Contrast Between Relational Database and Time Series Database on Handling Time Series Data

Zayed Uddin Chowdhury

***Abstract*— Various concepts of database have been changed from time to time depending on the data size, types, frequency, volume etc. In current era, the amount of data has rose exponentially with the increase of storage capacity and usage of all types of data. Most modern systems handle a lot of data within a very short time. As a result, time has become the key of storing data in many cases which made the scholars rethink about the structure of storing this type of data. This paper focuses on time series database (TSDB), their architecture and compares it with traditional relational database management systems (RDBMS) by experimenting on both type of database management systems.**

***Index Terms*— Database Management system, time series database, relational database**

I. INTRODUCTION

According to the classical definition, a time series is simply a sequence of numbers collected at regular intervals over a period of time. More generally, a time series is a sequence of data points (not necessarily numbers). And typically, time series consisting of successive measurements made over a time interval [9]. Time series databases are designed to optimize handling of time series data [5].

The increase in deployment of sensors for monitoring large industrial systems and the ability to analyze the collected data efficiently provide the means for automation and remote management to be utilized at an unprecedented scale [14]. For example, the sensors on a Boeing 787 produce upwards of half a terabyte of data per flight [12]. Large scale internet services(i.e. facebook, google etc) have to handle a huge amount of traffic every second. For example, Every 60 seconds on Facebook: 510,000 comments are posted, 293,000 statuses are updated, and 136,000 photos are uploaded [16]. This huge amount of data needs to be analyzed, stored and deliver when needed in a very fast and reliable manner [10]. For these data, time is not just a metrics but an axis.

However, traditional relational database systems do not scale well with large amounts of time series data, especially if, for example, one billion data points per day are stored, aggregated and queried. This is why there are so called time series databases, that are optimized to work with this form of data [3]. In addition, analysis of the collected time series often requires exporting the data to another application such as R or SPSS, as these provide additional capabilities and a simpler interface for time series analysis compared to an RDBMS, adding complexity to the analysis pipeline [13]. A review of recent technologies illustrates an emerging trend toward high performance, lightweight, databases specialized for time series data [8].

In the next section, this paper will go a bit deeper into the properties of time series database.

II. TIME SERIES DATABASE

Time Series Databases have key architectural design properties that make them very different from other databases. These include: time-stamp data storage and compression, data lifecycle management, data summarization, ability to handle large time series dependent scans of many records, and time series aware queries [5]. For example: With a Time Series Database, it is common to request a summary of data over a large time period. This requires going over a range of data points to perform some computation like a percentile increase this month of a metric over the same period in the last six months, summarized by month. This kind of workload is very difficult to optimize for with a distributed key value store. TSDBs are optimized for exactly this use case giving millisecond level query times over months of data. Another example: With Time Series Databases, its common to keep high precision data around for a short period of time. This data is aggregated and downsampled into longer term trend data. This means that for every data point that goes into the database, it will have to be deleted after its period of time is up. This kind of data lifecycle management is difficult for application developers to implement on top of regular databases. They must devise schemes for cheaply evicting large sets of data and constantly summarizing that data at scale. With a Time Series Database, this functionality is provided out of the box [5]. There are different algorithms used in time series databases. Some are discussed here:

A. Different Algorithms for Partial Data

- Random Sampling [2] samples the data stream at periodic intervals. In this approach, a reservoir is maintained from which a random sample can be generated. As the data stream flows, every new element has a certain probability of replacing an old element in the reservoir.
- Sliding window model [4] make decisions based only on recent data rather than running computations on all of the data seen so far, or on some sample (as the above-mentioned random sampling). So, any element for analysis, arrived at some time t will be declared as expired at time $t + w$, where w is the window size. In many practical use cases (e.g. sensing in IoT applications) only recent events may be important [9].
- Histogram approach partitions the data into a set of contiguous buckets [9].

- Titled frame model uses different granularities for time frames.The most recent time is registered at the finest granularity. The most distant time is registered at a coarser granularity [9].

