Modern OLAP Databases

Carnegie Mellon University
ADVANCED DATABASE SYSTEMS

Lecture #02

Andy Pavlo // 15-721 // Spring 2023
COURSE OUTLINE

Storage
- Columnar Storage
- Compression
- Indexes

Query Execution:
- Processing Models
- Scheduling
- Vectorization
- Compilation
- Joins
- Materialized Views

Query Optimization

Network Interfaces
TODAY’S AGENDA

Query Execution
Distributed System Architectures
OLAP Commoditization
DISTRIBUTED QUERY EXECUTION

Executing an OLAP query in a distributed DBMS is roughly the same as on a single-node DBMS. → Query plan is a DAG of physical operators.

For each operator, the DBMS considers where input is coming from and where to send output. → Table Scans → Joins → Aggregations → Sorting
DISTRIBUTED QUERY EXECUTION

Persistent Data

Worker Nodes

Intermediate Data

Persistent Data
DISTRIBUTED QUERY EXECUTION

Persistent Data

Worker Nodes

Intermediate Data

Persistent Data

Shuffle Nodes (Optional)
DISTRIBUTED QUERY EXECUTION

Persistent Data

Worker Nodes

Intermediate Data

Shuffle Nodes (Optional)

Worker Nodes

Persistent Data
DISTRIBUTED QUERY EXECUTION

Persistent Data

Worker Nodes

Intermediate Data

Shuffle Nodes (Optional)

Worker Nodes

Final Result

Persistent Data

Intermediate Data

Worker Nodes
DATA CATEGORIES

Persistent Data:
→ The "source of record" for the database (e.g., tables).
→ Modern systems assume that these data files are immutable but can support updates by rewriting them.

Intermediate Data:
→ Short-lived artifacts produced by query operators during execution and then consumed by other operators.
→ The amount of intermediate data that a query generates has little to no correlation to amount of persistent data that it reads or the execution time.
A distributed DBMS's system architecture specifies the location of the database's persistent data files. This affects how nodes coordinate with each other and where they retrieve/store objects in the database.

Two approaches (not mutually exclusive):
→ Push Query to Data
→ Pull Data to Query
PUSH VS. PULL

Approach #1: Push Query to Data
→ Send the query (or a portion of it) to the node that contains the data.
→ Perform as much filtering and processing as possible where data resides before transmitting over network.

Approach #2: Pull Data to Query
→ Bring the data to the node that is executing a query that needs it for processing.
→ This is necessary when there is no compute resources available where persistent data files are located.
Approach #1: Push Query to Data
→ Send the query (or a portion of it) to the node that contains the data.
→ Perform as much filtering and processing as possible where data resides before transmitting over network.

Approach #2: Pull Data to Query
→ Bring the data to the node that is executing a query that needs it for processing.
→ This is necessary when there is no compute resources available where persistent data files are located.
Each DBMS instance has its own CPU, memory, locally-attached disk. → Nodes only communicate with each other via network.

Database is partitioned into disjoint subsets across nodes. → Adding a new node requires physically moving data between nodes.

Since data is local, the DBMS can access it via POSIX API.
Each node accesses a single logical disk via an interconnect, but also have their own private memory and ephemeral storage. → Must send messages between nodes to learn about their current state.

Instead of a POSIX API, the DBMS accesses disk using a userspace API.
SYSTEM ARCHITECTURE

Choice #1: Shared Nothing:
→ Harder to scale capacity (data movement).
→ Potentially better performance & efficiency.
→ Apply filters where the data resides before transferring.

Choice #2: Shared Disk:
→ Scale compute layer independently from the storage layer.
→ Easy to shutdown idle compute layer resources.
→ May need to pull uncached persistent data from storage layer to compute layer before applying filters.
Traditionally the storage layer in shared-disk DBMSs were dedicated on-prem NAS.
→ Example: Oracle Exadata

Cloud **object stores** are now the prevailing storage target for modern OLAP DBMSs because they are "infinitely" scalable.
→ Examples: Amazon S3, Azure Blob, Google Cloud Storage
OBJECT STORES

Partition the database's tables (persistent data) into large, immutable files stored in an object store.
→ All attributes for a tuple are stored in the same file in a columnar layout (PAX).
→ Header (or footer) contains meta-data about columnar offsets, compression schemes, indexes, and zone maps.

