CelerData

Customer-Facing Analytics For Enterprises

Summary

Analytics is no longer just a tool for internal decision-makers. Modern applications embed analytical capabilities directly into the product experience, providing users, partners, and customers with real-time, personalized insights. This shift—from internal BI dashboards to customer-facing analytics—raises the bar for data infrastructure, including latency, concurrency, consistency, and availability at scale.

Legacy BI stacks and proprietary cloud data warehouses rarely meet all four requirements at once. CelerData Cloud—powered by the StarRocks engine and a lakehouse-native design—delivers where others cannot.

What You Get

1. **Consistently get sub-second latency** with stable p99 even during traffic spikes.
2. **Customer-facing analytics on open lakehouse tables**, such as Apache Iceberg, governance is always on.
3. **Seamless scale:** automatic materialised views, intelligent caching, and elastic compute adapt to workload changes without manual tuning.
4. **Up to 90 % lower costs** by eliminating redundant pipelines and right-sizing compute on demand.

Proof in Production

50%
Faster overall

10s
Data freshness

60%
↓ Hardware

- **TRM Labs** – Petabyte-scale blockchain lakehouse, P95 ≤ 3 s, 50 % overall better performance.
- **Pinterest Ads** – p90 latency ↓ 50 %, node count ↓ 68 %, 10 s data freshness.
- **Demandbase** – Hardware ↓ 60 %, storage ↓ 90 %, all JOINs on the fly with no more denormalized tables.

This paper explains why customer-facing analytics is so challenging, then shows how CelerData's real-time and lakehouse reference architectures meets these challenges—without trade-offs.

Customer-Facing Analytics Basics

Customer-facing analytics refers to analytical capabilities delivered directly to external users—customers, partners, and third-party stakeholders—through product interfaces such as dashboards, reports, alerts, or APIs. Unlike internal BI tools, which serve analysts and operators, these interfaces are part of the product experience itself. They must be fast, consistent, personalized, and embedded natively into user workflows.

Key characteristics of successful customer-facing analytics:

1. **Real-time insights:** Users expect current data, not yesterday's batch. Sub-second freshness and interaction speed are the norm.
2. **Performance consistency:** Consistency matters more than just speed. Outlier queries, cache misses, or infrastructure hiccups can destroy the experience. Consistent p99 latencies are essential.
3. **Personalized content:** Different users expect different views—per-tenant isolation, access controls, and tailored metrics are critical, pre-processed dashboards and reports are not enough.
4. **Scale:** Workloads often exceed tens of thousands of concurrent queries and must scale elastically.

Common industry applications:

- **AdTech & Creator Platforms:** Serve real-time campaign and content performance metrics to advertisers and creators.
  - **Pinterest** offers interactive dashboards to advertisers, displaying real-time spend, CTR, and audience breakdowns—queries encompass impressions, conversion logs, and attribution models.
  - **YouTube Studio** offers creators minute-level engagement metrics, subscriber trends, and monetization insights, supported by continuously updated aggregations and cohort analysis.
- **B2B SaaS Products:** Embed analytics to demonstrate product value and enable self-serve exploration.
  - **Demandbase** delivers ROI dashboards that show account-level engagement and campaign impact, utilizing shared Iceberg tables and isolated compute per tenant.
- **E-commerce Marketplaces:** Provide sellers with operational and customer behavior analytics.
  - **Amazon Seller Central** and **Shopify** expose dashboards showing revenue, inventory trends, and segmentation by region or product, often involving transactions, product catalogs, and clickstream data.
- **Financial Applications:** Deliver real-time insights on portfolios, transactions, and market exposure.
  - **Coinbase** supports second-level portfolio analytics and gain/loss reporting, combining live market feeds with user-specific holdings and transaction history summaries.
  - **Robinhood** embeds instant feedback on trade execution, P&L decomposition, and personalized alerting for every end user, powered by low-latency customer-facing analytics.

Unique Requirements For Customer-Facing Analytics

Customer-facing analytics must meet service-level objectives that far exceed those of internal BI dashboards. Latency must remain low and stable even when traffic is unpredictable and highly concurrent; data must remain fresh and access-controlled at a per-tenant level. These factors impose design constraints that general-purpose analytics stacks typically do not address out of the box.

Performance Expectations

1. **Sub-second latency** —Targets apply even during peak usage.
2. **High concurrency** —Tens of thousands of simultaneous queries, often spread across regions.
3. **Minute-to-second freshness** —New records should be visible within seconds of ingestion.
4. **Zero-tolerance reliability** —Timeouts or large tail-latency outliers directly erode user trust.

Meeting these objectives consistently is what distinguishes a production-grade customer-facing stack from an internal reporting system.

Resulting Engineering Challenges

These requirements, while individually manageable, create compounding pressure when combined. To meet them reliably, the system must address several interdependent sources of load and variability.

Key Challenges

Workload Volatility

Query volumes vary with customer behavior and are often unpredictable. Spikes may be triggered by usage surges from a single large tenant or synchronized activity across users (e.g., during business hours). Because queries are user-initiated and burst-prone, the system must handle load increases with little to no advance notice. Static concurrency limits and fixed provisioning are rarely sufficient.

Resource Contention

In multi-tenant environments, queries from different users and organizations compete for shared CPU, memory, and I/O. Without strict workload and resource isolation, high-volume tenants can degrade performance for others, resulting in latency outliers and SLA violations.

Forced Denormalization

Joins are fundamental to many analytics use cases, however, many OLAP systems built for low-latency workloads struggle to execute joins efficiently at scale. To compensate, teams often denormalize data into wide tables, leading to significant storage overhead, delayed freshness due to heavy preprocessing, rigid schemas that require full backfills on change, and complex pipelines that can’t scale.

Extreme Data Skew

In customer-facing systems, tenant data volumes often follow a long-tail distribution—where a small number of tenants account for a disproportionate share of records and queries. Without tenant-aware layout or execution strategies, this imbalance leads to cache contention, hot partitions, and degraded tail latency under concurrent load.

