

Towards Parallel Detection of Moving Flock Patterns in Large Spatiotemporal Datasets

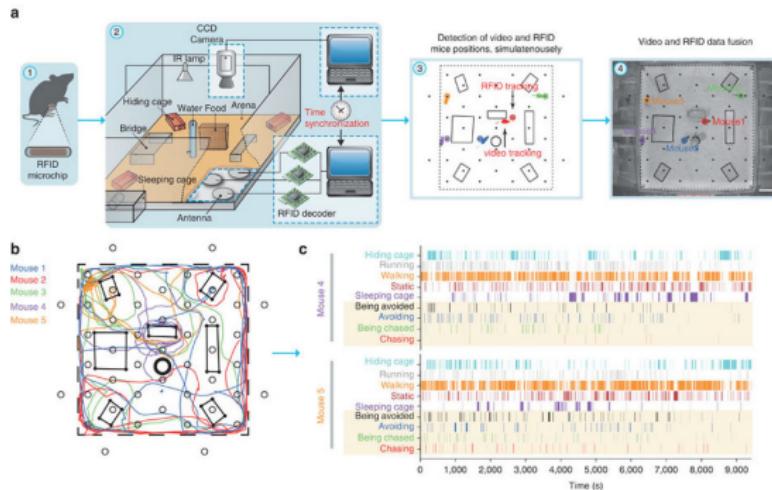
Final Report

Andres Calderon

December 4, 2016

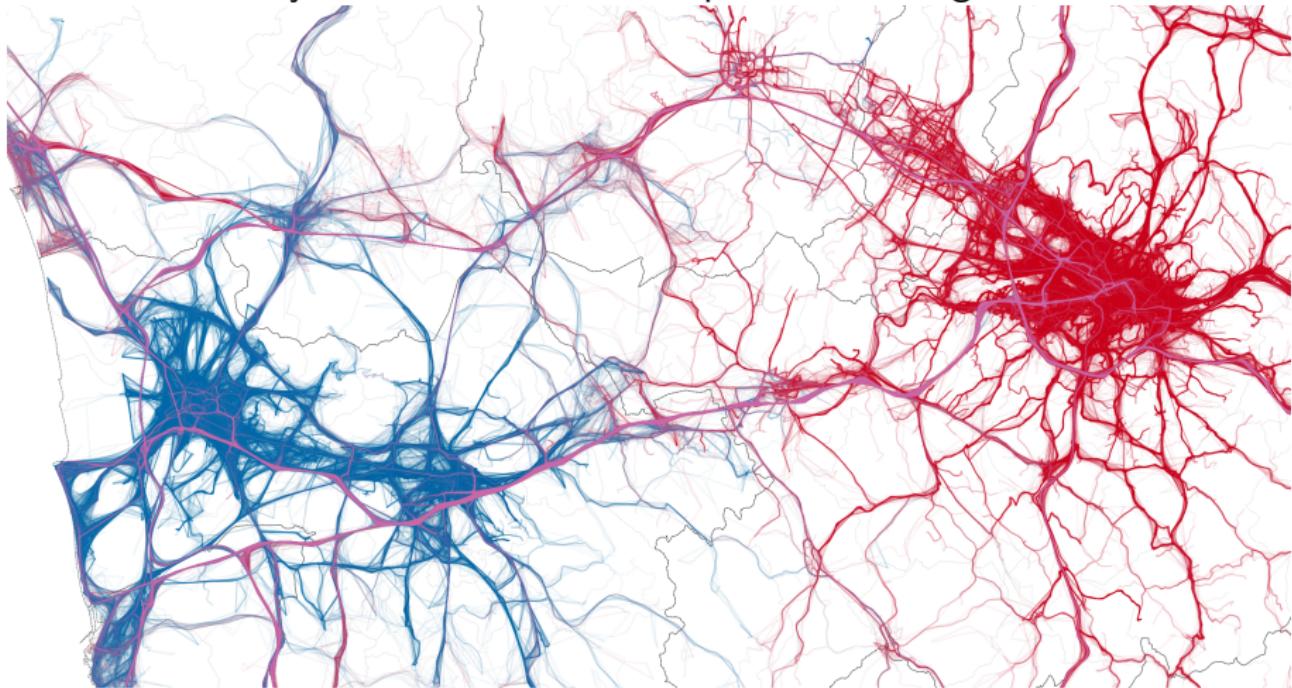
Trajectory datasets

- ▶ Anything that could move (or could be moved), will be tracked...
 - ▶ Smart phones, GPS, RFID, WiFi, Bluetooth, ZigBee, IoT, Satellites, Drones, Surveillance cameras...



