Benchmarking big observational health data

Bas Katsma[1,2] and Jeremy Georges-Filteau[1,3]

[1]The Hyve
[2]Vrije Universiteit Amsterdam
[3]Radboud University Nijmegen

May 4, 2020

Background

My internship at The Hyve is focused on addressing the performance issues encountered with data analysis of big observational health data (OHD) and analytical type (OLAP) workloads (e.g. "give me all the patients that smoke and have high blood pressure readings") composed of healthcare relevant aggregation queries.

Current implementations use traditional relational database management systems (RDBMS) to store OHD. An example use case is tranSMART (Athey et al., 2013) which uses PostgreSQL (*PostgreSQL: The world's most advanced open source database*, n.d.) as the main DBMS. The data is normalized in these systems; instead of having repeated data in a flat database, separate database tables are created to store this data and a key links it back to the main table. In terms of physical storage, all attributes of a record are stored contiguously on the disk.

Under RDBMS there is both OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing). In general, OLTP workloads are characterized by vast numbers of simple transactional SQL processes (INSERT, DELETE, UPDATE). The main focus lies on quick processing of queries and managing data integrity. RDBMSs effectively perform OLTP-style processes in high throughput due to the row store architecture.

On the other hand, OLAP processes are defined by complex ad-hoc queries involving aggregations and a rather low number of transactions.

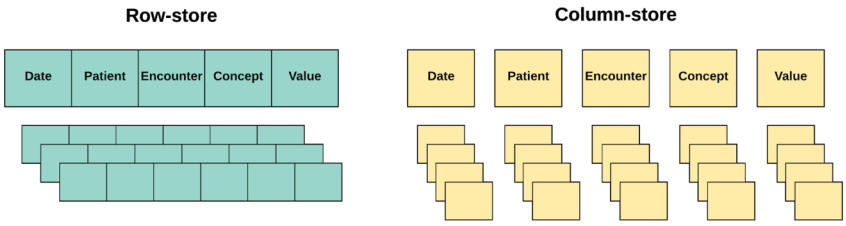

Figure 1: Differences in storage of database records between row-store and column-store DBMSs. Row-store systems contiguously store each complete record on the disk, whereas column values are grouped together in case of column-store systems.

Stonebraker et al. have performed pioneering work in the area of read-optimized DBMSs that store data by

column, also known as columnar or column-oriented DBMSs (Stonebraker et al., 2005). Figure 1 schematically depicts the differences in database storage between row-store and column-store systems.

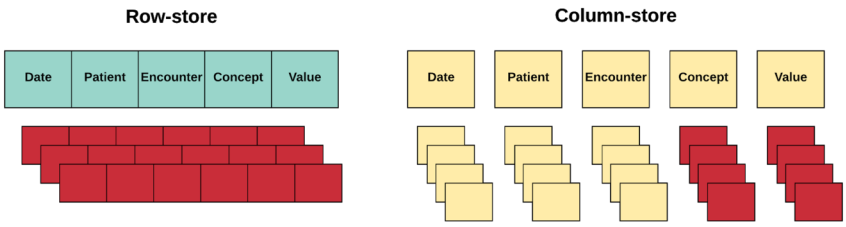

Figure 2: Differences in reading stored data between row-store and column-store DBMSs. An OLAP-type of workload is illustrated for which data from two attributes (Concept and Value) is required. Read data is colored in red.

OLAP workloads tend to answer specific business intelligence questions, such as the number of sales of product X over duration Y, which requires values from at most several columns to fulfill this request. Precise access to these needed columns can be granted instantly in column-store systems due to how all record attributes are stored, which increases query performance for these OLAP workloads since less data has to be read and processed. In contrast, row-store systems must retrieve all record attributes and dispose of irrelevant ones (Figure 2). This rationale has led to the question whether storing clinical data in columnar DBMSs could increase analytical workload performance.

Approach

For this benchmark, the openly accessible MIMIC-III v1.4 (Medical Information Mart for Intensive Care III) dataset is used (Johnson et al., 2016). This big dataset (˜33 GB) contains eleven years of observational health data resulting from approximately 60,000 intensive care unit admissions from over 40,000 patients and consists of bedside vital sign measurements (approximately 1 per hour), laboratory tests, medications, demographics, and multiple other variables.

The data is then conformed to a common data model: the i2b2 (*i2b2: Informatics for Integrating Biology & the Bedside*, n.d.) data mart model , which is based on the star schema structure (*I2B2 DATA MART - Server (Cells) Design - i2b2 Community Wiki*, n.d.). Here, a single central fact table is surrounded by dimension tables. The fact table solely stores "facts", which is a single observation on a specific patient in the field of healthcare. For example, the observation "110 bpm heart rate" or "diagnosis of diabetes." For each fact several attributes are stored, such as the IDs of the patient, provider and hospital admission, as well as the concept code for the specific observation and start and end dates.

Additional information related to the attributes of the fact table is stored in the dimension tables. For instance, the patient dimension table stores a patient's date of birth, sex, age, marital status, and other parameters.

The four different open-source database management systems (DBMS) that are being profiled in this benchmark are: **Apache Druid** (Yang et al., 2014), **ClickHouse** (*ClickHouse DBMS*, n.d.), **MonetDB** (Idreos et al., 2012), and **PostgreSQL** (*PostgreSQL: The world's most advanced open source database*, n.d.). These systems differ, amongst other things, in storage architecture and are explained more in-depth in the next section.