Impact When Requirements Are Not Met

Failure to meet the performance and scalability requirements of customer-facing analytics has downstream consequences across both product experience and engineering operations:

- **Inconsistent query latency** under load results in degraded responsiveness or even timeouts, rendering analytics interfaces less usable for end users.
- **System fragility increases** as teams build custom ingestion, denormalization, or pre-aggregations to compensate for limitations in the core engine. These workarounds add long-term complexity and limit agility.
- **Platform cost efficiency declines as** teams overprovision compute to guard against tail latency spikes, even during moderate loads.
- **Engineering velocity suffers** as efforts shift from product development to maintaining brittle, hand-tuned infrastructure.

Customer-Facing Analytics Essentials

Supporting external-facing workloads at scale requires more than just fast execution; it demands architectural guarantees around **scalability, low latency**, and **predictable performance under concurrency.** This section outlines the core architectural principles and execution techniques that make those guarantees possible.

Architectural Foundations for Scalability

Scalable systems depend on architectural choices that minimize coordination overhead, distribute execution effectively, and maintain performance as data volumes and query concurrency increase. A shared-data architecture, parallel planning, and pipelined execution must work together to enable predictable scale-out under real-world workloads.

Scalability Through Shared Data and Decoupled Architecture

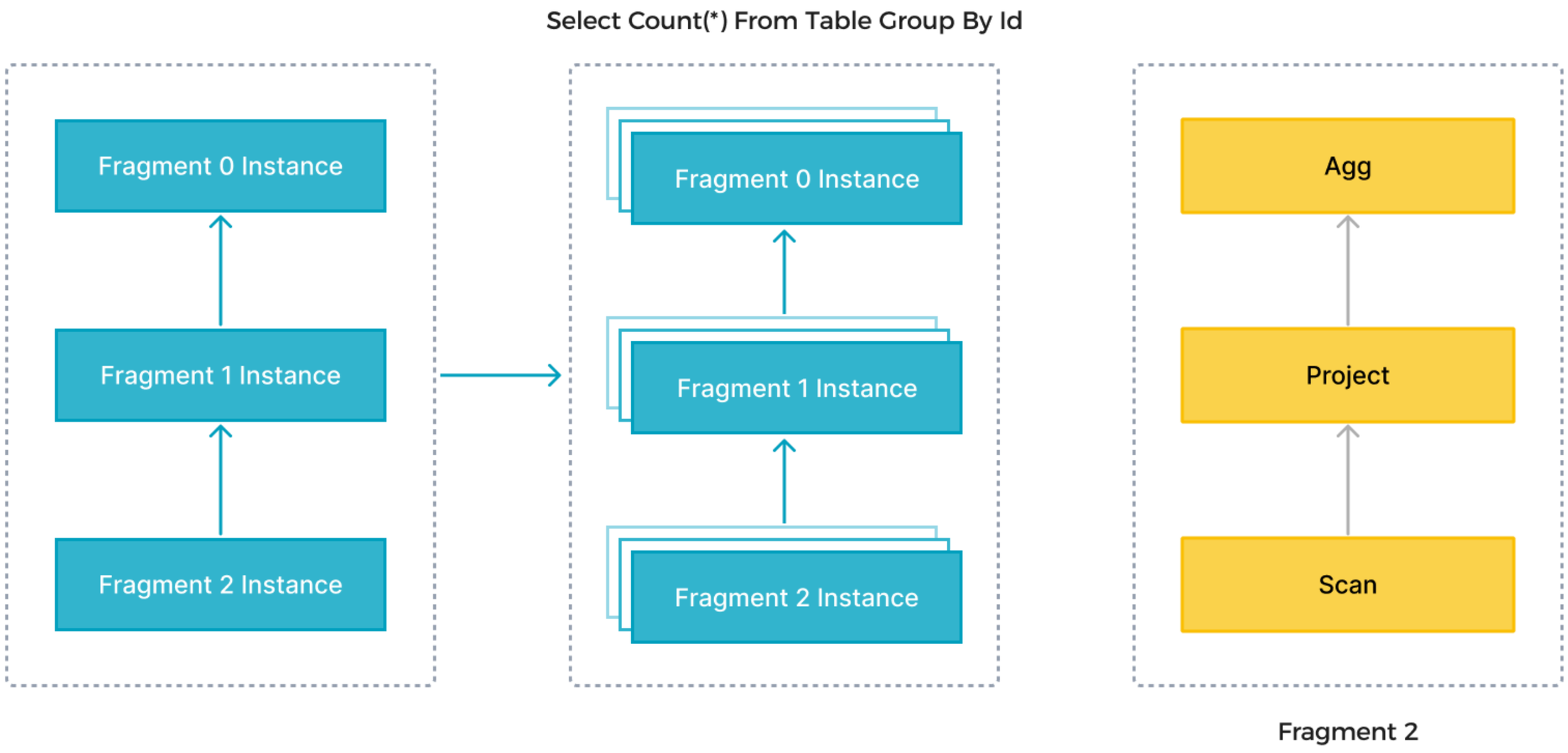
In a decoupled architecture, data is stored in a distributed object storage system (e.g., Amazon S3, GCS, or Azure Blob), while compute nodes remain stateless and are independently managed. These clusters access shared datasets through a centralized metadata layer, but do not own or persist with any data themselves.

- **Storage and compute scale independently,** allowing infrastructure to grow or contract based on workload.
- **Multiple compute services**—such as ingestion pipelines, analytics engines, and model inference—can access the same data concurrently.
- **No data movement or rebalancing** is required when compute nodes are added, removed, or reassigned.

This separation improves elasticity, simplifies failure recovery, and avoids resource contention associated with tightly coupled or shared-nothing systems.

Massively Parallel Processing (MPP)

To meet the throughput and concurrency demands of customer-facing analytics, systems must execute queries in a distributed fashion. Massively Parallel Processing (MPP) architecture achieves this by decomposing each query into fragments that are executed concurrently across multiple nodes. Each node processes a subset of the data independently, without shared state.



- **Horizontal scalability** — Processing capacity increases with node count, allowing systems to scale out to meet high-volume and high-concurrency workloads.
- **Distributed joins and high-cardinality aggregations** — MPP enables efficient execution of operations that are difficult to scale in centralized or scatter–gather models. Joins and aggregations over large, skewed, or multi-tenant datasets can be parallelized across nodes, avoiding bottlenecks and preserving latency guarantees.

These properties make MPP especially well-suited for customer-facing analytics, where queries often involve grouping or joining large datasets under strict latency constraints.

