APACHE DORIS

From Iceberg to AI

Accelerating Lakehouse Analytics and Hybrid Retrieval with Apache Doris

PRESENTER

Apache Doris Team

Open Source Community

github.com/apache/doris

doris.apache.org

Speaker

Matt Yi

Previously worked at Microsoft，Baidu and Tencent

Contribute 400+ Pull Requests to Apache Doris

Vectorized Excecution, MPP query engine, Workload isolation，

Memory Management

PMC Member of Apache Doris

Agenda

1. Iceberg as the Open Lakehouse Standard 1

2. Doris as the Lakehouse Query Layer 2

3. Customer Facing Analytics + Hybrid Retrieval 3

PART1

Iceberg
Open Lakehouse Standard

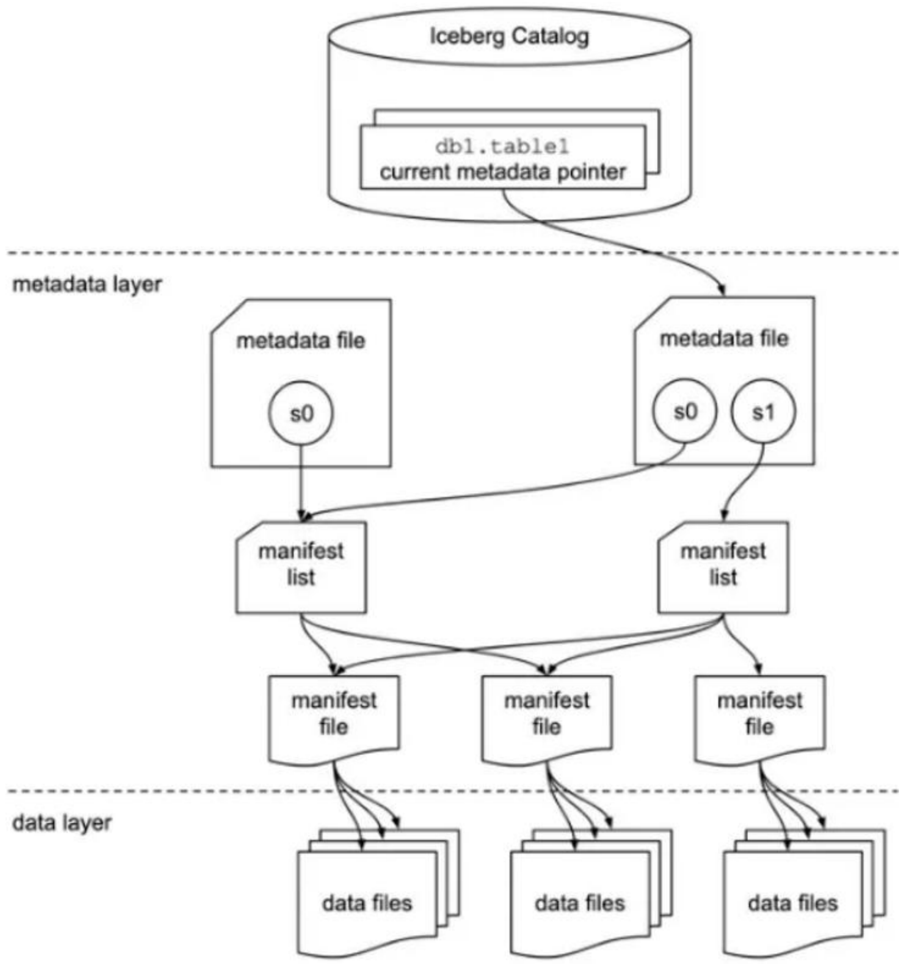
Iceberg as the Open Lakehouse Standard

Apache Iceberg is an open table format designed for huge analytic datasets. It brings the reliability and simplicity of SQL tables to the big data ecosystem, while making it possible for engines like Spark, Trino, Flink, and Apache Doris to safely work with the same tables concurrently.

ACID Transactions:
Iceberg provides atomic, consistent, isolated, and durable transactions that ensure data reliability even with concurrent reads and writes.

Schema Evolution:
Seamlessly add, drop, rename, or reorder fields in tables without affecting query performance or causing conflicts between concurrent reads and writes.

Time Travel:
Query data as it existed at a specific point in time or at a specific snapshot, enabling reproducible queries, auditing, and simplified data recovery.



Multi–Modal Data

1. Structured Data

Traditional Relational Model

- Fixed Schema
- Atomic Types (Int, String)
- Narrow Tables

ICEBERG

2. Semi-Structured

- Dynamic schema
- **JSON** / Map Types
- Sparse & Complex

{ "event": "click", ... }

3. Unstructured / AI

Multi-Modal Retrieval

- Images, Docs, Video
- Vector Embeddings
- Metadata Extraction

[0.1, 0.9...]

Diverse Query Patterns

The Lakehouse architecture centralizes storage, but efficiently serving diverse workloads from a single copy of data remains a critical bottleneck. A single engine must now adapt to conflicting resource requirements.

WORKLOAD	LATENCY REQ	ACCESS PATTERN
Batch SQL	Minutes / Hours	Full Scan / Shuffle
Interactive	second	Scan/Pruning / Aggregation
Customer Facing	Milliseconds	Pruning/Aggregation/Filte r
Vector/Text Search	Milliseconds	Point lookup

⚠ The Fragmentation Trap

Organizations often deploy separate specialized engines: Spark for ETL, Trino for queries, and Elasticsearch/Milvus for AI. This creates "Tool Silos," resulting in complex data synchronization pipelines, redundant storage costs, and inconsistent data governance across systems.

