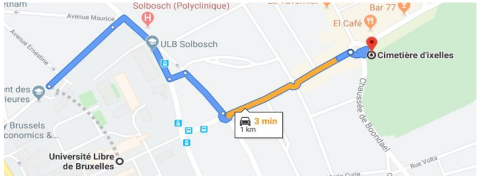
Distributed Moving Object Data Management in MobilityDB

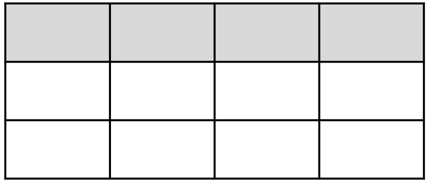
Esteban Zimányi, Mahmoud Sakr, Arthur Lesuisse and *Mohamed Bakli*
Université libre de Bruxelles, Belgium

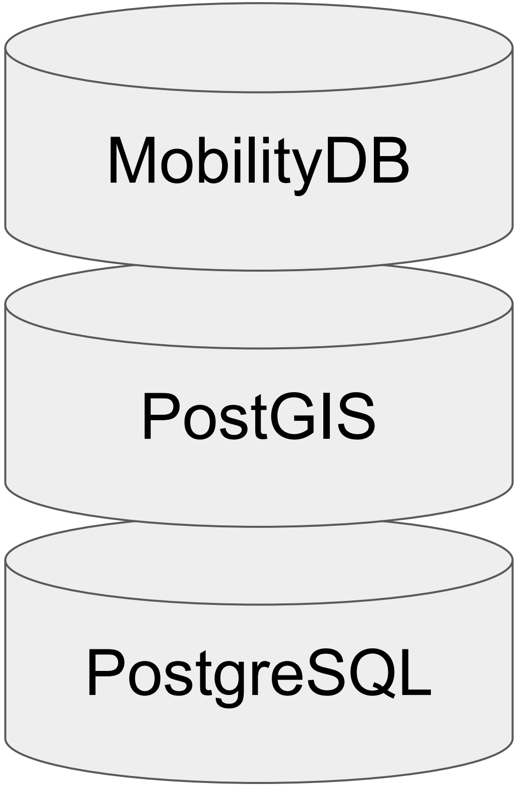
MobilityDB

- An SQL moving object database (MOD)
- Builds on PostgreSQL and PostGIS
- Developed by a team in Université libre de Bruxelles
- **Open source**: https://github.com/ULB-CoDE-WIT/MobilityDB
- Compliant with OGC standards on Moving Features, and in particular the OGC Moving Features Access