Applications

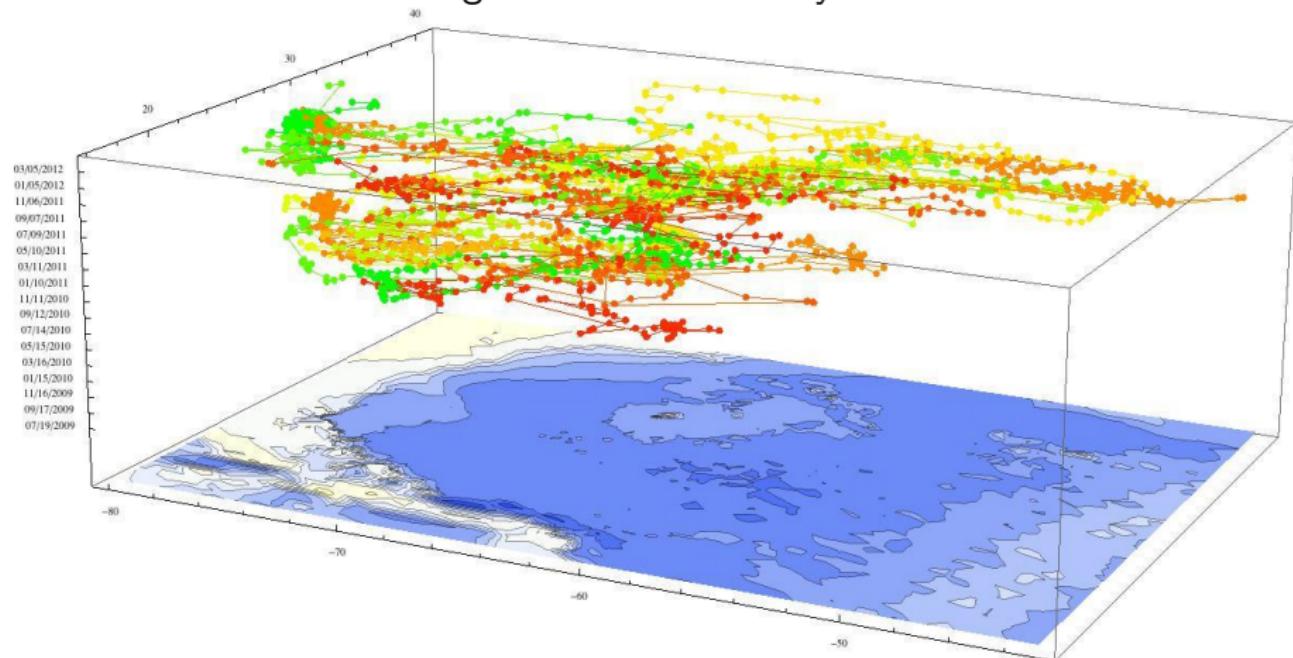
Are you a Returner or an Explorer? Ask Big Data.



<http://tinyurl.com/h14e9u8>

Applications

Tiger Shark Data Analysis.



<http://tinyurl.com/gtqjh4o>

Outline

Moving flock patterns

Implementation

Experiments

Conclusions and Future work

What is a flock???

Definition $((\mu, \epsilon, \delta) - \text{flock})$

Sets of at least μ objects moving close enough (ϵ) for at least δ time intervals (Benkert et al, 2008).



Motivation

Why is important of focus on flocks and finding disks???

- ▶ Why are moving flock patterns important?
 - ▶ They capture the collective behavior of trajectories as groups.
- ▶ Why is the finding of disks important?
 - ▶ It is the base of the algorithm.
 - ▶ It has a high complexity ($\mathcal{O}(2n^2)$).
 - ▶ It is no trivial, disks can be at any location.

Outline

Moving flock patterns

Implementation

Experiments

Conclusions and Future work

BFE implementation

- ▶ On-Line Discovery of Flock Patterns in Spatio-Temporal Data (Vieira et al, SIGSPATIAL'09).
- ▶ Implementation available at
<https://github.com/poldrosky/FPFlock>.

PBFE implementation

- ▶ Written in Scala using Simba 1.6.0.
 - ▶ DISTANCE JOIN and CIRCLE RANGE operators were the key operators!!!
 - ▶ Tested by visual inspection and counting number of found disks.
- ▶ Demo (in R language):
 - ▶ <http://tinyurl.com/jl55849>.
 - ▶ Password is 'nancy' (behave yourself).
- ▶ Code available at
[https://github.com/aocalderon/PhD/tree/master/Y2Q1/SDB/Project/Code/
Scripts/pbfe2](https://github.com/aocalderon/PhD/tree/master/Y2Q1/SDB/Project/Code/Scripts/pbfe2).

Bug report

DISTANCE JOIN does not work... #71

Closed aocalderon opened this issue 8 days ago · 4 comments

aocalderon commented 8 days ago

Hello there...

First of all, congratulations for your great job. I have been working on Simba for a while and I found it is such a great project. Recently, I am working in some queries focus on DISTANCE JOINS. However I have found some issues.

