viettel

TẬP ĐOÀN CÔNG NGHIỆP – VIỄN THÔNG QUÂN ĐỘI

BÁO CÁO MINI-PROJECT

Apache Doris

Phùng Huy Quang

phunghuyquang150904@gmail.com

Chương trình Viettel Digital Talent 2024

Lĩnh vực: Data Engineering

Mentor: Nguyễn Công Minh

Đơn vị: Viettel Solution

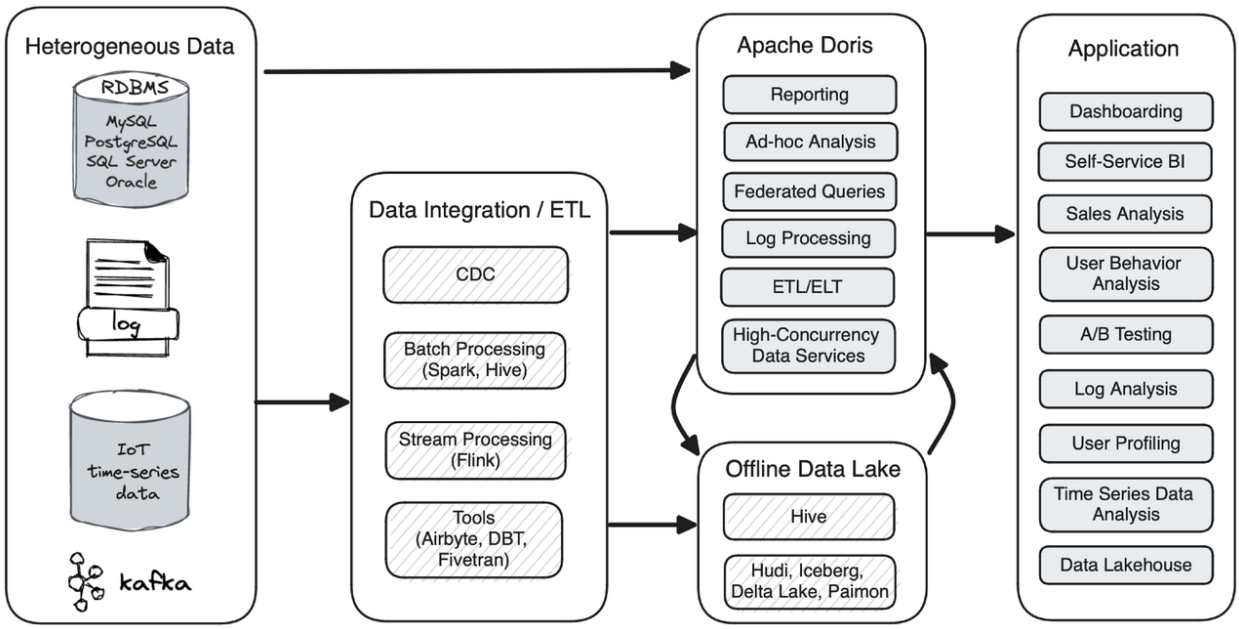
HÀ NỘI, 06/2024

Apache Doris

Table of Contents

What is Apache Doris?

Apache Doris is an MPP-based real-time data warehouse known for its high query speed. For queries on large datasets, it returns results in sub-seconds. It supports both high-concurrent point queries and high-throughput complex analysis. It can be used for report analysis, ad-hoc queries, unified data warehouse, and data lake query acceleration. Based on Apache Doris, users can build applications for user behavior analysis, A/B testing platform, log analysis, user profile analysis, and e-commerce order analysis.



The Background story of Doris

- At Baidu, the largest Chinese search engine, the need for a robust and efficient data warehousing system became clear as the complexity and volume of data grew. The existing two-tiered system, using technologies like Hadoop, Spark, and Storm for data processing and various tools for data serving, faced significant challenges. These included the inefficiencies of sharded MySQL and the complexities of a proprietary distributed statistical database, which required intricate maintenance and custom programming for complex queries.
- Different tools were used across departments, creating hybrid architectures that were cumbersome and difficult to manage. For example, the advertising platform needed to provide detailed, low-latency statistics and high-throughput data analysis, but the

existing MySQL and custom KV storage solutions could not scale efficiently. Internal BI reporting also relied on a mix of tools, further complicating data management.

- To address these challenges, Baidu developed Doris, a simple, integrated system combining the best features of Google Mesa and Apache Impala. Doris offers high concurrent low-latency query performance and high-throughput ad-hoc analysis. It supports bulk-batch and near real-time mini-batch data loading, ensuring high availability, reliability, fault tolerance, and scalability. By integrating a distributed storage engine with the Impala query engine, Baidu created a powerful MPP database that simplifies data processing and querying, eliminating the need for complex hybrid architectures.

Core features help users solve practical problems

1. **Ease of use:**

- It **supports ANSI SQL syntax**, including single table aggregation, sorting, filtering, and multi-table join, sub-query, etc.
- It also **supports complex SQL syntax** such as window functions and grouping sets.
- At the same time, users can **expand system functions through UDF, and UDAF**.
- In addition, Apache Doris is also **compatible with MySQL protocol**, which allows users to access Doris through various BI tools.

2. **High performance:**

- Doris is equipped with an **efficient column storage engine**, which reduces the amount of data scanning and implements an ultra-high data compression ratio.
- At the same time, Doris also **uses various index technology** to speed up data reading and filtering.
- Using the **partition and bucket pruning function**, Doris can support ultra-high concurrency of online service business, and a single node can support up to thousands of QPS.
- Further, Apache Doris **combines the vectorized execution engine** to give full play to the modern CPU parallel computing power. Doris supports the materialized view.
- In terms of the optimizer, Doris **uses a combination of CBO and RBO**, with RBO supporting constant folding, subquery rewriting, predicate pushdown, etc.

