Distributed Mobility Data Management in MobilityDB

Mohamed Bakli
Université libre de Bruxelles
Brussels, Belgium
mohamed.bakli@ulb.ac.be

Mahmoud Sakr
Université libre de Bruxelles
Brussels, Belgium
mahmoud.sakr@ulb.ac.be

Esteban Zimányi
Université libre de Bruxelles
Brussels, Belgium
ezimanyi@ulb.ac.be

***Abstract*—Mobility applications involve large amounts of data that must be managed and queried in a scalable way. MobilityDB[1] is an SQL moving object database system. It is an extension of PostgreSQL and PostGIS that supports storing and querying mobility data. This paper demonstrates the distributed query management capabilities in MobilityDB using a cluster that contains 2 billion real AIS ship trajectory points obtained from the Danish Maritime Authority.**

***Index Terms*—MobilityDB, Distributed Query Processing, AIS**

I. Introduction

The amount of location tracks generated daily from sensors, smart phones, and transportation systems is continuously increasing. This requires a production-scale data management system. Several extensions of distributed big data management platforms with trajectory analysis capabilities have been proposed, such as UlTraMan [3], HadoopTrajectory [6], and ST-Hadoop [4]. In the context of database systems, the SECONDO research prototype has a distributed algebra [5] that provides a procedural sharding and query distribution functions.

MobilityDB [2] is a production-scale open-source moving object database system that provides types and operations for managing mobility data in PostgreSQL and PostGIS. The types include temporal points, temporal floats, temporal Booleans, etc. For instance, a temporal point may be used to represent the evolution on time of the location of a car, as reported by GPS devices. A temporal float would then be used to represent its speed. MobilityDB has an extensive set of spatiotemporal operations such as distance, temporal predicates, range predicates, etc. In addition, the temporal types are supported by index access methods that extend PostgreSQL generalized search tree (GiST) and space partitioning search tree (SP-GiST) indexes. The query interface is standard SQL and hence, it is integrated with PostgreSQL query optimizer.

We have developed a distributed version of MobilityDB [1]. It is based on Citus[2], an extension that horizontally scales PostgreSQL across multiple machines using sharding and replication. We demonstrate it here using a real scenario.

[1]https://github.com/ULB-CoDE-WIT/MobilityDB
[2]https://github.com/citusdata/citus

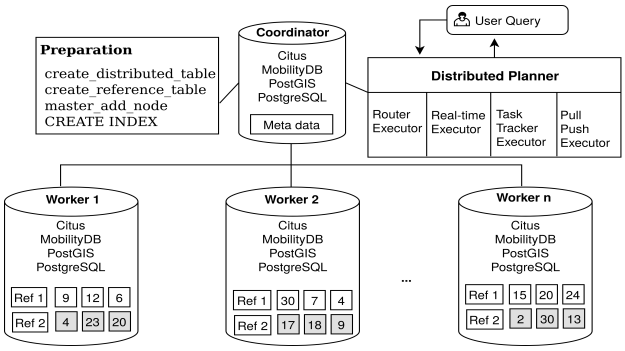


Fig. 1: Cluster architecture

II. Distributed MobilityDB

As shown in Fig. 1, the distribution is performed in two phases. In the *preparation phase*, the cluster is built and the data is sharded and replicated over worker nodes. In the *query phase*, the MobilityDB SQL queries are executed using the Citus distributed planners. The coordinator nodes distributes the data, creates distributed plans, and monitors the execution of the worker nodes. The worker nodes store data, execute queries, and generate local plans. All nodes have the same stack that consists of PostgreSQL, PostGIS, MobilityDB, and Citus. The big tables are distributed using the `create_distributed_table` function, which expects a sharding key. The coordinator routes the tuples to the worker nodes where they get physically stored. Descriptive information about the data shards is stored in the PostgreSQL catalog on the coordinator node. This information is used by the planner to distribute the queries. The other small tables are replicated on the worker nodes using `create_reference_table`.

III. Demonstration Overview

In the demonstration we execute a set of MobilityDB queries in a distributed manner. We deploy Distributed MobilityDB on a cluster of 4 nodes. The cluster is loaded with a real AIS ship dataset and will be accessed by using a VPN. Queries will be issued via the PgAdmin client and will get evaluated on the cluster.

A. Trajectory Dataset

The preloaded dataset contains ship trajectories obtained from the Danish Maritime Authority website[3]. The source data size is 500 GB. It has more than 2 billion spatiotemporal points and 800K trajectories. The schema is as follows:

```
Ships(id:int, trip:tpoint, SOG:tfloat, COG:tfloat)
Ports(id:int, name:text, geom:geometry)
```

The `Ships` table is sharded using hash-partitioning, which achieves load balancing. In addition, spatiotemporal GiST and SP-GiST indexes are built on the `trip` attribute. The `Ports` table is replicated on all workers.

B. Distributed Spatiotemporal Queries

We demonstrate Distributed MobilityDB using various SQL queries[4], for example:

- Q1 Departure time of all ships in the port of Kalundborg in Sept. 2, 2019 between 00:30 and 01:00 AM.
- Q2 Number of one-way trips that ships did on Sept. 2, 2019 between the ports of Rødby and Puttgarden.
- Q3 Trajectory and speed of the ships that spent more than 5 days to reach to the port of Kalundborg in Sept. 2019.
- Q4 Total travelled distance of all trips between the ports of Rdby and Puttgarden in Sept. 2019.