Pipeline Execution Within Nodes

In modern analytical engines, query execution within each node is organized as a non-blocking pipeline. Instead of materializing intermediate results between operators, data flows continuously through in-memory buffers—scan, filter, join, and aggregate stages remain active throughout execution.

This model improves overall throughput and reduces latency at each stage. More critically, it avoids coordination barriers and inter-operator pauses that often cause tail latency spikes under concurrent load. By keeping all operators active and eliminating idle transitions, pipelined execution helps maintain stable performance as query volume and complexity increase.

Combined with MPP, pipelined execution ensures that the system maintains steady performance as data volume and workload complexity grow.

Optimized for Low Latency and High Concurrency

Customer-facing analytics must deliver sub-second responses under unpredictable, high-concurrency workloads. To meet these demands, systems need efficient CPU utilization at the operator level and adaptive query planning that avoids reliance on brittle, precomputed pipelines.

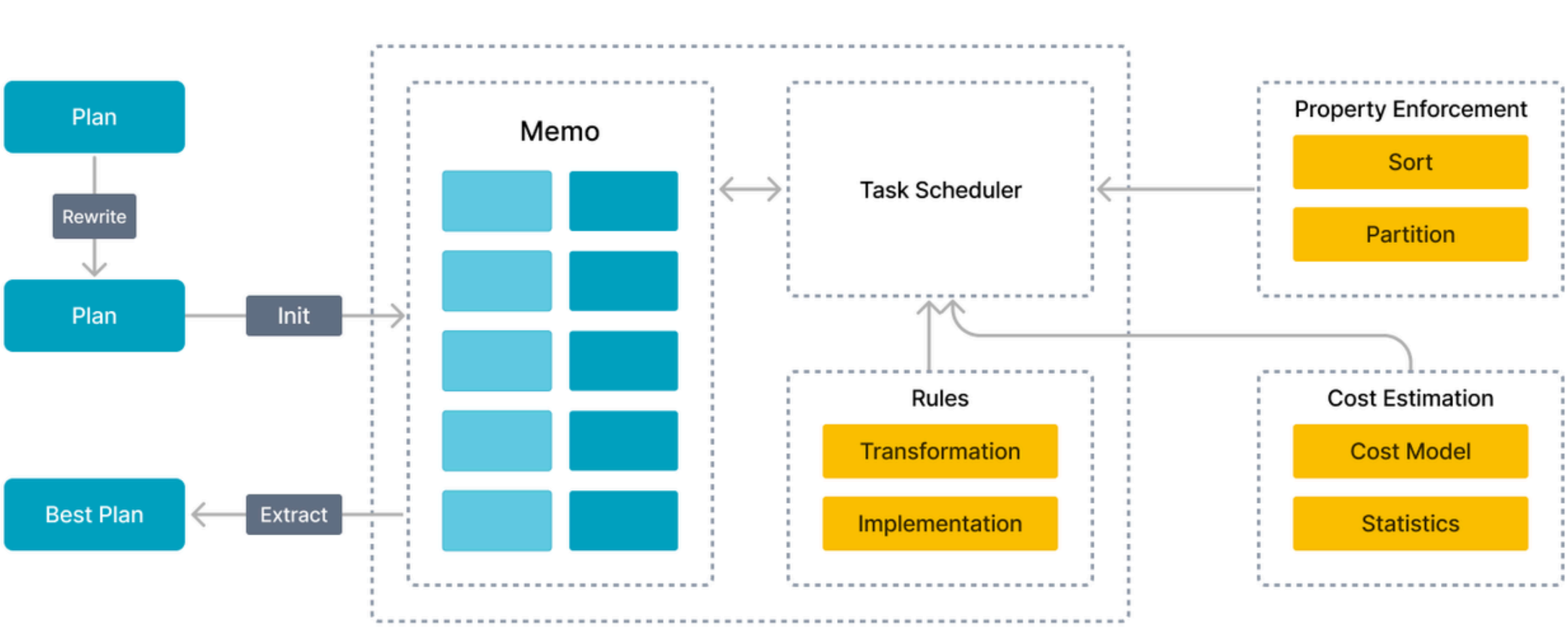
Efficient CPU Utilization through Vectorized Execution

To meet sub-second latency targets at scale, the query engine must maximize CPU efficiency across a wide range of input sizes and query shapes. Vectorized execution addresses this by processing data in columnar batches, allowing each operator to execute across thousands of values per instruction cycle.

1. **Maximizes throughput per core** – SIMD-based operators reduce CPU cycles, enabling more queries to execute concurrently.
2. **Reduces per-query overhead** – Fewer function calls and memory branches improve stability under variable workloads.
3. **Improves latency predictability** – Execution cost scales smoothly with data volume and filters, minimizing p95/p99 outliers.

On-the-fly Joins & High-Cardinality Aggregations via Cost-Based Optimization

Many systems handle customer-facing queries by denormalizing data up front to avoid runtime joins. While avoiding on-the-fly joins, this approach introduces long-term complexity: preprocessing pipelines delay data freshness, schema changes necessitate costly backfills, and storage usage increases rapidly.



To avoid these trade-offs, customer-facing analytics systems must support **efficient joins and aggregations at query time.** This requires a **cost-based optimizer (CBO)** that selects the optimal join order and aggregation strategy based on data statistics and filter selectivity. At runtime, the engine applies **runtime filters** to eliminate non-matching rows early, reducing shuffle and I/O cost.

These techniques make it feasible to serve customer-facing workloads over normalized data, even when queries involve high-cardinality group-by operations and dynamic filtering, without compromising latency or reliability.

Consistent Performance Under Load

Even with scalable execution and low average latency, real-world workloads can still produce unpredictable outliers, especially under bursty or skewed access patterns. This section focuses on the system-level mechanisms that ensure stability at the tail.

Low-Latency Reads via Robust Data Caching

To maintain stable query latency at scale, systems must avoid hitting object storage for every request. A well-designed caching hierarchy enables fast reads while preserving performance isolation.

- **Segmented cache:** Hot data blocks are cached on local SSDs and divided into segments based on priority or access pattern. This prevents large scans from evicting data needed by frequent, lightweight queries, which are critical for heterogeneous workloads.
- **Proactive warm-up:** Preloads expected access paths after cluster restarts or deployments to minimize cold start latency.