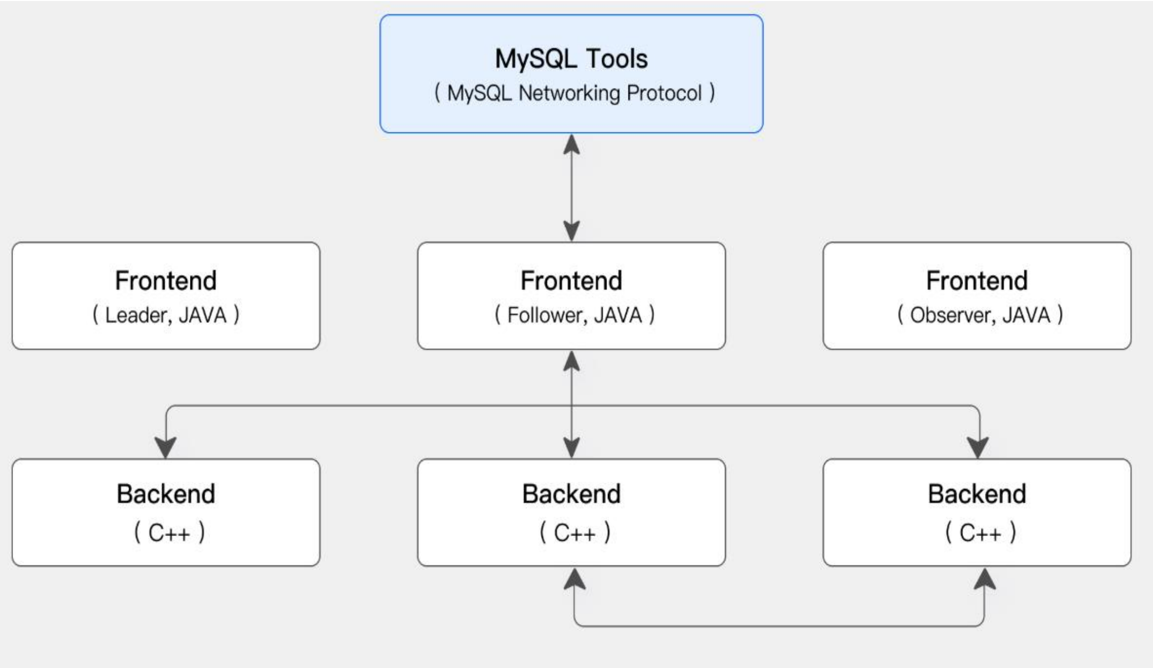
The Unified Goal

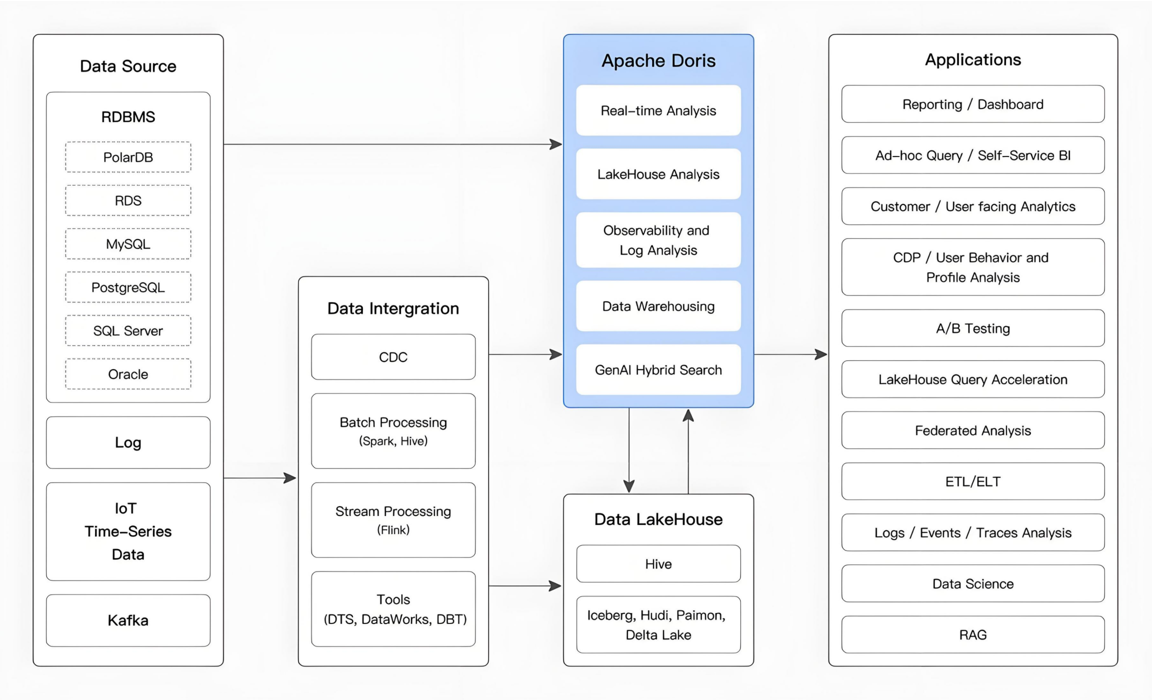
A true Lakehouse needs a Unified Query Layer capable of serving high-performance SQL, keyword search, and vector retrieval directly on open data formats (Iceberg). This eliminates data movement and provides a consistent "Source of Truth" for both BI and AI apps.

PART2

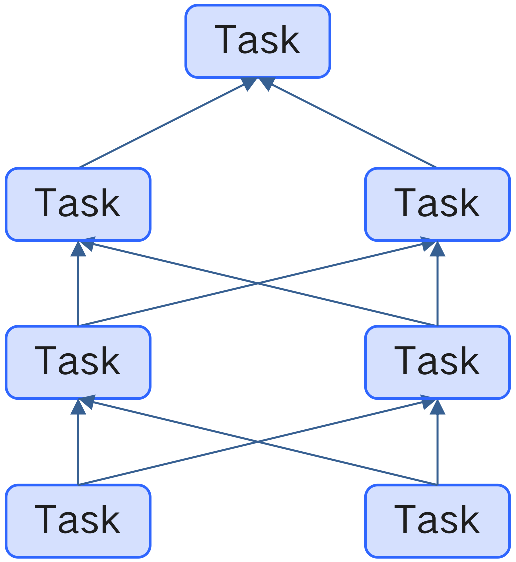
Doris Accelerates Iceberg Analytics

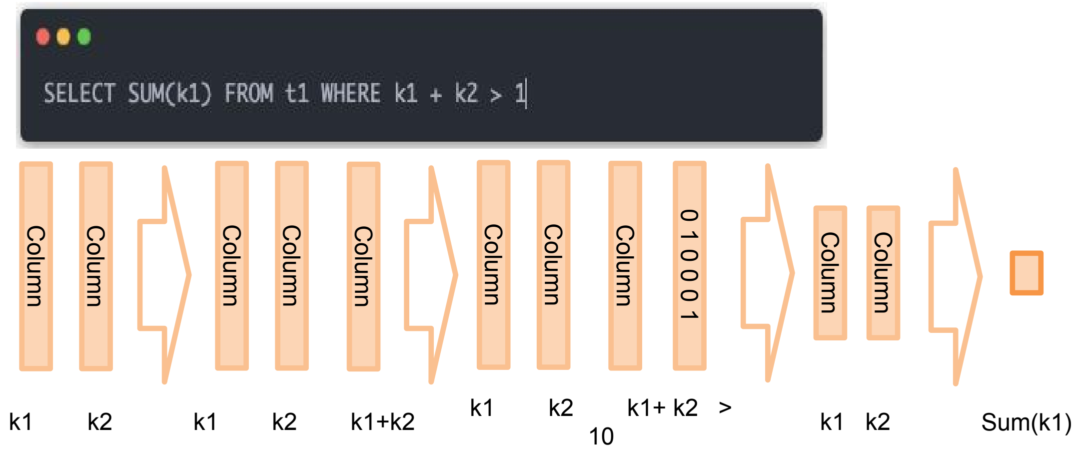
Hybrid Search and Analytics Database





Fast Execution Engine





CBO & HBO Optimizer

Pushes filters and predicates down to the storage layer (Iceberg manifest/files). Reduces I/O by skipping irrelevant data blocks early, minimizing the amount of data transferred and processed by the execution engine.

MPP Query Engine

- Tasks are executed in a pipeline and concurrently.
- Data is transmitted in memory.

Vectorized Execution Engine

- **Compile-time optimization**
- SIMD instructions
- CPU Cache affinity

X86: SSE, AV2,AVX512

ARM: neon，sv2（ARMV9）

Native Parquet Reader

Apache Doris implements several key optimizations for efficient file reading from Iceberg tables, dramatically improving query performance on data lakes:

Predicate Push Down

Filters pushed into storage layer, eliminating unnecessary data reads. Leverages Iceberg metadata for precise filtering.

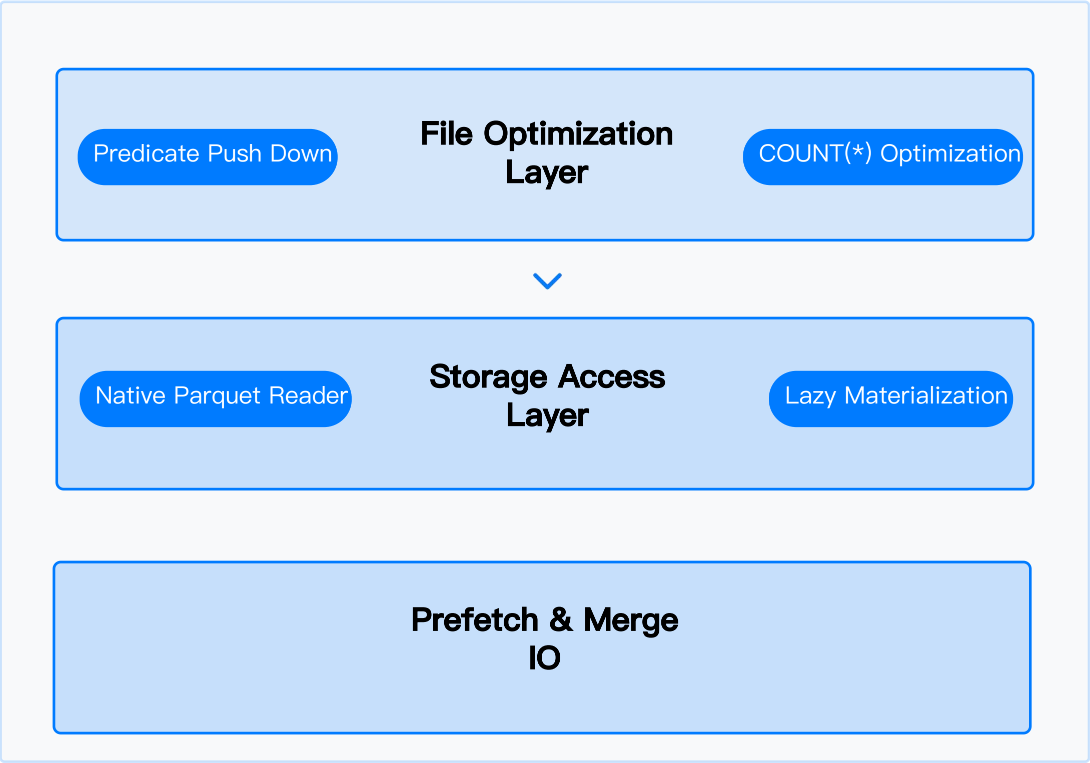
Lazy Materialization

Defers column loading until actually needed, reducing memory usage and I/O by 40-60% for wide tables.

Prefetch & Merge IO:

Intelligent data prefetching mechanisms with adaptive read-ahead strategies reduce I/O wait times by 60-80%. Batch read requests are merged to minimize network round trips.

> Performance Impact: These optimizations reduce data scan volume by up to 95% and cut query latency by 5-10x compared to traditional data lake queries.



Multi-Level Cache

Apache Doris achieves high-performance real-time queries on Iceberg tables through a series of sophisticated engine and I/O optimizations designed specifically for lake data processing.

