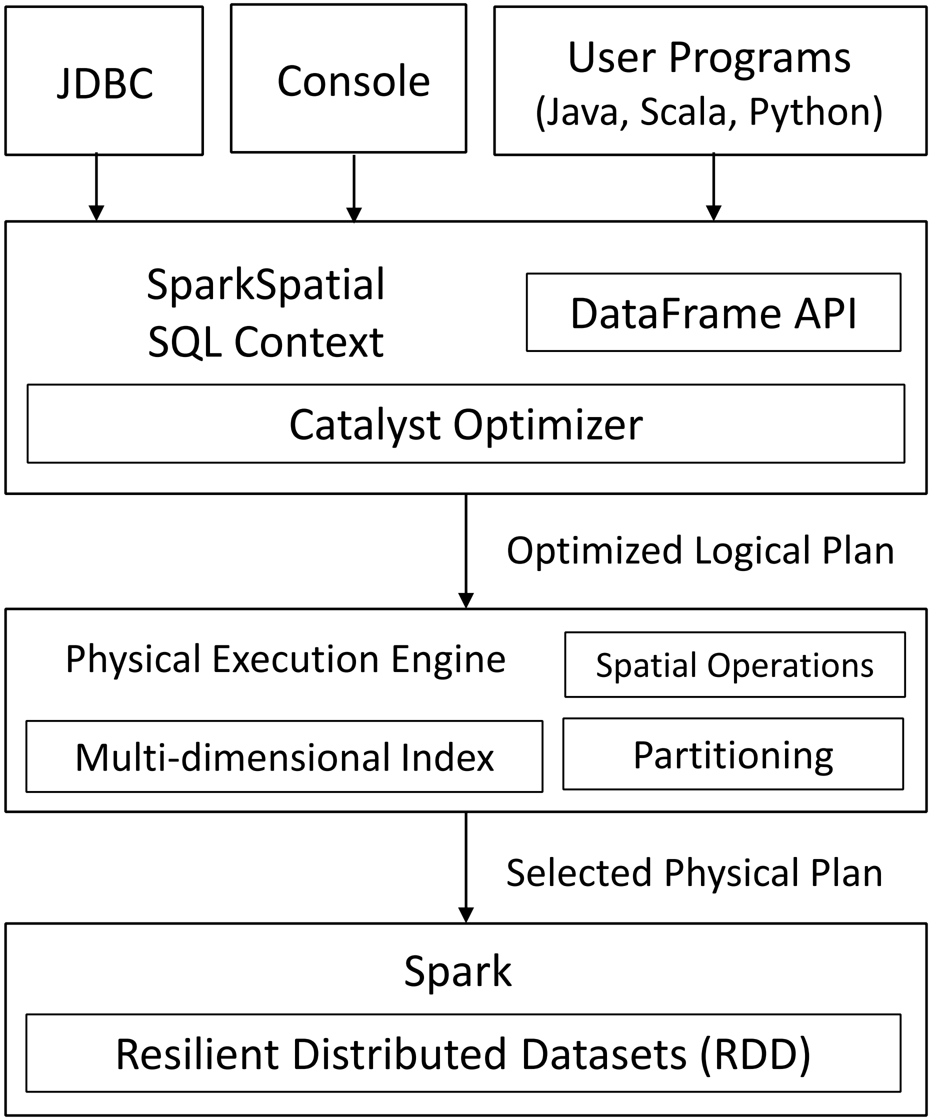
SparkSpatial: Distributed Spatial Data Processing System



- Distributed Spatial Data Processing System based on Spark
- Supporting compound queries on multi-dimensions (spatial, textual, temporal)
- Supporting fast spatial joins on large spatial data
- Adopting a user-friendly SQL-like query language as main interface
- DataFrame API
- Providing several kinds of multi-dimensional index and partitioning strategies
- More complex optimizers based on cost evaluation.

Programming Interface

- SQL-Like Query Language

```
SELECT r.foodtype, count(*) AS c
FROM queries AS q DISTANCE JOIN rests AS r
    ON POINT(r.x, r.y) IN CIRCLERANGE(POINT(q.x, q.y), 10.0)
GROUP BY r.foodtype
ORDER BY c
```

- DataFrame API

```
rest.knn(Point(rest(“x”), rest(“y”)), Point(3.0, 5.0), 10)
rest.distancejoin(query, Point(query(“x”), query(“y”)),
             Point(rest(“x”), rest(“y”)), 10.0)
    .groupBy(rest(“foodtype”))
    .select(rest(“foodtype”), count())
```

- Index Management:

```
CREATE INDEX pointIndex ON point1(x, y) USE rtree
```

Comparison

- Spark SQL:

```
SELECT id, x, y
FROM points
ORDER BY (x - 3) * (x - 3) + (y - 4) * (y - 4)
LIMIT 10
```

- SparkSpatial:

```
SELECT id, x, y
FROM points
WHERE POINT(x, y) in KNN(POINT(3, 4), 10)
```

- Expressing Ability: KNN Join

```
SELECT *
FROM point1 KNN JOIN point2
  ON POINT(point2.x, point2.y) IN KNN(POINT(point1.x, point1.y), 10)
```

Indexing Strategy

- Motivation:

  Prune useless scanning, save time & resources.

- Challenge:

  RDD → Distributed Set → Poor in Random Access

- Solution:

  Storage Format Changing & Local + Global Indexing

- Three phases:
  - Data Partitioning
  - Local Indexing
  - Global Indexing

Indexing Strategy (Cont.d)

- Data Partitioning
- Three Main Concerns:
  - Partition Size Fitness: Memory Overflow
  - Data Locality: Query Acceleration.
  - Load Balancing
- Abstract Class: ‘Partitioner’

  number of partitions + mapping from key to partition
- Solution:

  One Dimensional: Equal Depth Range Partitioner

  Multi-Dimensional: STRPartitioner

Indexing Strategy (Cont.d)

- Local Indexing
- Partition Packing:

  Pack all elements of a partition into an array.