B. Lambda Architecture

The algorithms described above works only on some recent data. But, the full log of data is also needed. For this reason, lambda architecture is used. It is illustrated in figure 1. On this picture the data source is constantly being updated

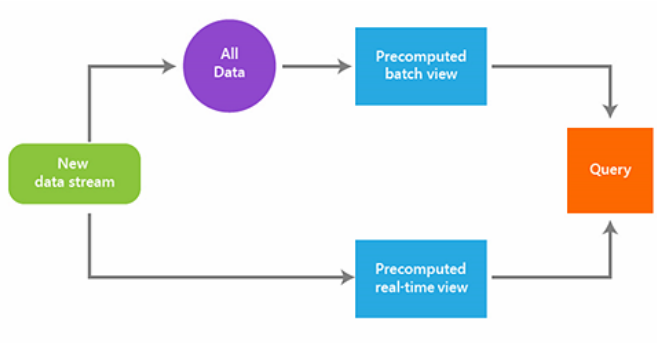

Fig. 1. Lambda Architecture [6]

with data. Most of the data is processed in real time. It is specially needed for Internet of Things and M2M systems, where time series processing is the main source behind control actions and conclusions(alerts). In the same time, some processing without the strong limitations for time-to decision can be done. Database could save processed data for queries from users and applications [9]. In the next section, some recent proven time series database management systems will be discussed.

III. Proposed DBMS

In this section, the paper discusses mainly on two TSDB. One is TimescaleDB [1] and Gorilla [11].

A. Gorilla

Gorilla is a fast, scalable, in-memory time series database developed by Facebook. It is an opensource in-memory TSDB that can store tens of millions of datapoints (e.g., CPU load, error rate, latency etc.) every second and respond queries over this data within milliseconds. The strength of Gorilla are these:

- Time series compression
  - Compressing time stamps: According to Facebook, about 96 percent of all time stamps can be compressed to a single bit using Gorilla [11].
  - Compressing values: A two-hour block allows Gorilla to achieve a compression ratio of 1.37 bytes per data point. 1.37 bytes per data point.
- In-memory data structures
- On disk structures
- Handling failures

However, the features in this TSDB is mainly based on Facebook's need. So, it is not proven for general time series

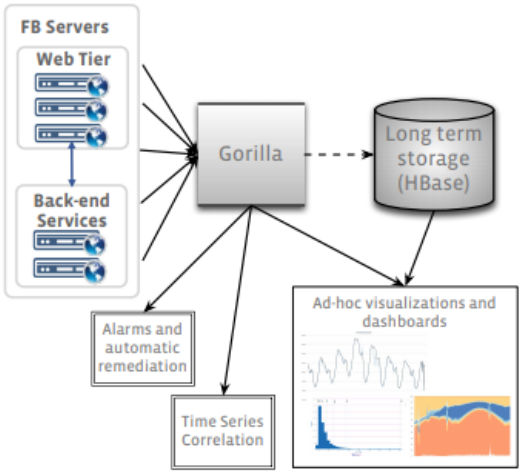

Fig. 2. High level overview of the ODS monitoring and alerting system using Gorilla [11]

data. It is also available only for Linux currently. This takes us to the next option in the next section.

B. TimescaleDB

TimescaleDB is a time series database engineered up from PostgreSQL (packaged as an extension) and yet scales out horizontally, which means it supports normal SQL and all of the features you expect from a relational database: JOINs, secondary indexes, complex predicates and aggregates, window functions, CTEs, etc [7]. TimescaleDB stores data as

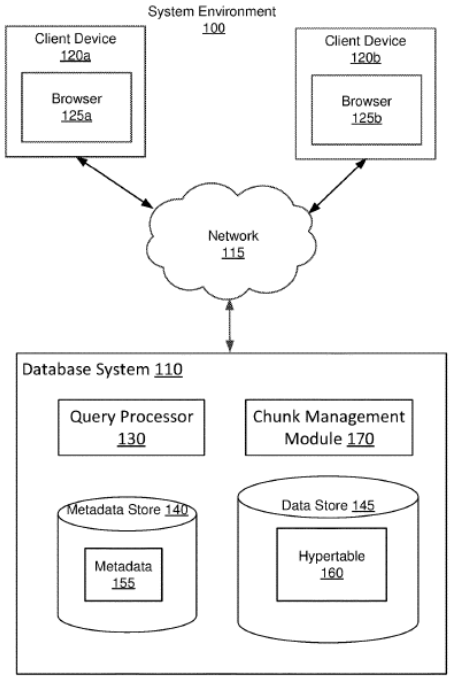

Fig. 3. High level overview of TimescaleDB Architecture [1]

hypertables that represent partitioned database tables as seen in figure 3. Each hypertable comprises chunks of data that

may be distributed across multiple locations , each location comprising at least a storage device . The database system provides an interface that allows data base queries seamlessly to hypertables as well as standard tables . The database system dynamically creates chunks as records are added to a hypertable [1]. TimescaleDB is available in different platform (i.e. windows, mac, linux). We will test this out with traditional RDBMS like SQL Server and Oracle in the next section.

IV. Experiment and comparison

In this section, a practical comparison between TimescaleDB, SQL Server 2017 and Oracle 13 is carried out. As this paper is focused on time series data, a sample dataset of NYC Taxi and Limousine Commission [15] is used. For saving time and space we have grabbed only January 2016 data. The dataset contains **10906858** rows.

