HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

MSE Database Seminar - Fall 2017

GPU Databases

Author:
Kurath Samuel

Supervisor:
Prof. Stefan Keller

January 22, 2018

Abstract

The content of this paper is splitted into three parts, it begins with an overview about GPU Database Systems. Followed by a part about MapD, that is an example of the available GPU database products. And finally a benchmark comparing the performance of MapD and PostgreSQL. The data of the benchmark is based on a excerpt of the New York City Taxi and Limousine Commission (TLC) Trip Record Data.

Contents

1. Introduction

Durring the Master of Science in Engineering the students have to participate at two seminars. The goal of these are to elaborate a theme on their own, discuss the result in group and write a paper about the topic.

The Databasesystems Seminar does a focus on GPU Database Systems. The students do have a closer look at a certain GPU Database product and have to do a benchmark.

The benchmark is based on a excerpt of the New York City Taxi and Limousine Commission (TLC) Trip Record Data and the queries are predetermined by Prof. Stefan Keller.

2. GPU Databases

Related to the massive amount of data that is collected nowadays, the stagnation of CPU speed and the trend to use the GPU for tasks like machine learning, the database developers have discovered the GPU to improve the performance of their products, too. Hence the main idea of GPU databases is to perform some operations on the GPU for acceleration purposes.

Strengths GPUs do have their strengths on parallelable tasks. This is due to the fact that GPUs can have thousands of cores and high bandwidth memory on each card. Thus most of the GPU databases products focus on analytics.

Weaknesses Beside of these strengths, GPU databases host several pitfalls like transfer of the data from the CPU to the GPU, the memory limitations and the massiv costs of GPU servers.

2.1. Components of a GPU database

The current section will give you an overview of the components a GPU database consists of. The paper [Bre+14] of Sebastian Bress et al. does split the components into Functional and Non-Functional properties. Since those categories are reasonable they are used in this paper as well. The next sub section will explain and list these properties.

2.1.1. Non-Functional properties

Performance

Performance is the biggest advantage of GPU database, but not all tasks are automatic faster on GPUs. Due to the huge amount of processors tasks that are able to parallelise easily are incredibly fast. Unfortunately task which require a complex control flow or are hard to parallelise don't profit from the cores. And there is always the bottleneck of the data transfer to the GPU.

Portability

In terms of portability we talk about if the GPU database is able to run on different GPUs or CPUs vendors/architectures, like NVIDIA or AMD graphic cards. Often this requirement is in contrast to the performance property, since the vendors offer special implementation or hardware details to accelerate their products.

Scalability

To my point of view the paper [Bre+14] of Sebastian Bress et al. missed the important criteria of the scalability. The collected data of companies grow and grow, thus you need to have memory for all that data. Sadly the memory of GPUs is often limited, hence the only way to handle this issue it to scale your GPU database verticaly over multiple GPUs.

2.1.2. Functional properties

Storage system

If we talk about storage systems there are several scenarios conceivable. First with the Video Random Access Memory (VRAM) of the GPUs we have an additional storage medium next to the Random-Access Memory (RAM) and the hard disk. The different mediums provide different advantages and disadvantages. Hard disks are persistent, cheap and have a huge capacity. Unfortunately they are pretty slow. Random-Access Memory is very fast compared to hard disks but the transfer to the GPU is still a bottleneck and it isn't a persistent storage, either. With Video Random Access Memory the bottleneck of the transfer disappears, but most of the time the storage capacity is highly limited.

Storage model

In terms of storage model, there are row stores or column stores [AMH08]. Row stores store table records in a sequence of rows. Column stores store table records in a sequence of columns, the entries of a column is stored in contiguous memory locations [Bre14]. The advantages of column stores are tasks like aggregations, though row stores are more efficient if the result of a query returns multiple rows.