Manifest Cache

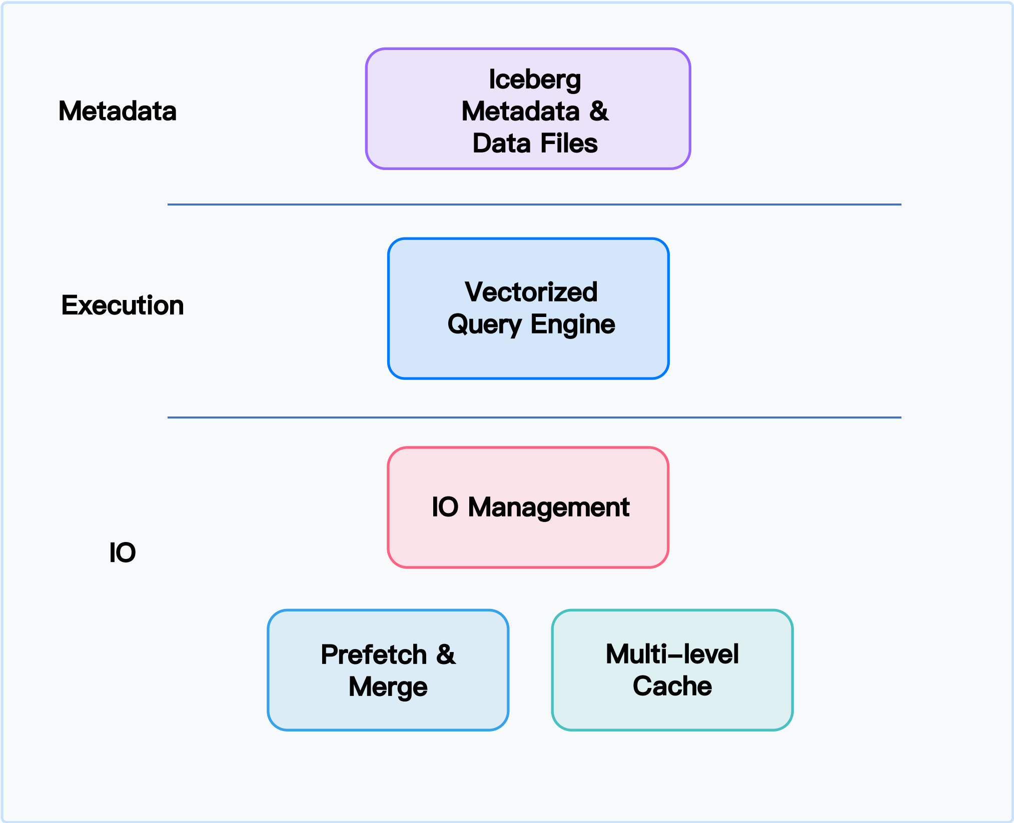
The deserialized metadata to decrease the parsing latency. And will support distributed metadata cache.

Data Cache:

Local SSD cache for hot data blocks, support warmup and LRU. And will support uncompressed data cache in next version.

Iceberg Metadata Optimization:

Native integration with Iceberg's metadata layer for efficient partition pruning, statistics utilization, and snapshot selection, reducing data scan volumes by up to 95% for typical analytical queries.

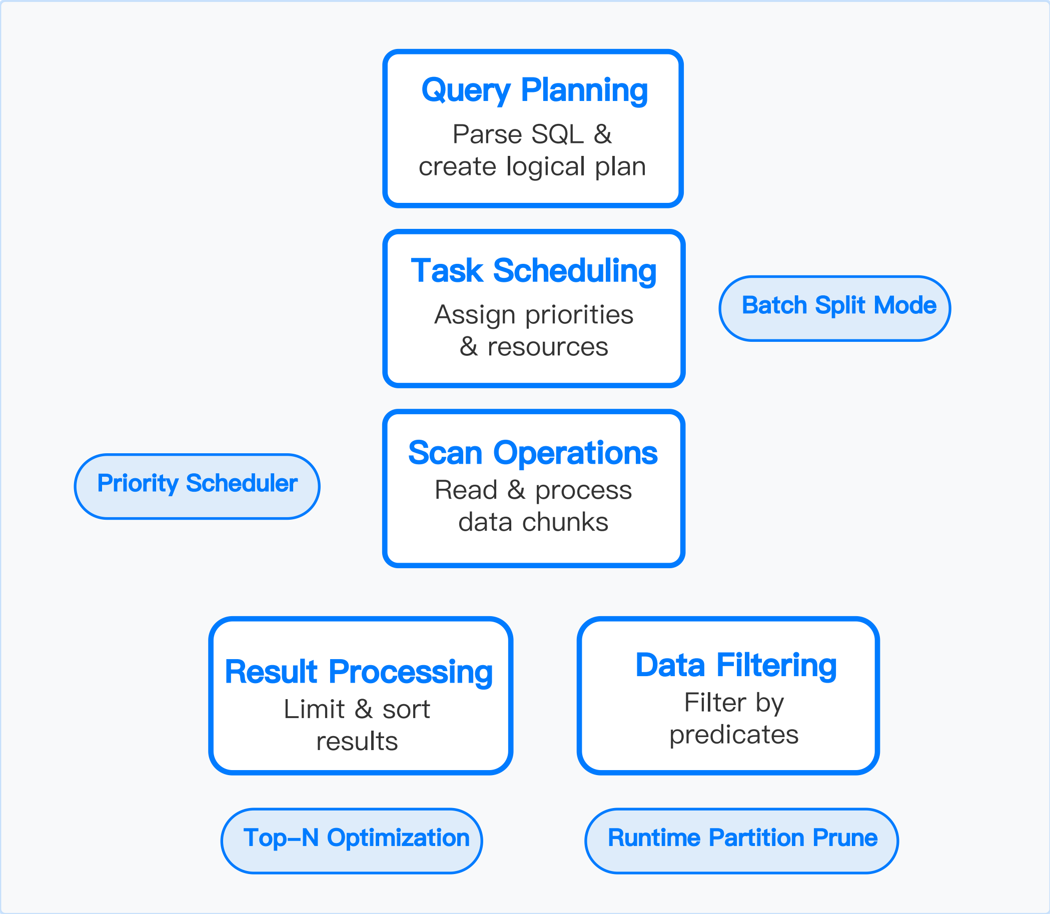


Smart Schedule

Achieving real-time query performance on Iceberg tables requires sophisticated scheduling and scanning optimizations. Apache Doris implements several advanced techniques to maximize efficiency:

- **Priority Scan Scheduler:** Intelligently prioritizes scan tasks based on data locality, task complexity, and cluster load balance to minimize query latency.
- **Runtime Partition Prune:** Eliminates unnecessary partition scans at runtime using predicate conditions, dramatically reducing data scan volume.

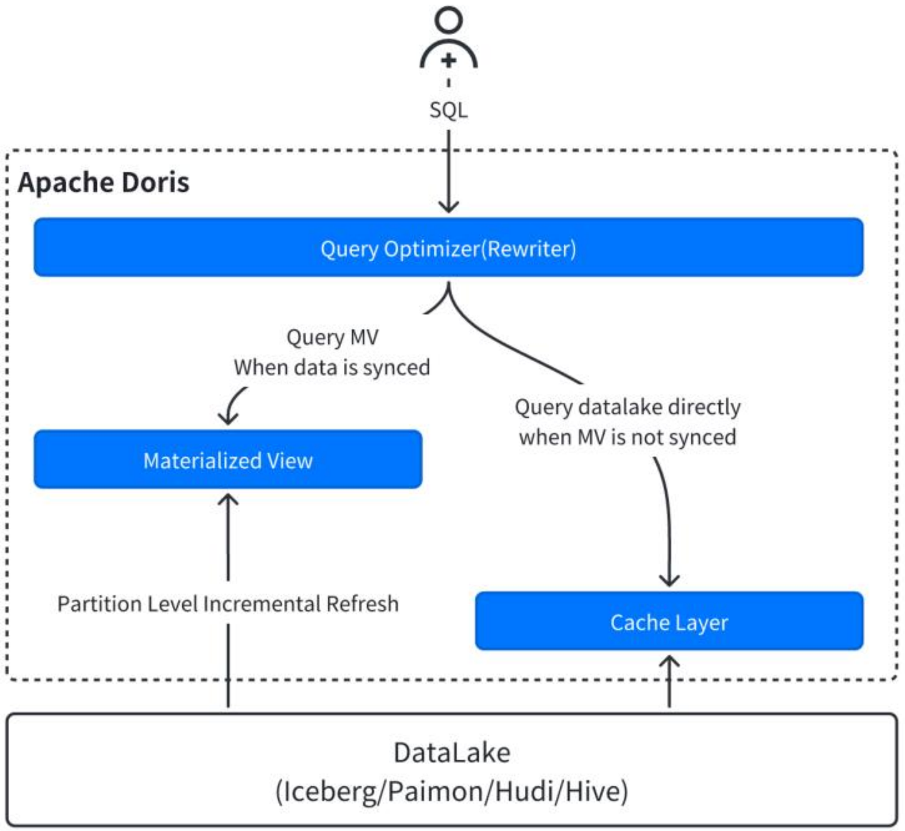
> Performance Impact:
> These optimizations collectively reduce query latency by 70–90% compared to standard Iceberg scans, delivering near-database speed for data lake queries with **predictable latency.**



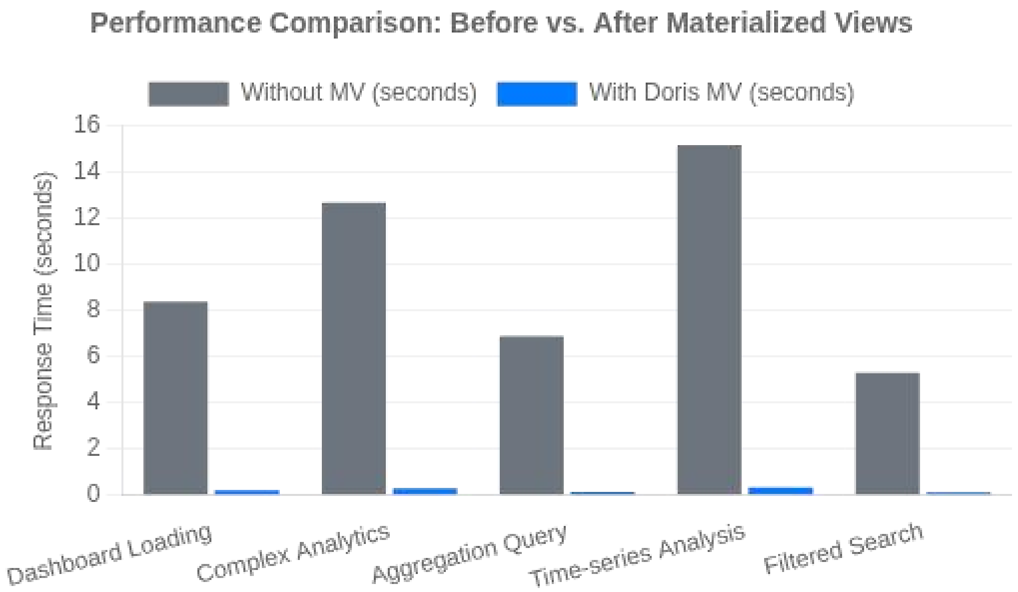
Materialized View

Materialized views are pre-computed, persistent result sets that transform complex analytical queries into simple table lookups, significantly accelerating query performance while maintaining data freshness.