A. Insert

We have inserted all rows from the CSV file to 3 of the databases. Table I shows the execution time. From Table I,

TABLE I
INSERT TIME

TimescaleDB	SQL Server	Oracle
20 mins	$>$ 2 hours	$>$ 2 hours

we can see that TimescaleDB is far ahead from the other RDBMS.

B. Select

We have carried out some queries here. The main criteria of choosing the queries is time dependency. We will see how they perform when they are sorted by time and when they are not. Only the timescaledb version of the query is written in the tables, but the syntax were changed according to the database system we used while testing.

1) Query 1: "SELECT Distinct rate_code from rides"

TABLE II
QUERY 1

TimescaleDB	SQL Server	Oracle
13.8 sec	$<$1 sec	$<$1 sec

In Table II we can see that the RDBMS performs better than the time series database. Now, we will modify this query to have some time restriction.

2) Query 2: "SELECT Distinct rate_code, pickup_datetime from rides order by pickup_datetime"

TABLE III
QUERY 2

TimescaleDB	SQL Server	Oracle
7.14 sec	18 sec	19 sec

In Table III, time series database performs better than the RDBMS ones.

TABLE IV
QUERY 3

TSDB	SQL Server	Oracle
3.28 sec	2 sec	2 sec

3) Query 3: "SELECT * FROM rides WHERE (dropoff_datetime-pickup_datetime) $>$ '1 hour' LIMIT 1000"

In Table IV, RDBMS performs better than TSDB. The query does not sort the output according to time.

4) Query 4: "SELECT * FROM rides ORDER BY pickup_datetime desc LIMIT 1000"

TABLE V
QUERY 4

TSDB	SQL Server	Oracle
2.2 sec	104 sec	60 sec

In Table V, again we can see that as soon as the output is ordered according to time TSDB outperforms RDBMS.

V. CONCLUSIONS

Different TSDB and RDBMS use different techniques for storing and retrieving data. But for the database systems used in this paper we can conclude that TSDB outperforms top notch RDBMS when it comes to manipulating time series data. But, other than that in most of the cases, RDBMS performs better. So, which type of database should be used completely depends on the type of data we are handling.

References

[1] Matvey Arye, Michael J Freedman, Robert Kiefer, Ajay A Kulkarni, Erik Nordström, and Olof Rensfelt. Scalable database system for querying time-series data, August 30 2018. US Patent App. 15/907,105.

[2] Chris Chatfield. *The analysis of time series: an introduction*. CRC press, 2016.

[3] Julian Ganz, Matthias Beyer, and Christian Plotzky. Time-series based solution using influxdb.

[4] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[5] Influxdata. Time series database (tsdb) explained. [Online; Available: https://www.influxdata.com/time-series-database/].

[6] Daniel Jebaraj. Lambda architecture: Design simpler, resilient, maintainable and scalable big data solutions. [Online; Available: https://www.infoq.com/articles/lambda-architecture-scalable-big-data-solutions].

[7] Ajay Kulkarni. When boring is awesome: Building a scalable time-series database on postgresql. [Online; Available: https://blog.timescale.com/when-boring-is-awesome-building-a-scalable-time-series-database-on-postgresql-2900ea453ee2].

[8] Benjamin Leighton, Simon JD Cox, Nicholas J Car, Matthew P Stenson, Jamie Vleeshouwer, and Jonathan Hodge. A best of both worlds approach to complex, efficient, time series data delivery. In *International Symposium on Environmental Software Systems*, pages 371–379. Springer, 2015.

[9] Dmitry Namiot. Time series databases. In *DAMDID/RCDL*, pages 132–137, 2015.

[10] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[11] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[12] Jussi Ronkainen and Antti Iivari. Designing a data management pipeline for pervasive sensor communication systems. *Procedia Computer Science*, 56:183–188, 2015.

[13] Ilari Shafer, Raja R Sambasivan, Anthony Rowe, and Gregory R Ganger. Specialized storage for big numeric time series. *HotStorage*, 13:1–5, 2013.

[14] Abhishek B Sharma, Franjo Ivančić, Alexandru Niculescu-Mizil, Haifeng Chen, and Guofei Jiang. Modeling and analytics for cyber-physical systems in the age of big data. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):74–77, 2014.

[15] NYC Taxi and Limousine Commission. Tlc trip record data. [Online; Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml].

[16] zephoria. The top 20 valuable facebook statistics updated november 2018, November 2018. [Online; Available: https://zephoria.com/top-15-valuable-facebook-statistics/].