Processing model

There are two processing models in modern databases tuple-at-a-time and operator-at-a-time. The tuple-at-a-time is similar to an iterator, which iterates over the relevant tuples and applies the operations. The operator-at-a-time fits to GPUs, since it applies the same operation on a bulk of data.

Query processing

GPU database systems are able to use the GPU and the CPU, hence it is necessary to choose the right processor for the right task.

Query placement It is a extremely difficult task to decide which processing device is the most accurate for the current query.

Optimization To optimize the performance in therms of query execution time, several factors are include. For example the operations, the data, hardware specifications and even more.

Transaction support

An other problem are transactions and consistency on GPUs, until now there is almost no research done in this area. Thus most of the GPU database don't support transactions.

2.1.3. Proposed architecture

The paper [Bre+14] proposes an architecture with an in-memory storage, using a column store, an operator-at-a-time processing model, cross device processing and no transaction support. With regard to the portability a hardware oblivious architecture is suggested. To my point of view a hardware aware GPU database would make more sense. Since GPU databases are all about speed, use hardware specific tuning would result in more acceleration. And the huge amount of data makes scalability absolutely necessary.

3. MapD

MapD is a GPU database with the goal to speed up queries and analytic tasks with the power of GPU's and their massive parallel architectures consisting of thousands of cores. The first prototype of MapD was develop in 2012 by Todd Mostak. A year leater MapD was incubated at the MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) database group and in September 2013 Todd Mostak founded MapD Technologies, Inc. They have got two products called MapD Core [Map17c] and MapD Immerse [Map17d]. The MapD Core SQL engine is an open source in-memory, SQL, GPU database and MapD Immerse is a tool for visual analytics on top of the MapD Core SQL engine.

Figure 3.1.: *MapD Logo*

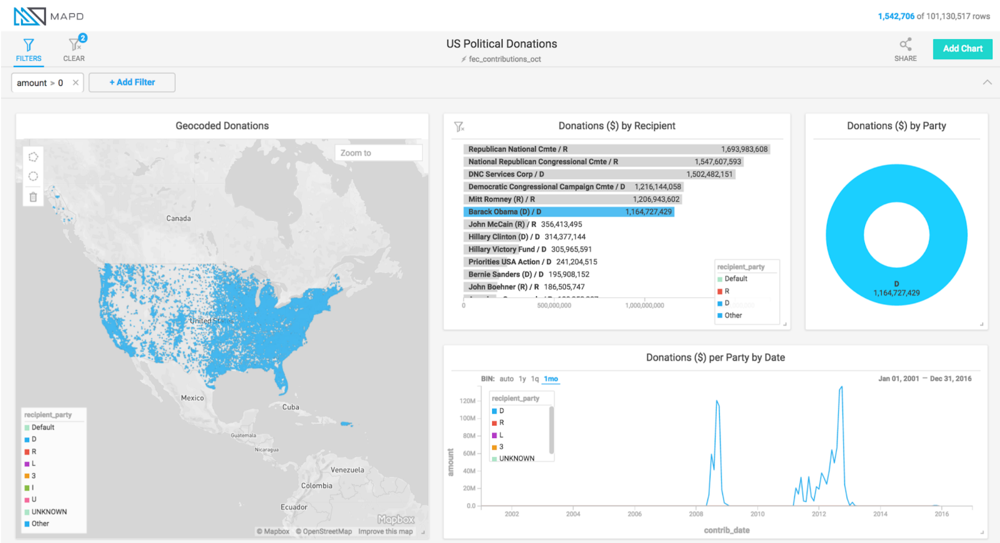

Figure 3.2.: *MapD Immerse [Dup17]*

3.0.1. Functional Properties

Related to the excellent Survey of Sebastian Bress et al. [Bre+14] MapD has the following functional properties.

Storage system

MapD has got a relational DBMS that is able to handle data amounts bigger than the memory space. But tries to hold as much data as possible in-memory to to improve the performance.

Storage model

MapD uses a columnar layout to store the data and uses so called chunks, which split the columns in smaller pieces. The chunks are the basic units of the memory manager.

