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 port 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 part of the Unified New York City Taxi and Uber 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 the public NYC Taxi Rides dataset 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 database 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 interesting Non-Functional properties of GPU database are performance and portability. The next sub section will explain and list the Functional Properties.

### 2.1.1. Functional properties

#### Storage system

If we talk about storage systems there are several scenarios conceivable. First we 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 provides 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. 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.

### 2.1.2. 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.

# 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 MapD Core SQL engine.


**Figure 3.1.:** *MapD Logo*

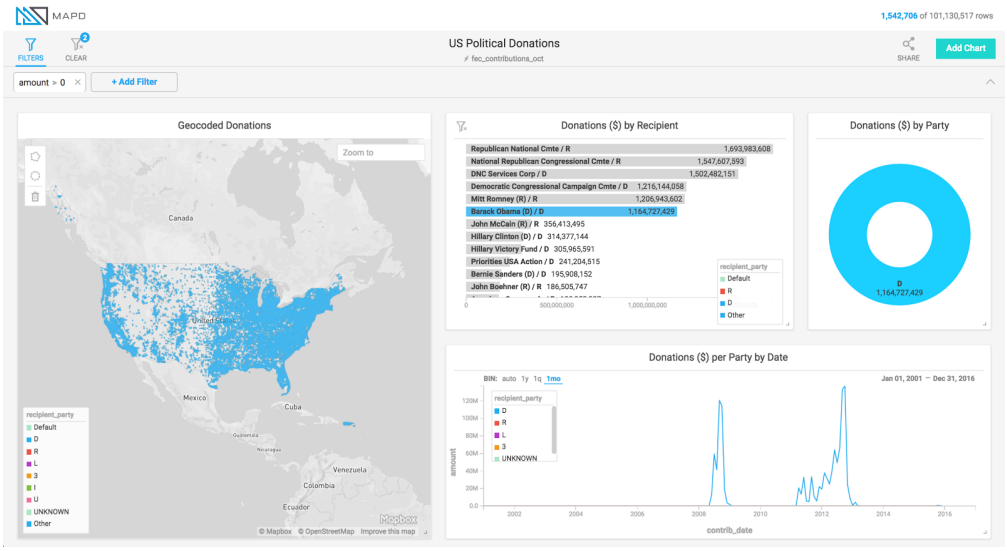


**Figure 3.2.:** *MapD Immerse [Dup17]*

### 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 in to 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

# 4. Benchmark

Benchmark

## 4.1. NYC Taxi Rides


**Figure 4.1.:** *Taxi Dropoffs*

## 4.2. Queries

The following list is about the queries

## 4.3. Results

The benchmark led to the following results.

# 5. Conclusion

# A. Architecture

# Appendices

# 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.

[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.

[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/.