3. **Unified data warehouse:**

- Thanks to the well-designed architecture, Doris can easily handle both low-latency, high-concurrency scenarios and high-throughput scenarios.

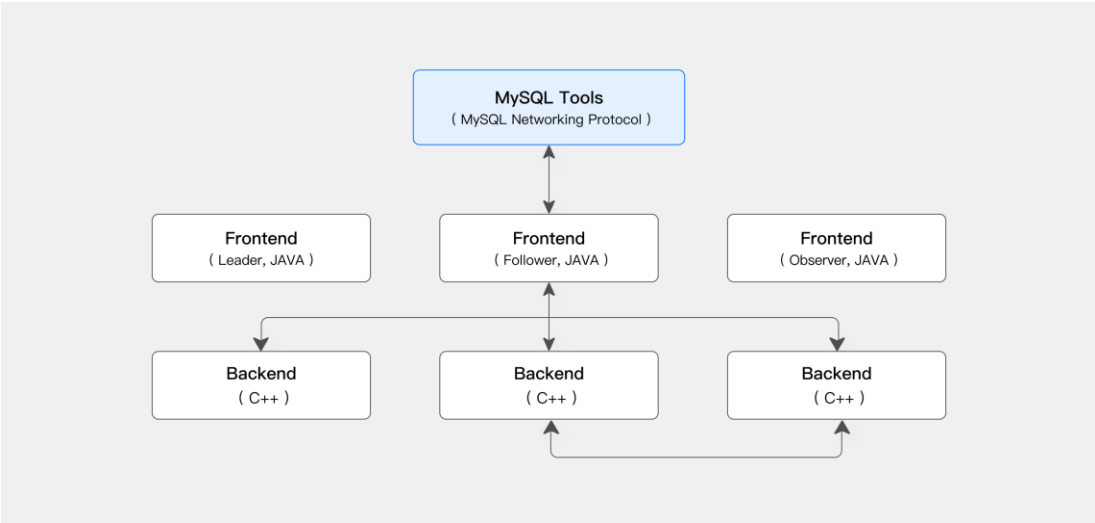
4. **Federated query analysis:**

- With the help of Doris's complete **distributed query engine**, Doris can access data lakes such as Hive, Iceberg, and Hudi, as well as high-speed queries to external data sources such as Elasticsearch and MySQL.

5. **Ecological enrichment:**

- Doris provides rich data ingest methods, supports fast loading of data from localhost, Hadoop, Flink, Spark, Kafka, SeaTunnel, and other systems, and can also directly access data in MySQL, PostgreSQL, Oracle, S3, Hive, Iceberg, Elasticsearch, and other systems without data replication.
- At the same time, the data stored in Doris can also be read by Spark and Flink and can be output to the upstream data application for display and analysis.

Doris System Overview



Doris' implementation consists of two daemons: frontend (FE) and backend (BE):

1. **Frontend (FE):**

- The Frontend daemon consists of a **query coordinator** and **catalog manager**.
- The **query coordinator** is responsible for **receiving users' SQL queries**, **compiling queries**, and **managing query execution**.

- The **catalog manager** is responsible for **managing metadata** such as databases, tables, partitions, replicas, etc.
- **Several frontend daemons** could be deployed to **guarantee fault tolerance** and **load balancing**.

2. Backend (BE):

- The Backend daemon **stores the data** and **executes the query fragments**.
- A table is split into many tablets. Tablets are managed by backends. The Backend daemon could be configured to use multiple directories. Any directory's IO failure doesn't influence the normal running of the Backend daemon. Doris will recover and rebalance the whole cluster automatically when necessary.
- **Many Backend daemons** could also be deployed **to provide scalability** and **fault tolerance**.

A typical Doris cluster is generally composed of several frontend daemons and dozens to hundreds of backend daemons.

3. How does the system work?

- Clients can use MySQL-related tools to connect any frontend daemon to submit SQL queries.
- The Frontend receives the query and compiles it into query plans executable by the backends. Then frontend sends the query plan fragments to backend.
- Backends will build a query execution DAG (Directed acyclic graph). Data is fetched and pipelined into the DAG. The result response is sent to the client via the Frontend.
- The distribution of query fragment execution takes minimizing data movement and maximizing scan locality as the main goal.
- Because Doris is designed to provide interactive analysis, the average execution time of queries is short. Considering this, adopting query re-execution meets the fault tolerance of query execution.

4. Main features of Doris' Frontend design:

4.1. In-Memory Catalog:

- **Traditional data warehouse always uses an RDBMS database** to store their catalog metadata. To produce a query execution plan, Frontend needs to look up the catalog metadata. This kind of catalog storage may be enough for low concurrent ad-hoc

analysis queries. But **for online high concurrent queries, its performance is very bad**, resulting in increased response latency. For example, Hive metadata query latency is sometimes up to tens of seconds or even minutes. **To speed up the metadata access**, Doris adopted the **in-memory catalog storage**.

- In-memory catalog storage has **three functional modules**: **real-time memory data structures**, **memory checkpoints on the local disk**, and **an operation relay log**. When modifying the catalog, the mutation operation is written into the log file first. Then, the mutation operation is applied to the memory data structures. Periodically, a thread does the checkpoint that dumps memory data structure image into the local disk. The checkpoint mechanism enables the fast startup of the Frontend and reduces the disk storage occupancy. The in-memory catalog also simplifies the implementation of multiple frontends.