  → explicit index & faster random access within partition.
- Local Index Building:

  Build requested index structure for each packed partition

  Co-locate packed partition with its local index.
- As a result:

  Storage format for tables changed

  RDD[Row] => RDD[PackedPartitionWithIndex]

  A Partition => an element with original data & local index

Indexing Strategy (Cont.d)

- Global Indexing

  Index for pruning ***partitions*** in query processing.

- Required Info:
  - Partition Boundaries (Data Partitioning)
  - Partition Statistics (Local Indexing)
- Index Structure choosing:
  - One-dimensional data: sorted range bounds.
  - Multi-dimensional data: R-Tree over partition MBRs.

Spatial Operations

- Developer API: PartitionPruningRDD

  Skip tasks on specified partition.

- Range Query & Circle Range Query
- Step 1: Query for all partitions that intersect query area.
- Step 2: Invoke range query on remaining partitions.

Spatial Operations (Cont.d)

- KNN Query
- Safe pruning bound: Top k maximum distance.
  - Maximum distance as Distance Function.
  - Step 1: KNN Query on all partition boundaries.
  - Step 2: KNN Query on remaining partition.
  - Take k-th distance as pruning bound.
- Only step 1 is safe, but loose.
- Step 2: Much stronger pruning power:
  - 27 partitions remaining → 4 partitions remaining.

Spatial Operations (Cont.d)

- Distance Join:

  R join S on distance between point in R and S less than r.

- Theta-Join => Cartesian Product + Filter (SLOW!!)

- Native Implementation:
  - Nested-Loop Distance Join (with/without R-Tree)
  - SJMR Distance Join
  - R-Tree Distance Join

Spatial Operations (Cont.d)

- KNN Join

  R join S on point in S is k-nearest neighbor of point in R over data set S.

- In Spark SQL: unavailable.
- Implemented Solutions:
  - Cartesian KNN Join
  - NestedLoop KNN Join (with/without R-Tree)
  - Voronoi KNN Join
  - R-Tree KNN Join