MobilityDB: Architecture





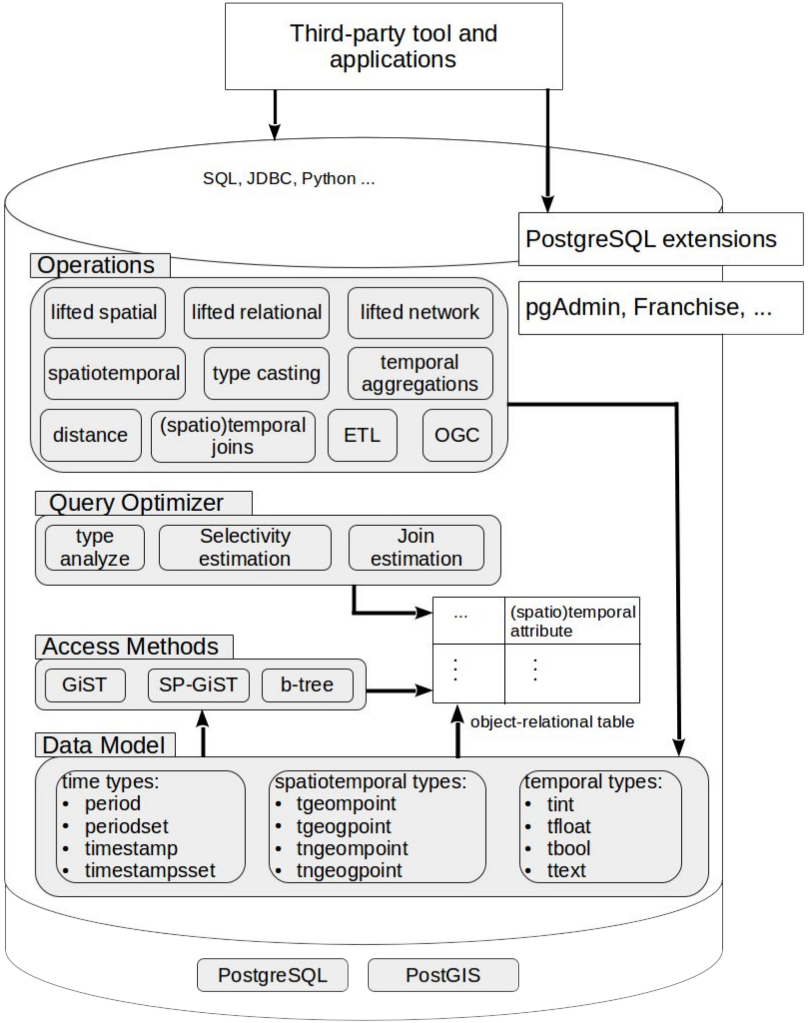


tgeompoint, tgeogpoint,
tint, tfloat, ttext, tbool

geometry, geography

numeric, monetary, character,
data/time, boolean, enum,
arrays, range,
XML, JSON, ...

MobilityDB: Features



MobilityDB: Query Examples

- Trips whose speed was ever greater than 90

```
SELECT TripID
FROM Trips
WHERE speed(Trip) ?> 90
```

- Trips that ever intersect a point of interest

```
SELECT TripID, POIDescription
FROM Trips t, POIs p
WHERE tintersects(t.Trip, p.Geom) ?= TRUE
```

- Temporal count of the number of trips

```
SELECT tcount(Trip)
FROM Trips
```

MobilityDB Ecosystem

MobilityDB MapMatch	MobilityDB Exchange	MobilityDB ETL	MobilityDB View

MobilityDB Distributed	MobilityDB Network	MobilityDB Stream	MobilityDB Python	MobilityDB JDBC
Citus	PgRouting	PipelineDB	Psycopg 2.8	PostgreSQL JDBC 42.2.6

