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

- PostgreSQL
- PostGIS
- MobilityDB

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

- Trips that ever intersect a point of interest
  
  ```sql
  SELECT TripID, POIDescription
  FROM Trips t, POIs p
  WHERE tintersects(t.Trip, p.Geom) != TRUE
  ```

- Temporal count of the number of trips
  
  ```sql
  SELECT tcount(Trip)
  FROM Trips
  ```
## MobilityDB Ecosystem

<table>
<thead>
<tr>
<th>MobilityDB MapMatch</th>
<th>MobilityDB Exchange</th>
<th>MobilityDB ETL</th>
<th>MobilityDB View</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobilityDB Distributed</td>
<td>MobilityDB Network</td>
<td>MobilityDB Stream</td>
<td>MobilityDB Python</td>
</tr>
<tr>
<td>Citus</td>
<td>PgRouting</td>
<td>PipelineDB</td>
<td>Psycopg 2.8</td>
</tr>
<tr>
<td>PostgreSQL 11 PostGIS 2.5</td>
<td>Python 3.7</td>
<td>Java 11</td>
<td></td>
</tr>
</tbody>
</table>

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

Preparation
create_distributed_table
create_reference_table
master_add_node
CREATE INDEX

Coordinator
Citus
MobilityDB
PostGIS
PostgreSQL
Meta data

Distributed Planner
Router
Executor
Real-time
Executor
Task
Tracker
Executor
Pull
Push
Executor

User Query

Worker 1
Citus
MobilityDB
PostGIS
PostgreSQL
Ref 1 9 12 6
Ref 2 4 23 20

Worker 2
Citus
MobilityDB
PostGIS
PostgreSQL
Ref 1 30 7 4
Ref 2 17 18 9

Worker n
Citus
MobilityDB
PostGIS
PostgreSQL
Ref 1 15 20 24
Ref 2 2 30 13
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

<table>
<thead>
<tr>
<th>Query</th>
<th>Workers</th>
<th>Coordinator</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT * FROM Weather WHERE City= 'Brussels'</code></td>
<td><code>SELECT * FROM Weather_1 WHERE City= 'Brussels'</code></td>
<td><code>SELECT * FROM Result_1 UNION SELECT * FROM Result_1 ...</code></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Query</th>
<th>Workers</th>
<th>Coordinator</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT COUNT(Temperature) FROM Weather WHERE City= ‘Brussels’</code></td>
<td><code>SELECT COUNT(Temperature) cnt FROM Weather_1 WHERE City= ‘Brussels’</code></td>
<td><code>SELECT SUM(cnt) FROM (SELECT * FROM Result_1 UNION SELECT * FROM Result_2 ...)</code></td>
</tr>
</tbody>
</table>
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, leisure
  - 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)
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?

```sql
1 Custom Scan (Citus Router)
2  ->  Distributed Subplan 54_1
3  ->  GroupAggregate, Group Key: remote_scan.pointid
4     ->  Sort, Sort Key: remote_scan.pointid
5     ->  Custom Scan (Citus Real-Time)
6         Task Count: 32
7         Tasks Shown: One of 32
8     ->  Task
9             Node: host=pgx12 port=5432 dbname=sf21_0
10            ->  HashAggregate, Group Key: p.pointid, t.carid
11                ->  Nested Loop
12                ->  Seq Scan on points_102041 p
13                ->  Bitmap Heap Scan on trips_102008 t
14                  Recheck Cond: (trip & p.geom)
15                  Filter: _intersects(trip, p.geom)
16                  ->  Bitmap Index Scan on
17                                  trips_spgist_idx_carid_102008
18                  Index Cond: (trip & p.geom)
```
Experimental Results: Overall Gain

Run time gain on a cluster of 4 nodes

Run time gain on a cluster of 28 nodes
Experimental Results: Gain Per Query

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?