4.2. Multiple Frontends:

- **Many data warehouses only support single frontend-like nodes**. There are some systems supporting master and slave deploying. But **for online data serving, high availability is an essential feature**. Further, **the number of queries per second may be very large**, so **high scalability is also needed**. Doris provides the feature of multiple frontends using **replicated-state-machine technology**.
- Frontends can be configured to **three kinds of roles**: **leader, follower, and observer**. Through a voting protocol, follower frontends first elect a leader frontend. All the write requests of metadata are forwarded to the leader, then the leader writes the operation into the replicated log file. If the new log entry will be replicated to at least quorum followers successfully, the leader commits the operation into memory and responds to the write request. Followers always replay the replicated logs to apply them to their memory metadata**. If the leader crashes, a new leader will be elected from the leftover followers**. Leader and follower **mainly solve the problem of write availability** and **partly solve the problem of read scalability**.
- **Usually, one leader frontend and several follower frontends can meet most applications' write availability and read scalability requirements**. For very high concurrent reading, continuing to increase the number of followers is not a good practice. Leader replicates log stream to followers synchronously, so adding more followers will increase write latency. **Like Zookeeper, Doris has introduced a new type of frontend node called observer that helps address this problem** and further

improves metadata read scalability. Leader replicates log stream to observers asynchronously. Observers don't involve leader election.

4.3. Consistency Guarantee:

- If a client process connects to the leader, it will see up-to-date metadata, so that strong consistency semantics is guaranteed. If the client connects to followers or observers, it will see metadata lagging a little behind the leader, but monotonic consistency is guaranteed. In most of Doris' use cases, monotonic consistency is accepted.
- If the client always connects to the same Frontend, monotonic consistency semantics are guaranteed; however, if the client connects to other Frontends due to failover, the semantics may be violated. Doris provides a SYNC command to guarantee metadata monotonic consistency semantics during failover. When failover happens, the client can send a SYNC command to the newly connected Frontend, which will get the latest operation log number from the leader. The SYNC command will not return to the client if the local applied log number is still less than the fetched operation log number. This mechanism can guarantee the metadata on the connected Frontend is newer than the client has seen during its last connection.

4.4. MySQL Networking Protocol:

- MySQL-compatible networking protocol is implemented in Doris' Frontend. Firstly, SQL interface is preferred for engineers; Secondly, compatibility with MySQL protocol makes integrating with current existing BI software, such as Tableau, easier; Lastly, rich MySQL client libraries and tools reduce our development costs, but also reduce the user's use cost.
- Through the SQL interface, the administrator can adjust system configuration, add and remove Frontend nodes or backend nodes, and create a new database for the user; the user can create tables, load data, and submit SQL queries.
- Online help documents and Linux Proc-like mechanisms are also supported in SQL. Users can submit queries to get the help of related SQL statements or show Doris' internal running state.
- In the Frontend, a small response buffer is allocated to every MySQL connection. The maximum size of this buffer is limited to 1MB. The buffer is responsible for buffering the query response data. Only if the response is finished or the buffer size reaches 1MB, the response data will begin to be sent to the client. Through this small trick, the Frontend can re-execute most of the queries if errors occur during query execution.

4.5. Two-Level Partitioning:

- Like most distributed database systems, data in Doris is horizontally partitioned. However, a single-level partitioning rule (hash partitioning or range partitioning) may not be a good solution to all scenarios. For example, there is a user-based fact table that stores rows of the form (date, user-id, metric). Choosing only hash partitioning by column user-id may lead to uneven distribution of data when one user's data is very large. If choosing range partitioning according to column date, it will also lead to uneven distribution of data due to the likely data explosion in a certain period.
- Therefore, Doris supports the two-level partitioning rule. The first level is range partitioning. Users can specify a column (usually the time series column) range of values for the data partition. In one partition, the user can also specify one or more columns and several buckets to do the hash partitioning. Users can combine with different partitioning rules to better divide the data. Figure 4 gives an example of two-level partitioning.
- Three benefits are gained by using the two-level partitioning mechanism. Firstly, old and new data could be separated, and stored on different storage mediums; Secondly, the storage engine of the Backend can reduce the consumption of IO and CPU for unnecessary data merging, because the data in some partitions is no longer updated; Lastly, every partition's bucket number can be different and adjusted according to the change of data size.

5. Main features of Doris' Backend design:

5.1. Data Storage Model:

- **Mesa Data Model:** Data in Mesa is a multi-dimensional fact table with dimensional attributes (keys) and measure attributes (values). The schema specifies an aggregation function for values with the same key.
- **ORCFile/Parquet:** Used for efficient, columnar storage and processing, enabling faster querying and high compression.
- **Non-Aggregation Table:** Columns are not differentiated as dimensions or metrics but are sorted by specified columns without aggregation.
- **Aggregation Table:** Follows Mesa's model, distinguishing between key and value columns, and requires data to be sorted on all key columns for aggregation.
- Materialized views containing a subset of columns to improve query performance. Rollups are sorted differently and are updated atomically.

- Updates are applied in batches (at least every minute) to enhance throughput. Each batch generates a delta data file and uses a versioning system to ensure consistency.
- It uses columnar storage with block-level indexing (min/max, bloom filter) to speed up queries by filtering out irrelevant data blocks.
- Sort key index files are used for quick lookups, combining binary search on index and data blocks.
- **Benefits: (1) Efficient Storage and Querying:** Columnar storage allows efficient aggregation and compression, leading to better performance for large-scale data analysis. (2) **Flexibility:** Supports aggregation and non-aggregation tables to cater to different query needs. (3) **Atomic Updates:** Ensures data consistency and reliability through MVCC (Multi-Version Concurrency Control).
- **Challenges:** (1) **Complexity:** Users may struggle with understanding the key-value space and aggregation functions. (2) **Read Overhead:** Single-column count operations can incur additional read overheads due to the need to read all key columns. (3) **Sorting Costs:** Sorting all key columns in large tables can become a bottleneck in the ETL (Extract, Transform, Load) process.