Processing model MapD processes on operator-at-a-time or one chunk per operation. Thus it is a block-oriented processing. The queries are compiled for the CPU and the GPU.

Query processing

Query placement In contrast to [Bre+14] the gained experience with MapD showed that MapD tries to run the queries on the GPU even if there isn't enough space and isn't able to handle such queries on his own. Hence the user has to switch the execution mode from GPU to CPU.

Optimization MapD's optimizer tries to execute the queries on the most suitable device, like text searching using an index on the CPU and table scans on the GPU.

Transactions support

MapD doesn't not support transactions and doesn't support UPDATE or DELETE operations on the inserted data, either. That could be related to the difficult task of handling persistence between the CPU and GPU or even distributed systems at all.

3.1. Overview

The following section will give you an overview about the handling of MapD. Often there will be a comparison between MapD and PostgreSQL to point out the differences and similarities of these two databases.

3.1.1. Data Definition Language (DDL)

The DDL seams familiar since it uses SQL syntax [Map17a]. The syntax to handle users, databases, tables, and views is listed below.

User

- CREATE USER
- DROP USER
- ALTER USER

Database

- CREATE DATABASE
- DROP DATABASE

Table

- CREATE TABLE
- CREATE TABLE AS SELECT
- ALTER TABLE
- DROP TABLE
- TRUNCATE TABLE

View

- CREATE VIEW
- DROP VIEW

Datatypes

MapD supports the data types shown in table 3.1. To get a better intuition the corresponding PostgreSQL data types are listed as well.

MapD		PostgreSQL	
Data type	Size [bytes]	Data type	Size [bytes]
TEXT	variable	text	variable
TIMESTAMP	8	timestamp	8
TIME	8	time	8
DATE	8	date	4
FLOAT	4	real	4
DOUBLE	8	double precision	8
INTEGER	4	integer	4
SMALLINT	2	smallint	2
BIGINT	8	bigint	8
BOOLEAN	1	boolean	1
DECIMAL	8	numeric	variable

Table 3.1.: *Data types [Map17b] [Gro17a]*

As you can see MapD supports the common data types. And if you compare them to the corresponding PostgreSQL types they have got nearly the same names. In addition PostgreSQL provides further data types like json or box which allow extended possibilities of use.

3.1.2. Data Manipulation Language (DML)

As well as the DDL of MapD the DML uses the SQL syntax too. MapD currently supports the instructions:

- INSERT INTO
- SELECT

Until now there is no support for the operations:

- DELETE
- UPDATE

But they are in development, since MapD don't want to compromising the speed much with these instructions it may take a while.

Furthermore, MapD provides operations like EXPLAIN, LIKELY/UNLIKELY, Aggregate Functions and Conditional Expressions to improve the DML operations and extend the functionality.

3.1.3. Data import

MapD allows to import data from different sources as you can see in the following section.

COPY FROM

The COPY FROM operation is callable from the mapdql terminal and imports data from a local CSV or related format file into the database.

SQL Importer

The SQL Importer is a java tool that allows to run queries on other database via JDBC and stores the results into MapD.

StreamInsert

The StreamInsert program could be attached at the end of a real-time stream processing engine like Kafka or a similar product to import stream data into MapD for further analytic tasks.

HDFS

The tool sqoop-export offers the possibility to import CSV or Parquet files from a HDFS file system into MapD database.

3.1.4. Data export

To export data from MapD, mapdql provides the command COPY TO that allows to export the result of a SELECT statement to a file. For example like:

- COPY (SELECT * FROM tweets) TO '/tmp/tweets.csv';

3.1.5. Client interfaces

MapD provides a tool called mapdql [Map17e] as a client-side SQL console that displays query results you submit to the MapD Core Server. The counterpart of PostgreSQL is psql [Gro17b].