For example, I have this code ([Distance.Join.tar.gz](#) and [P10K.csv](#)):

```

from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext
from pyspark.sql import Row

conf = (SparkConf()\
        .setMaster("local")\
        .setAppName("My app")\
        .set("spark.executor.memory", "1g"))
sc = SparkContext(conf = conf)
sqlContext = SQLContext(sc)

epsilon = 100
points = sc.textFile("P10K.csv")\
    .map(lambda line: line.split(","))\
    .map(lambda p: Row(id=int(p[0]), lat=float(p[1]), lng=float(p[2])))\
    .toDF()

npoints = points.count()
npoints

points.registerTempTable("p1")
points.registerTempTable("p2")
p1 = sqlContext.sql("""
    CREATE INDEX

```

Projects
None yet

Labels
None yet

Milestone
No milestone

Assignees
No one assigned

3 participants

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

<https://github.com/InitialDLab/Simba/issues/71>



Bug report

- ▶ A SQLParser error in the DISTANCE JOIN examples.
- ▶ A requirement failure when JOIN's were combined with filters and inequations.
- ▶ Caused by the Simba Optimizer:
 - ▶ DistanceJoin node does not support inequations inside (another Filter node is needed).
- ▶ Quickly fixed at RP #72.

Bug report

- Both were been throwing a 'requirement failure' error:

```
# Code for Spark-SQL (Python)
sql = """
    SELECT *
    FROM p1
    DISTANCE JOIN p2
    ON POINT(p2.lng, p2.lat) IN CIRCLE RANGE(POINT(p1.lng, p1.lat), {0})
    WHERE p1.id < p2.id""".format(epsilon)
pairs = sqlContext.sql(sql)
```

```
// Code for Spark DataFrame API (Scala)
val pairs = p1.distanceJoin(p2, Point(p1("x"), p1("y")), Point(p2("x2"), p2("y2")), epsilon)
  .filter("id < id2")
```

Bug report

- ▶ Found a workaround using the DataFrame API:

```
// Code for Spark DataFrame API (Scala)
val pairs = p1.distanceJoin(p2, Point(p1("x"), p1("y")), Point(p2("x2"), p2("y2")), epsilon)
val disks = pairs.rdd.filter( (x:Row) => x.getInt(0) > x.getInt(3) )
```

Outline

Moving flock patterns

Implementation

Experiments

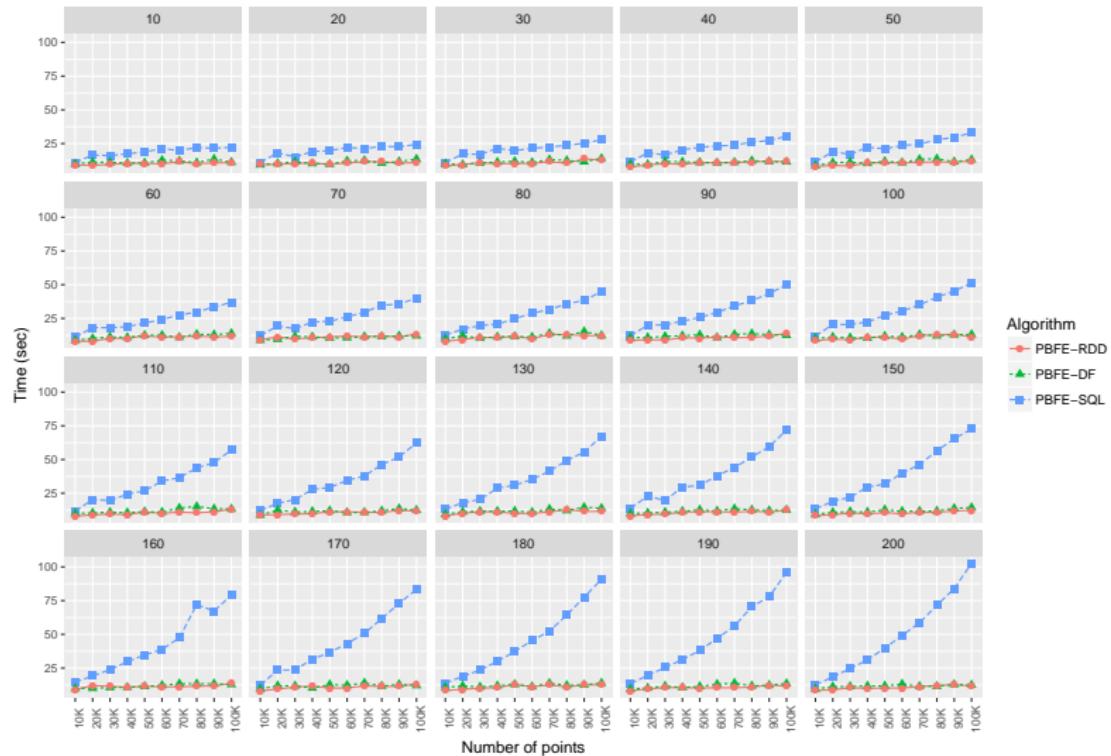
Conclusions and Future work

RDD vs DF vs SQL filters

- ▶ Which have the best performance???
 - ▶ Spark SQL WHERE clause.
 - ▶ Spark DataFrame filter function.
 - ▶ Spark RDD filter function.

