Paper Sharing:
Tensor Data Platform & Deep Lake

Rui LIN

12 July, 2023

1 Basic Concepts

1. **Relational Database Engine**

   it is also known as the Query Processor, and is responsible for parsing, which is the activity in which the user input (high-level language) is converted to a machine-understandable code (low-level language).

   Relational databases are tools for storing various types of information that are related to each other in some way. Data engineers build and design relational databases (and other data management systems) to assist organizations in collecting, storing, and analyzing data.

2. **DBMS**

   A Database Management System (DBMS) is software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs that manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software store and retrieve data. DBMS allows users to create their own databases as per their requirements.

3. **Columnar Database**

   A columnar database is a database management system (DBMS) that stores data in columns rather than in rows as relational DBMSs do. The main differences between a columnar database and a traditional row-oriented database are centered around performance, storage necessities and schema modifying techniques.

4. **Metadata**

   Metadata is data that describes other data. In the context of a relational database, metadata is the data that describes the structure of the database itself. It provides information about the tables, columns, and relationships

between them. Metadata can also include information about the database schema, such as constraints and indexes.

Metadata can be stored in various formats such as string, integer, date/time, etc. depending on the type of data it describes. For example, the data type for a column name would typically be a string, while the data type for a date/time constraint would be a date/time value.

5. **Physical Plan**

   A physical plan is a sequence of operations that are executed by a database management system (DBMS) to execute a query. It describes how the query will be executed, including which tables will be accessed, which indexes will be used, and how the data will be sorted and joined.

6. **Execution Plan**

   The main difference between an execution plan and a physical plan is that an execution plan is generated at runtime by the DBMS's query optimizer, while a physical plan is generated at design time by the database developer.

7. **UDFs**

   User-defined functions (UDFs) are functions that are defined by users in a database management system (DBMS). They allow users to extend the functionality of the DBMS by creating their own functions that can be used in queries. UDFs can be used to perform complex calculations, manipulate data, or perform other operations that are not supported by the built-in functions of the DBMS.

8. **TVFs**

   Table-valued functions (TVFs) are user-defined functions that return a table as their result set. They can be used to encapsulate complex queries and calculations that involve multiple tables or views. TVFs can be used in the FROM clause of a SELECT statement just like a table or view.

9. **LLP**

   LLP is the abbreviation of Learning from Label Proportion, which is a weakly supervised classification problem. In LLP, data points are grouped into bags, and the label proportions within each bag are observed instead of the instance-level labels. For example, in a bag of 100 images, 80% of them are labeled as "cat" and 20% as "dog". The task is to learn a classifier to predict the individual labels of future individual instances.

10. **Differential Privacy**

    A mathematical framework that defines and provides privacy guarantees for algorithms that access data

11. **ACID Transactions**

    A transaction is any operation that is treated as a single unit of work, which either completes fully or does not complete at all, an leaves the storage system in a consistent state. The classic example of a transaction is what occurs when you withdraw money from your bank account. Either the money has left your bank account, or it has not — there cannot be an in-between state.

    A - Atomicity: each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed. This property prevents data loss and corruption from occurring if, for example, if your streaming data source fails mid-stream.

    C - Consistency: ensures that transactions only make changes to tables in predefined, predictable ways. Transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table.

    I - Isolation: when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously.

    D - Durability: ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

12. **Data Silos**

    A data silo is a collection of data held by one group that is not easily or fully accessible by other groups in the same organization. Finance, administration, HR, marketing teams, and other departments need different information to do their work.

13. **Petabytes**

    A petabyte is a multiple of a byte, which is the unit of storage size for digital information. Since peta indicates multiplication by the fifth power of 1,000, a petabyte is equal to one quadrillion (1,000,000,000,000,000) bytes, 1,000,000 GBs, or 1,000 TBs. Nowadays, it is often used when labeling network hard drives, total server farm capacity, or storage media with large capacity.

14. **First-Generation Data Lake**

    1. HDFS
    2. AWS S3

    Drawbacks: traditionally collected data into distributed storage systems, and the unorganized collections of the data turned data lakes into "data swamps".

15. **Second-Generation Data Lake**
    1. Delta
    2. Iceberg
    3. Hudi

    They strictly operate on top of standardized structured formats[1] such as:
    1. Parquet (Apache Parquet is an open-source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. Parquet is available in multiple languages, including Java, C++, Python, etc...)
    2. ORC
    3. Avro,

    and provide features like time travel, ACID transactions, and schema evolution.