Database connection

To following commands compares the connection to MapD respectively PostgreSQL with mapdql and psql.

mapdql: mapdql <database> -u <user> -p <password> –port <port> -s <host>

pgsql: psql -h <host> -p <port> -U <user> -W <password> <database>

Commands

The table 3.2 lists some basic commands of mapdql and pgsql. It is only a slight slice of all possible commands, but another good example to point out how much in common those two products have.

Command	mapdql	pgsql
List databases	\l	\l
List tables in database	\t	\d
Describe a table	\d <table>	\t <table>
Connect to a database	\c	\c
Print timing information	\timing	\timing
Switch to GPU mode	\gpu	-
Switch to CPU mode	\cpu	-
Quit	\q	\q

Table 3.2.: *Commands*

Alternative interfaces

Beside of mapdql MapD provides the following interfaces:

- JDBC
- ODBC
- pymapd
- Python JayDeBeApi
- SQuirreL SQL
- RJDBC
- Apache Thrift

3.1.6. Outlook

As MapD begin their development there already have been companies with a huge amount of data, but the focus was to do visualization and analytic task on only a part of this data. Since the data should fit on a single GPU. In recent times the customer of MapD and similar vendors like to use the GPU acceleration on all their data. Hence they would like to use multiple GPUs to do their analytic tasks. With the Version 3.0, MapD realized these requirements and is doing ongoing reseach and development to improve and accelerate the scaling task. [Mor18] And as already mentioned MapD tries to support operations to alter the stored data.

4. Benchmark

This section does concerned with a benchmark between the two databases PostgreSQL 10.0 and MapD 3.3.1. The dataset is based on a part of the New York City Taxi and Limousine Commission (TLC) Trip Record Data [New17]. All queries were executed on the same server.

4.1. Dataset

Figure 4.1.: *TLC Trip Record Data*

The New York City Taxi and Limousine Commission (TLC) Trip Record Data [New17] consists of trip records from the yellow and green taxis. Furthermore there are data from the For-Hire Vehicle (FHV). The data includes information about capturing pick-up and drop-off locations, times, trip distances, fares, rate types, and driver-reported passenger counts.

To simplify the handling with the huge amount of data we used the scripts from the Github repository [Sch17b], that is able to download all the data, provides the schema for PostgreSQL with the PostGIS [Dev17] extension and consists of scripts to import the data.

4.1.1. Specification

To get a brief overview of the specification have a look at the following list.

Format:	CSV
Yellow:	January 2009 - June 2017
Green:	August 2013 - June 2017
FHV:	January 2015 - June 2017
Taxi trips:	Over 1.1 billion [Sch17a]
Size:	267 GB [Sch17a]

4.1.2. Data subset

Due to the fact that not all GPU databases are able to handle queries based on data larger than the GPU memory, we had to shrink the dataset fitting into the available memory.

The GPU used on the server for the benchmark is a Nvidia Tesla K40m with 12 GB memory. Hence we used the yellow and green taxi trip data of the year 2015, that has a size of 12 GB, too. Consequently all queries fit into the GPU memory.

4.2. Queries

The Queries were predefined by Prof. Stefan Keller and are inspired by the Google BigQuery examples [LLC17]. All the queries are listed below, the used syntax is able to run on MapD.

```
SELECT passenger_count, Avg(total_amount)
FROM   trips
GROUP  BY 1;
```

Listing 4.1: *Query 1, Counts all the yellow taxi trips*

```
SELECT cab_type_id, Count(*)
FROM   trips
GROUP  BY 1;
```

Listing 4.2: *Query 2, Calculates the average passenger amount per trip*

```
SELECT passenger_count, Extract(year FROM pickup_datetime), Count(*)
FROM   trips
GROUP  BY 1, 2;
```

Listing 4.3: *Query 3, Sums the yearly amount of passengers*

```
SELECT passenger_count, Extract(year FROM pickup_datetime), Cast(
    trip_distance AS INT), Count(*)
FROM   trips
GROUP  BY 1, 2, 3
ORDER  BY 2, 4 DESC;
```