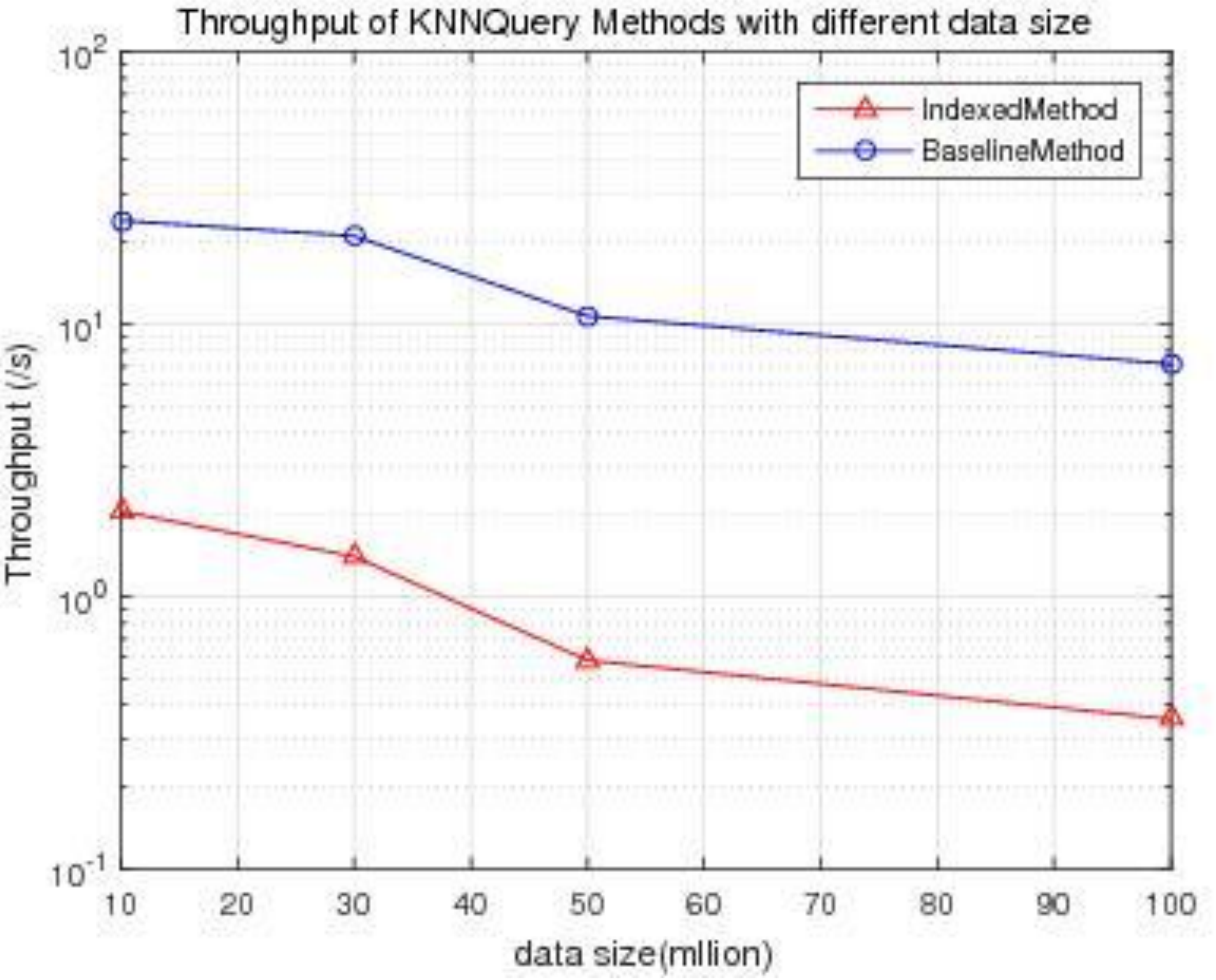
Optimization

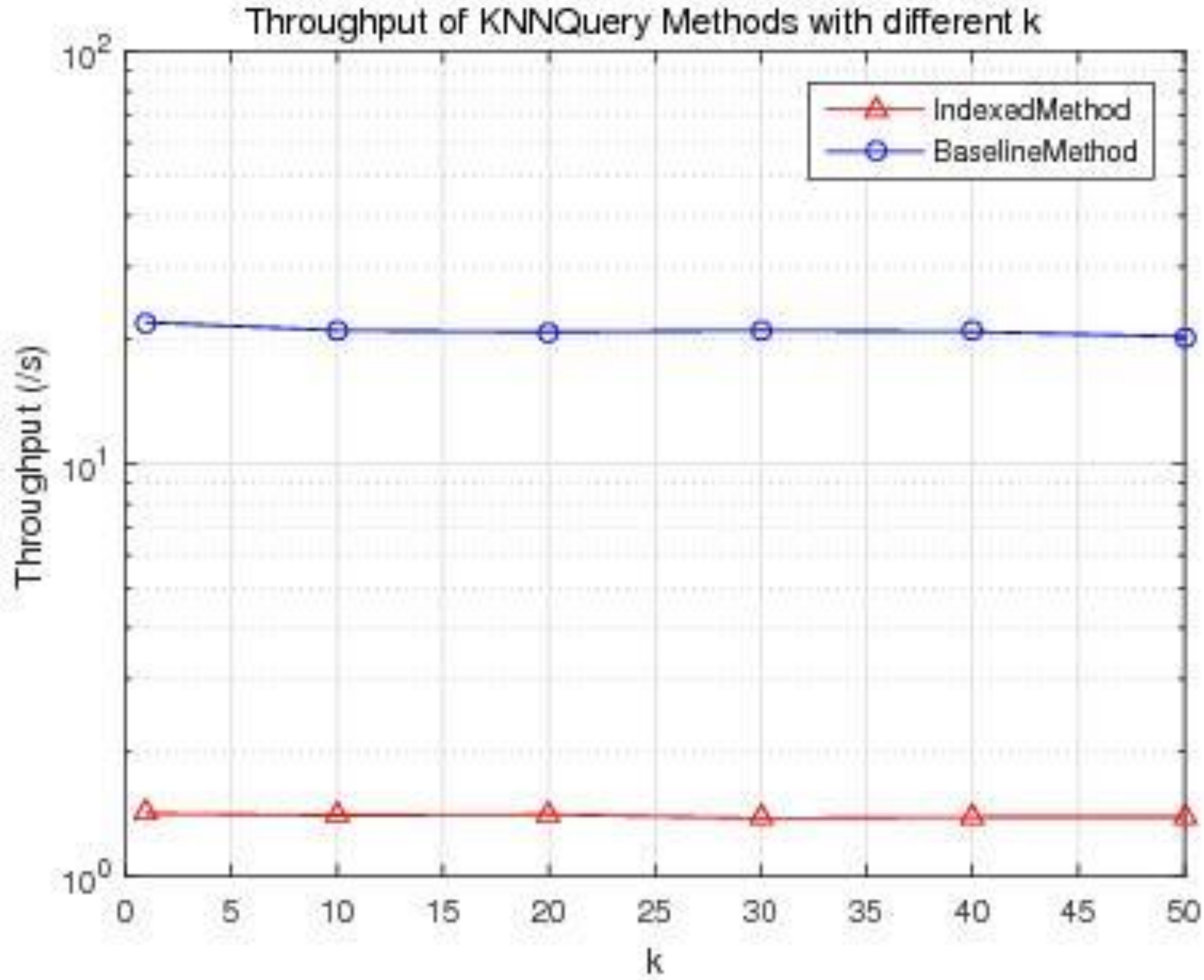
- Automatic optimization according to built indexes.
- Formating Query Predicates
  - CNF (Conjunctive Normal Form)
  - DNF (Disconjuctive Normal Form)
- Predicate Combination:

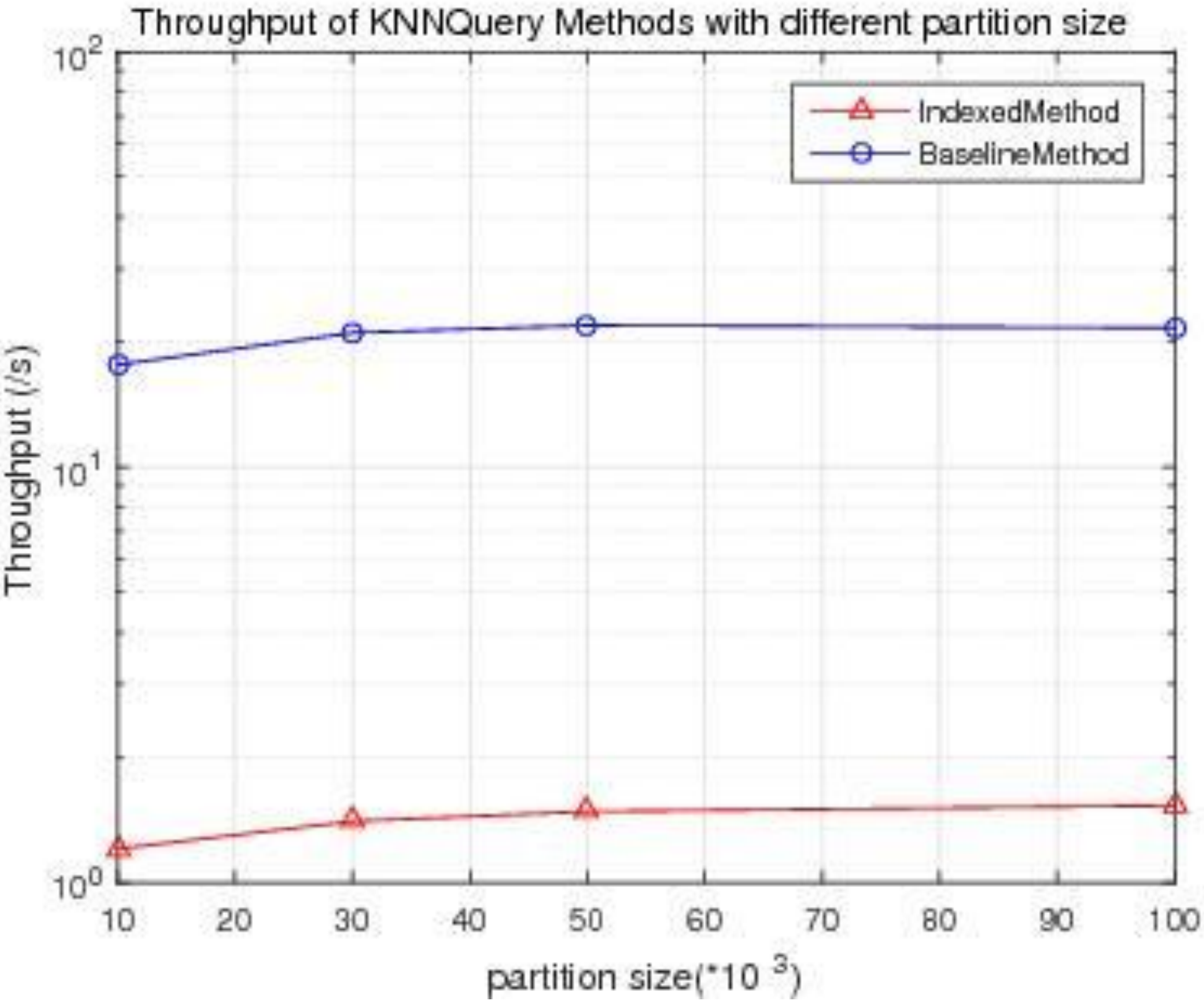
  e.g. x > 4 && x < 6 && y > 3 && y < 7

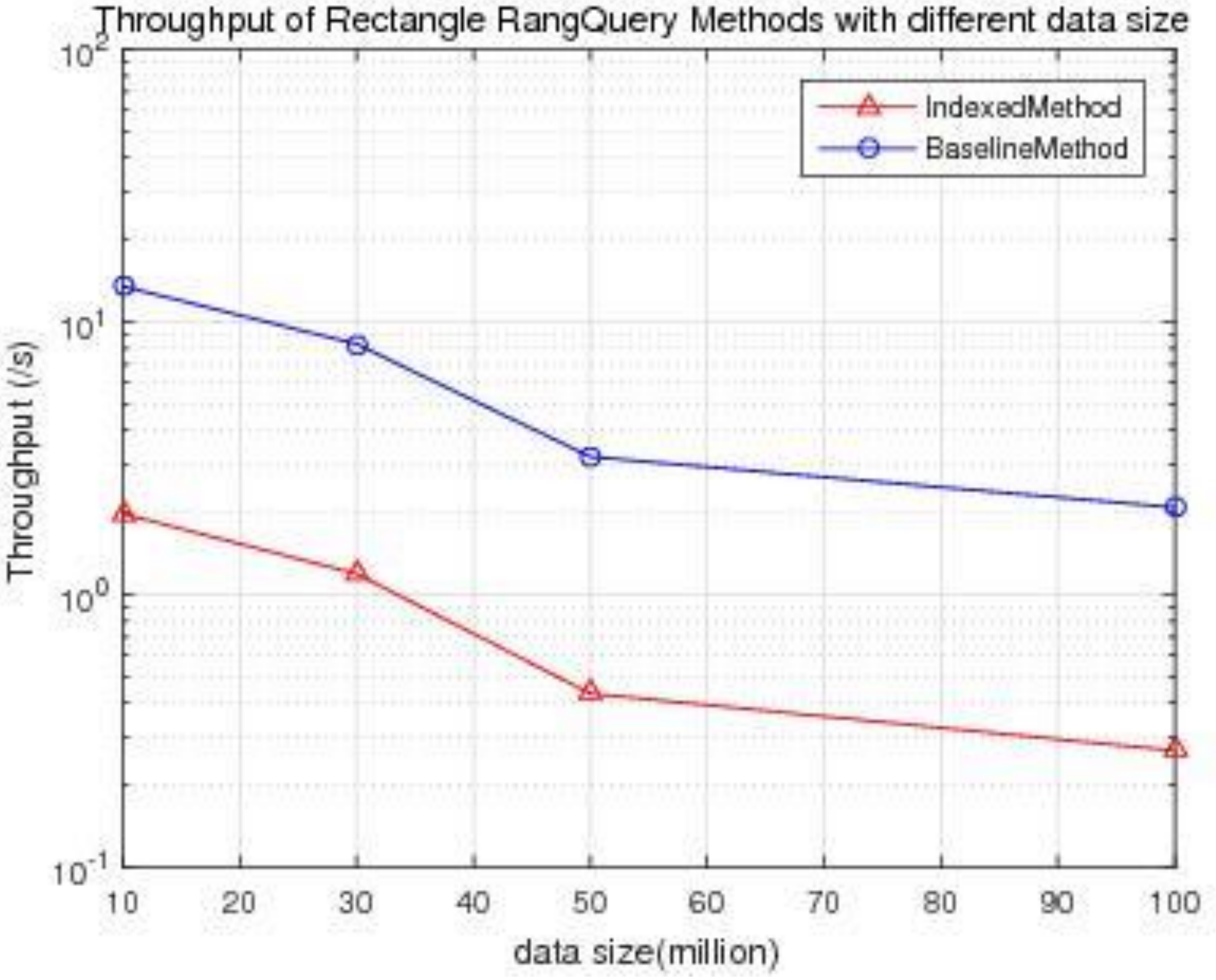
  → InRange(Point(4, 3), Point(6, 7))
- Alternative Execution Path for Indexed Relation:

  IndexedRelationScan


Throughput of KNNQuery Methods with different data size
IndexedMethod
BaselineMethod
Throughput (/s)
10^2
10^1
10^0
10^{-1}
10
20
30
40
50
60
70
80
90
100
data size(mllion)

