**Paper name: Gorilla:** A Fast, Scalable, In-Memory Time Series Database.

**Authors**: Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, Kaushik Veeraraghavan.

**Problem statement**:

Facebook maintains one of the largest databases. In 2013, Facebook’s time series database monitoring system was not scaling sufficiently. Thus, Facebook developed Gorilla, an in-memory Time Series Database (TSDB) for storing massive real time data.

**Hypothesis**:

An in-memory TSDB that functions as a write through cache for monitoring data that prioritizes reads/writes over availability of older data is scalable for monitoring massive time series data.

**Specific aims**:

Gorilla aimed to store 2 billion unique time series records identified by a string key. It has to add 700 million data points per minute. It stored data for 26 hours. It needs to execute more than 40,000 queries per second at peak. It has to read data within one millisecond. It needs to support time series with 15 second granularity. It has to read even when a single server crashes. It needs to have the ability to quickly scan over all in memory data. It has to support at least 2x growth per year.

**Domain importance**:

Large-scale internet service providers like Facebook are storing massive quantity of time series data. They have scaled beyond a few systems running on hundreds of machines to thousands of individual systems running on many thousands of machines across different continents.

These large-scale services need to accurately monitor the health and performance of the underlying system and quickly identify and diagnose problems as they arise. These systems need to have observability over many different systems. They need to identify issues with new software releases. They also need to be fault tolerant with multi-region architecture and operable with network failures.

**Data importance and format**:

Timeseries Map (TSmap) is the primary data structure in Gorilla. TSmap includes a vector of C++ shared-pointers to time series and a case-insensitive map from time series names.

**Data example**:

Gorilla stores and monitoring server/system parameters at Facebook. Each data point is a pair of 64-bit values representing the time stamp and value at that time. Timestamps and values are compressed separately using information about previous values. The later time stamps are compressed as the time gap is usually the time (one point every minute). A block contains data of every two hours.

**Validation/success criterion**:

The success criterion is related to the commercial need of the database – it needs to store massive time series record, it has to add millions of data points per minute, it needs to execute thousands of queries per second, it has to read data within milliseconds. Additionally, it needs to be robust against server crashes. Finally, it has to support at least 2x growth per year.

**Methodology**:

Compressed in-memory storage: Facebook generates million of time series on small individual data points. This is compressed using a delta-of-delta technique. Subsequent time stamps are expressed as compressed difference of time stamps.

Storage model: Gorilla is shared system that assigns different servers a specific subset of time series. Thus, more data can be managed by increasing the number of serves.

Query model: Gorilla uses a simple query model – it allows retrieval of time series unique name and time range.

**Key result**:

After the first 6 months of usage, Gorilla showed positive results.

Fault tolerance: it experiences 3 unplanned network cuts – and detected all of them. It also shifted major load to different region after a major bug was discovered.

Site wide error rate debugging: Gorilla detected a site-wide error rate increase within minutes.

**Key conclusion**:

Gorilla achieves very good compression of time series data. It has fast retrieval time with high scalability. However, all of its achievements are due to specialization for a specific application – monitoring server/system parameters at Facebook. Gorilla’s data is also single-dimensional and specialized. Its high compression is the result of low time resolution and infrequent update for monitoring. Its simple query model is designed for fast retrieval. Thus, it can’t be thought of as an alternative for all other time series data.

**References:**

* https://www.vldb.org/pvldb/vol8/p1816-teller.pdf

**Paper name**: Neural Databases

Authors: James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, Alon Halevy.

**Problem statement**:

The main goal of NeuralDB is to support applications where users do not need to pre-define a schema. Instead, they can use natural language to express facts and generate queries. This has a lot of use cases in personal assistance, natural search answering etc.

**Hypothesis**: NeuralDB can use transformer language model to generate database and output query results from natural language input.

**Specific aims**:

Applying neural nets to store data and answer query without having predefined schema. Update the underlying data representation based on data update. Make use of the knowledge embedded in large-scale pre-trained transformer language model.

**Domain importance**:

Neural networks have achieved significant progress in tasks such as speech recognition, natural language understanding, and computer vision. Based on these, researchers have experimented with the application of neural nets to data management problems like query optimization and entity matching [1, 2]. In applying neural nets to data management, research has so far assumed that the data was modeled by a database schema. This paper proposes NeuralDB, that requires no pre-defined schema. Thus, the database scope does not need to be defined beforehand, and any relevant data can be stored and queried as needed. Queries and updates can be posed in natural language forms convenient for non-technical users.

**Data importance and format**:

There is no existing data sets that are directly applicable for the new NeuralDBs. The paper developed a new database for testing purpose. The training, validation and test set contains 535, 50 and 50 databases, each having 50 facts. Each database has 100-200 question and answer pairs yielding total of 60000 training, 5500 validation and 6000 test sets.

**Data example**:

Training a NeuralDB requires supervision in the form of (𝐷, 𝑄, 𝐴) triples, where 𝐷 is a set of facts, 𝑄 is a query and 𝐴 is the correct answer to 𝑄 over 𝐷. The training data was generated from Wikidata [48] to express facts in natural language.

The input data can be in the form of text like: ‘Sarah lives in London’. If subsequent text is ‘Sarah now lives in Manchester’, then asking the model for ‘Where does Sarah live now’ will generate answer of ‘Manchester’. The underlying data is stored in neural network.

**Validation/success criterion**:

Answer accuracy evaluation: the answers generated by NeuralDB was compared against reference data that contained correct answers – following Exact Match method (1 for correct and 0 for wrong answer). The available data was segmented into training and test sets.

**Methodology**:

NeuralDB uses an underlying large-language transformer model as a data source. It also took Wikidata for training and testing. It uses Neural SPJ for answering queries. Neural SPJ is capable of aggregation for answering questions. It support set generation, supervision and transfer learning.

**Key result**:

The first contribution of this paper is to show that state of the art transformer models [3] can be adapted to answer simple natural language queries. Specifically, the models can generate answers combining multiple facts – just like performing joins. However, the model do not perform well on aggregation queries like counting. They also replicate the biases of the underlying language model.

The second contribution of the paper is to propose an architecture for neural databases that uses the power of transformers at its core.

Finally, the paper describes an experimental study that validates the different components - Neural SPJ to answer queries.

**Key conclusion**:

NeuralDB is able to answer queries from natural sentences that do not depend on a pre-defined schema. The experiment shows that it is possible to have accurate query result that involve select, project, join, aggregation. However, it is also prone to bias in the base language model.

**Reference**:

1. Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. CoRR, abs/2004.00584, 2020.
2. Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, pages 19–34. ACM, 2018
3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lilon Jones, Aidan Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.