Consistent SLAs through Physical Resource Isolation

In customer-facing environments, workloads often originate from many tenants with differing usage patterns and data volumes. To maintain consistent query latency, compute resources must be physically isolated.

A common and practical approach is to deploy **multiple dedicated compute warehouses,** each operating as an independent cluster over a shared storage layer. These warehouses access the same catalog and object store, but enforce strict boundaries in resource allocation:

- **Dedicated resource envelopes:** CPU, memory, and concurrency limits are scoped per warehouse, preventing noisy-neighbor effects.
- **Tenant isolation:** High-volume or latency-sensitive tenants can run in separate environments to protect SLA targets.
- **Workload isolation:** Analytical queries, background compaction, and data ingestion can be isolated across different clusters to prevent resource contention.

This architecture avoids the interference common in shared infrastructure models, enabling predictable performance with heterogeneous workloads.

CelerData Solution Overview

CelerData is a high-performance analytical engine purpose-built for customer-facing analytics. It is based on the StarRocks open-source project and extends it with enterprise capabilities for the cloud. CelerData is designed to serve real-time, high-concurrency analytics workloads directly to end users, whether embedded in a SaaS product, exposed through APIs, or powering self-serve dashboards.

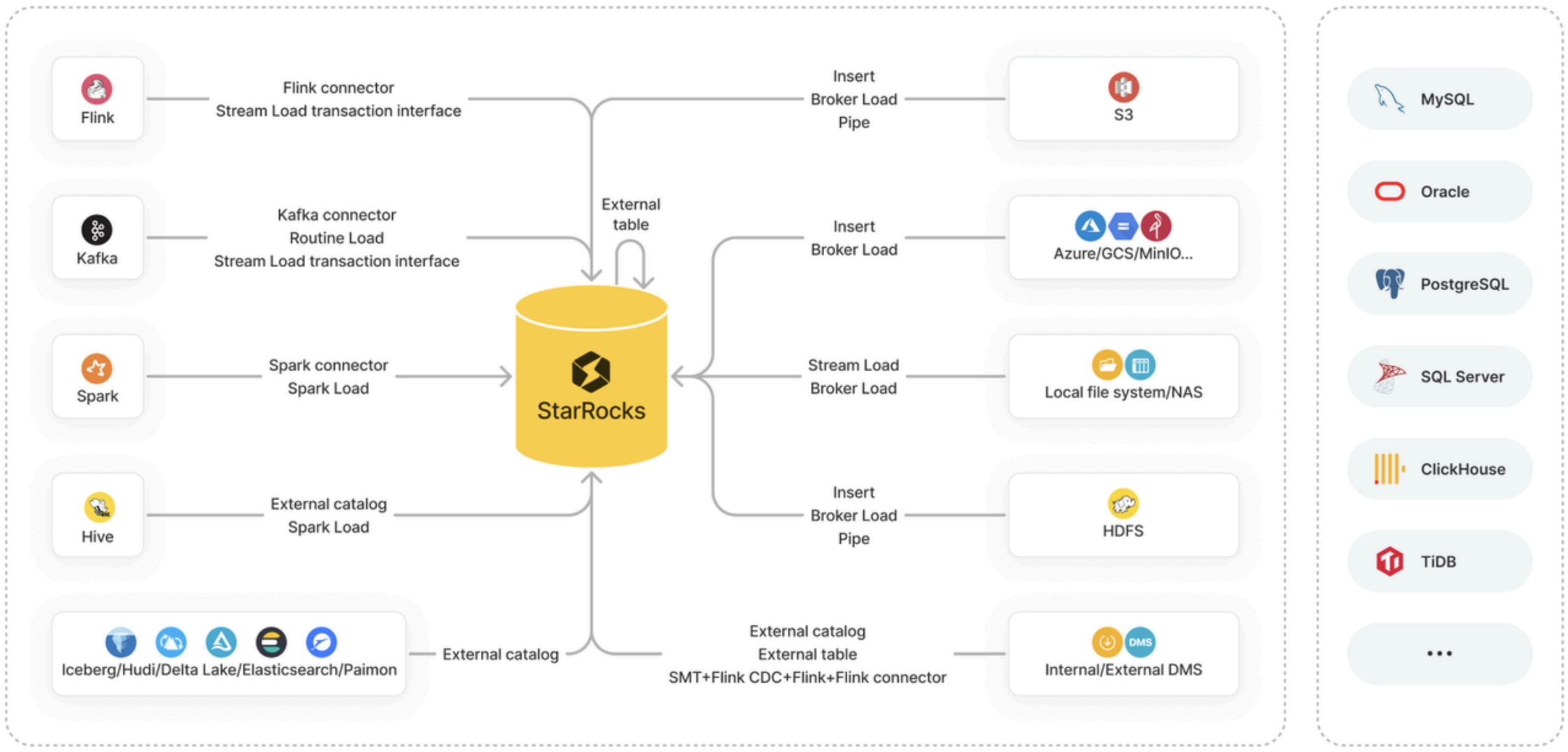
This section outlines how CelerData addresses these requirements through a specialized architecture that strikes a balance between performance, scalability, and governance. Instead of forcing every workload through a single processing path, CelerData supports two reference architectures optimized for distinct freshness and governance needs:

- **A real-time architecture** designed for frequently updated datasets that targets < 10-second data freshness
- **A lakehouse-native architecture** optimized for query acceleration on open-format tables, where freshness is measured in minutes, and governance is a priority

Real-Time Reference Architecture: Optimized Internal Tables

The real-time reference architecture is designed for applications that demand sub-second latency and second-level data freshness. In this mode, data is ingested into StarRocks' internal tables using its optimized storage format, in real-time, and served directly to end-users.

Ingestion Architecture



CelerData supports multiple streaming ingestion patterns—designed for different scalability and freshness requirements—while ensuring managed delivery and fault tolerance:

Routine Load (Pull-based via Kafka/Pulsar)

CelerData continuously pulls data from topics in Kafka or Pulsar using its built-in Routine Load. This approach ensures exactly-once delivery, configurable with batch size and concurrency.

Kafka Connect (Push-Based CDC)

For change data capture (CDC) from upstream databases, Kafka Connect enables external systems to push change events into Kafka topics. These events are then ingested by CelerData through its Kafka connector, supporting flexible data formats (e.g., Avro, JSON, Protobuf) and tunable parameters for latency, batching, and throughput.