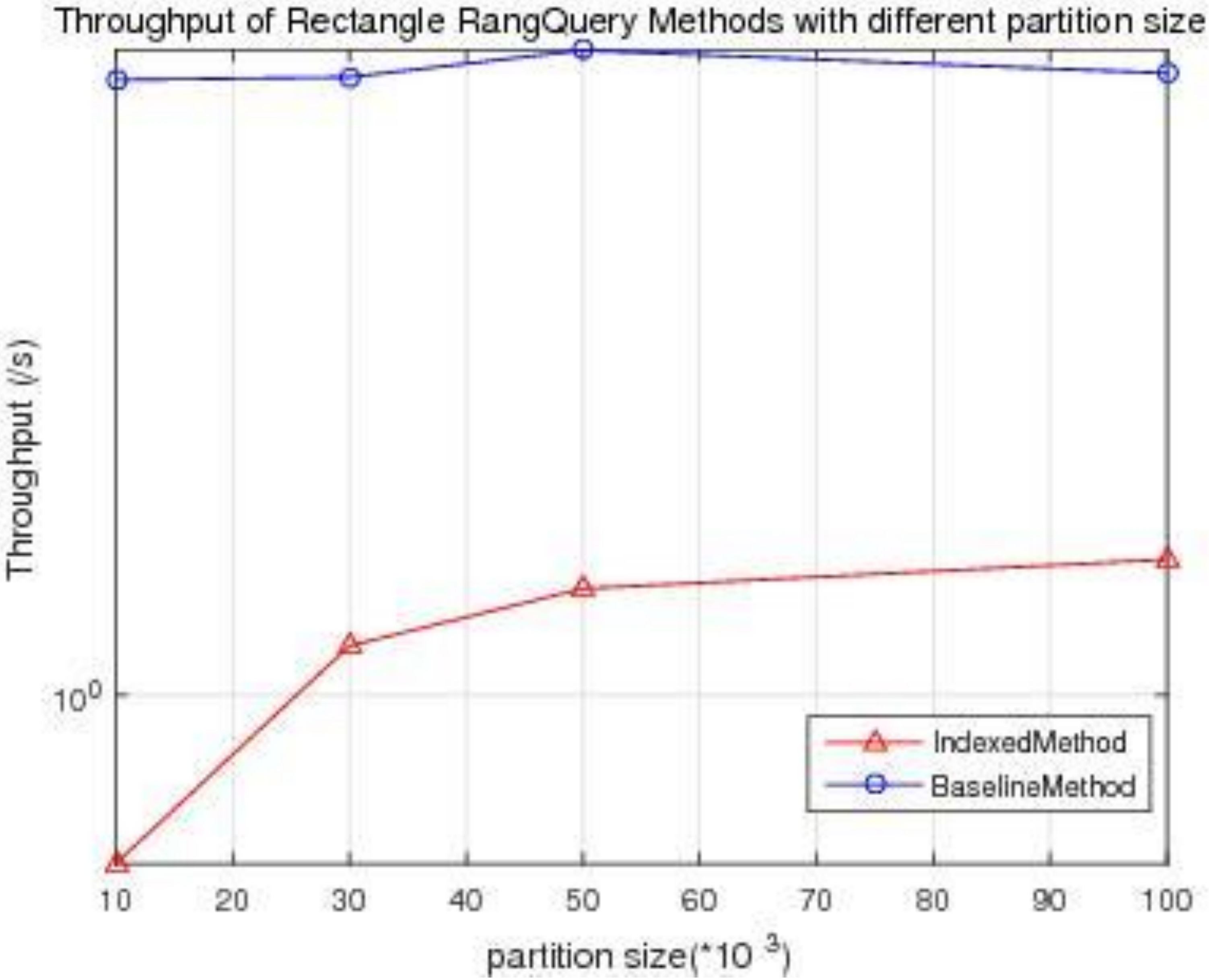

Throughput of KNNQuery Methods with different k
IndexedMethod
BaselineMethod
Throughput (/s)
10^2
10^1
10^0
0
5
10
15
20
25
30
35
40
45
50
k