Benchmarking on Beijing dataset

Execution time by ϵ (radius of disk in mts) in Beijing dataset.



Benchmarking on Beijing dataset

Execution time by ε (radius of disk in mts) in Beijing dataset.

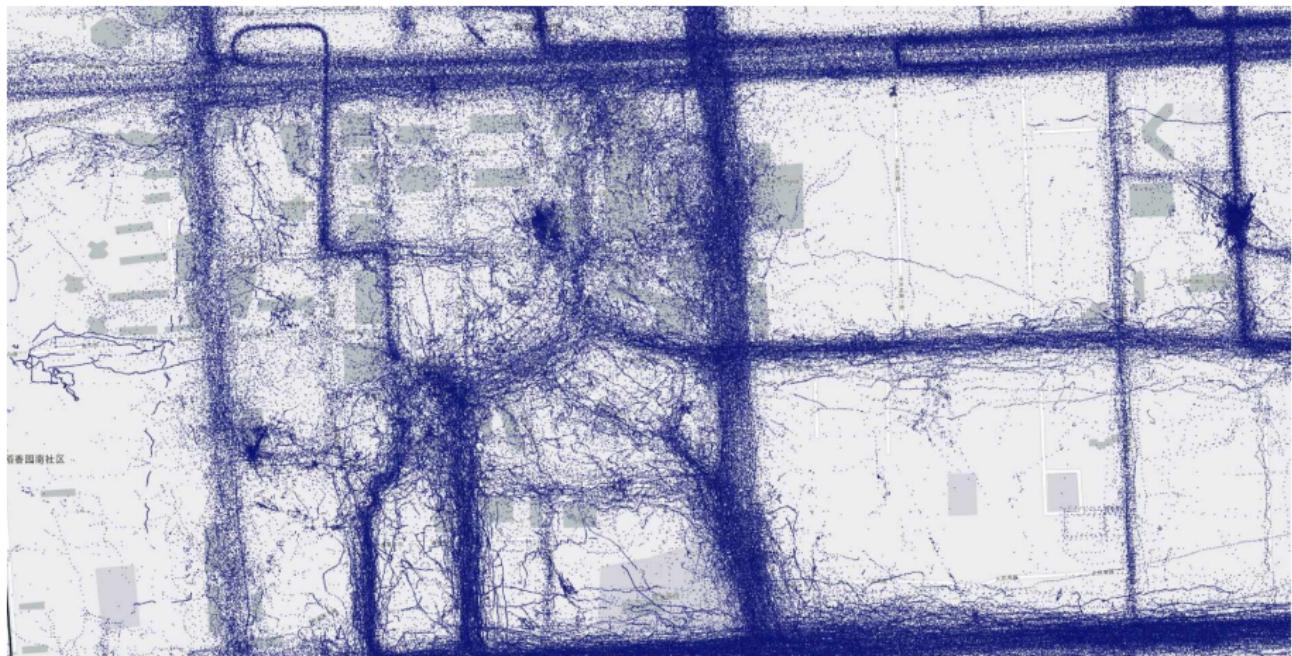


Dataset

- ▶ **Beijing** from Geolife project¹.
 - ▶ 182 users in a period of over three years (from April 2007 to August 2012).
 - ▶ 17,621 trajectories.
 - ▶ ≈18 million points (no duplicates).