MobilityDB	PostgreSQL 11 PostGIS 2.5	Python 3.7	Java 11

Ubuntu 18.04.2 LTS

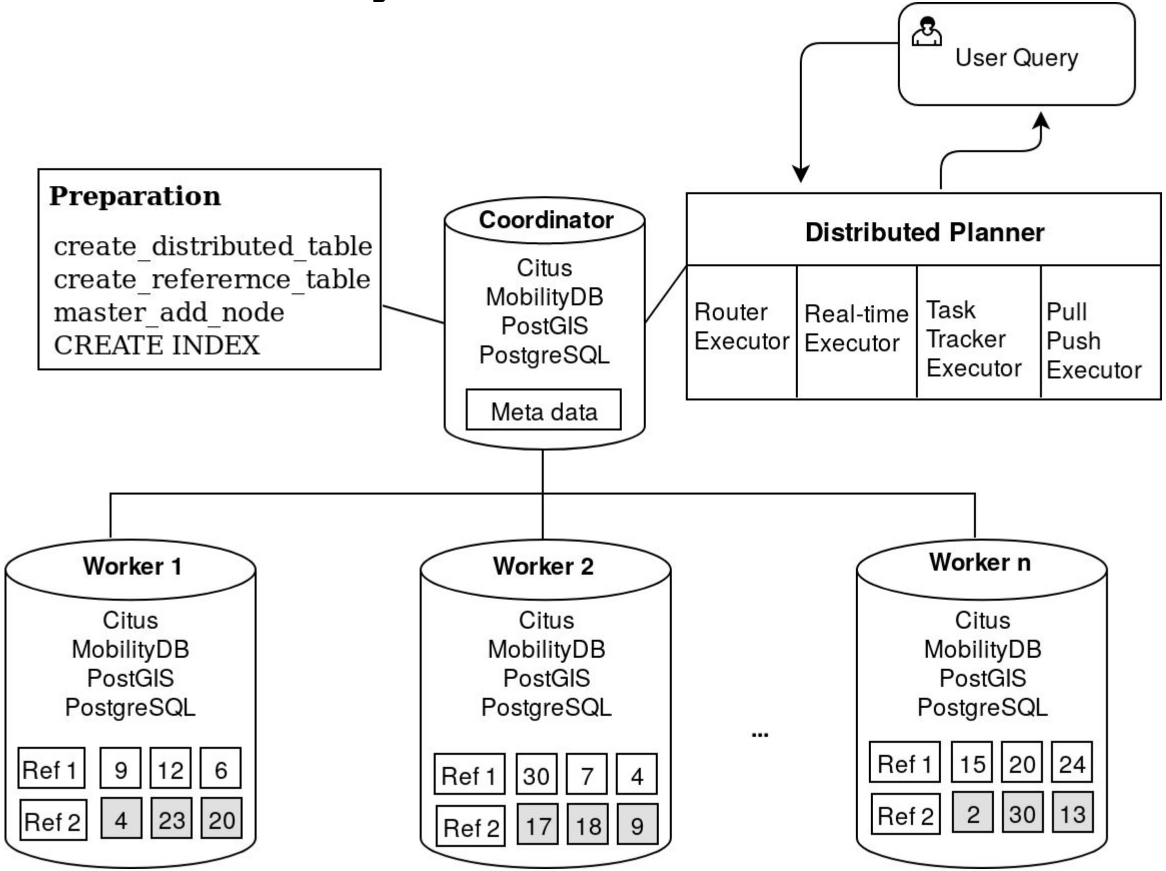
MobilityDB: Coping with Big Data

- Mobility data is typically huge
- MobilityDB builds on PostgreSQL, which is not distributed
- For big data management it is essential to scale out => Distributed MobilityDB
- The solution cannot change the PostgreSQL core

Distributed PostgreSQL: Open Source Extensions

- PostgreXL
  - A fork, not an extension of PostgreSQL
  - Newest version is based on PostgreSQL-10
- TimescaleDB
  - Extension of PostgreSQL, not a fork
  - Partition the data horizontally based on the time interval
  - Planner functions are modified from the PostgreSQL core planner
  - Optimized for storing and analyzing time series data
- Citus
  - Extension of PostgreSQL, not a fork
  - Acquired by Microsoft (2019)
  - Horizontally scales PostgreSQL across multiple machines using sharding and replication
  - Every worker receives a SQL query not an execution plan
  - Workers have their own planner and different distributed plan executors

Distributed MobilityDB: Cluster Architecture



Citus Distributed Query Planner: Query Classes

- **Routable queries:** Queries that can be fully evaluated on a subset of workers, the final result is a simple concatenation of are the workers results
- Query sent to worker nodes, which optimize it using the regular PostgreSQL planner, executes it, and returns the result to the route executor

Query	Workers	Coordinator

`SELECT *`
`FROM Weather`
`WHERE City= ‘Brussels’`
	`SELECT *`
	`FROM Weather_1`
	`WHERE City= ‘Brussels’`
		`SELECT *`
		`FROM Result_1`
		`UNION`
		`SELECT *`
		`FROM Result_1`
		`...`

Citus Distributed Query Planner: Query Classes

- **Push-downable queries**: Queries that span multiple shards and use aggregates, GROUP BY, ORDER BY, and LIMIT
- Executed by the Citus real-time executor
- Workers use PostgreSQL planner to optimize the execution of their fragments and return their result to the executor
- Executor merge results, do post processing, and produce the final result

Query	Workers	Coordinator

`SELECT COUNT(Temperature)`
`FROM Weather`
`WHERE City= ‘Brussels’`
	`SELECT COUNT(Temperature) cnt`
	`FROM Weather_1`
	`WHERE City= ‘Brussels’`
		`SELECT SUM(cnt)`
		`FROM`
		`(SELECT * FROM Result_1`
		`UNION`
		`SELECT * FROM Result_2`
		`...)`