The benchmark consists of over 40 highly variable queries, each with different complexity and applicability. Each query will be repeated Q times, while the benchmark itself is run B times. Random parametrization for each b round ensures fairer performance testing. Several components will be profiled at query level during each benchmark:

- Query execution time;
- Current CPU usage (average of all cores);
- CPU load average (1, 5, and 15 minutes);
- Free/available memory;
- Buffered memory;
- Cached memory;

In addition, DBMS optimization will be analyzed by comparing benchmarks with default settings versus hardware and software optimized configurations. Downscaling (removing data) and upscaling (copy data) the MIMIC-III dataset allows for scalability analysis. Lastly, database stability testing is performed.

Databases

Apache Druid is a columnar data store written in Java designed to handle enormous amounts of event data with sub-second queries. In-house development of Druid started in 2011 by Metamarkets to operate their analytics product, after which it was open-sourced in 2012 and advanced under the Apache License in 2015. Since then, dozens of powerhouses like Alibaba, Airbnb, and Netflix incorporated Druid to run business intelligence in production (*Druid — Powered by Apache Druid*, n.d.). Druid's internal architecture is split up into several distinct processes:

- **Broker**, which handle queries;
- **Coordinator**, which handle availability of data;
- **Historical**, which store data ready for querying;
- **MiddleManager**, which handle ingestion of data;
- **Overlord**, which assign workloads w.r.t. data ingestion;
- **Router**, [optional] which forward requests to Brokers, Coordinators, and Overlords;

Each of those processes can be composed and deployed independently.

Druid uses so-called "datasources" to store the data, which are functionally equivalent to tables in relational DBMSs (RDBMS). In contrast to RDBMS, normalization — to eliminate data redundancy — is not supported in Druid: the datasources must be totally flat.

Datasources are always partitioned by time, which generates "chunks" spanning a certain time range (e.g. a single day of events if partitioning is set to "day"). Each chunk is additionally partitioned into "segments," single files that contain several million rows of data.

Another column-oriented DBMS of interest marketed for high performance real-time data analysis is **ClickHouse**, which is developed by Yandex and was open-sourced under the Apache License in 2016. This DBMS has been successfully used in production in massive analytical undertakings by Cloudflare to perform real-time analysis of DNS and HTTP traffic and CERN to handle and store metadata on billions of events of the LHCb experiment, among many others (*How Cloudflare analyzes 1M DNS queries per second*, n.d.; Vasile et al., 2019). To obtain the advertised blazing fast processing speeds ("[ClickHouse] processes hundreds of millions to more than a billion rows per second"), the following items play an important role:

- Columnar nature of stored data enables more "hot" data in RAM;
- High CPU performance by processing parts of columns at a single moment (vectorized query execution) using specialized CPU instructions;
- Data compression;

- Linear scalability by adding additional servers;

As with Druid, the data should fit in a single wide fact table. ClickHouse lacks database transactions as well as fully-developed UPDATE and DELETE implementations. Since all ClickHouse nodes are constructed equally, there is no single point of failure.

One of the earliest developed columnar DBMSs is **MonetDB**, in its current form initially created in 2002, and is globally used by research institutes and businesses, as well for educational purposes. The open-source DBMS claims "innovations in all layers of the DBMS," such as:

- CPU optimized query execution architecture;
- Automatic and self-tuning indexes;
- Run-time query optimization;
- Vertical fragmentation to store data;

The database architecture of MonetBD is represented in three layers; the first layer accommodates the SQL query interface and parses this into custom MonetDB Assembly Language (MAL) instructions. These instructions are sent to the next — middle — layer and optimized. The bottom layer connects to the data, which is stored in Binary Association Tables (BATs). Each data column in MonetDB is stored in a {`object-identifier, value`} table, creating a BAT.

In contrast to the previously mentioned DBMSs, **PostgreSQL** is a traditional DBMS; it stores data by row. With over 30 years of development, it has established itself as a reliable, performant, extensible, and feature robust general purpose open-source DBMS used by a great number of products and organizations to safely and robustly store data.

It is designed to be the single source of truth, which is accomplished by featuring transactions that have ACID properties (atomicity, consistency, isolation, duration).

Predictions

My prediction in general is that all columnar DBMSs outperform the traditional DBMS PostgreSQL in query execution time, solely due to the read-optimized nature of these column-oriented systems when performing OLAP-style queries.

Between columnar DBMSs, I expect ClickHouse and MonetDB — which are DBMSs written in low-level programming languages (i.e. C++ and C, respectively) — to outperform Apache Druid that is written in Java given the ability of these languages to interact closer to bare metal and having less overhead compared to higher-level programming languages.

References

tranSMART: an open source and community-driven informatics and data sharing platform for clinical and translational research. (2013). *AMIA Summits on Translational Science Proceedings*, *2013*, 6.

https://www.postgresql.org/. https://www.postgresql.org/

C-store: a column-oriented DBMS. (2005). *Proceedings of the 31st International Conference on Very Large Data Bases*, 553–564. https://dl.acm.org/doi/10.5555/1083592.1083658

MIMIC-III, a freely accessible critical care database.. (2016). *Sci Data*, *3*, 160035.

https://www.i2b2.org/. https://www.i2b2.org/

https://community.i2b2.org/wiki/display/ServerSideDesign/I2B2+DATA+MART. https://community.i2b2.org/wiki/display/ServerSideDesign/I2B2+DATA+MART

Druid. (2014). *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data - SIGMOD 14*. https://doi.org/10.1145/2588555.2595631

https://clickhouse.tech/. https://clickhouse.tech/

Monetdb: Two decades of research in column-oriented database. (2012). *IEEE Data Engineering Bulletin*.

https://druid.apache.org/druid-powered. https://druid.apache.org/druid-powered

https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second/. https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second/

Evaluating InfluxDB and ClickHouse database technologies for improvements of the ATLAS operational monitoring data archiving. (2019). ATL-COM-DAQ-2019-063.