¹<http://tinyurl.com/j7t2cao>

Beijing dataset

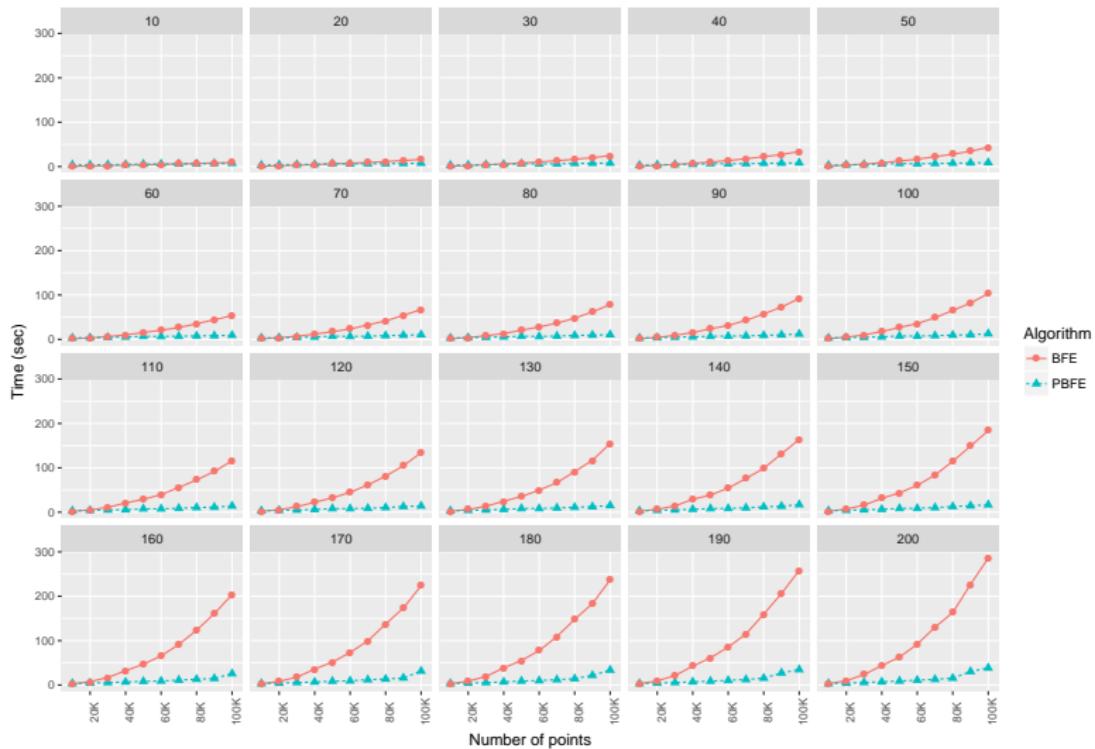


Setup

- ▶ Single-node.
- ▶ Processor: 4-core Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz
- ▶ RAM: 8 GB.
- ▶ Ubuntu 16.04 LTS, Simba/Spark 1.6.0.

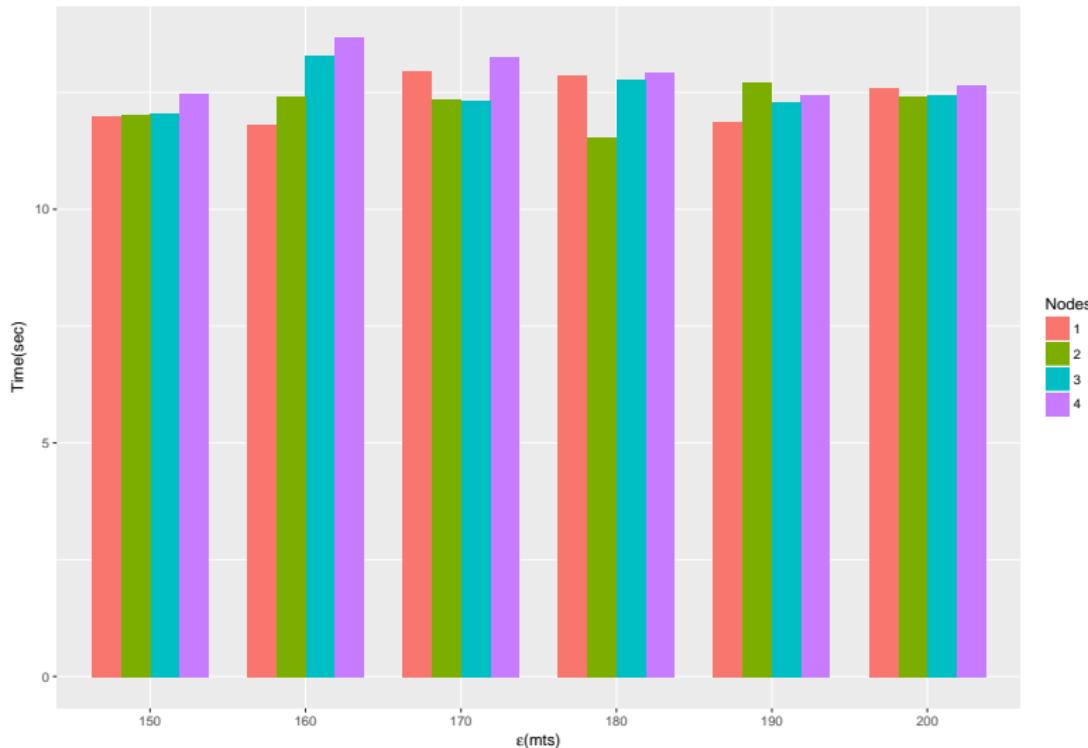
Execution time

Execution time by ε (radius of disk in mts) in Beijing dataset.