- **Materialized Storage:** Apache Doris stores pre-computed aggregates, joins, and filtered datasets in its highly optimized columnar format, dramatically reducing query processing time.
- **Transparent Query Rewrite:** Queries automatically leverage materialized views without requiring any changes to application code, making integration seamless.
- **Partition Level Refresh:** Instead of rebuilding the entire materialized view when source data changes, Doris intelligently refreshes only affected partitions, dramatically reducing refresh overhead and ensuring data freshness.



Materialized View



42x
Average query speedup

98%
Reduction in scan size

23 ms
Avg. dashboard response

5x
Concurrent user capacity

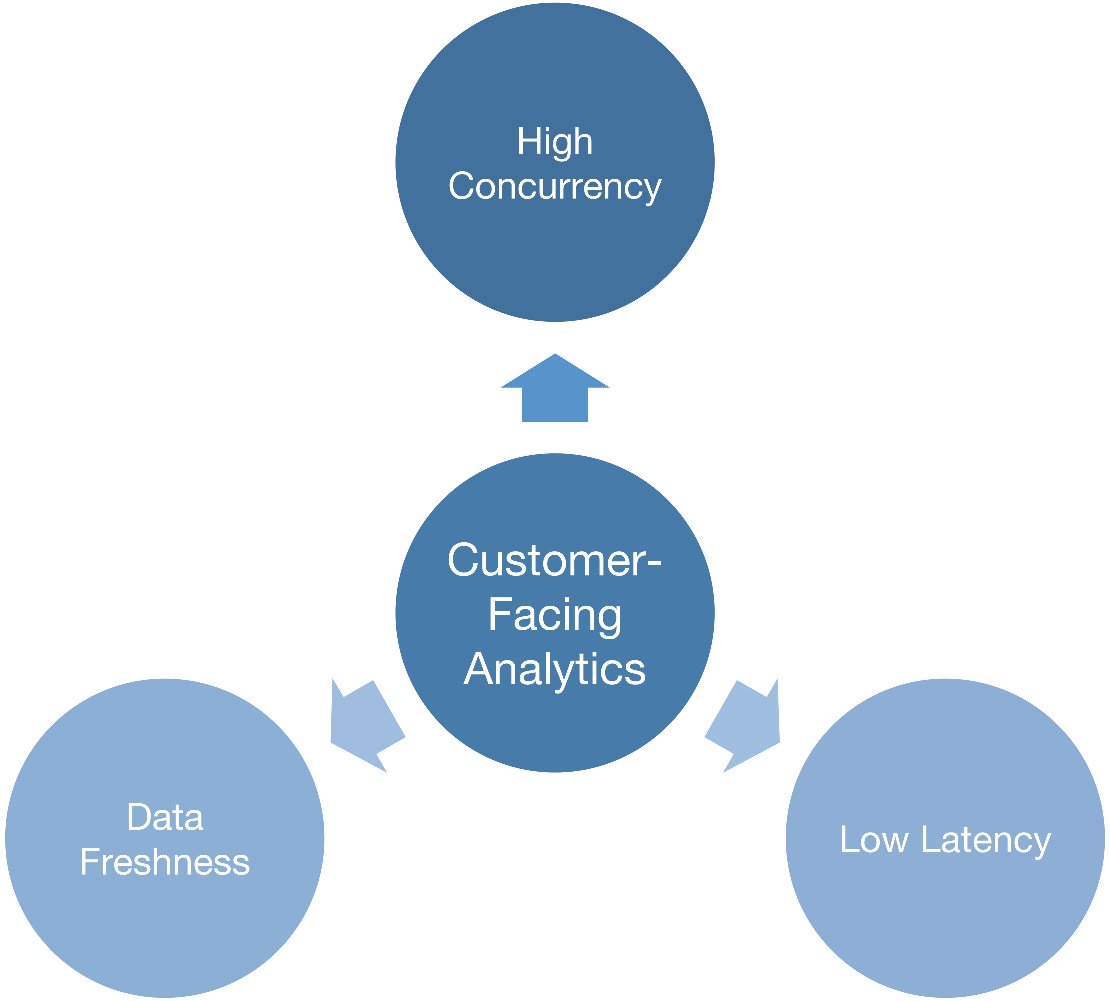
PART3

Customer Facing Analytics + Hybrid Retrieval

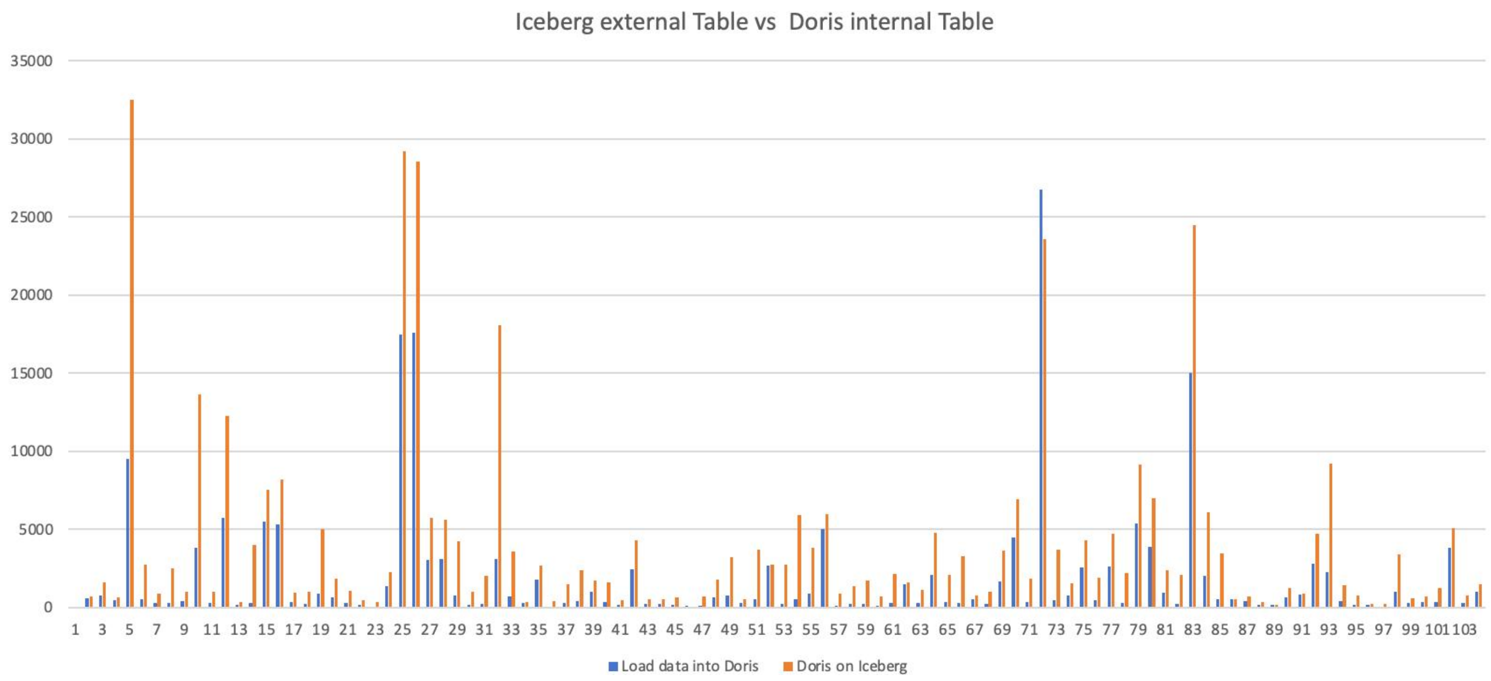
Key Challenges

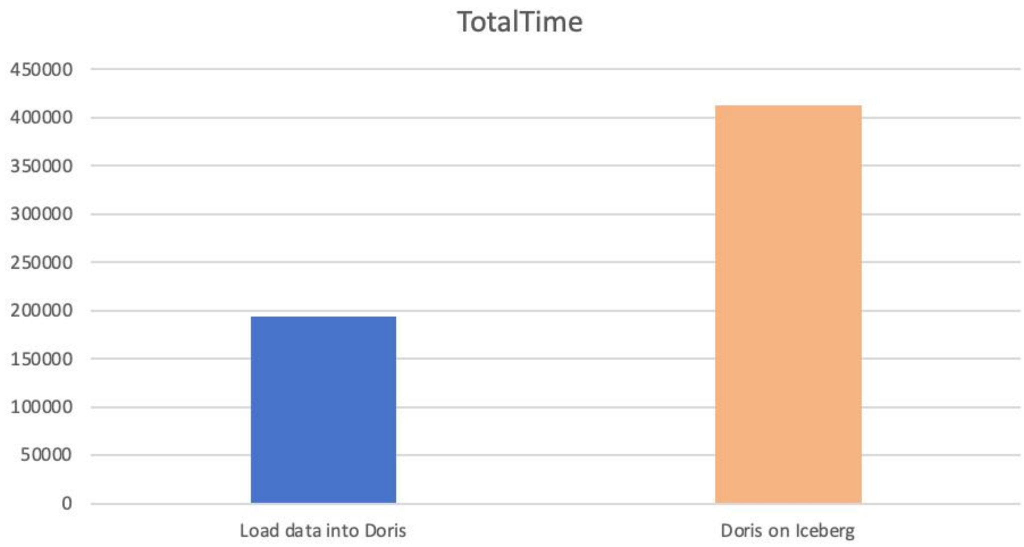
High Concurrency & low latency:
Provide high-concurrency data query services for users, with latency controlled at the second or millisecond level to ensure a good user experience. AI Agent will trigger more queries for a single task, placing greater pressure on the system.