The DBMS retrieves a block's header to determine what byte ranges it needs to retrieve (if any).

Each cloud vendor provides their own proprietary API to access data (PUT, GET, DELETE).
→ Some vendors support predicate pushdown (S3).
OBJECT STORES

Partition the database's tables (persistent data) into large, immutable files stored in an object store.

→ All attributes for a tuple are stored in the same file in a columnar layout (PAX).

→ Header (or footer) contains meta-data about columnar offsets, compression schemes, indexes, and zone maps.

The DBMS retrieves a block's header to determine what byte ranges it needs to retrieve (if any).

Each cloud vendor provides their own proprietary API (PUT, GET, DELETE).

→ Some vendors support predicate pushdown (S3).

Workers

- Separated compute / storage

- One Worker pod per compute node
  - Executes portions of the query plan

- Custom network protocol over UDP
  - Data distribution between workers
  - Uses Intel DPDK
  - 50% higher throughput on AWS over TCP/IP

- Shard files cached in local NVMe SSD

- Shards persisted in object store
  - Custom AWS S3 access library
  - 3X better throughput than stock S3 lib
ADDITIONAL TOPICS

File Formats
Table Partitioning
Data Ingestion / Updates / Discovery
Scheduling / Adaptivity
OBSERVATION

Snowflake is a monolithic system comprised of components built entirely in-house. Most of the non-academic DBMSs we will cover this semester will have a similar overall architecture.

But this means that multiple organizations are writing the same DBMS software...
One recent trend of the last decade is the breakout OLAP engine sub-systems into standalone open-source components.

→ This is typically done by organizations not in the business of selling DBMS software.

Examples:
→ System Catalogs
→ Query Optimizers
→ File Format / Access Libraries
→ Execution Engines
One recent trend of the last decade in OLAP engine sub-systems into standalone open-source components. This is typically done by organizations of selling DBMS software.

Examples:
- System Catalogs
- Query Optimizers
- File Format / Access Libraries
- Execution Engines
A DBMS tracks a database's schema (table, columns) and data files in its catalog.

→ If the DBMS is on the data ingestion path, then it can maintain the catalog incrementally.

→ If an external process adds data files, then it also needs to update the catalog so that the DBMS is aware of them.

Notable implementations:

→ **HCatalog**

→ **Google Data Catalog**

→ **Amazon Glue Data Catalog**
QUERY OPTIMIZERS

Extendible search engine framework for heuristic- and cost-based query optimization.
→ DBMS provides transformation rules and cost estimates.
→ Framework returns either a logical or physical query plan.
This is the hardest part to build in any DBMS.

Notable implementations:
→ Greenplum Orca
→ Apache Calcite
FILE FORMATS

Most DBMSs use a proprietary on-disk binary file format for their databases. The only way to share data between systems is to convert data into a common text-based format.

→ Examples: CSV, JSON, XML

There are open-source binary file formats that make it easier to access data across systems and libraries for extracting data from files.

→ Libraries provide an iterator interface to retrieve (batched) columns from files.
UNIVERSAL FORMATS

Apache Parquet (2013)
→ Compressed columnar storage from Cloudera/Twitter

Apache ORC (2013)
→ Compressed columnar storage from Apache Hive.

Apache CarbonData (2013)
→ Compressed columnar storage with indexes from Huawei.

Apache Iceberg (2017)
→ Flexible data format that supports schema evolution from Netflix.

HDF5 (1998)
→ Multi-dimensional arrays for scientific workloads.

Apache Arrow (2016)
→ In-memory compressed columnar storage from Pandas/Dremio.
EXECUTION ENGINES

Standalone libraries for executing vectorized query operators on columnar data.
→ Input is a DAG of physical operators.
→ Require external scheduling and orchestration.

Notable implementations:
→ Velox
→ DataFusion
→ Intel OAP
CONCLUSION

Today was about understanding the high-level context of what modern OLAP DBMSs look like. 
→ Fundamentally these new DBMSs are not different than previous distributed/parallel DBMSs except for the prevalence of a cloud-based object store for shared disk.

Our focus for the rest of the semester will be about state-of-the-art implementations of these systems' components.
NEXT CLASS

Storage Models
Data Representation
Partitioning
Catalogs