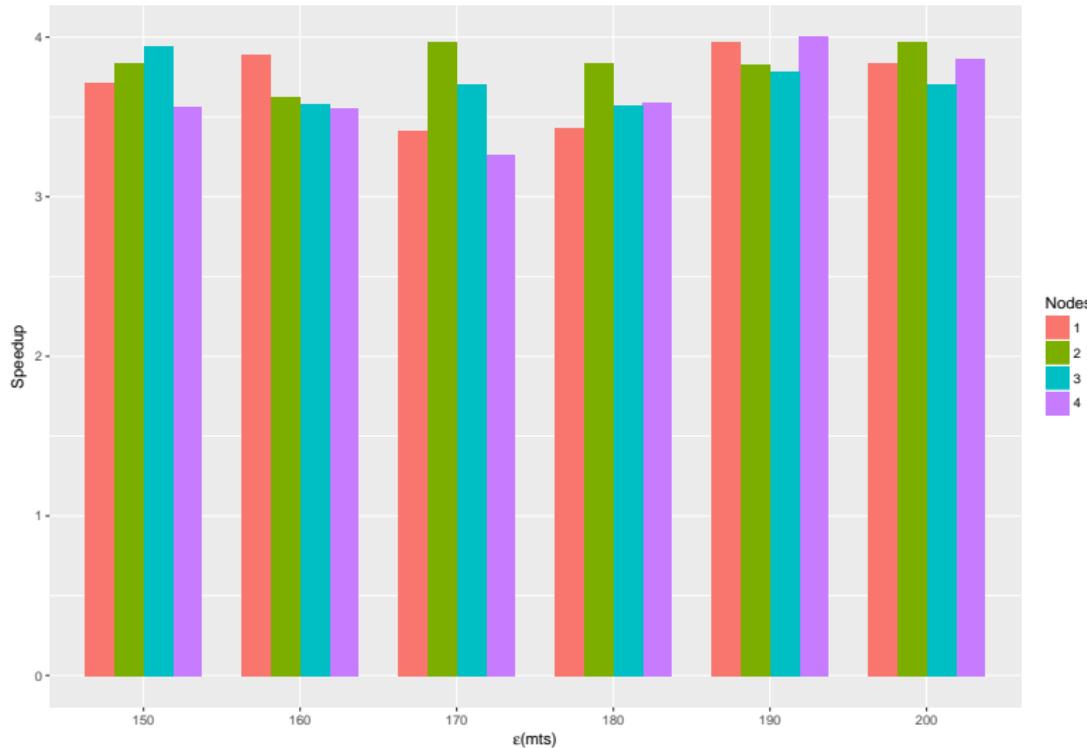
Scaleup

Scaleup in Beijing dataset.



Speedup

Speedup in Beijing dataset.

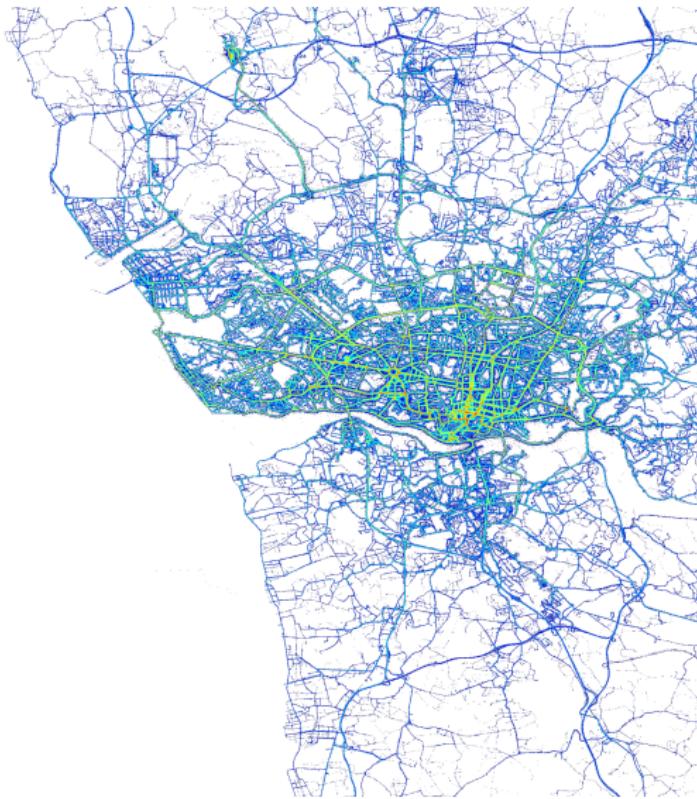


Dataset

- ▶ **Porto** from ECML/PKDD 15 Taxi Trajectory Prediction Challenge².
 - ▶ A complete year (from 01/07/2013 to 30/06/2014).
 - ▶ Trajectories for all the 442 taxis running in the city of Porto, in Portugal.
 - ▶ ≈17.7 million points (no duplicates).