5.2. Data Loading:

- **Hadoop-Batch Loading:** Ideal for loading large volumes of data. Data batches are produced by an external Hadoop system, typically every few minutes. Hadoop handles data preparation (shuffle, sort, aggregate) instead of Doris.
- **Benefits:** Offloads heavy computations to Hadoop, improving efficiency. Reduces performance pressure on the Doris cluster. Ensures stability of the query service, crucial for online data services.
- **Loading:** Recommended for incremental data loading. Utilizes Doris' query engine for data import after deploying fs-brokers.
- **Mini-Batch Loading:** Suitable for loading small amounts of data with low latency. Raw data is pushed into a backend via an HTTP interface. The backend performs data preparation and final loading. HTTP tools can connect to either the Frontend (which redirects requests to a backend) or directly to the backend.
- **Benefits:** Efficient handling of large and small data volumes. Improved stability and performance by leveraging external systems for heavy computations. Flexible loading methods to suit different data volume and latency requirements.

5.3. Resource Isolation:

- **Multi-tenancy Isolation:** Multiple virtual clusters can be created in one physical Doris cluster. Every backend node can deploy multiple backend processes. Every backend process only belongs to one virtual cluster. Virtual cluster is one tenancy.
- **User Isolation:** There are many users in one virtual cluster. You can allocate the resources among different users and ensure that all users' tasks are executed under a limited resource quota.
- **Priority Isolation:** There are three priority isolation groups for one user. Users could control resources allocated to different tasks submitted by themselves, for example, user query tasks and loading tasks require different resource quotas.

5.4. Multi-Medium Storage:

- Most machines in modern data centers are equipped with both SSDs and HDDs. SSD has good random read capability which is the ideal medium for queries that need a large number of random read operations. However, SSD's capacity is small and is very expensive, so we could not deploy it at a large scale. HDD is cheap and has a huge capacity that is suitable for storing large-scale data but with high read latency.
- In the OLAP scenario, we find users usually submit a lot of queries to query the latest data (hot data) and expect low latency. The user occasionally executes queries on historical data (cold data). This kind of query usually needs to scan large scale of data and is high latency.
- Multi-medium storage allows users to manage the storage medium of the data to meet different query scenarios and reduce the latency. For example, the user could put the latest data on SSD and historical data, which is not used frequently on HDD, user will get low latency when querying the latest data while getting high latency when querying historical data which is normal because it needs to scan large scale data.

5.5. Vectorized Query Execution:

- Runtime code generation using LLVM is one of the techniques employed extensively by Impala to improve query execution times. Performance could gain 5X or more is typical for representative workloads.
- But runtime code generation is not suitable for low latency queries, because the generation overhead costs about 100ms. Runtime code generation is more suitable for large-scale ad-hoc queries. To accelerate the small queries (of course, big queries will also obtain benefits), we introduced vectorized query execution into Doris.

- **Vectorized query execution** is a feature that greatly reduces the CPU usage for typical query operations like scans, filters, aggregates, and joins.
- A standard query execution system processes one row at a time. This involves long code paths and significant metadata interpretation in the inner loop of execution. Vectorized query execution streamlines operations by processing a block of many rows at a time. Within the block, each column is stored as a vector (an array of a primitive data type). Simple operations like arithmetic and comparisons are done by quickly iterating through the vectors in a tight loop, with no or very few function calls or conditional branches inside the loop. These loops compile in a streamlined way that uses relatively few instructions and finishes each instruction in fewer clock cycles, on average, by effectively using the processor pipeline and cache memory.

Architectures that make Apache Doris analyze data in real-time fast

1. MPP-based (Massively Parallel Processing):

a. Definition:

- Massively parallel processing (MPP) is a collaborative processing of the same program using two or more processors. By using different processors, speed can be dramatically increased.
- Since the computers running the processing nodes are independent and do not share memory, each processor handles a different part of the data program and has its operating system. Companies use a messaging interface so that the different MPP processors can arrange thread handling for faster analytics for business intelligence in large volumes of data.
- MPP databases are shared among processors. The MPP architecture allows relevant information to be communicated between processors. Large datasets in data warehouses connect independent processing nodes.

b. MPP – Database:

- An MPP database is a data warehouse or type of database where processing is split among servers or nodes.
- A leader node handles communication with each of the individual nodes. The computer nodes carry out the process requested by dividing up the work into units and more

manageable tasks. An MPP process can scale horizontally by adding additional computing nodes rather than having to scale vertically by adding more servers.

- The more processors attached to the data warehouse and MPP databases, the faster the data can be sifted and sorted to respond to queries. This eliminates the long time required for complex searches on large datasets.
- Data warehouse appliances, used for big data analysis and deep insight, typically combine MPP architecture into the database to provide high performance and easier platform scalability.

c. Benefits:

- **Scalability:** You can scale out in a nearly unlimited way. MPP databases can add additional nodes to the architecture to store and process larger data volumes.
- **Cost-efficiency:** You don't necessarily have to buy the fastest or most expensive hardware to accommodate tasks. When you add more nodes, you distribute the workload, which can then be handled with less expensive hardware.
- **Eliminating the single point of failure:** If a node fails for some reason, other nodes are still active and can pick up the slack until the failed node can be returned to the mix.
- **Elasticity:** Nodes can be added without having to render the entire cluster unavailable.