Flink Connector (Stream Transformation and CDC)

For use cases needing stream processing (e.g., joins or windowing), CelerData provides a Flink connector. This enables ingestion via Flink jobs, supporting complex pipeline logic and idempotent writes into tables.

Real-Time Upsert With Primary Key Tables

IMany customer-facing analytics workloads require frequent, low-latency updates, such as tracking user sessions, real-time pricing, or campaign metrics. Traditional columnar engines often struggle in these scenarios, as they are optimized for append-only data and lack efficient mechanisms for point updates or overwrite semantics.

CelerData addresses this through native support for primary key tables. Updates are applied via a write-optimized delta layer, which is periodically merged into the base columnar storage in the background. This design maintains the compression and scan efficiency of a columnar engine, while enabling high-throughput upserts.

Key benefits for customer-facing systems include:

1. **Second-level freshness** even for update-heavy workloads
2. **Storage efficiency** without reverting to row-based engines
3. **Simplified ingestion pipelines** for workloads that require overwrite semantics or de-duplication

This model is especially useful for applications that ingest streaming data, such as user activity, ad events, or rapidly changing product states, while still demanding low-latency query performance on up-to-date results.

Performance-Optimized Storage

To ensure consistently fast queries, StarRocks internal tables rely on StarRocks' optimized storage engine, which includes features such as:

- **Partitioning:** StarRocks internal tables allow advanced partitioning strategies. Both range and list partitioning are supported, along with secondary partitioning, for example, by tenant ID and time, to address data skew across tenants. This ensures efficient pruning, balanced parallelism, and stable performance, even when query volume and data distribution vary widely across users.
- **Secondary Indexes:** StarRocks implements bitmap and inverted indexes to accelerate high-selectivity filters. Bitmap indexes are ideal for low-cardinality columns, while inverted indexes support keyword search and tokenized values.
- **Sorting (Clustering Keys):** StarRocks physically sorts data within each tablet based on a clustering key, reducing the number of rows read and improving performance for range queries, merge joins, and sort-based aggregations.
- **Bucketing:** Rows can be evenly distributed into hash buckets to balance IO and improve join efficiency. Bucketing is particularly effective when joining large fact tables with small dimension tables.
- **Low-Cardinality Optimization with Dictionary Encoding:** StarRocks applies dictionary encoding to low-cardinality columns, replacing repeated values with compact integer codes. This reduces storage, improves scan efficiency, and accelerates predicate evaluation—particularly for common filters on user-facing dimensions.

These optimizations not only lower query latency but also maintain stable performance under high concurrency and data skew, which are critical for real-time, customer-facing applications.

Lakehouse-Native Reference Architecture

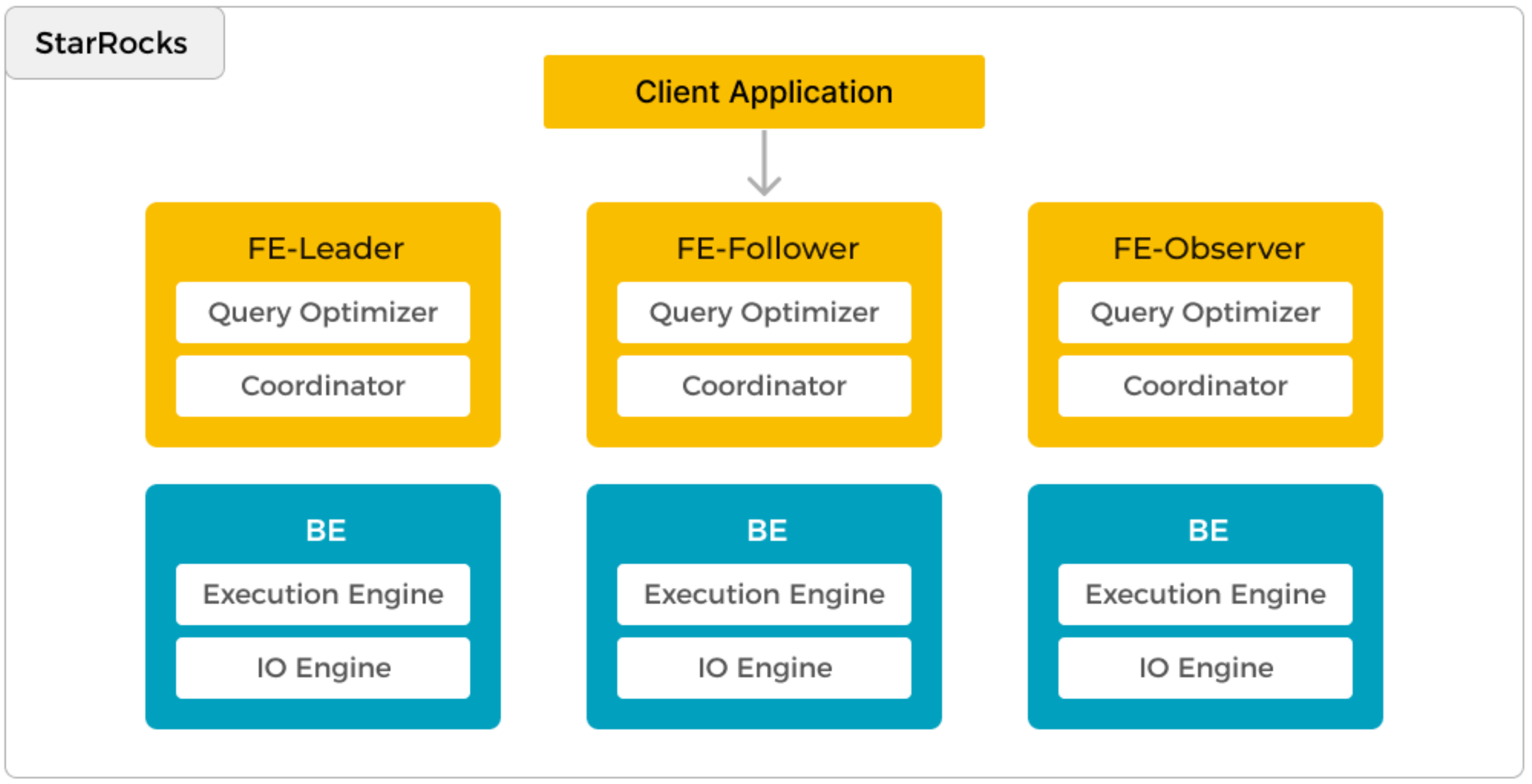
For organizations prioritizing governance, cost-efficiency, and simplicity, the lakehouse-native mode allows CelerData to serve customer-facing workloads directly from open table formats like Apache Iceberg—without ingestion, ETL, or duplication.