²<http://tinyurl.com/zzbtlt9>

Porto dataset

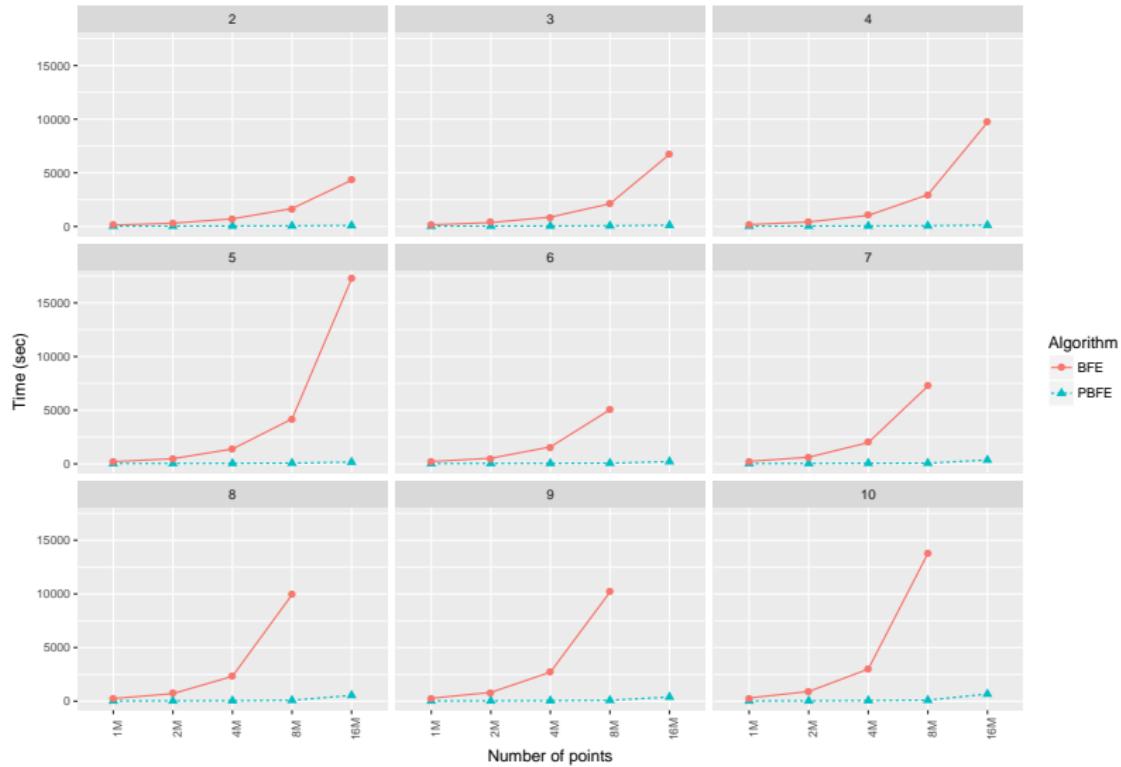


Setup

- ▶ 4-node cluster at DBLab.
- ▶ Processors: 8-core Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
- ▶ RAM: 15.5 GB.
- ▶ Centos 6.8, Simba/Spark 1.6.0.

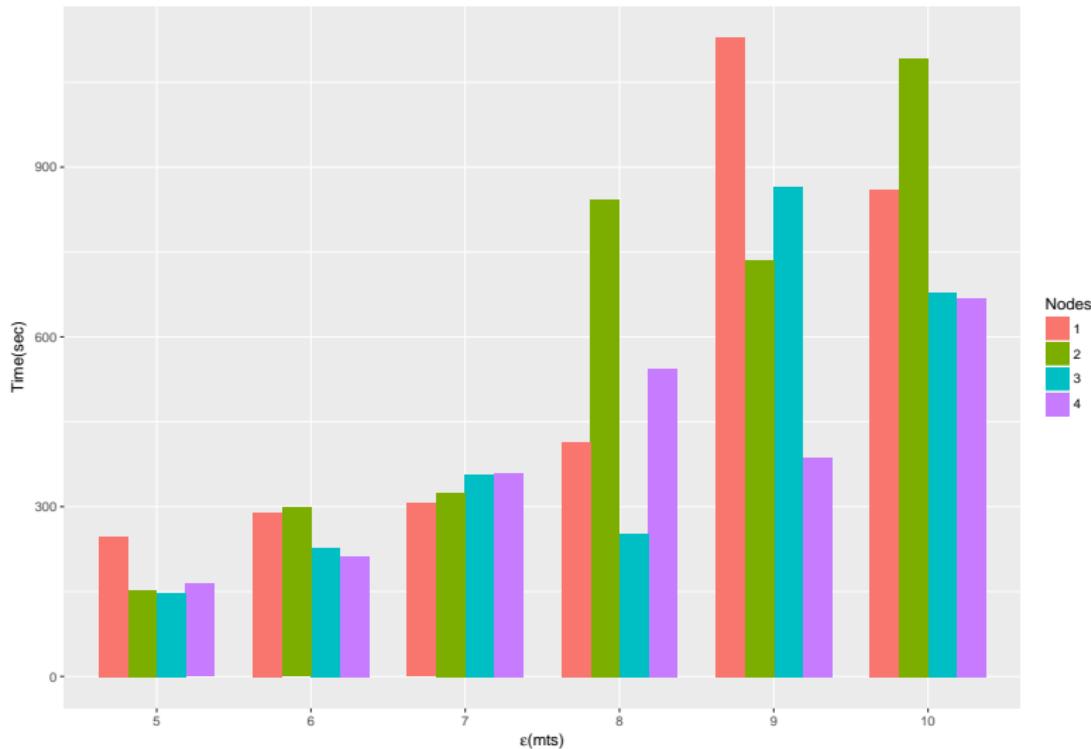
Execution time

Execution time by ϵ (radius of disk in mts) in Porto dataset.