16. **Query Engine**
    1. Presto: Presto is an open-source SQL query engine that's fast, reliable, and efficient at scale. Use Presto to run interactive/ad hoc queries at sub-second performance for your high-volume apps.
    2. Athena: Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon S3 using standard SQL. With a few clicks in the Amazon Web Services Management Console, customers can point Athena at their data stored in S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.
    3. Hive: Apache Hive is a data warehouse system built on top of Apache Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in various databases and file systems that integrate with Hadoop. Hive offers a simple way to apply structure to large amounts of unstructured data and then perform batch SQL-like queries on that data.
    4. Photon: Photon is the next generation engine on the Databricks Lakehouse Platform that provides extremely fast query performance at low cost – from data ingestion, ETL, streaming, data science and interactive queries – directly on your data lake.

17. **Frameworks**
    1. Hadoop:
    2. Spark:
    3. Airflow:

18. **Data Warehouse**
    1. Snowflake:
    2. BigQuery:
    3. Redshift:

[1] `https://www.upsolver.com/blog/the-file-format-fundamentals-of-big-data`

4. Clickhouse:

19. **Chunks**
    A chunk is the largest unit of physical disk dedicated to database server data storage.

20. **Scan Queries**
    Scan Queries is a separate data access interface designed primarily for running analytical ad hoc queries against a DB.

21. **Distributed Storage**
    Distributed storage is a software-defined storage system that enables access to data. It includes hardware and software enabling a scale-out distributed file system technology targeted at unstructured data growth. Data can be split across various physical servers and multiple data centers. It generally utilizes a cluster of storage units with synchronization and coordination methods between nodes. Storing data in multiple computers or in computers that are geographically dispersed is a form of distributed storage.

22. **Data Ingestion**
    Data ingestion is the process of moving data from various sources to a destination where it can be accessed, used, and analyzed. The destination can be a cloud data lake, cloud data warehouse, database, data mart, or a document store. Data ingestion may or may not involve transformation or manipulation of data during the process.

2 Tensor Data Platform

Learning notes of Ref. [1].

2.1 Summary

1. **Why**

   1) **Problem to solve**: Integrate relational and ML workloads. SQL can be used as a higher-level abstraction or orchestrator between data operations and machine learning transforms.

   2) **Previous solutions**:

   - try to optimize the hand-off of data between separate ML and DB systems. → reduce the conversion time

   - integrate ML as a UDF through an external specialized system, to express ML algorithms directly in SQL

   3) **Bottlenecks**:

   - ML is merely a guest in the relational house owned by the DBMS

   - the existing methods poorly suited to handle non-relational data

   - the existing methods miss out on the virtuous cycle among HW vendors/OSS/ML academics/app developers that TCR engines enjoy

2. **What**

   1) **Proposed Methods:** Tensor Data Platform (TDP), a database built upon a tensor runtime, namely, PyTorch.

   2) **Advantages:**

   - TDP leverages PyTorch to run queries over structured and unstructured data on a wide range of hardware devices.

   - TDP integrates the flexibility of PyTorch's programming model with the declarative power of SQL.

   - TDP leads to a hybrid ML-SQL experience that is appealing to database users without forcing data scientists outside of their comfort zone.

   - PyTorch in TDP allows trainable queries.

   3) **Performance:**

   Available functions:

- ML within SQL: UDF-based programming model. Use UDFs/TVFs to parse unstructured data into a structured representation (cf. Example 3.1, use UDF to get the Digit & Size tables).

- SQL within ML: Embedding queries in PyTorch programs.

- Trainable query (cf. Example 3.2).

- Multi-modal queries (cf. Figure 2, search for the images and the information in the receipts).

- SQL queries over OCRed documents (cf. Listing 8, extract the information in the image and then use SQL to do query).

- Learning from Label Proportions (LLP).

- Learning to answer queries over images.

Compare with others:

- Faster on GPU (cf. Figure 2 (right))

- No conversion time (cf. Figure 3 (left))

- More efficient training (cf. Figure 3 (right))

- Better generalization

3. **How**

- **Storage:** store relational data in a columnar format. TDP accepts input data in different formats. When data is registered into TDP, it is first transformed into a tensor and subsequently encoded. Data can be stored both on the CPU and GPU.