2. CBO (Cost-based optimizer):

a. Definition:

- A Cost-Based Optimizer (CBO) is an advanced type of query optimizer that enhances query performance by evaluating multiple query execution plans and choosing the one with the lowest estimated cost.
- Unlike simpler optimizers that rely solely on heuristic or rule-based approaches, CBOs use detailed statistical information about the data to make informed decisions. By leveraging data statistics such as data distribution, table sizes, and index availability, the CBO can estimate the cost of various execution plans and select the most efficient one, reducing both processing time and resource consumption.
- This sophisticated approach allows the CBO to handle complex queries effectively, optimizing the performance and overall efficiency of the database system.

b. How it works:

- The CBO operates on the principle of analyzing various potential execution paths for a query and selecting the one that incurs the lowest cost in terms of system resources like CPU, memory, and I/O operations. Here's a breakdown of its process:

- **Query Analysis**: Initially, the CBO breaks down the SQL query to understand the required operations, like table joins, data sorting, and indexing.
- **Execution Plan Generation:** It then explores numerous ways to execute the query, each constituting a different execution plan.
- **Cost Estimation:** For each plan, the CBO estimates a cost, considering factors like estimated I/O and CPU usage. This cost is not measured in traditional units but in an abstract measure that might be referred to as "Query Bucks."
- **Optimal Plan Selection:** After evaluating the costs, the CBO selects the execution plan with the lowest estimated resource usage.

c. Benefits:

- **Improved Query Performance:**
  - **Optimal Execution Plans:** By considering various execution strategies and their associated costs, CBOs select the most efficient plan for a given query. This leads to faster query execution times, especially in complex databases.
  - **Dynamic Plan Selection:** CBOs can adapt to changing data patterns and select plans that are optimized for the current state of the data, ensuring consistently high performance.
- **Efficient Resource Utilization:**
  - **Balanced Resource Use:** CBOs estimate the cost of different execution plans in terms of CPU, memory, and I/O resources. This helps in selecting plans that make optimal use of available resources, preventing over-utilization or under-utilization of system capabilities.
  - **Reduced Operational Overhead:** Efficient query plans mean less strain on database resources, which can reduce operational costs related to hardware, energy, and maintenance.
- **Scalability and Flexibility:**
  - **Handling Large Datasets:** CBOs are particularly effective in environments with large and complex datasets. They can scale efficiently as data volume grows, maintaining performance without the need for constant manual tuning.
  - **Adaptability to Data Changes:** CBOs continuously update their understanding of data distribution and statistics, making them adept at handling evolving data landscapes without manual intervention.
- **Enhanced Accuracy in Query Optimization:**

    - **Data-Driven Decisions:** Unlike Rule-Based Optimizers, CBOs rely on actual data statistics and distributions, leading to more accurate and effective optimization decisions.
    - **Reduced Guesswork:** The reliance on data over fixed rules reduces guesswork in query planning, leading to more consistent and predictable query performance.
- **Better Support for Complex Queries:**
    - **Handling Sophisticated Query Structures:** CBOs are adept at optimizing complex queries involving multiple joins, subqueries, and advanced functions, where rule-based systems might struggle.
    - **Customization for Specific Workloads:** CBOs can tailor execution plans to fit specific workload patterns, which is particularly beneficial in specialized or analytical query environments.

3. Fully Vectorized Execution Engine:

a. Definition:

- **Vectorized Query Execution** refers to a method in database engines that enhances query performance by processing data in batches, rather than row by row. This methodology improves the CPU's data processing efficiency by taking advantage of modern CPU architecture and its ability to perform Single Instruction, Multiple Data (SIMD) operations.

b. How it works:

- Vectorized Query Execution operates by loading chunks of data, called vectors, into the CPU cache and performing batch operations on this data. The main features include:
    - **Batch Processing:** Processes data in large chunks, reducing the overhead related to processing individual data points.
    - **Single Instruction, Multiple Data (SIMD) Optimization:** Maximizes efficiency by performing the same operation on multiple data points simultaneously.
    - **Columnar Read and Write:** Streamlines operations by only processing columns relevant to the query, as opposed to the entire dataset.

c. Benefits:

- Vectorized Query Execution provides an array of benefits that find use cases in various fields, especially in scenarios with voluminous data processing, such as data science and business intelligence. These benefits include:
    - **Increase in Query Performance:** The primary advantage is the speedup of query execution, often by an order of magnitude.
    - **Efficient CPU Usage:** Using SIMD operations and batch processing, reduces CPU cycles per row.
    - **Data Compression:** The column-oriented nature of Vectorized Query Execution allows for better data compression, saving memory space.

4. Point Query:

a. Definition:

- A **point query** is a type of database query that retrieves a specific row or record based on its unique identifier, such as a primary key. This contrasts with range queries, which retrieve a set of rows that match a range of values. Point queries are highly efficient for accessing individual records directly and quickly.

b. How it works:

- **Index Utilization:**
    - Most databases utilize indexes to improve the efficiency of point queries. An index is a data structure that allows quick retrieval of records from a database table.
    - When a point query is executed, the database engine uses the index to locate the specific record associated with the unique identifier.
- **Key-Value Lookup:**
    - The process involves a key-value lookup where the key is the unique identifier, and the value is the record associated with that key.
    - For example, in a SQL database, a point query might look like **SELECT * FROM table WHERE id = 123**. The database engine uses the primary key index to quickly find the record with **id = 123**.
- **Direct Access:**
    - After identifying the location of the record using the index, the database directly accesses the storage location to fetch the data.