Listing 4.4: *Query 4, Groups the amount of passenger by year regarding the trip distance*

```
SELECT *
FROM   trips
WHERE  ( pickup_longitude BETWEEN -74.007511 AND -73.983479 )
       AND ( pickup_latitude BETWEEN 40.7105 AND 40.731071 ) LIMIT 10;
```

Listing 4.5: *Query 5, Queries all trips in a certain bounding box*

```
SELECT Extract(HOUR FROM pickup_datetime) AS h, AVG(trip_distance / NULLIF(
    TIMESTAMPDIFF(HOUR, pickup_datetime, dropoff_datetime),0)) AS speed
FROM   trips
WHERE  ( pickup_longitude BETWEEN -74.007511 AND -73.983479 )
       AND ( pickup_latitude BETWEEN 40.7105 AND 40.731071 )
       AND trip_distance > 0
       AND fare_amount / trip_distance BETWEEN 2 AND 10
       AND dropoff_datetime > pickup_datetime
       AND cab_type_id = 1
GROUP  BY h
ORDER  BY h;
```

Listing 4.6: *Query 6, Determines the average speed of the yellow taxi trips by hour of the day in a bounding box*

```
SELECT Extract(HOUR FROM pickup_datetime) AS h, Avg(trip_distance / NULLIF(
    TIMESTAMPDIFF(HOUR, pickup_datetime, dropoff_datetime),0))AS speed
FROM   trips
WHERE  trip_distance > 0
       AND fare_amount / trip_distance BETWEEN 2 AND 10
       AND dropoff_datetime > pickup_datetime
       AND cab_type_id = 1
GROUP  BY h
ORDER  BY h;
```

Listing 4.7: *Query 7, Computes the average speed of the yellow taxi trips by hour of the day*

```
SELECT Extract(DOW FROM pickup_datetime) AS dow, Avg(trip_distance / NULLIF(
    TIMESTAMPDIFF(HOUR,pickup_datetime,  dropoff_datetime), 0)) AS speed
FROM   trips
WHERE  trip_distance > 0
       AND fare_amount / trip_distance BETWEEN 2 AND 10
       AND dropoff_datetime > pickup_datetime
       AND cab_type_id = 1
GROUP  BY dow
ORDER  BY dow;
```

Listing 4.8: *Query 8, Calculates the average speed of the yellow taxi trips by day of the week*

```
SELECT Extract(DOW FROM pickup_datetime) AS dow, Avg(trip_distance / NULLIF(
    TIMESTAMPDIFF(HOUR,pickup_datetime,  dropoff_datetime), 0)) AS speed
FROM   trips
WHERE  ( pickup_longitude BETWEEN -74.007511 AND -73.983479 )
       AND ( pickup_latitude BETWEEN 40.7105 AND 40.731071 )
       AND trip_distance > 0
       AND fare_amount / trip_distance BETWEEN 2 AND 10
       AND dropoff_datetime > pickup_datetime
       AND cab_type_id = 1
GROUP  BY dow
ORDER  BY dow;
```

Listing 4.9: *Query 9, Determines the average speed of the yellow taxi trips by day of the week in a bounding box*

4.3. Results

As already mentioned the queries of the benchmark were applied on a database without GPU acceleration, in our case PostgreSQL 10.0 with the PostGIS 2.4 extension and a on MapD 3.3.1 a GPU powered database. The dataset was introduced in section 4.1 and the queries are listed in the section 4.2.

4.3.1. PostgreSQL vs. MapD cold

The diagram 4.2 compares the query duration of PostgreSQL and MapD at the first time of execution (cold). That means there might are no caching effects influencing the measurement.