Citus Distributed Query Planner: Query Classes

- **Recursive CTE queries**: can not be pushed down
- Planner is recursively called for the subqueries.
- Coordinator pushes back the result of the subquery to the worker nodes, these results are used as reference tables in the evaluation of the main query
- Example:

```
WITH PointCount AS (
     SELECT P.PointId, COUNT(DISTINCT T.CarId) AS Hits
     FROM Trips T, Points P WHERE tintersects(T.Trip, P.geom)
     GROUP BY P.PointId )
SELECT PointId, Hits FROM PointCount AS P
WHERE P.Hits = ( SELECT MAX(Hits) FROM PointCount )
```

Citus Distributed Query Planner: Query Classes

- **Complex queries**: Non-co-located joins, which are expensive as they involve a lot of network I/O for re-partitioning the data
- Citus planner rejects this class of queries by default: To activate it, an option needs to be set
- Queries that involved non-co-located joins always broke in our experiments
- One query might be split into multiple parts and different executors might be invoked for the parts depending on the structure of each query.

Experimental Evaluation

- **Goal**: Assess how to distribute queries by integrating MobilityDB and Citus
- Two clusters of 4 and 28 nodes
- Dataset generated by BerlinMOD, a benchmark for MOD
  - Simulated trips: to work, from work, leasure
  - Size can be controlled by a scale factor
- Workload: 17 BerlinMOD/R range queries of four categories
  - Object, Temporal, Spatial, Spatiotemporal
- We experiment with three data partitioning methods:
  - Object based (by carId)
  - 3D grid partitioning (space partitioning)
  - GiST partitioning (data partitioning, r-tree)

BerlinMOD Query in MobilityDB: Example

- Q17: Which point(s) from Points have been visited by a maximum number of different vehicles?

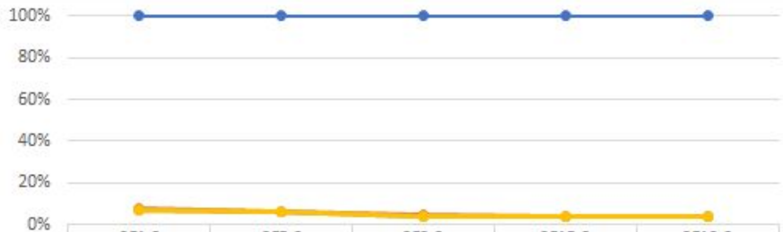
```
WITH PointCount AS (
    SELECT P.PointId, COUNT(DISTINCT T.CarId) AS Visits
    FROM Trips T, Points P
    WHERE st_intersects(trajectory(T.Trip), P.geom)
    GROUP BY P.PointId )
SELECT PointId, Visits
FROM PointCount AS P
WHERE P.Visits = ( SELECT MAX(Visits) FROM PointCount )
```

Distributed Plans in Citus: Example

- Q17: Which point(s) from Points have been visited by a maximum number of different vehicles?

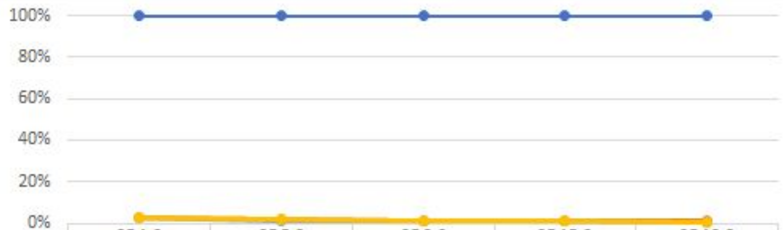
```
Custom Scan (Citus Router)
 ->  Distributed Subplan 54_1
 ->  GroupAggregate, Group Key: remote_scan.pointid
   ->  Sort, Sort Key: remote_scan.pointid
     ->  Custom Scan (Citus Real-Time)
       Task Count: 32
       Tasks Shown: One of 32
       ->  Task
         Node: host=pgxl2 port=5432 dbname=sf21_0
         ->  HashAggregate, Group Key: p.pointid, t.carid
           ->  Nested Loop
             ->  Seq Scan on points_102041 p
             ->  Bitmap Heap Scan on trips_102008 t
               Recheck Cond: (trip && p.geom)
               Filter: _intersects(trip, p.geom)
               ->  Bitmap Index Scan on
                    trips_spgist_idx_carid_102008
                 Index Cond: (trip && p.geom)
```