- This direct access mechanism significantly reduces the time required to retrieve the record compared to a full table scan, where the database would need to check each row.

c. Benefits:

- **High Performance:** Point queries are extremely fast due to the use of indexes and direct access methods. They typically have constant time complexity (O(1)) for retrieval, making them suitable for real-time applications.
- **Reduced Latency:** Because point queries do not require scanning the entire table, they have minimal latency. This is particularly beneficial for applications that require quick responses, such as web applications, real-time analytics, and transactional systems.
- **Efficient Resource Utilization:** Point queries minimize the computational and I/O resources required for data retrieval. This efficiency can lead to better overall system performance and reduced load on the database server.
- **Scalability:** As the size of the database grows, the performance of point queries remains consistent, provided the indexes are properly maintained. This scalability is crucial for large-scale applications and big data systems.
- **Simplicity:** Point queries are straightforward to implement and understand. They involve simple query syntax and clear logic, making them easy to use for developers and database administrators.

5. Query execution is data-driven:

a. Definition:

- **"Query execution is data-driven"** refers to a paradigm where the execution flow of a query is determined dynamically based on the data it processes. This means that the operations performed, and their order can change depending on the characteristics of the data encountered during execution. This approach contrasts with more static query execution plans, where the query execution path is predetermined and does not adapt to the data.

b. How it works:

- **Adaptive Query Execution:**
  - The query execution plan is not fixed. Instead, the system monitors the data and adjusts the execution strategy in real time.

    - For instance, if the system detects that a particular join operation is more efficient using a hash join instead of a nested loop join based on the data size and distribution, it can switch to the more optimal method during execution.

- **Data Flow Control:**
    - The execution engine dynamically routes data through different execution paths based on the data's characteristics.
    - Operators in the query plan can adapt their behavior. For example, a filter operation might change its strategy based on the selectivity of the predicate (i.e., how many rows are filtered out).
- **Feedback Loop:**
    - There is a continuous feedback loop where the execution engine collects statistics and performance metrics as it processes the data.
    - These metrics are used to make real-time decisions about optimizing the query execution path.
- **Heuristics and Cost Models:**
    - The system uses heuristics and cost models to estimate the cost of different execution strategies. These models consider factors like data size, distribution, and available system resources (e.g., memory, CPU).
    - Based on these estimates, the system chooses the most cost-effective execution path.

c. Benefits:

- **Adaptive Query Execution:**
    - The query execution plan is not fixed. Instead, the system monitors the data and adjusts the execution strategy in real time.
    - For instance, if the system detects that a particular join operation is more efficient using a hash join instead of a nested loop join based on the data size and distribution, it can switch to the more optimal method during execution.
- **Data Flow Control:**
    - The execution engine dynamically routes data through different execution paths based on the data's characteristics.
    - Operators in the query plan can adapt their behavior. For example, a filter operation might change its strategy based on the selectivity of the predicate (i.e., how many rows are filtered out).

- **Feedback Loop:**
    - There is a continuous feedback loop where the execution engine collects statistics and performance metrics as it processes the data.
    - These metrics are used to make real-time decisions about optimizing the query execution path.
- **Heuristics and Cost Models:**
    - The system uses heuristics and cost models to estimate the cost of different execution strategies. These models consider factors like data size, distribution, and available system resources (e.g., memory, CPU).
    - Based on these estimates, the system chooses the most cost-effective execution path.

Doris Real-Life Use Cases

1. Scenario: Stable ingestion of 15 billion logs per day

- In the user's case, their business churns out 15 billion logs every day. Ingesting such data volume quickly and stably is a real problem. With Apache Doris, the recommended way is to use the Flink-Doris-Connector. It is developed by the Apache Doris community for large-scale data writing. The component requires simple configuration. It implements Stream Load and can reach a writing speed of 200,000~300,000 logs per second, without interrupting the data analytic workloads.
- A lesson learned is that when using Flink for high-frequency writing, you need to find the right parameter configuration for your case to avoid data version accumulation. In this case, the user made the following optimizations:
    - **Flink Checkpoint:** They increase the checkpoint interval from 15s to 60s to reduce writing frequency and the number of transactions processed by Doris per unit of time. This can relieve data writing pressure and avoid generating too many data versions.
    - **Data Pre-Aggregation:** For data of the same ID but comes from various tables, Flink will pre-aggregate it based on the primary key ID and create a flat table, to avoid excessive resource consumption caused by multi-source data writing.
    - **Doris Compaction:** The trick here includes finding the right Doris backend (BE) parameters to allocate the right amount of CPU resources for data

compaction, setting the appropriate number of data partitions, buckets, and replicas (too much data tablets will bring huge overheads), and dialing up “max_tablet_version_num” to avoid version accumulation.

- These measures together ensure daily ingestion stability. The user has witnessed stable performance and a low compaction score in the Doris Backend. In addition, the combination of data pre-processing in Flink and the Unique Key model in Doris can ensure quicker data updates.

2. Scenario: Storage strategies to reduce costs by 50%

- The size and generation rate of logs also impose pressure on storage. Among the immense log data, only a part of it is of high informational value, so storage should be differentiated. The user has three storage strategies to reduce costs.
    - **ZSTD (ZStandard) compression algorithm:** For tables larger than 1TB, specify the compression method as "ZSTD" upon table creation, it will realize a compression ratio of 10:1.
    - **Tiered storage of hot and cold data:** This is supported by the new feature of Doris. The user sets a data "cooldown" for 7 days. That means data from the past 7 days (namely, hot data) will be stored in SSD. As time goes by, hot data "cools down" (getting older than 7 days), and it will be automatically moved to HDD, which is less expensive. As data gets even "colder", it will be moved to object storage for much lower storage costs. Plus, in object storage, data will be stored with only one copy instead of three. This further cuts down costs and the overheads brought by redundant storage.
    - **Differentiated replica numbers for different data partitions:** The user has partitioned their data by time range. The principle is to have more replicas for newer data partitions and less for the older ones. In their case, data from the past 3 months is frequently accessed, so they have 2 replicas for this partition. Data that is 3~6 months old has two replicas, and data from 6 months ago has one single copy.
- With these three strategies, the user has reduced their storage costs by 50%.