Data Freshness:
Data can be ingested on a large scale in near real time, with updates integrated into the data, ensuring real-time insights.



Performance

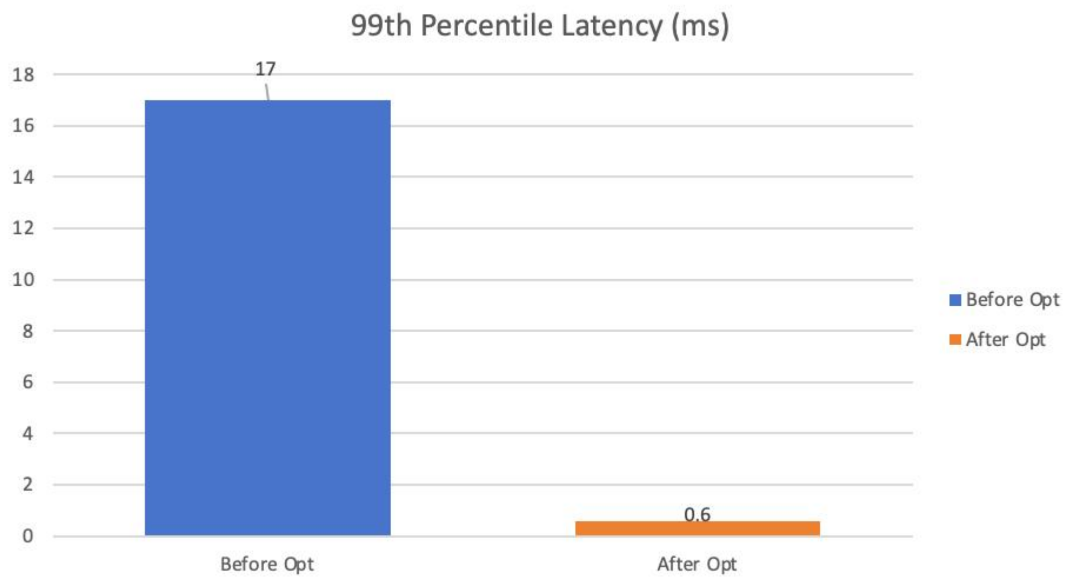


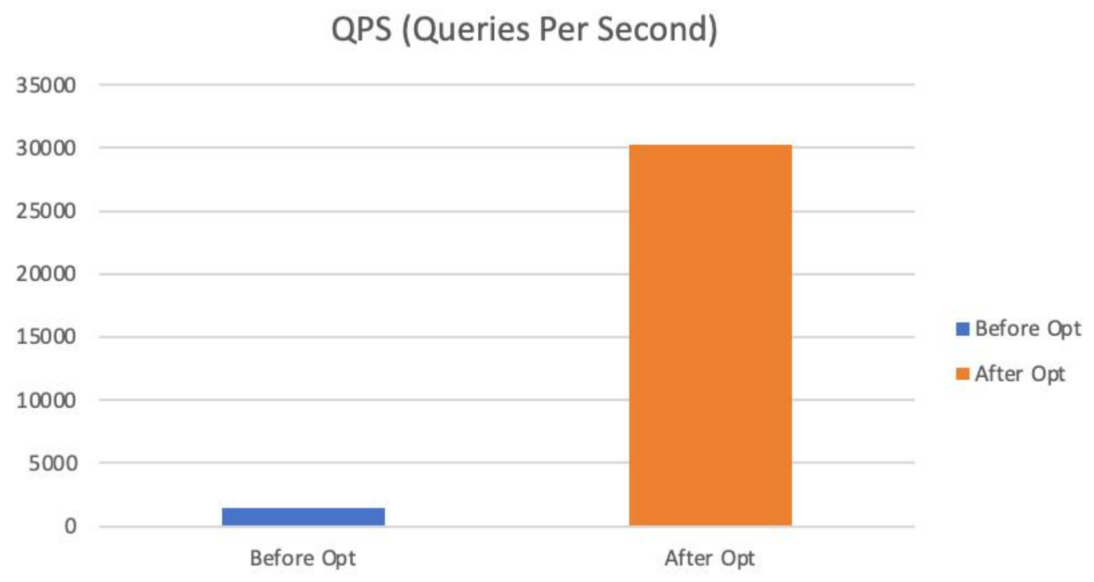


- TPC-DS 1TB
- 193 seconds vs 413 seconds
- All data is cached in local cache

Although TPC-DS is already a CPU-intensive benchmark, we found that the performance of Doris internal tables is still **2x faster** that of Iceberg external tables.

Blazing fast in simple key-value queries



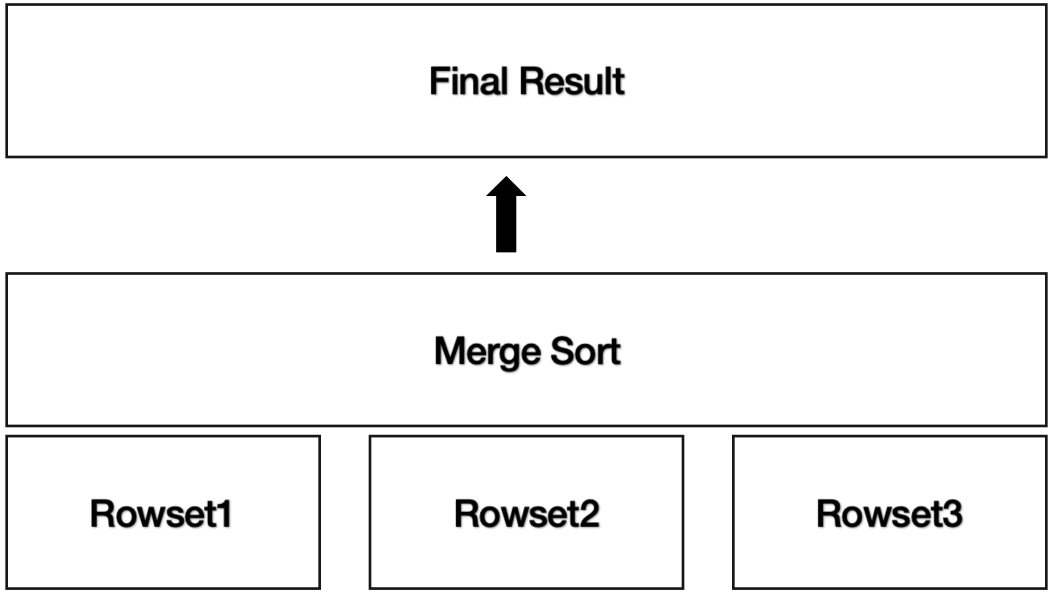


- Row Format instead of column format
- Short circuit query, skip sql parser and planning

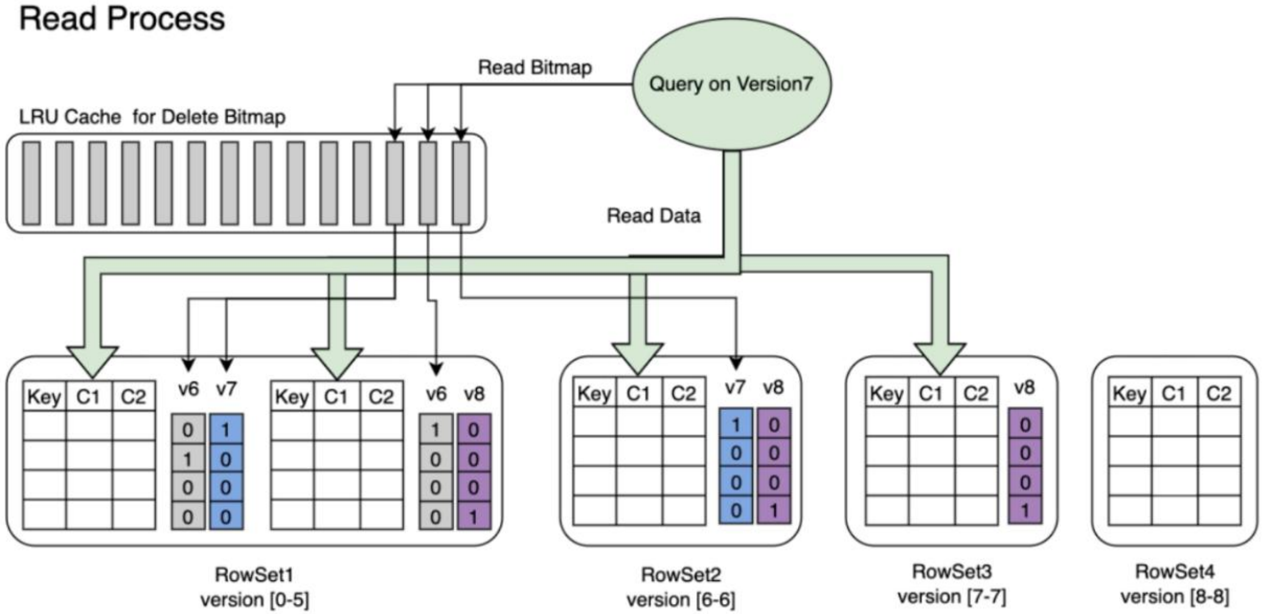
- CPU：48 core
- Benchmark：YCSB
- 100M records