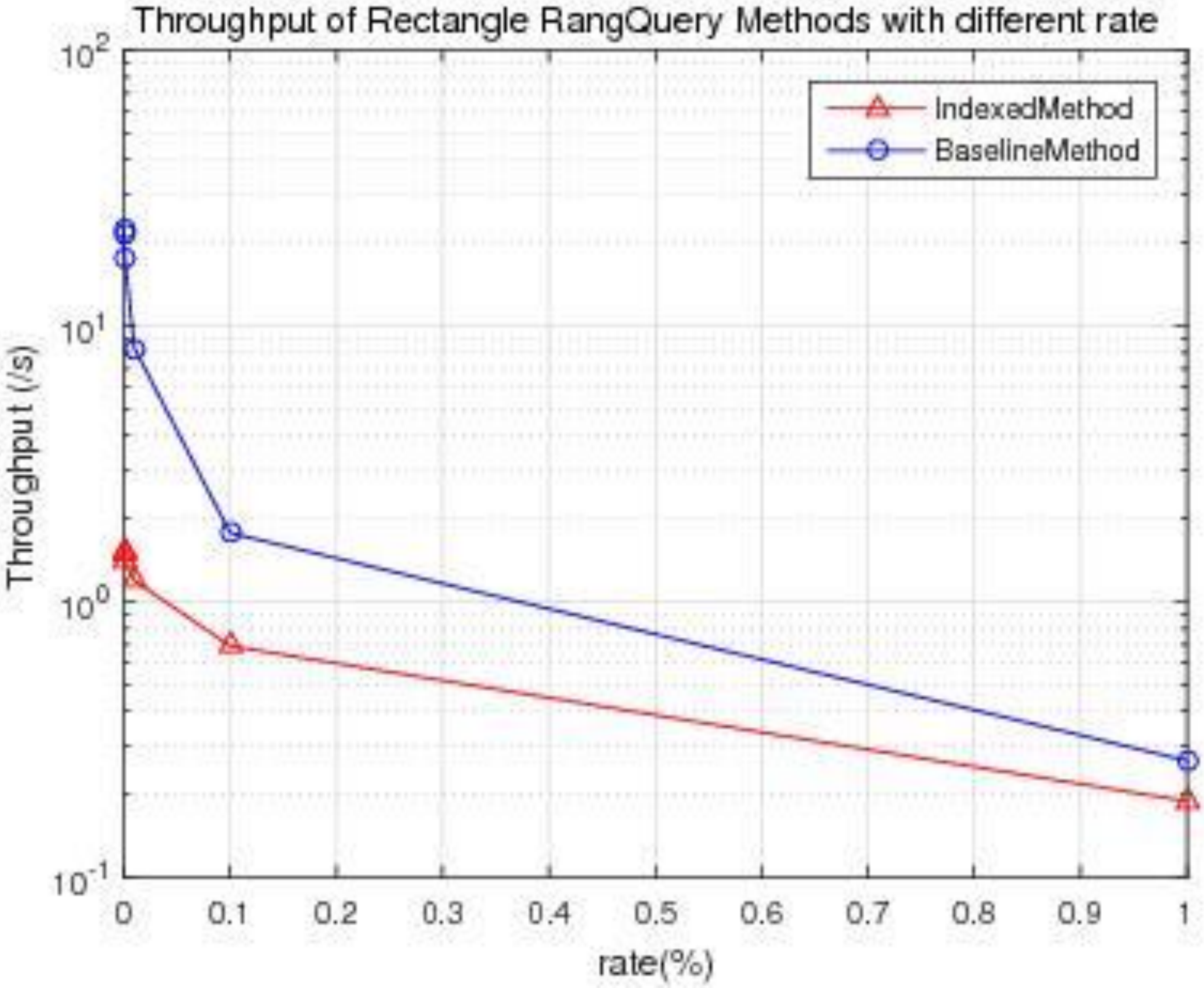

Throughput of KNNQuery Methods with different partition size
IndexedMethod
BaselineMethod
Throughput (/s)
10^2
10^1
10^0
10
20
30
40
50
60
70
80
90
100
partition size(*10 3)