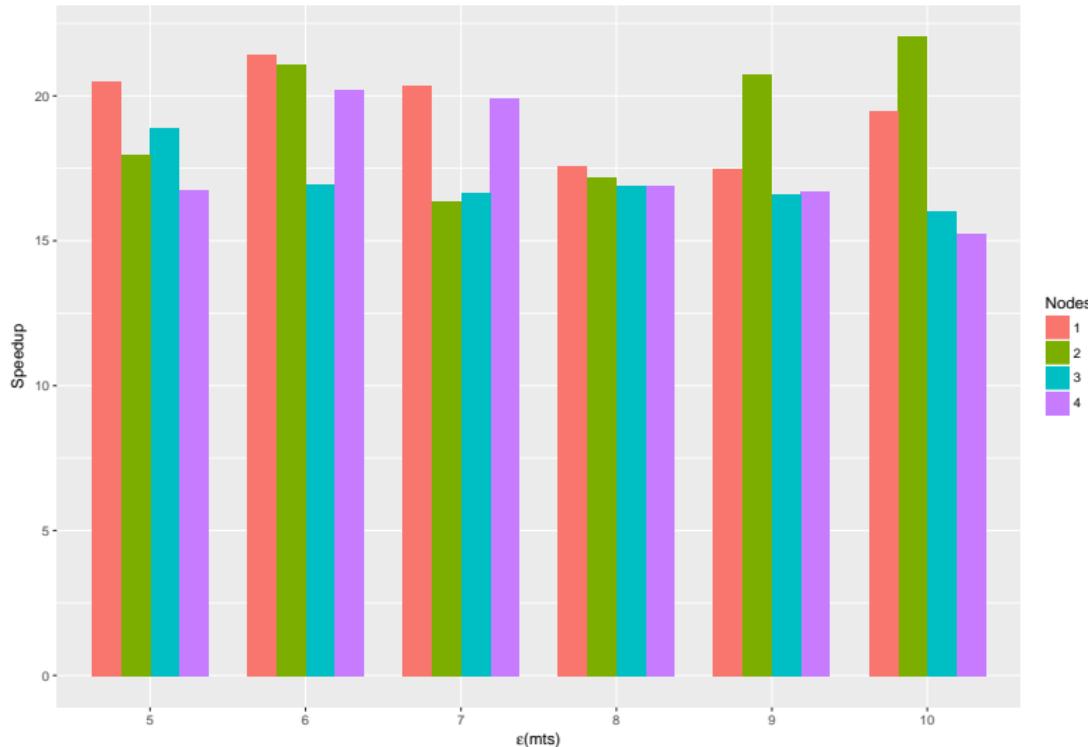
Scaleup

Scaleup in Porto dataset.



Speedup

Speedup in Porto dataset.



Outline

Moving flock patterns

Implementation

Experiments

Conclusions and Future work

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Conclusions

- ▶ An implementation of a parallel method to detect disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.
- ▶ DISTANCE 'SELF' JOINS depict a better performance using DataFrame or RDD filters (avoid Spark-SQL).

Future work

- ▶ Work on a parallel strategy to prune redundant and ‘under μ ’ disks.
- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.
- ▶ Explore more datasets to optimize Speedup and Scaleup values.
- ▶ Data pre-processing and result visualization are still open issues.

Future work

- ▶ Work on a parallel strategy to prune redundant and ‘under μ ’ disks.
- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.
- ▶ Explore more datasets to optimize Speedup and Scaleup values.
- ▶ Data pre-processing and result visualization are still open issues.

Future work

- ▶ Work on a parallel strategy to prune redundant and ‘under μ ’ disks.
- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.
- ▶ Explore more datasets to optimize Speedup and Scaleup values.
- ▶ Data pre-processing and result visualization are still open issues.

Future work

- ▶ Work on a parallel strategy to prune redundant and ‘under μ ’ disks.
- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.
- ▶ Explore more datasets to optimize Speedup and Scaleup values.
- ▶ Data pre-processing and result visualization are still open issues.

Future work

- ▶ Work on a parallel strategy to prune redundant and ‘under μ ’ disks.
- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.
- ▶ Explore more datasets to optimize Speedup and Scaleup values.
- ▶ Data pre-processing and result visualization are still open issues.

Thank you!!!

Do you have any question?