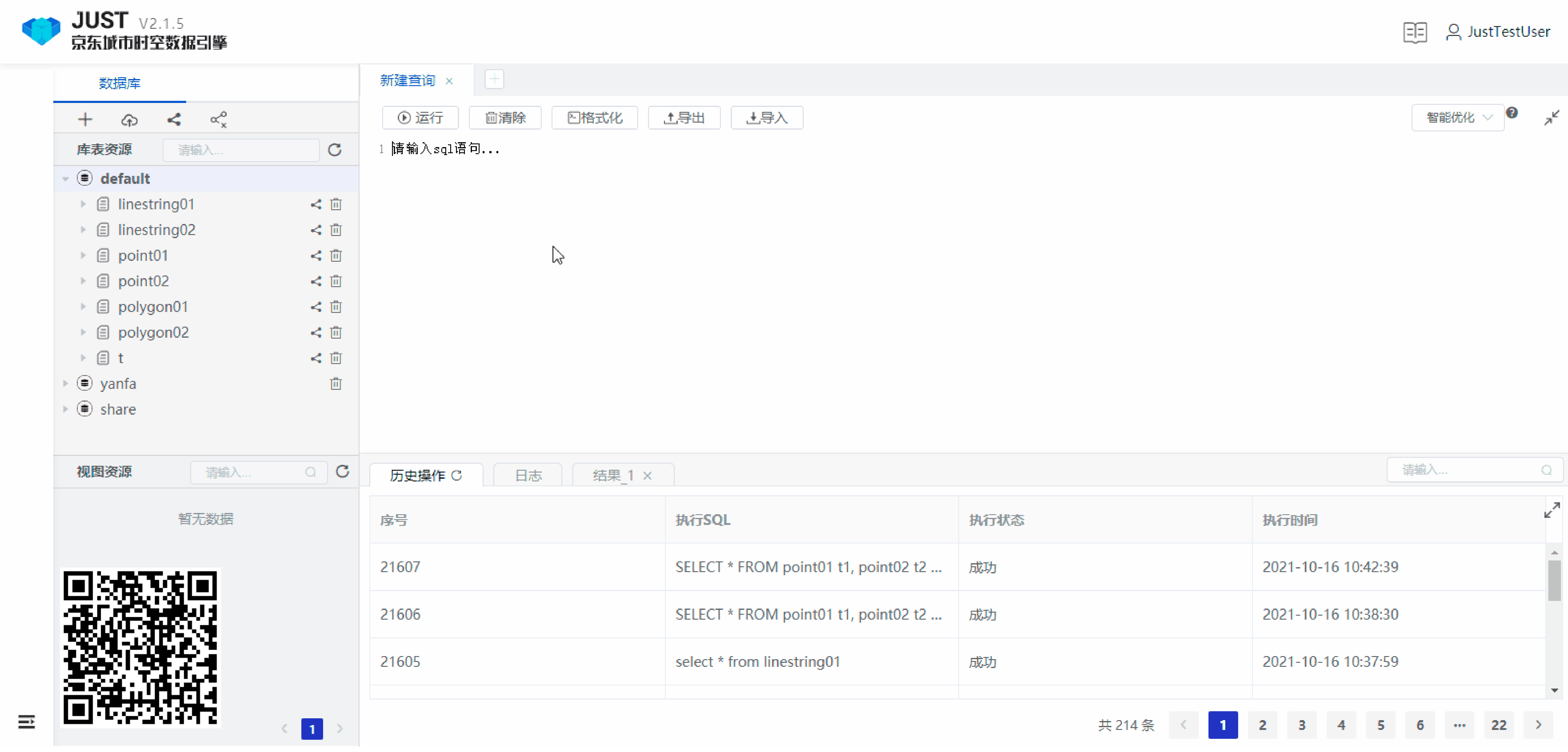




More Scalable
9X Faster

System Demonstration

http://stknnjoin.urban-computing.com/



1 请输入sql语句...

序号	执行SQL	执行状态	执行时间
21607	SELECT * FROM point01 t1, point02 t2 ...	成功	2021-10-16 10:42:39
21606	SELECT * FROM point01 t1, point02 t2 ...	成功	2021-10-16 10:38:30
21605	select * from linestring01	成功	2021-10-16 10:37:59

共 214 条 ‹ 1 2 3 4 5 6 ··· 22 ›

Conclusion

- ❑ Contribution
  - ❑ Propose a novel and useful ST-kNN Join problem
  - ❑ Propose a two-round join framework based on Spark
    - ❑ A new spatio-temporal partition method
    - ❑ A new lightweight and effective index structure TRC-index
    - ❑ Remove duplicates based on spatio-temporal reference points
  - ❑ Extensive experiments based on three real datasets shows the effectiveness
  - ❑ Deploy it to our product JUST, and public the source code
    - ❑ Source Code: https://github.com/1085904057/spatialjoin
- ❑ Future Works
  - ❑ Cache some intermediate results
  - ❑ Cost models to determine good system parameters, e.g., α, β, *binNum*

Distributed Spatio-Temporal *k* Nearest Neighbors Join

Thanks!

Ruiyuan Li, Chongqing University
liruiyuan@cqu.edu.cn

WeChat Official Account
Download the Slides by Inputting “ST-kNNJ_Slides”