Sub-second Real-time Update

Merge On Read

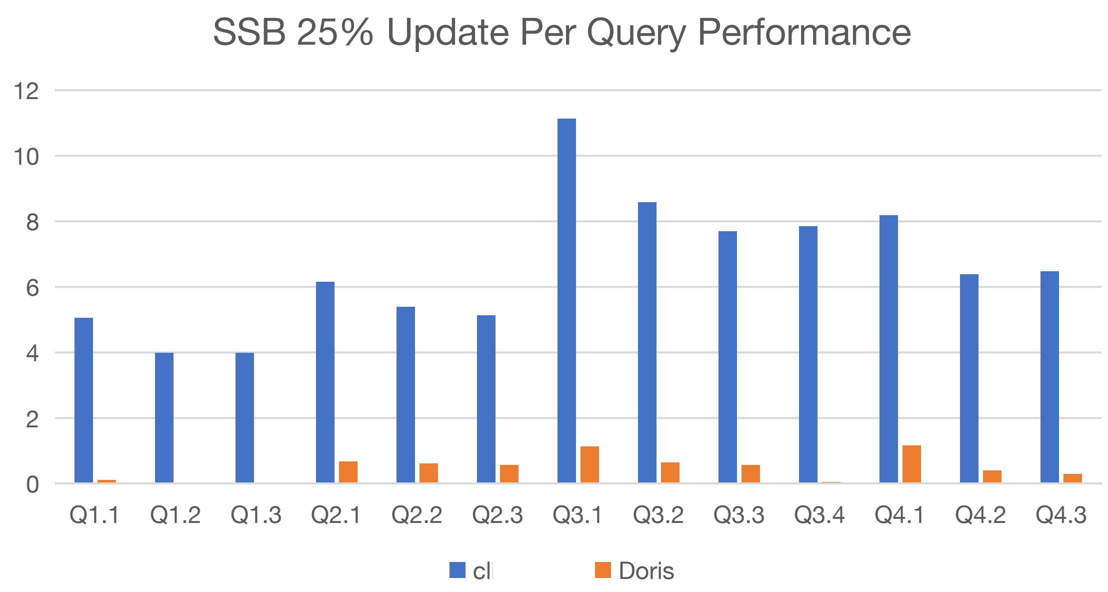


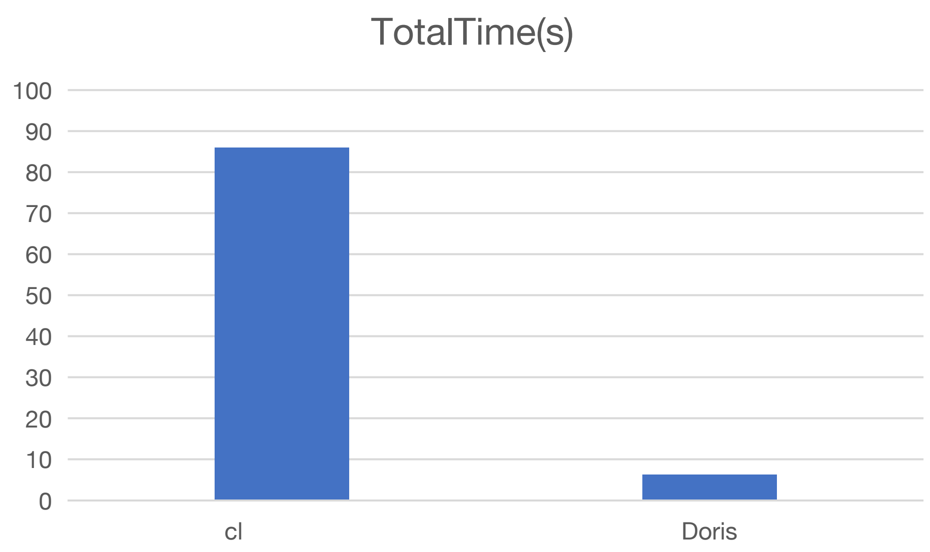
Merge On Write



Rowset can be treated as a parquet file in iceberg.

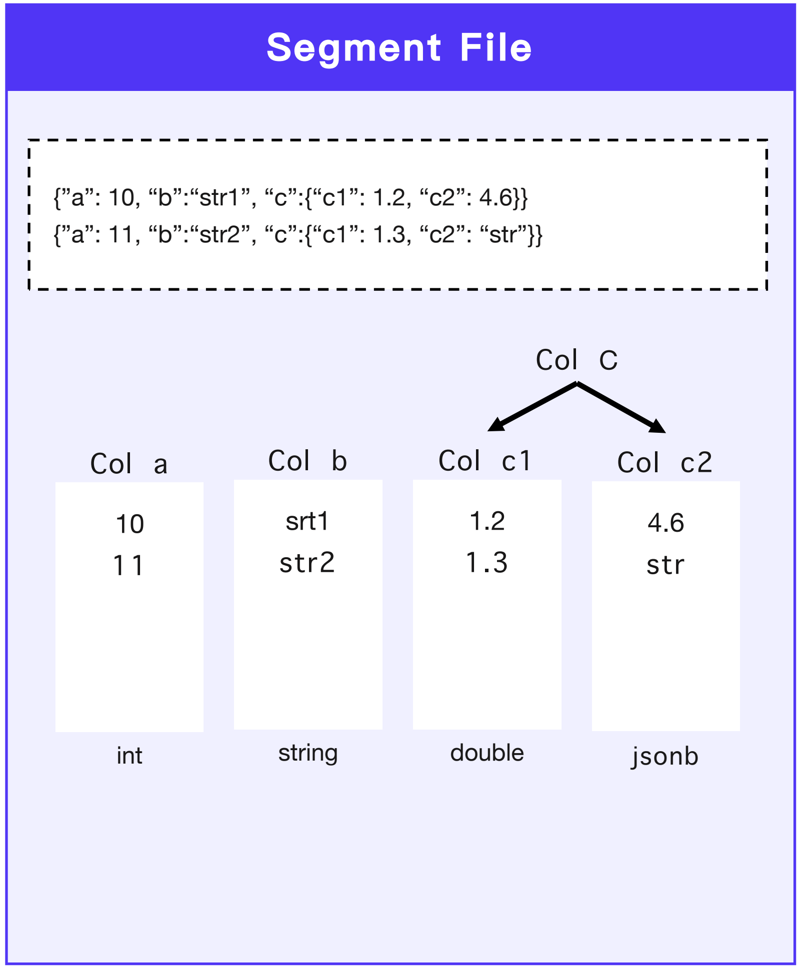
Performance during real-time update





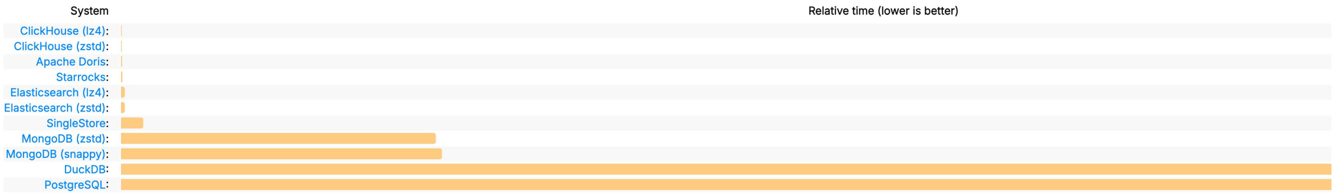
Doris is 25x faster than other

Variant: Columnar JSON Format



- Convert JSON into a columnar format
- Support array and nested JSON

```sql
CREATE TABLE IF NOT EXISTS ${table_name} (
    k BIGINT,
    v VARIANT
)
PROPERTIES("replication_num" = "1");
-- CAST to a concrete type before aggregation
SELECT CAST(v['properties']['title'] AS STRING) AS title
FROM ${table_name}
GROUP BY title;
```



Detailed Comparison

	ClickHouse (lz4)	ClickHouse (zstd)	Apache Doris	Starrocks	Elasticsearch (lz4)	Elasticsearch (zstd)
Data size:	150.32 GiB (×1.62)	**92.72 GiB (×1.00)**	199.88 GiB (×2.16)	179.73 GiB (×1.94)	454.54 GiB (×4.90)	359.58 GiB (×3.88)
Data quality:	999999258	999999258	999999994	996999662	999999153	999999101
Q1.	0.411s (×1.04)	**0.396s (×1.00)**	1.610s (×3.99)	1.580s (×3.92)	3.026s (×7.48)	3.884s (×9.59)
Q2.	**5.526s (×1.00)**	5.543s (×1.00)	6.180s (×1.12)	11.260s (×2.04)	27.380s (×4.95)	28.548s (×5.16)
Q3.	**2.651s (×1.00)**	2.869s (×1.08)	5.450s (×2.05)	3.320s (×1.25)	24.787s (×9.32)	23.570s (×8.86)
Q4.	**0.862s (×1.00)**	1.001s (×1.16)	1.120s (×1.30)	2.590s (×2.98)	8.731s (×10.02)	8.080s (×9.28)
Q5.	**0.969s (×1.00)**	1.086s (×1.12)	1.150s (×1.18)	2.740s (×2.81)	9.733s (×9.95)	8.994s (×9.20)