3. Scenario: Differentiated query strategies based on data size

- Some logs must be immediately traced and located, such as those of abnormal events or failures. To ensure real-time response to these queries, the user has different query strategies for different data sizes:
    - **Less than 100G:** The user utilizes the dynamic partitioning feature of Doris. Small tables will be partitioned by date and large tables will be partitioned by hour. This can avoid data skew. To further ensure balance of data within a partition, they use the snowflake ID as the bucketing field. They also set a starting offset of 20 days, which means data from the recent 20 days will be kept. In this way, they find the balance point between data backlog and analytic needs.
    - **100G~1T:** These tables have their materialized views, which are the pre-computed result sets stored in Doris. Thus, queries on these tables will be much faster and less resource-consuming. The DDL syntax of materialized views in Doris is the same as those in PostgreSQL and Oracle.
    - **More than 100T:** These tables are put into the Aggregate Key model of Apache Doris and pre-aggregate them. In this way, we enable queries of 2 billion log records to be done in 1~2s.
- These strategies have shortened the response time of queries. For example, a query of a specific data item used to take minutes, but now it can be finished in milliseconds. In addition, for big tables that contain 10 billion data records, queries on different dimensions can all be done in a few seconds.

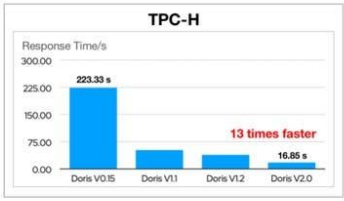
Compare Doris with similar tools

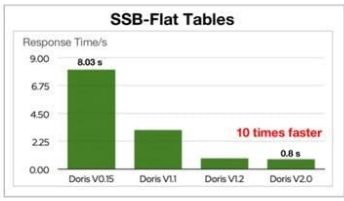
As a real-time OLAP engine, Apache Doris has a competitive edge in query speed. According to the TPC-H and SSB-Flat benchmarking results, Doris can deliver much faster performance than Presto, Greenplum, and ClickHouse.

As for its self-volution, it has increased its query speed by over ten times in the past two years, both in complex queries and flat table analysis.

TPC-H	Doris 2.0	Presto	Greenplum	ClickHouse
Q1	1.57	13.9	30.9	2.74
Q2	0.14	7.31	3.2	108.88
Q3	0.54	12.88	7.9	248.13
Q4	0.26	9.2	8.8	8.54
Q5	1.15	17.89	7.9	235.92
Q6	0.04	4.63	3.9	0.54
Q7	0.73	21.49	11.5	OOM
Q8	0.33	29.9	8.9	OOM
Q9	3.41	60.3	37.8	OOM
Q10	1.34	14.46	7	OOM
Q11	0.1	4.7	1.9	9.71
Q12	0.12	11.17	5.2	OOM
Q13	1.77	10.57	10.4	317.63
Q14	0.14	10.93	3.7	25.38
Q15	0.26	13.46	9.5	1.41
Q16	0.25	3.1	2.3	28.88
Q17	0.12	20.74	71.2	42.32
Q18	2.38	36.28	26	OOM
Q19	0.21	11.34	4.4	54.96
Q20	0.46	14.41	11.7	90.86
Q21	1.06	45.71	16.4	OOM
Q22	0.47	3.58	7.3	10.3
Sum (s)	16.85	377.95	297.8	1186.2

SSB-Flat	Doris 2.0	Presto	ClickHouse
Q1.1	0.02	5.43	0.06
Q1.2	0.01	3.55	0.02
Q1.3	0.03	3.58	0.17
Q2.1	0.08	6.66	0.25
Q2.2	0.08	3.74	0.25
Q2.3	0.06	2.97	0.22
Q3.1	0.16	9.29	0.43
Q3.2	0.07	8.56	0.34
Q3.3	0.06	5.39	0.33
Q3.4	0.01	5.04	0.03
Q4.1	0.14	9.26	0.45
Q4.2	0.05	9.81	0.17
Q4.3	0.03	6.54	0.13
Sum (s)	0.8	79.82	2.85





Test Environment

3 × 16 Core, 64GB cloud virtual machines, SF100

1. Doris vs Clickhouse:

	Doris	Clickhouse
Overview	- Apache Doris is an MPP-based interactive SQL data warehousing system designed for reporting and analysis. - It is known for its high performance, real-time analytics capabilities, and ease of use. - Apache Doris integrates technologies from Google Mesa and Apache Impala. - Unlike other SQL-on-Hadoop systems, Doris is designed to be a simple and tightly coupled system that does not rely on external dependencies. It aims to provide a streamlined and efficient solution for data warehousing and analytics.	- ClickHouse is an open-source columnar database management system designed for high-performance online analytical processing (OLAP) tasks. - It was developed by Yandex, a leading Russian technology company. ClickHouse is known for its ability to process large volumes of data in real-time, providing fast query performance and real-time analytics. - Its columnar storage architecture enables efficient data compression and faster query execution, making it suitable for large-scale data analytics and business intelligence applications.