Experimental Results: Overall Gain



	SF1.0	SF5.0	SF9.0	SF15.0	SF19.0
Single Node	100%	100%	100%	100%	100%
Object-Based	7.97%	6.48%	5.01%	4.02%	3.90%
3D-Grid	6.62%	5.90%	4.06%	3.97%	3.88%
GiST	6.66%	6.02%	4.09%	4.00%	3.90%

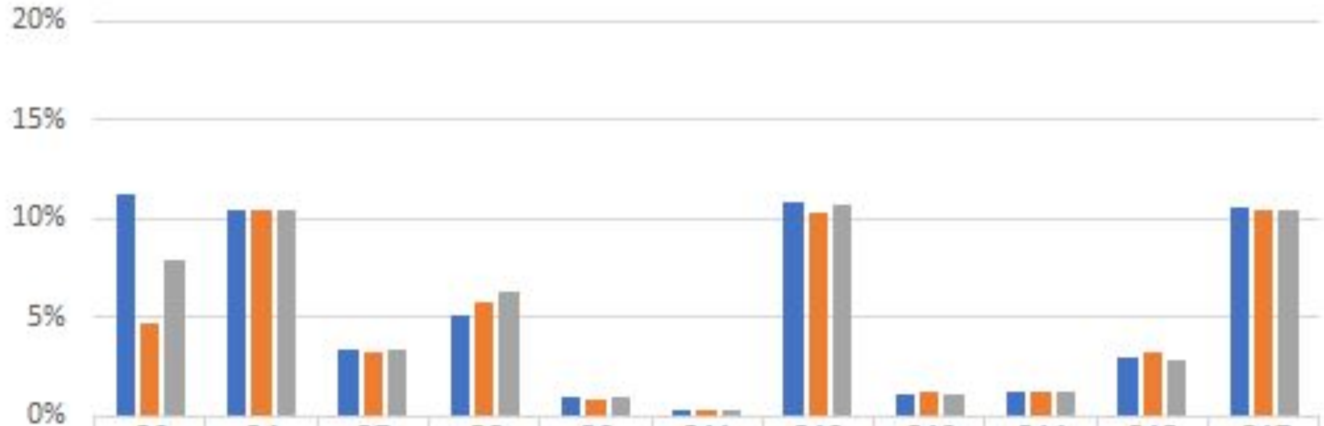
Run time gain on a cluster of 4 nodes



	SF 1.0	SF 5.0	SF 9.0	SF 15.0	SF 19.0
Single Node	100%	100%	100%	100%	100%
Object-Based	2.92%	1.87%	1.14%	0.95%	0.91%
3D-Grid	2.57%	1.58%	1.08%	0.88%	0.80%
GiST	2.66%	1.62%	1.11%	0.89%	0.79%

Run time gain on a cluster of 28 nodes

Experimental Results: Gain Per Query



	Q3	Q4	Q7	Q8	Q9	Q11	Q12	Q13	Q14	Q15	Q17
Object-Based	11%	10%	3%	5%	1%	0.21%	11%	1.13%	1%	3%	10%
3D-Grid	5%	10%	3%	6%	1%	0.21%	10%	1.23%	1.26%	3%	10%
GiST	8%	10%	3%	6%	1%	0.21%	11%	1.13%	1%	3%	10%

Run time gain per query on a cluster of 4 nodes

Discussion of Results

- Experiments done using the BerlinMOD benchmark for moving object databases
- Results show a **significant gain** in the performance of the distributed queries wrt single-node queries
- No significant differences between the data partitioning methods
- 3D-grid is slightly better than the other two methods, but with a small margin
- 13 queries out of 17 could be distributed out of the box
- **However**, BerlinMOD benchmark was not designated to distributed MOD
- A specific benchmark is required to assess the performance of different classes of distributed MOD queries

Future Work: Roadmap

- Enabling non-co-located spatiotemporal joins
- Supporting MobilityDB temporal aggregate functions
- Extending the distributed planner of Citus

Thanks for listening !

Questions ?