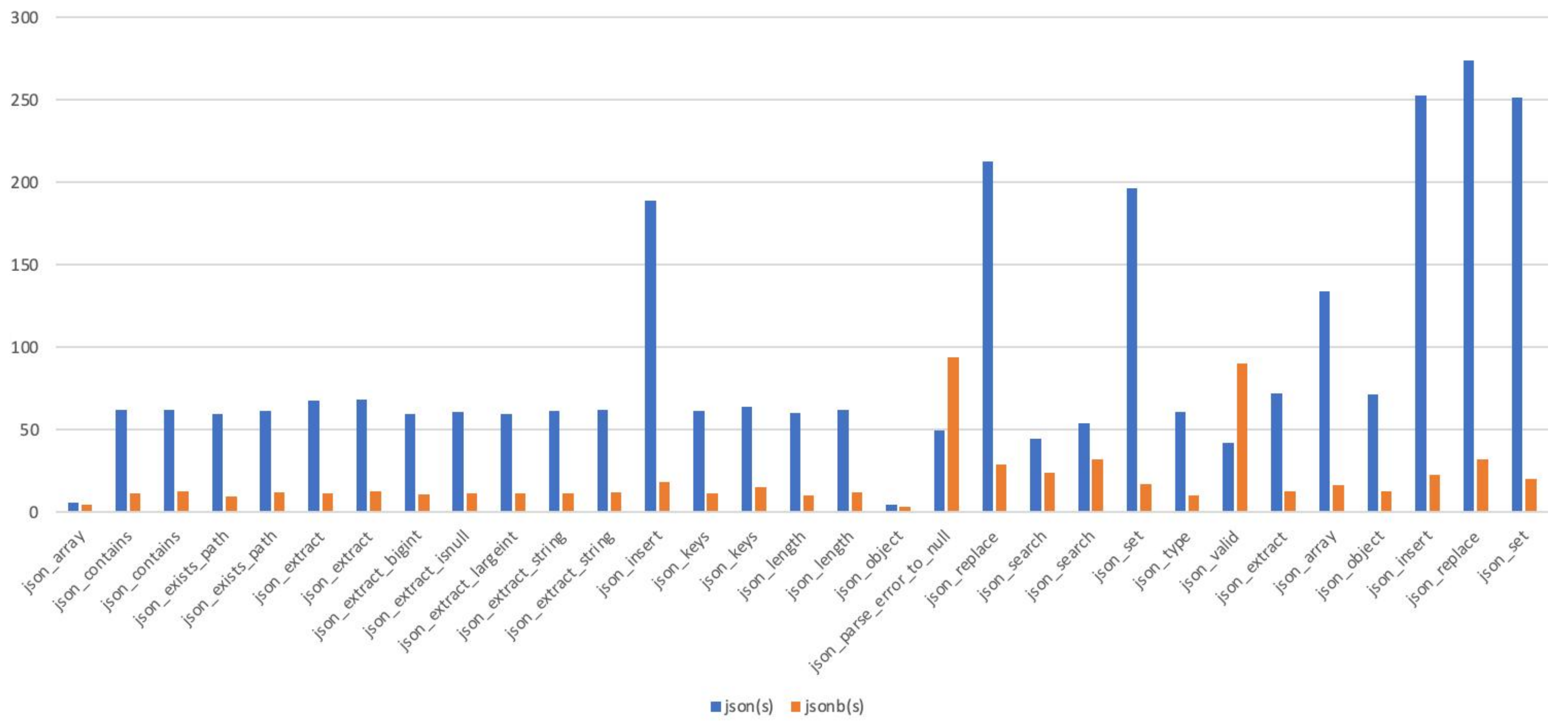
	SingleStore	MongoDB (zstd)	MongoDB (snappy)	DuckDB	PostgreSQL
Data size:	218.75 GiB (×2.36)	164.60 GiB (×1.78)	246.35 GiB (×2.66)	440.14 GiB (×4.75)	615.01 GiB (×6.63)
Data quality:	811999990	893632990	893632990	974400000	804000000
Q1.	43.980s (×108.35)	1054.370s (×2597.00)	1060.360s (×2611.75)	3717.611s (×9156.70)	3884.170s (×9566.95)
Q2.	196.390s (×35.48)	20461.800s (×3696.14)	22193.600s (×4008.96)	3721.045s (×672.16)	4277.800s (×772.73)
Q3.	111.105s (×41.76)	1216.240s (×457.07)	1229.170s (×461.92)	3717.631s (×1397.08)	4253.340s (×1598.40)
Q4.	32.407s (×37.18)	168.797s (×193.59)	169.060s (×193.89)	3719.273s (×4265.23)	4907.090s (×5627.41)
Q5.	40.644s (×41.53)	173.268s (×176.99)	172.833s (×176.55)	3722.804s (×3802.67)	4913.500s (×5018.91)

Hot Run

JSONB: Fast row format

```sql
CREATE TABLE test_json (
  id INT,
  j JSON
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

```sql
SELECT
    id,
    j,
    json_extract(j, '$.a1[0]'),
    json_extract(j, '$.a1[0].k1')
FROM
    test_json
ORDER BY
    id;
```



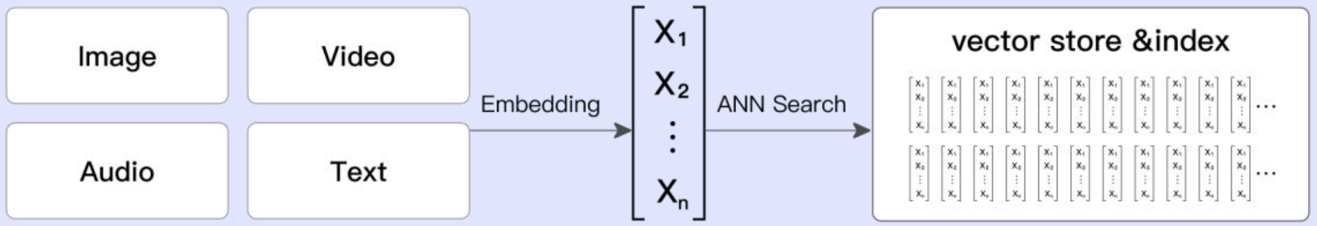
5-10x improvement

Vector Search

ANN

- Algos: HNSW, IVF
- Vector type: Array<float/double>
- Distance: L1, L2, COSINE, Product
- Quantization: SQ4,SQ8,PQ

```
CREATE TABLE sift_1M (
  id int NOT NULL,
  embedding array<float>  NOT NULL  COMMENT "",
  INDEX ann_index (embedding) USING ANN PROPERTIES(
      "index_type"="hnsw",
      "metric_type"="l2_distance",
      "dim"="128",
      "quantizer"="pq",     -- Specify using PQ for quant
      "pq_m"="2",           -- Required when using PQ, in
      sub-vectors
      "pq_nbits"="2"        -- Required when using PQ, in
  )
) ENGINE=OLAP
DUPLICATE KEY(id) COMMENT "OLAP"
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```