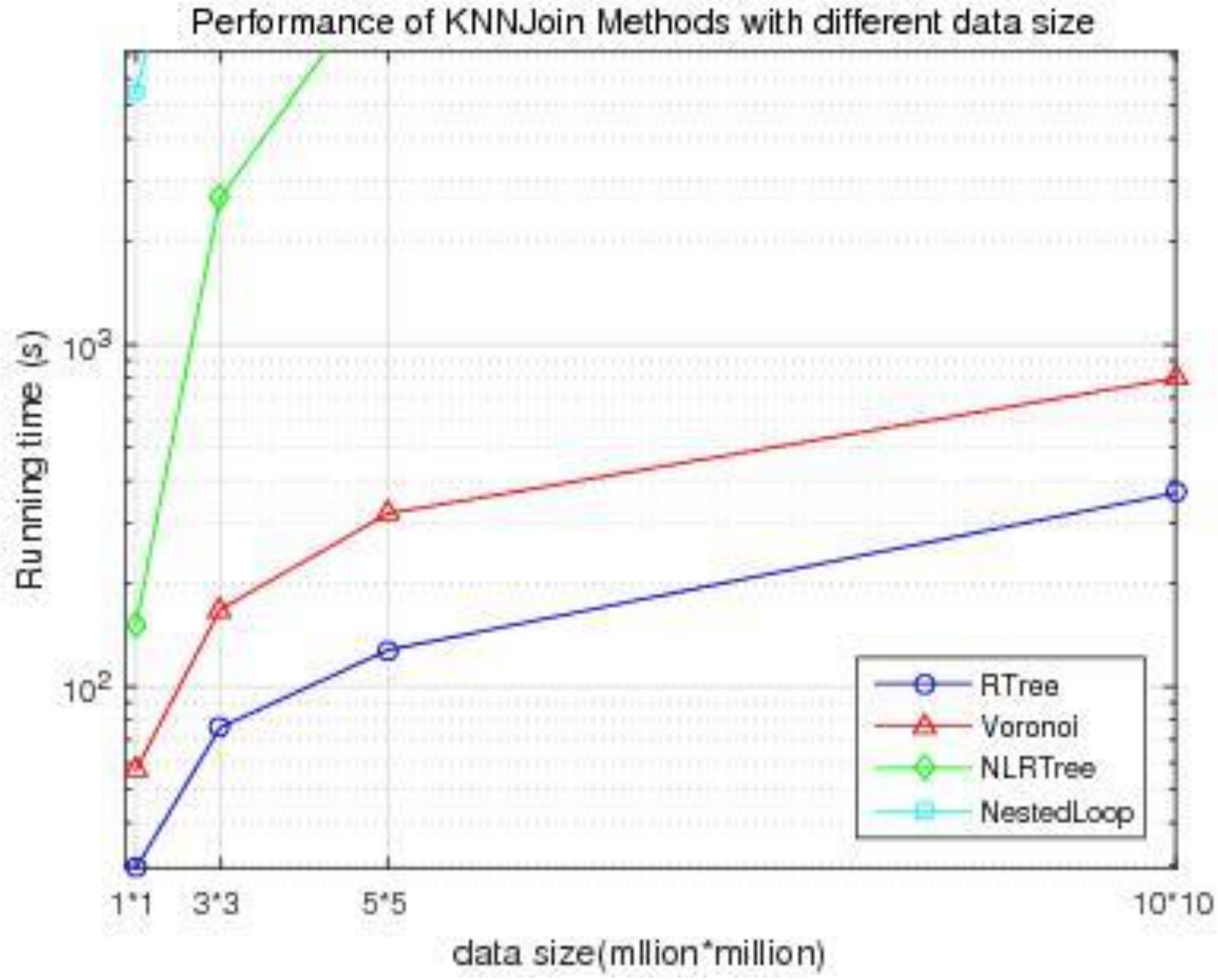


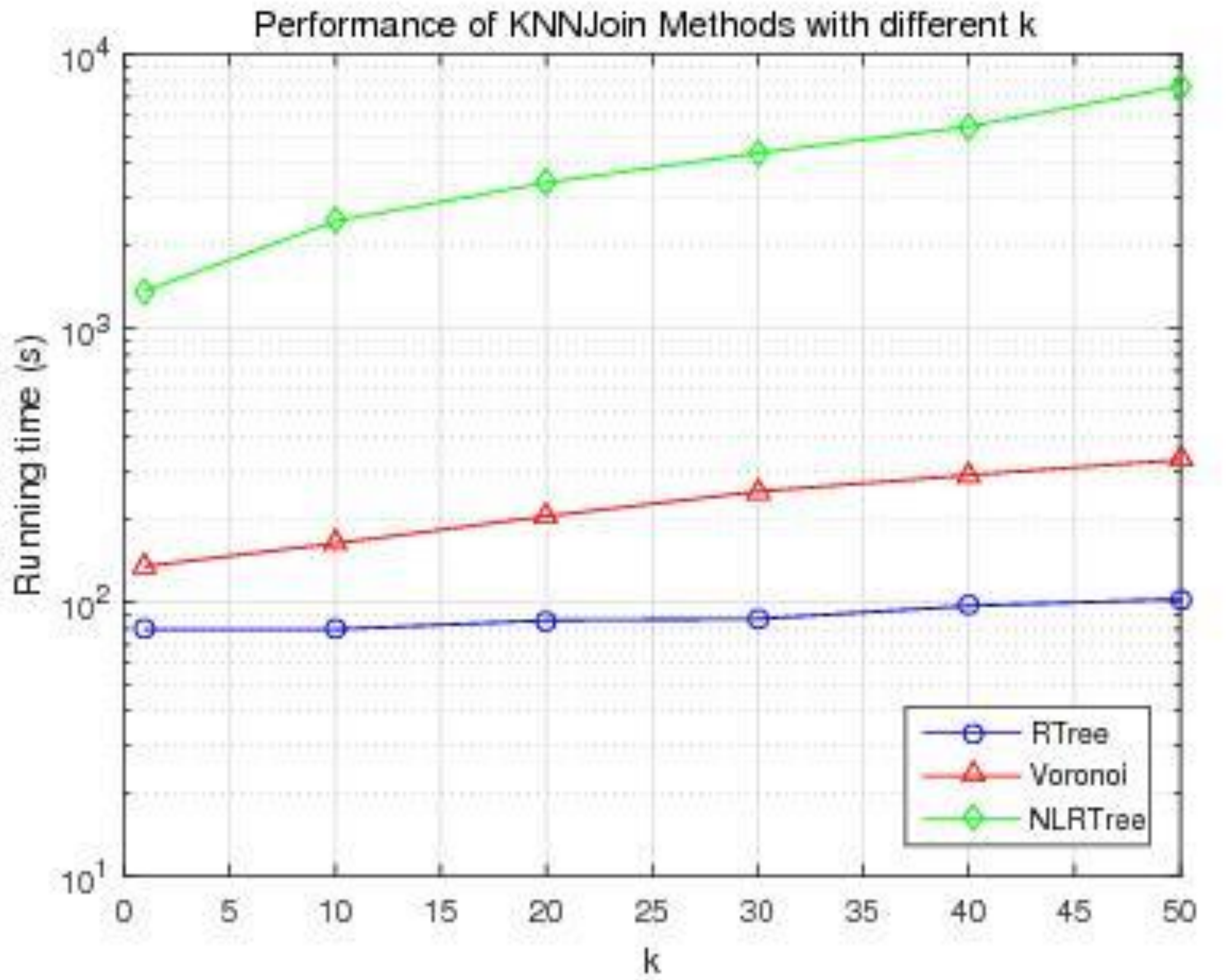
Throughput of Rectangle RangQuery Methods with different data size
IndexedMethod
BaselineMethod
Throughput (/s)
10^2
10^1
10^0
10^{-1}
10
20
30
40
50
60
70
80
90
100
data size(million)