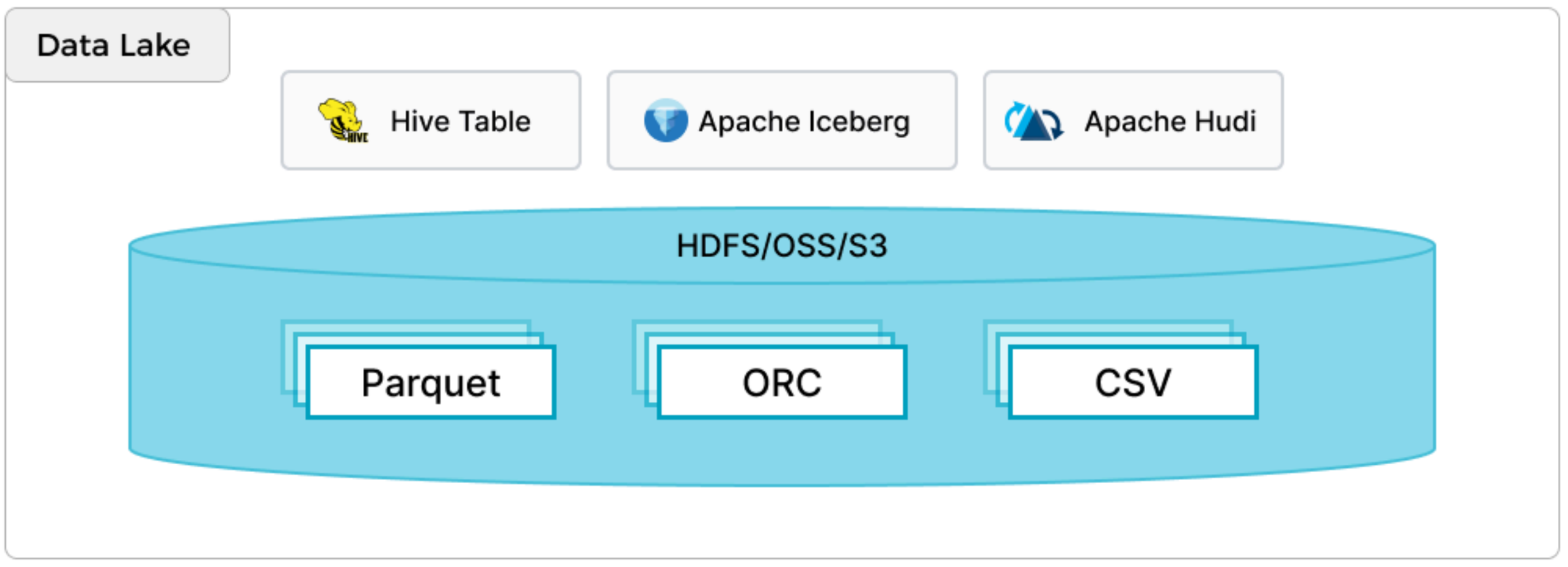
This architecture is built around a shared data lake and consists of the following components:

- **Cloud Object Storage (e.g., Amazon S3, Google Cloud Storage, Azure Blob)** — A static, durable object store that holds the table's physical data files (Parquet or ORC). Objects are immutable and versioned, allowing rollback and cross-region replication.

- **Open Table Format (Apache Iceberg / Delta Lake / Hudi)** — A logical layer that maps objects to tables, handles schema evolution, partitioning, and ACID snapshot isolation.
- **Catalog Service (Unity Catalog, AWS Glue, Hive Metastore)** — A central metadata repository that stores table definitions, statistics, and access policies; every query engine consults the catalog for authoritative metadata.
- **Compaction & Metadata Jobs** — Scheduled services that merge small files, rewrite manifests, and refresh column statistics so planners can generate efficient query plans.
- **CelerData Compute Warehouse (Stateless StarRocks Clusters)** — Executes vectorised SQL directly on Iceberg data. Features include distributed manifest scanning, runtime filter push-down, SSD segment cache, and automatic query rewrite to materialised views. Warehouses scale elastically, isolate tenants via resource groups, and can be suspended to zero during idle windows.

CelerData connects to the catalog and storage directly. It pushes down filters and projections, prunes partitions, and parallelizes metadata scanning. Most importantly, it allows queries to run **without the need to maintain a copy of the data in a proprietary system**, ensuring a single source of truth across platforms.





Accelerating Lakehouse Queries with Lakehouse Map Materialized Views

Table formats like Iceberg are well-suited for governance, schema evolution, and multi-engine interoperability. However, they are not optimized for the low-latency, high-concurrency workloads typical of customer-facing analytics. Querying directly from object storage can introduce unpredictable performance due to metadata traversal, small file overhead, and limited storage-layer optimizations.

To address this, CelerData Cloud introduces a Lakehouse map materialized view pattern: a physically optimized representation of the most active portion of a lakehouse table, maintained within the query engine's native storage format.

What Is a Lakehouse Map Materialized View?

CelerData's **materialized views** provide a way to cache frequently accessed query results or table fragments in the engine's native storage format, with support for automatic refresh and transparent query rewrite. When applied to external tables, such as Iceberg, this mechanism enables a performant execution layer without duplicating pipelines or compromising governance.

A **Lakehouse Map materialized view** is a direct projection of a lakehouse table, mirroring its schema and partitioning, typically limited to the most active data window (e.g., the last 30 to 90 days). Unlike aggregated views, a 1:1 MV retains every row and column in scope, but stores them in the native StarRocks format. This enables the engine to fully leverage internal optimizations—such as partitioning, dictionary encoding, secondary indexes, and bucketing—without duplicating the entire dataset.

These views are incrementally maintained, and automatically rewritten at query time with 100% rewrite consistency, meaning no application or dashboard logic needs to change. Queries that reference the base tables (such as Apache Iceberg) are intercepted by the planner and redirected to the MV whenever the relevant partitions are available.