- **Data Encoding:** TDP does not use PyTorch tensor directly, but rather provides its own encoded tensor abstraction.

- **Query Processor:**

1) Sparks/Substrait can generate the physical plan,

2) TDP can compile each physical operator in the physical plan to PyTorch models

3) TDP has a mapping dictionary (maybe physical operator → PyTorch APIs)

4) For a physical operator, there may exist several mappings

4. **Limitations**

1) It is still heuristics to pick which PyTorch implementation is used to deploy each physical operator.

2) The supported ML algorithms are still limited in TDP.

3 Deep Lake

Learning notes of Ref. [2].

3.1 Summary

1. **Why**

   1) **Problem to solve:** Design a lakehouse that can support large-scale, complex, and unstructured datasets (e.g., CoCo (330K images), ImageNet (1.2M images), Oscar (multilingual text corpus)) used for deep learning workflows.

   2) **Previous solutions and bottlenecks:**

   Complex Data Types in a Database

   - Databases are not optimized for storing and serving large files and can cause performance issues.
   - Binary data does not fit well into a database's structured format, making it difficult to query and manipulate. Storing large amounts of binary data in a database can be more costly than other storage solutions.

   Complex Data Along with Tabular Formats

   - Tabular formats extended for deep learning, such as Petastorm or Feather, have yet to gain wide adoption.

   Object Storage for Deep Learning

   - Current cloud-native choices for storing large unstructured datasets have four drawbacks:
     a) they introduce significant latency overhead,
     b) unstructured data ingestion without metadata control can produce "data swamps",
     c) object storage has built-in version control, which is rarely used in data science workflow, and
     d) data on object storage gets copied to a virtual machine before training, thus resulting in storage overhead and additional costs.

   Second Generation of Data Lakes

   - The second-generation data lakes are still bound by the limitations of the inherent data formats to be used in deep learning.

2. **What**

   1) **The proposed method:** Deep Lake is designed to help deep learning workflows run as seamlessly as analytical workflows run on Modern Data

Stack (MDS).

2) **Advantages:**

- Fast data ingestion speed → Ingesting 10000 images from FFHQ dataset from different format, Deep Lake has the best performance.
- Fast iteration speed of images against other dataloaders.
- More efficient streamable dataloader from different locations (e.g., Local, AWS S3, MinIO).
- Faster training on the cloud.
- Allow distributed training of a large multi-modal dataset with high GPU utilization.
- Version control.
- Visualization of tensors.
- Tensor Query Language.

3. **How**

Deep Lake extends the second generation of data lake capabilities for deep learning use cases by rethinking the format and upstream features, including querying, visualization, and native integration, to keep learning frameworks to complete the ML lifecycle.

Tensor Storage Format:

(a) Deep Lake datasets follow columnar storage architecture, with tensors as columns.

(b) Tensors can contain dynamically shaped arrays, also called ragged tensors.

(c) Htype (e.g., image, video, audio, bbox, dicom, etc.) defines the expectations on samples in a tensor such as data type, shape, number of dimensions, or compression.

(d) Deep Lakes chunks are constructed based on the lower and upper bound of the chunk size to fit a limited number of samples.

(e) Deep Lake implements an on-the-fly re-chunking algorithm to optimize the data layout. One of the key access patterns of it is shuffled stream access for training machine learning models.

(f) Deep Lake can be plugged into any storage provider.

4. **Limitations**

(a) The storage format does not support custom ordering for an even more efficient storage layout required for vector search or key-value indexing.

(b) Deep Lake implements branch-based locks for concurrent access. Similar to Delta ACID transaction model, Deep Lake can be extended to highly-performant parallel workload.

(c) The current implementation of TQL only supports a subset of SQL operations.

References

[1] A. Gandhi, Y. Asada, V. Fu, A. Gemawat, L. Zhang, R. Sen, C. Curino, J. Camacho-Rodríguez, and M. Interlandi, "The tensor data platform: Towards an ai-centric database system," 2022.

[2] S. Hambardzumyan, A. Tuli, L. Ghukasyan, F. Rahman, H. Topchyan, D. Isayan, M. McQuade, M. Harutyunyan, T. Hakobyan, I. Stranic, and D. Buniatyan, "Deep lake: a lakehouse for deep learning," 2022.