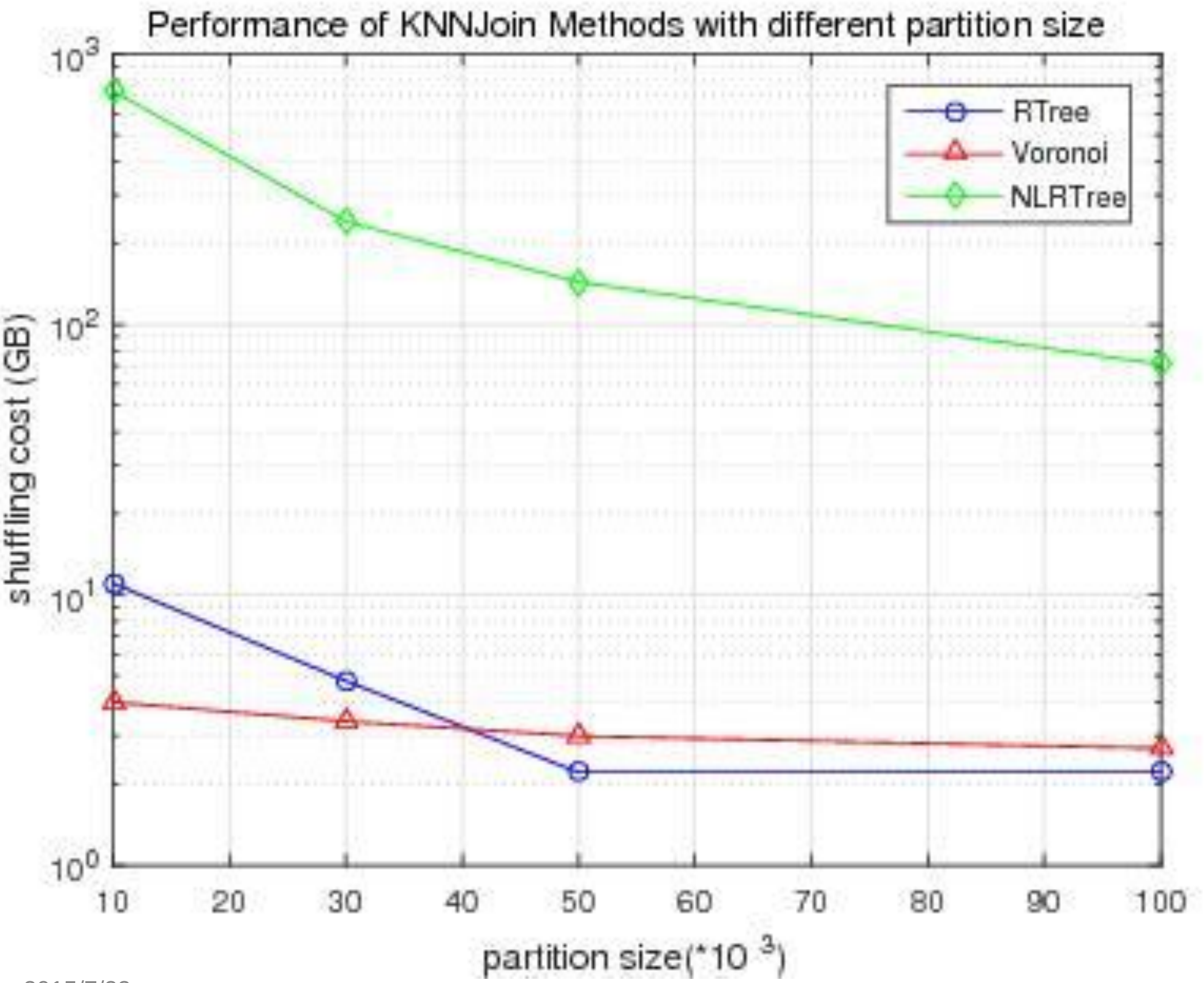
2015/7/22

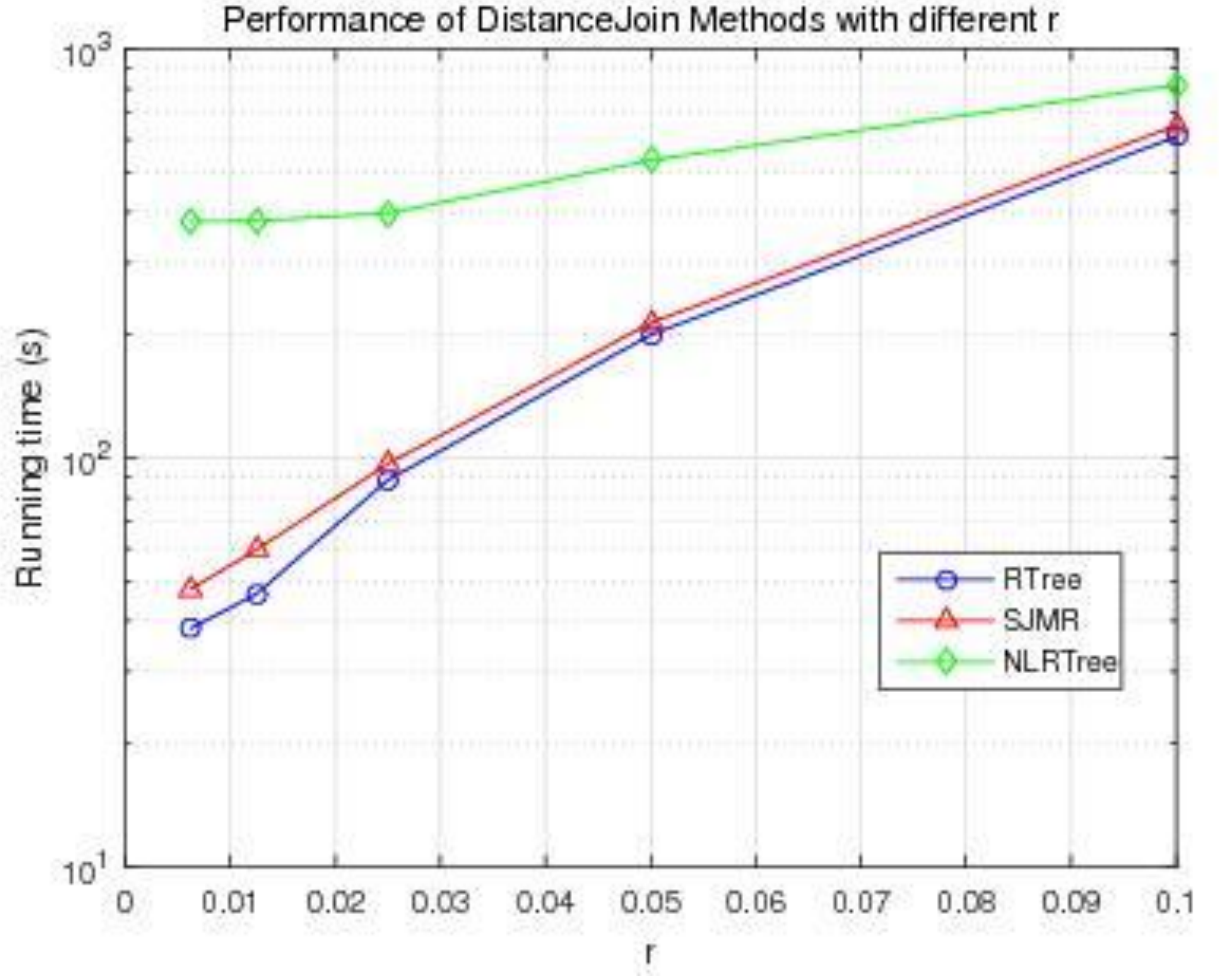

Throughput of Rectangle RangQuery Methods with different partition size
Throughput (/s)
10^0
10
20
30
40
50
60
70
80
90
100
partition size(*10 ^{3})
IndexedMethod
BaselineMethod


Throughput of Rectangle RangQuery Methods with different rate
IndexedMethod
BaselineMethod
Throughput (/s)
10^2
10^1
10^0
10^{-1}
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
rate(%)


Performance of KNNJoin Methods with different data size
Running time (s)
10^3
10^2
1*1
3*3
5*5
10*10
data size(mllion*million)
RTree
Voronoi
NLRTree
NestedLoop


Performance of KNNJoin Methods with different k
Running time (s)
10^4
10^3
10^2
10^1
0
5
10
15
20
25
30
35
40
45
50
k
RTree
Voronoi
NLRTree


Performance of KNNJoin Methods with different partition size
shuffling cost (GB)
10^3
10^2
10^1
10^0
10
20
30
40
50
60
70
80
90
100
partition size(*10 ^3)
RTree
Voronoi
NLRTree


Performance of DistanceJoin Methods with different r
Running time (s)
10^3
10^2
10^1
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
r
RTree
SJMR
NLRTree