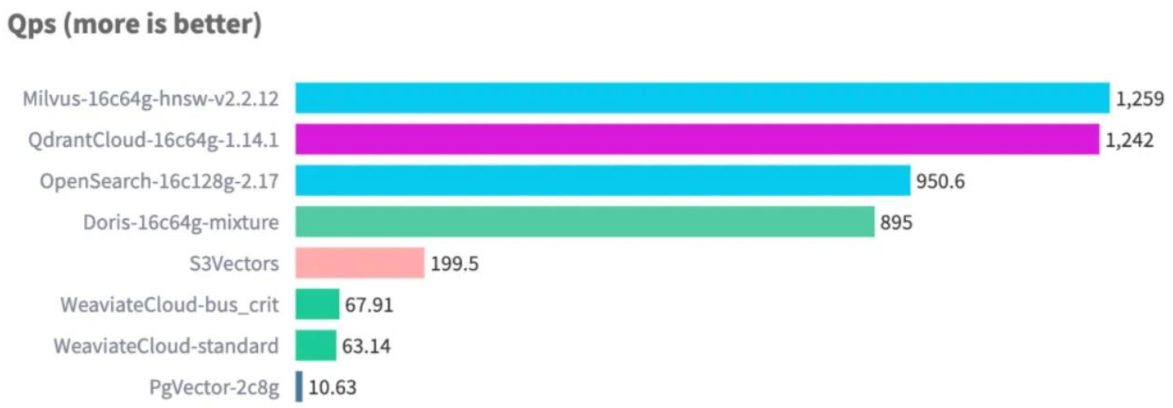
Top N Query

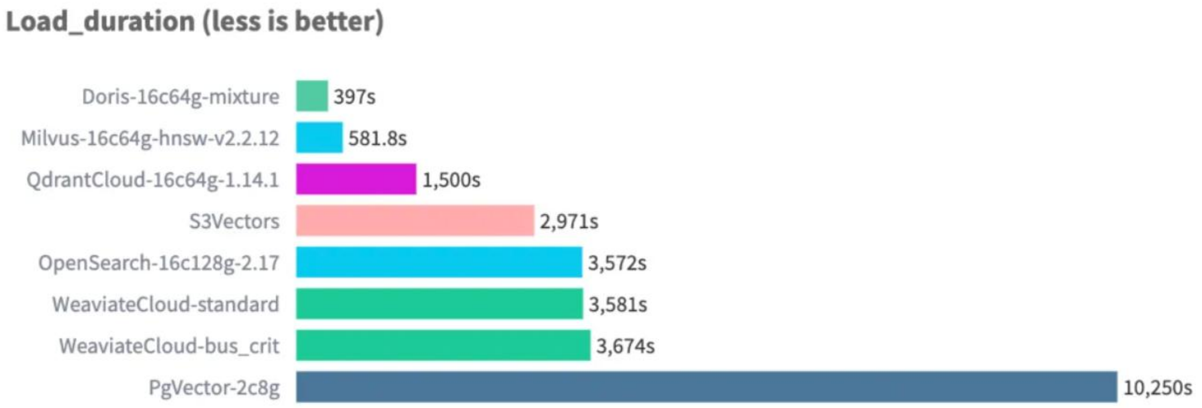
```
SELECT id,
       L2_distance(
        embedding,
        [0,11,77,24,3,0,0,0,28
        0,6,92,8,14,73,125,29,
        50,25,70,64,7,59,18,7,
       ) AS distance
FROM sift_1m
ORDER BY distance
LIMIT 10;
```

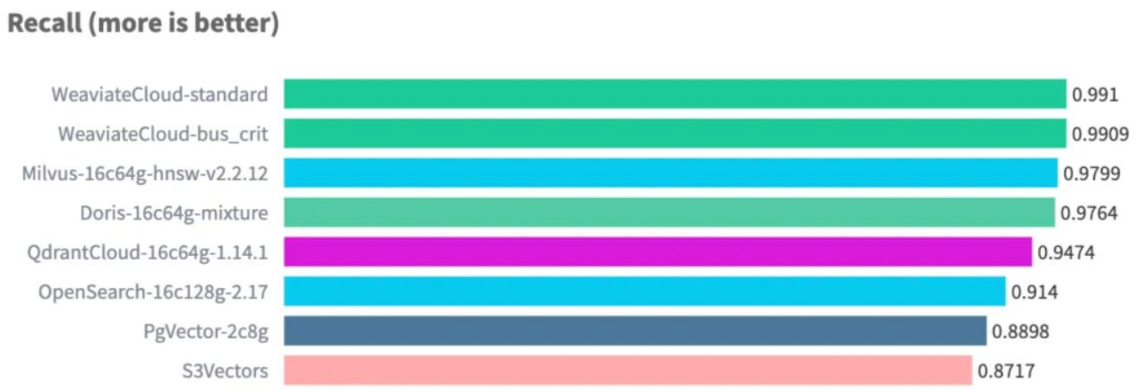
Range Query

```
SELECT count(*)
FROM   sift_1m
WHERE  l2_distance_approximate(
        embedding,
        [0,11,77,24,3,0,0,0,28,70
        0,6,92,8,14,73,125,29,7,0
        50,25,70,64,7,59,18,7,16,
        > 300;
```

Performance



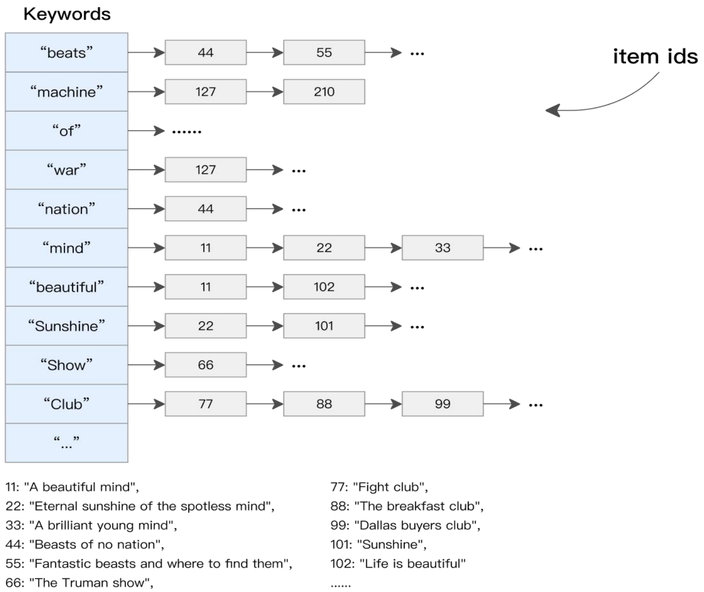




- CPU：Intel Xeon Platinum 8369B @ 2.70GHz (16 核)
- Memory：64GB
- Benchmark：https://github.com/zilliztech/VectorDBBenc
- 768D 1M

Fulltext Search

Inverted Index



- Equality and set: =, !=, IN, NOT IN
- Range: >, >=, <, <=, BETWEEN
- Null checks: IS NULL, IS NOT NULL
- Arrays: array_contains, array_overlaps

Search Syntax

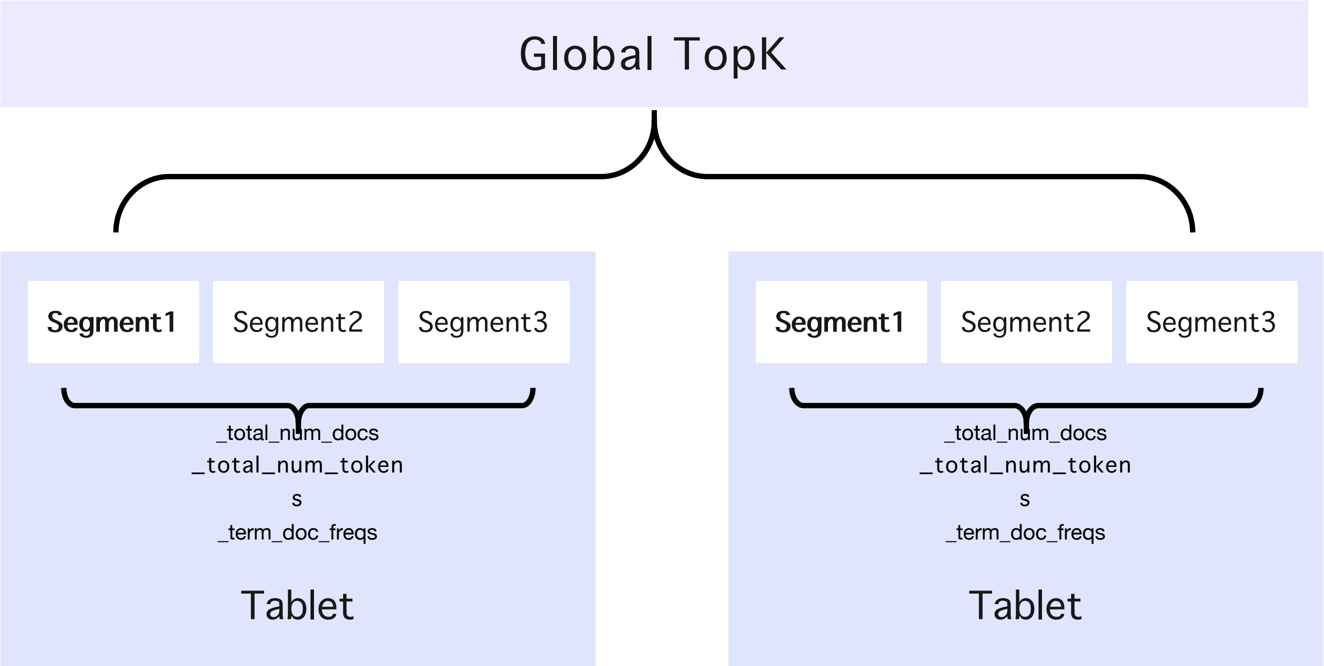
Tokenizer
- Ik
- Jieba
- Pinyin
- ICU

Match
- MATCH_ANY
- MATCH_PHRASE
- MATCH_ALL

Query DSL

```sql
SELECT id, title
FROM search_test_basic
WHERE SEARCH('category:Technology AND NOT title:Machine');
```

BM25 Score




```
SELECT *,
       score() AS relevance
FROM search_demo
WHERE content MATCH_ANY 'text search test'
ORDER BY relevance DESC
LIMIT 10;
```

- MATCH_ANY
- MATCH_ALL
- MATCH_PHRASE
- MATCH_PHRASE_PR
EFIX
- SEARCH

Key Takeaways

1. Fast Query Acceleration Layer

Apache Doris acts as a powerful query layer, significantly accelerating Iceberg analytics without data migration.

2. High Concurrent & Low Latency Customer Facing Analytics

High concurrency and low latency are core to customer-facing analytics systems, ensuring smooth user experiences amid massive simultaneous requests.

3. AI–Ready Foundation

Apache Doris unifies analytics, full–text, and vector search, enabling complex hybrid retrieval workloads for modern AI applications.

Presented by:

Apache Doris Community

Let's Connect

Q & A

Open for Discussion

Join the Community

GitHub github.com/apache/doris

Website doris.apache.org

Slack Channel #apache–doris

Thank You for Listening!