Lecture #01: Modern Analytical Database Systems

15-721 Advanced Database Systems (Spring 2024)
`https://15721.courses.cs.cmu.edu/spring2024/`
Carnegie Mellon University
Prof. Andy Pavlo

1 Background

On-Line Analytical Processing (OLAP) systems to extract new information from existing data sets. Historically these workloads were run in a monolithic DBMS that had all of an organization's data in centralized managed storage.

1.1 1990s - Data Cubes

DBMSs would maintain multi-dimensional arrays as pre-computed aggregations to speed up queries, often augmented onto pre-existing operational DBMSs designed to operate on row-oriented data. These materialized views would be periodically refreshed and the cubes had to be specified by system administrators ahead of time.

1.2 2000s - Data Warehouses

Data warehouses are monolithic DBMSs built for the purpose of efficiently executing OLAP workloads. They are shared-nothing architectures with column-oriented data. Many of these systems started as forks of Postgres with DBMS-managed storage using proprietary data encoding / formats. They processed data from operational OLTP databases through ETL (Extract Transform Load) data integration processes.

1.3 2010s - Shared Disk Engines

Shared-disk DBMS architectures relied on third-party distributed storage (object stores) instead of using custom storage managers. The first generation of these systems managed data files themselves, but newer systems allow external entities to add data files to the storage without enforcing a schema.

1.4 2020s - Lakehouse Systems

[2] Lakehouse systems are middlware for data lakes that add support for better schema control / version with transactional CRUD (create, read, update, delete) operations. Changes stored in row-oriented log-structured files indices which are periodically compacted into read-only columnar files. These Lakehouse systems make use of three observations.

- People want to execute more than just SQL queries on data, often more procedural code.
- Decoupling data store from DBMS reduces ingest/egress barriers.
- Most data is unstructured / semi-structured (images, video).

2 OLAP DBMS Architectures

The architecture of an OLAP DBMS contains many design choices that make each individual DBMS unique. However, some design choices and architecture are common / shared among many systems.

2.1 OLAP DBMS Components

[1] OLAP DBMS components often made into standalone services and libraries. There are many engineering challenges to make components interoperable + performant. These include system catalogs, intermediate representations, query optimizers, file formats, access libraries, and execution engines.

2.2 Architecture Overview

- **Front-end** takes in the user query and parses it into a intermediate representation using the language parser.
- **Planner** takes in the intermediate representation from the front-end and uses the binder, rewriter, and optimizer and cost model to generate a query plan.
- **Scheduler** take in the query plan, organizes workers nodes, and schedules execution by breaking up the plan into plan fragments.
- **Execution Engine** takes in plan fragments and executes them.
- **I/O Service** takes in block requests from the execution engine and returns the requested data after retrieving it from the object store.
- **Catalog** keeps track of data locations and metadata for the DBMS and communicates with all components of the DBMS except the front-end.

2.3 Distributed Query Execution

Executing an OLAP query on a distributed DBMS system is roughly the same as on a single-node DBMS. A query plan can be thought of as a directed acyclic graph of physical operators, where for each operator the DBMS is considering where the input data is coming from and where it is sending the output data to. This data is split into two categories.

Persistent Data is the source of record for the database (e.g. tables). Modern systems assume these data files are immutable but can update them by rewriting them.

Intermediate Data consists of short-lived artifacts produced by query operators during execution to be consumed by other query operators. There is little to no correlation between the amount of intermediate data and the amount of persistent data read or the execution time.

2.4 Distributed System Architecture

A distributed DBMSs architecture specifies the location of the database's persistent data files. This affects how different nodes coordinate with each other and where they retrieve / store objects in the database. There are two approaches to connecting the query with the data.

Push query to data is when the query (or a portion the query) is sent to the node that contains the data. As much filtering and processing as possible is done where the data resides before transmitting the result over the network.

Pull data to query consists of bringing the data to a node that is executing a query that needs the data for processing. This method is necessary when there are no compute resources at the location where persistent data is stored.

2.5 System Architecture

There are two common system architectures, Shared-Nothing and Shared-Disk. Shared-Disk is the system of focus and modern importance.

Shared-Nothing requires that each DBMS instance has its own CPU, memory, and locally-attached disk. These nodes then communicate via network, with the database partitioned into disjoint subsets split across the nodes (this means adding a new node requires physically moving data between nodes). Since the data is local, the DBMS can access it via POSIX API. This architecture is harder to scale due to data movement, but is potentially more efficient and performant. It also allows for push query to data as each persistent memory has compute resources.

Shared-Disk has each node connect to a single logical disk via interconnect, but may also have their own private memory and ephemeral storage. Nodes must still communicate to learn about each others internal state. Instead of a POSIX API, the DBMS accesses data through a userspace API. There is more flexibility as the compute layer and storage layer can scale independently, it is easy to shutdown idle compute layer resources. However, it may be required that data is pulled from storage to compute before applying filters (though not necessarily).

2.6 Shared-Disk Implementation

Traditionally, the storage layer in Shared-Disk DBMSs were located on-prem NAS (e.g. Oracle Exadata). Cloud **object stores** are now the prevailing storage target for modern OLAP DBMSs because of near-infinite scalability (e.g. Amazon S3, Azure Blob, Google Cloud Storage).

3 Object Stores

To store a DBMSs contents in an object store, partition the tables into large immutable files to be stored in the object store. All attributes for a tuple are stored in the same file in a columnar layout (PAX). The header or footer should contain the metadata about columnar offsets, compression schemes, indices, and zone maps. The DBMS retrieves the header / footer first to know what byte range it needs to retrieve the data. Each cloud vendor implements their own proprietary API to access data (GET, PUT, DELETE).

References

[1] P. Pedreira, O. Erling, K. Karanasos, S. Schneider, W. McKinney, S. R. Valluri, M. Zait, and J. Nadeau. The composable data management system manifesto. *Proc. VLDB Endow.*, 16(10):2679–2685, jun 2023. doi: 10.14778/3603581.3603604.

[2] M. A. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *Conference on Innovative Data Systems Research*, 2021.