Time Series Data	- Apache Doris can be effectively used with time series data for real-time analytics and reporting. - With its high performance and sub-second response time, Doris can handle massive amounts of time-stamped data and provide timely query results. - It supports both high-concurrent point query scenarios and high-throughput complex analysis scenarios, making it suitable for analyzing time series data with varying levels of complexity.	- ClickHouse can be used for storing and analyzing time series data effectively, although it is not explicitly optimized for working with time series data. - While ClickHouse can query time series data very quickly once ingested, it tends to struggle with very high write scenarios where data needs to be ingested in smaller batches so it can be analyzed in real time.
Key Concepts	- **MPP (Massively Parallel Processing):** Apache Doris leverages MPP architecture, which allows it to distribute data processing across multiple nodes, enabling parallel execution and scalability. - **SQL:** Apache Doris supports SQL as the query language, providing a familiar and powerful interface for data analysis and reporting. - **Point Query:** Point query refers to retrieving a specific data point or a small subset of data from the database. - **Complex Analysis:** Apache Doris can handle complex analysis scenarios that involve processing large volumes of data and performing advanced computations and aggregations.	- **Columnar storage:** ClickHouse stores data in a columnar format, which means that data for each column is stored separately. This enables efficient compression and faster query execution, as only the required columns are read during query execution. - **Distributed processing:** ClickHouse supports distributed processing, allowing queries to be executed across multiple nodes in a cluster, improving query performance and scalability. - **Data replication:** ClickHouse provides data replication, ensuring data availability and fault tolerance in case of hardware failures or node outages. - **Materialized Views:** ClickHouse supports materialized views, which are precomputed query results stored as

		tables. Materialized views can significantly improve query performance, as they allow for faster data retrieval by avoiding the need to recompute the results for each query.
Architecture	- Apache Doris is based on MPP architecture, which enables it to distribute data and processing across multiple nodes for parallel execution. - It is a standalone system and does not depend on other systems or frameworks. Apache Doris combines the technology of Google Mesa and Apache Impala to provide a simple and tightly coupled system for data warehousing and analytics. - It leverages SQL as the query language and supports efficient data processing and query optimization techniques to ensure high performance and scalability.	- ClickHouse's architecture is designed to support high-performance analytics on large datasets. - ClickHouse stores data in a columnar format. This enables efficient data compression and faster query execution, as only the required columns are read during query execution. - ClickHouse also supports distributed processing, which allows for queries to be executed across multiple nodes in a cluster. ClickHouse uses the MergeTree storage engine as its primary table engine. MergeTree is designed for high-performance OLAP tasks and supports data replication, data partitioning, and indexing.
Features	- **High Performance:** Apache Doris is designed for high-performance data analytics, delivering sub-second query response times even with massive amounts of data. - **Real-Time Analytics:** Apache Doris enables real-time data analysis, allowing users to gain insights and make informed decisions based on up-to-date information.	- **Real-time analytics:** ClickHouse is designed for real-time analytics and can process large volumes of data with low latency, providing fast query performance and real-time insights. - **Data compression:** ClickHouse's columnar storage format enables efficient data compression, reducing storage requirements and improving query performance.

	- **Scalability:** Apache Doris can scale horizontally by adding more nodes to the cluster, allowing for increased data storage and processing capacity.	- **Materialized views:** ClickHouse supports materialized views, which can significantly improve query performance by precomputing and storing query results as tables.
Use Cases	- **Real-Time Analytics:** Apache Doris is well-suited for real-time analytics scenarios where timely insights and analysis of large volumes of data are crucial. It enables businesses to monitor and analyze real-time data streams, make data-driven decisions, and detect patterns or anomalies in real time. - **Reporting and Business Intelligence:** Apache Doris can be used for generating reports and conducting business intelligence activities. It supports fast and efficient querying of data, allowing users to extract meaningful insights and visualize data for reporting and analysis purposes. - **Data Warehousing:** Apache Doris is suitable for building data warehousing solutions that require high-performance analytics and querying capabilities. It provides a scalable and efficient platform for storing, managing, and analyzing large volumes of data for reporting and decision-making.	- **Large-scale data analytics:** ClickHouse's high-performance query engine and columnar storage format make it suitable for large-scale data analytics and business intelligence applications. - **Real-time reporting:** ClickHouse's real-time analytics capabilities enable organizations to generate real-time reports and dashboards, providing up-to-date insights for decision-making. - **Log and event data analysis:** ClickHouse's ability to process large volumes of data in real-time makes it a suitable choice for log and event data analysis, such as analyzing web server logs or application events.
Pricing Model	- As an open-source project, Apache Doris is freely available for usage and does not require any licensing fees.	- ClickHouse is an open-source database and can be deployed on your own hardware.

	- Users can download the source code and set up Apache Doris on their own infrastructure without incurring any direct costs. - However, it's important to consider the operational costs associated with hosting and maintaining the database infrastructure.	- The developers of ClickHouse have also recently created ClickHouse Cloud which is a managed service for deploying ClickHouse.

Reference:

https://doris.apache.org/docs/get-starting/what-is-apache-doris/

https://github.com/apache/doris/wiki/Doris-Overview

https://hackernoon.com/introduction-to-apache-doris-a-next-generation-real-time-data-warehouse

https://doris.apache.org/blog/summit/

https://www.factioninc.com/blog/it-challenges/massively-parallel-processing/

https://www.dremio.com/wiki/vectorized-query-execution/

https://celerdata.com/glossary/cost-based-optimizer#:~:text=A%20Cost%2DBased%20Optimizer%20(CBO,with%20the%20lowest%20estimated%20cost.

https://dev.to/apachedoris/log-analysis-how-to-digest-15-billion-logs-per-day-and-keep-big-queries-within-1-second-4f4n

https://www.influxdata.com/comparison/clickhouse-vs-doris/