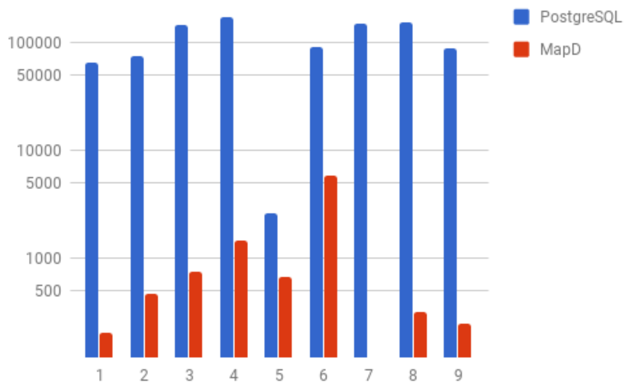

Figure 4.2.: *Diagram PostgreSQL vs. MapD cold (logarithmic scale)*

Query	PostgreSQL [ms]	MapD [ms]
1	64878	203
2	74404	456
3	145749	742
4	172592	1431
5	2556	663
6	89029	5756
7	150823	119
8	152883	315
9	87535	243

Table 4.1.: *Data PostgreSQL vs. MapD cold*

4.3.2. PostgreSQL vs. MapD warm

The diagram 4.3 compares the query duration of PostgreSQL and MapD after the first time of execution (warm). That means there might are caching effects speeding up the measurement. The query was processed 5 times and the values listed in the table 4.2 are the mean time of these measurements.

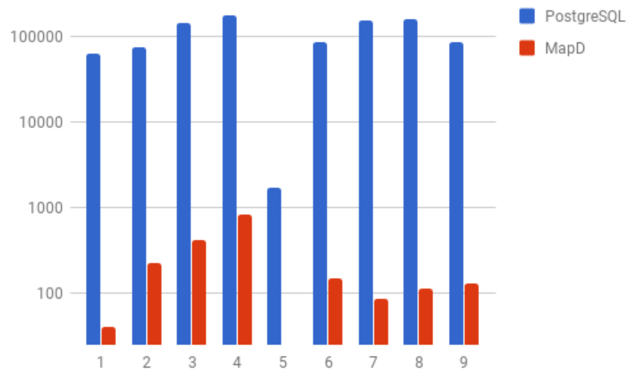

Figure 4.3.: *Diagram PostgreSQL vs. MapD warm (logarithmic scale)*

Query	PostgreSQL [ms]	MapD [ms]
1	62684	41
2	75140	223
3	142338	412
4	172052	840
5	1685	25
6	84439	149
7	150073	86
8	155107	115
9	86060	131

Table 4.2.: *Data PostgreSQL vs. MapD warm*

4.3.3. PostgreSQL cold vs. warm

The diagram 4.4 compares the query duration of PostgreSQL at the first execution with the mean of the next 5 executions.

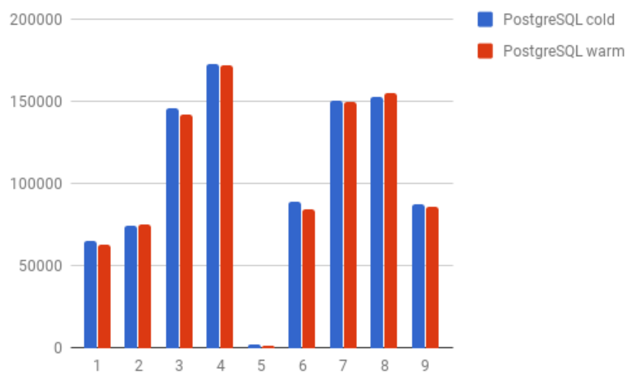

Figure 4.4.: *Diagram PostgreSQL cold vs. PostgreSQL warm*

Query	PostgreSQL cold [ms]	PostgreSQL warm [ms]
1	64878	62684
2	74404	75140
3	145749	142338
4	172592	172052
5	2556	1685
6	89029	84439
7	150823	150073
8	152883	155107
9	87535	86060

Table 4.3.: *Data PostgreSQL cold vs. PostgreSQL warm*

4.3.4. MapD cold vs. warm

The diagram 4.5 compares the query duration of MapD at the first execution with the mean of the next 5 executions.