Because the MV typically retains only recent partitions, storage cost remains bounded, and optimizer statistics can be refreshed more frequently over a smaller, denser working set. This improves the quality of query plans—particularly for joins and filters—further reducing tail latency.

Importantly, the Iceberg table remains the single source of truth. Any data not found in the MV is automatically read from the lake and unioned at runtime, preserving correctness while optimizing for the most latency-sensitive range of queries.

This pattern offers a practical bridge between open governance and performance—allowing systems to serve customer-facing analytics workloads on lakehouse data without maintaining redundant ETL pipelines or full-table copies.

Typical Use Cases

1. **Recent activity dashboards:** Powering customer-facing views over the last 7–90 days of user, transaction, or engagement data.
2. **Multi-tenant reporting:** Serving recent analytics for hundreds or thousands of tenants with strong performance isolation.
3. **Ad-hoc filtering:** Supporting interactive queries across large, partitioned tables with unpredictable filter patterns (e.g., by region, product, time).
4. **Time-windowed joins:** Accelerating recent joins between fact tables and slowly changing dimensions such as user attributes or campaign metadata.

Case Studies

Herdwatch Enables Customer-Facing Analytics Using Apache Iceberg And CelerData Cloud

Herdwatch is a farm management SaaS platform that enables livestock farmers across Europe to log compliance records, track animals, and manage daily operations via web and mobile apps. As the platform expanded internationally, its legacy architecture—based on isolated MySQL RDS instances per region—began to show critical limitations.

Key Challenges

- **Siloed Analytics:** Each region had its own database, which meant users had to consult separate dashboards for insights. Unified reporting was impossible.
- **Performance Bottlenecks:** Dashboards querying RDS replicas were slow, prone to timeouts, and lacked scalability — making them unsuitable for customer-facing dashboards.

The Solution

Herdwatch initially used Athena to query data stored in Apache Iceberg, but as performance and cost limitations emerged, the team rebuilt its analytics architecture using CelerData's lakehouse-native approach—retaining Iceberg for governance while adopting CelerData Cloud to meet performance and scalability requirements.

- **Centralized Data Lake:** All regional pipelines now flow into a unified Iceberg data lake on AWS S3.
- **ETL Pipelines:** AWS Glue and DBT transform data into bronze, silver, and gold layers, streamlining aggregation and optimization.
- **StarRocks as Query Layer:** StarRocks dramatically accelerates queries on the gold layer, powering both internal BI tools and customer-facing applications.
- **Materialized Views:** Every major table was accelerated using 1:1 materialized views. For complex queries, transformations were also applied with the materialized views, ensuring optimal performance.
- **Customer-Facing Dashboards:** Dashboards directly query StarRocks via MySQL-compatible APIs, delivering sub-second response times.

Results

- **Unified analytics:** All regional pipelines now write into a shared lakehouse catalog, enabling cross-region reporting and consistent governance.
- **Performance improvement:** Dashboard latency dropped to 700 ms–1.5 seconds, even under load.
- **Improved cost-efficiency and scalability:** By replacing Athena’s per-scan pricing model with materialized views and caching, Herdwatch aligned compute cost with actual usage patterns—supporting long-term growth without linear increases in query cost.
- **Real-time freshness:** Pipelines now reflect recent operational data without complex backfills.

By modernizing its architecture around a decoupled, lakehouse-native analytics stack, Herdwatch delivered faster, more consistent insights directly to its users—without sacrificing flexibility or cost efficiency.

Pinterest

Pinterest serves over 500 million monthly users and thousands of advertisers, offering real-time campaign metrics via its Partner Insights platform. As data volume and analytical complexity grew, the company encountered scaling limits with its previous architecture, which was built on Apache Druid.

Key Challenges

- **Latency and cost:** Multi-dimensional, user-defined dashboards suffered from high query latency and rising compute costs.
- **Limited flexibility:** Druid's restricted SQL support slowed development for internal analytics teams.
- **Complex ingestion:** Data pipelines required external MapReduce jobs and JSON-based configuration, increasing maintenance overhead.
- **Denormalization overhead:** The lack of efficient join support resulted in redundant data copies and slower iteration cycles.

The Solution

To address these issues, Pinterest migrated its analytics backend to StarRocks. The new architecture included:

- **A fully SQL-native engine** supporting real-time ingestion and rich analytical queries
- **A custom-built middleware service (Archmage)** to translate application requests into optimized SQL
- **Direct ingestion pipelines**—eliminating the need for external ingestion config and preprocessing
- **A scaled deployment on AWS** with 70 back-end nodes and 11 front-end nodes using FE/BE separation and load balancing
- **MySQL compatibility** to support existing internal tooling

Results

- **50% reduction in p90 query latency** with only **32% of the prior hardware footprint**
- **3× improvement in cost-efficiency,** enabling better concurrency at lower operational cost
- **10-second data freshness,** powering real-time campaign feedback loops for advertisers
- **Simplified engineering workflows,** accelerating onboarding, and new metric delivery

By replatforming to a real-time, SQL-native engine with efficient join support, Pinterest delivered faster insights to advertisers while reducing infrastructure complexity and cost.

Demandbase Ditches Denormalization By Switching off ClickHouse

Demandbase, a leading B2B go-to-market platform, provides advanced analytics and account intelligence to thousands of enterprise customers. As usage expanded, its existing ClickHouse-based architecture struggled to keep pace with growing data volumes and concurrency requirements.