For example, Q1 above is given next:

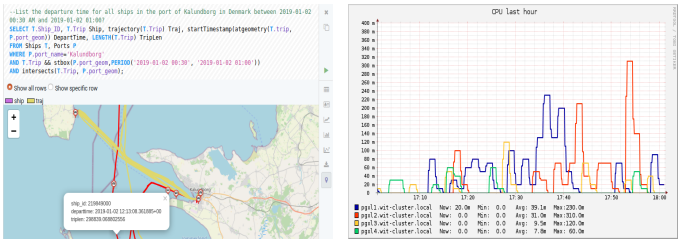
```
1 SELECT T.ship_id, startTimestamp(atGeometry(T.trip,
      P.port_geom)) AS DepartTime
2 FROM Ships T, Ports P
3 WHERE P.port_name='Kalundborg' AND T.Trip &&
4     STBOX(P.port_geom, period('2019-01-02 00:30',
          '2019-01-02 01:00'))
5     AND intersects(T.Trip, P.port_geom);
```

This query finishes in 62 seconds on a cluster of 4 nodes. It shows a *co-located join* between the distributed `Ships` table and the replicated `Ports` table. The `STBOX` function is used to construct a 3D bounding box using a `geometry` and a `period`. The *overlaps* operator `&&` checks the overlapping between the trip and the `STBOX` (Line 4). Then, the `intersects` function checks whether a temporal point has ever intersected a geometry (Line 5). The `atGeometry` function is used to retrieve the part of the trajectory that is inside the port geometry. Finally, the `startTimestamp` function is used to get the departure time (Line 1). The query plan generated by MobilityDB is as follows:

```
1 Custom Scan (Citus Real-Time)
2 Task Count: 32
3 Tasks Shown: One of 32
4   -> Task (Node: host=pgxl2 port=5432 dbname=ships)
5     -> Nested Loop
6       -> Seq Scan on ports_102046 p
7         -> Index Scan using ships_spgist_idx_102008
              on ships_102008 t
8           Index Cond: ((trip && stbox(p.port_geom,
              '[2019-01-02 00:30:00+00, 2019-01-02
              01:00:00+00)'::period))
9           Filter: intersects(trip, p.port_geom)
```

[3]AIS Dataset: https://www.dma.dk/SikkerhedTilSoes/Sejladsinformation/AIS/Sider/default.aspx

[4]The SQL queries are available at https://github.com/mbakli/MobilityDB-AIS-Queries



(a) Query visualization in Franchise (b) Cluster management in Ganglia

Fig. 2: Query execution management

This query is distributed to 32 shards on the cluster nodes by the Citus real-time planner (Lines 1-2). The number of shards is equal to the number of cores that we have on the cluster nodes. Each shard contains part of the data and the query plan above represents one of the 32 plans (Lines 3-10). The query first filters trips that were at that period in the port of Kalundborg (Lines 7-9) using the `&&` (overlaps) operator. This operator uses the SP-GiST index to prune trips that do not contribute to the results. The `intersects` function comes at the end to check the intersection between the trip points and the geometry of the port.

C. Demonstration Scenario

In the demonstration, the queries described in Section III-B are executed. The query plan is shown using the PgAdmin client. In addition, we visualize the results (i.e., trajectories and regions) using an interactive visualization tool for MobilityDB based on Franchise as shown in Fig. 2a. Finally, we will observe the query performance and resource utilization of the cluster nodes using Ganglia[5], a cluster management application. It visualizes the load on the cluster nodes after the query execution as shown in Fig. 2b.

ACKNOWLEDGMENTS

This work is done within the MobiPulse project, partially funded by Innoviris, Brussels.

REFERENCES

[1] Mohamed Bakli, Mahmoud Sakr, and Esteban Zimányi. Distributed moving object data management in MobilityDB. In Proceedings of BigSpatial '19, p. 1–10 (2019).

[2] Esteban Zimányi, Mahmoud Sakr, Arthur Lesuisse, and Mohamed Bakli. MobilityDB: A Mainstream Moving Object Database System. In Proceedings of SSTD 2019, p. 206–209 (2019).

[3] Xin Ding, Lu Chen, Yunjun Gao, Christian S. Jensen, Hujun Bao: UlTraMan: A Unified Platform for Big Trajectory Data Management and Analytics. PVLDB 11(7):787-799 (2018).

[4] Alarabi, L., Mokbel, M.F. & Musleh, M. ST-Hadoop: a MapReduce framework for spatio-temporal data. Geoinformatica 22(4): 785–813 (2018).

[5] Thomas Behr and Ralf Güting. 2016. http://dna.fernuni-hagen.de/secondo/files/Documentation/General/DistributedQuery/ProcessinginSecondo.pdfDistributed Query Processing in Secondo. Technical Report, FernUniversity Hagen. (2016).

[6] Bakli, M., Sakr, M. & Soliman, T.H.A. HadoopTrajectory: a Hadoop spatiotemporal data processing extension. Journal of Geographical Systems 21(2), 211-235 (2019).

[5]http://ganglia.sourceforge.net/