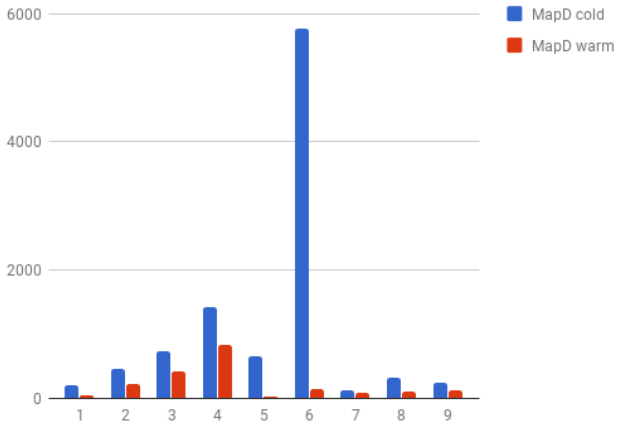

Figure 4.5.: *Diagram MapD cold vs. MapD warm*

Query	MapD cold [ms]	MapD warm [ms]
1	203	41
2	456	223
3	742	412
4	1431	840
5	663	25
6	5756	149
7	119	86
8	315	115
9	243	131

Table 4.4.: *Data MapD cold vs. MapD warm*

4.3.5. Conclusion

As you can see in the diagrams 4.2 and 4.3, MapD is up to 1500 times faster then PostgreSQL. I even had to use a logarithmic scale to make the duration of the PostgreSQL queries viewable on the diagrams. The performance of MapD is really astonishing. The huge speedup compared to PostgreSQL might due to the fact that MapD is an in-memory database and the GPU acceleration is a big improvement as well. Further, the MapD usage of a column store instead of a row-oriented storage like PostgreSQL does increase the performance, too. And the fact that MapD is developed for an analytical purpose and doesn't support transaction in contrast to PostgreSQL must not be forgotten. Another interesting insight can be observed at the comparison of the query execution at the first time (cold) or the following executions (warm). There is no significant difference between the cold and warm execution of the PostgreSQL queries shown at 4.4. In contrast to PostgreSQL, all warm queries were faster than the cold ones as illustrated at 4.5.

Appendices

A. Repositories

The current section is about the used source code and the related repositories, that are used during the seminar.

A.1. MSE-Database-Seminar-GPU-Databases

The repository MSE-Database-Seminar-GPU-Databases[1] is provided by Prof. Stefan Keller. It is the main entrypoint for the database seminar fall 2017.

Content

The repository consists of the queries for the benchmark. The queries are available for PostgreSQL, PostGIS and MapD. In addition there is a schema for MapD for the New York City Taxi drives dataset and an data import script for MapD as well. Further, there are Dockerfiles for PostgreSQL with the PostGIS extension, BlazingDB, MapD and PG-Strom.

PostgreSQL with the PostGIS extension is based on the Dockerfile of Mike Dillon [Dil17], who provides Dockerfiles for several versions of PostgreSQL with PostGIS. Additionally there are fine tuned settings for PostgreSQL related to the powerful server.

MapD docker image is based on the Dockerfile [Sei17] maintained by Andrew Seidl, who is a developer at MapD Technologies, Inc.

PG-Strom [Koh17] is an extension for PostgreSQL that adds GPU acceleration to the database.

[1] `https://github.com/geometalab/MSE-Database-Seminar-GPU-Databases`

A.2. nyc-taxi-data

The repository nyc-taxi-data[2] of Todd W. Schneider provides scripts for:

- Data download
- Database schema for PostgreSQL with the PostGIS extension
- Data import script
- Scripts for analysis purpose

It is the code base of Todd W. Schneider's article about the New York City Taxi and Uber data [Sch17a].