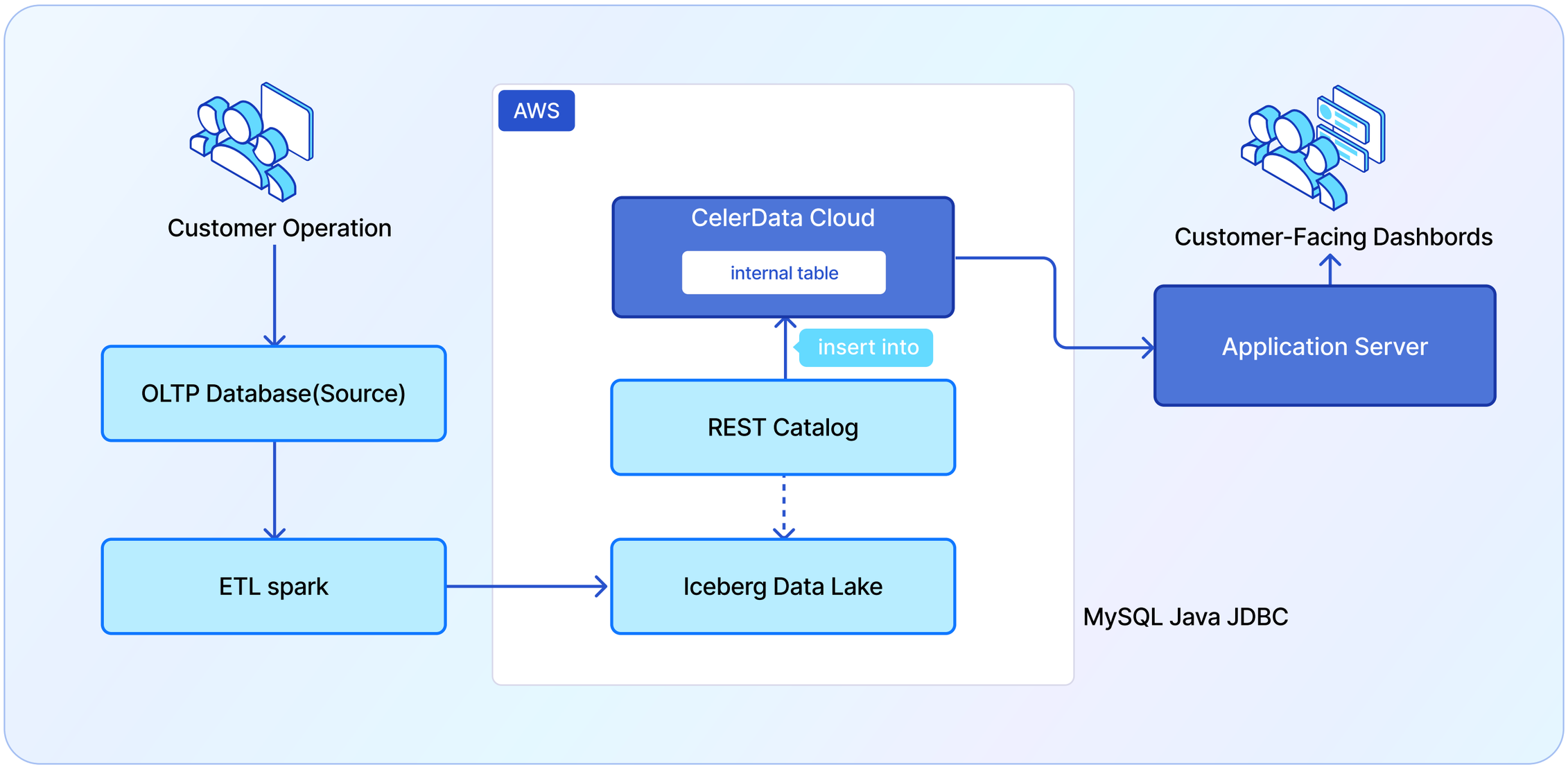
Key Challenges

- **Inability to scale joins:** Full denormalization of all data was required because ClickHouse lacked efficient join support at scale, which created added overhead to the infrastructure.
- **Storage inefficiency:** Denormalized tables resulted in a 10-fold increase in storage footprint.
- **ETL limitations:** Complex pipelines made real-time data freshness unachievable, and schema changes required costly data backfills.

- **Operational complexity:** Scaling ClickHouse requires manual data balancing. Demandbase ultimately maintained 49 overprovisioned ClickHouse clusters, which increased costs and management burden.

The Solution

Demandbase modernized its analytics architecture by adopting **Apache Iceberg** as its central data lakehouse and **CelerData Cloud** as the high-performance engine that serves customer-facing queries.



Key architectural benefits included:

- **Apache Iceberg data lakehouse:** Demandbase's centralized data store and source of truth for analytics data
- **ETL Pipelines:** Data flows from OLTP databases serving production systems through ETL pipelines, undergoing cleaning, transformation, and selective pre-computations before being ingested into Apache Iceberg.
- **CelerData Cloud:** Data is ingested into CelerData Cloud, which is in CelerData's optimized format for performance.
- **Customer-facing applications:** CelerData Cloud serves customer-facing applications directly via a MySQL connector, offering second-level query latencies.
- **Greatly reduced the need for denormalization:** CelerData Cloud's efficient JOIN performance enables denormalization only for specific use cases, unlike the previous ClickHouse solution that required it for all data, leading to significant savings in storage and resources.

Results

- **Cluster Costs:** The original ClickHouse deployment that consisted of 49 clusters of 3 nodes per cluster was replaced with a more efficient 45-node StarRocks cluster, resulting in a 60% reduction in hardware resources.
- **Storage Costs:** By dramatically reducing denormalized data, Demandbase reduced storage costs by 90%.
- **ETL Costs:** Eliminating the need for heavy ETL pipelines to maintain denormalized data simplified their data pipeline and reduced the associated operational burden.

These improvements enabled Demandbase to achieve a more scalable, cost-effective infrastructure without sacrificing query performance or data freshness for their customers.

How CelerData Can Help

The architectural patterns and techniques discussed—decoupled storage and compute, vectorized execution, multi-level caching, materialized views, and workload isolation—form the foundation for building customer-facing analytics systems that are fast, consistent, and scalable.

CelerData implements these patterns as first-class capabilities. Built on open-source StarRocks, it allows engineering teams to meet strict SLA requirements without maintaining complex, hand-tuned pipelines.

If you're designing analytics for external users, CelerData provides a practical path to production. Learn more at celerdata.com or try CelerData Cloud for free at cloud.celerdata.com.

CelerData | Powered by StarRocks

CelerData Cloud is an analytics service that delivers data warehouse performance on the data lake. CelerData shortens your time for production by eliminating external data transformation and the need to ingest data into a database for query acceleration. With its performance, CelerData meets business and application latency and concurrency requirements with fewer computing resources.

Learn about CelerData

Join us on Slack

Try CelerData Cloud at cloud.celerdata.com and claim your credit card-free 30-DAY FREE TRIAL.