Figure A.1.: *New York City Taxi Drop Offs [Sch17a]*

A.3. GpuDbSeminar

The repository GpuDbSeminar [3] of Samuel Kurath hosts the documentation of this paper and provides Dockerfiles for the benchmark.

[2] `https://github.com/toddwschneider/nyc-taxi-data`
[3] `https://github.com/Murthy10/GpuDbSeminar`

Bibliography

[AMH08] Daniel J Abadi, Samuel R Madden, and Nabil Hachem. "Column-stores vs. row-stores: How different are they really?" In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 967–980.

[Bre+14] Sebastian Breß et al. "Gpu-accelerated database systems: Survey and open challenges". In: *Transactions on Large-Scale Data-and Knowledge-Centered Systems XV*. Springer, 2014, pp. 1–35.

[Bre14] Sebastian Breß. "The design and implementation of CoGaDB: A column-oriented GPU-accelerated DBMS". In: *Datenbank-Spektrum* 14.3 (2014), pp. 199–209.

[Dev17] PostGIS Developers. *Spatial and Geographic objects for PostgreSQL*. Dec. 2017. URL: https://postgis.net/.

[Dil17] Mike Dillon. *mdillon/postgis*. Dec. 2017. URL: https://github.com/appropriate/docker-postgis.

[Dup17] Tai Dupree. *Quick Insight with MapD Immerse Cross Filtering*. Dec. 2017. URL: https://www.mapd.com/blog/2017/04/14/quick-insight-with-mapd-immerse-cross-filtering/.

[Gro17a] The PostgreSQL Global Development Group. *Chapter 8. Data Types*. Dec. 2017. URL: https://www.postgresql.org/docs/10/static/datatype.html.

[Gro17b] The PostgreSQL Global Development Group. *psql*. Dec. 2017. URL: https://www.postgresql.org/docs/10/static/app-psql.html.

[Koh17] KaiGai Kohei. *PG-Strom*. Dec. 2017. URL: http://strom.kaigai.gr.jp/.

[LLC17] Google LLC. *NYC Taxi and Limousine Trips*. Dec. 2017. URL: https://cloud.google.com/bigquery/public-data/nyc-tlc-trips?hl=en.

[Map17a] Inc. MapD Technologies. *Data Definition (DDL)*. Dec. 2017. URL: https://www.mapd.com/docs/latest/mapd-core-guide/data-definition/.

[Map17b] Inc. MapD Technologies. *Datatypes and Fixed Encoding*. Dec. 2017. URL: https://www.mapd.com/docs/latest/mapd-core-guide/fixed-encoding/.

[Map17c] Inc. MapD Technologies. *MapD Core - In-Memory, Open Source SQL engine*. Dec. 2017. URL: https://www.mapd.com/platform/core/.

[Map17d] Inc. MapD Technologies. *MapD Immerse Visual Analytics*. Dec. 2017. URL: https://www.mapd.com/platform/immerse/.

[Map17e] Inc. MapD Technologies. *mapdql*. Dec. 2017. URL: https://www.mapd.com/docs/latest/mapd-core-guide/mapdql/.

[Mor18] Timothy Prickett Morgan. *Pushing A Trillion Row Database With GPU Acceleration*. Jan. 2018. URL: https://www.nextplatform.com/2017/04/26/pushing-trillion-row-database-gpu-acceleration/.

[New17] The City of New York. *TLC Trip Record Data*. Dec. 2017. URL: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[Sch17a] Todd W. Schneider. *Analyzing 1.1 Billion NYC Taxi and Uber Trips, with a Vengeance*. Dec. 2017. URL: http://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/.

[Sch17b] Todd W. Schneider. *Unified New York City Taxi and Uber data*. Dec. 2017. URL: https://github.com/toddwschneider/nyc-taxi-data.

[Sei17] Andrew Seidl. *MapD container*. Dec. 2017. URL: https://github.com/mapd/mapd-core/tree/master/docker.