Understanding Cache Service & Object Storage in Starburst

Welcome to this comprehensive exploration of the Starburst caching ecosystem. Throughout this presentation, we'll dive deep into the intricate relationship between cache services and object storage within the Starburst Enterprise Platform (SEP).

We'll examine how cache services dramatically improve query performance, explore the differences between embedded and standalone cache implementations, and provide clear guidance on when to use each option based on your specific requirements and operational constraints.

Let's begin by understanding the fundamental components and how they work together to accelerate your data analytics environment.

A **by Atish Kumar Sinha**

Object Storage in Starburst

Amazon S3

The most widely used object storage service, offering excellent scalability and durability for Starburst implementations. Provides fine-grained access controls and regional availability.

Azure Data Lake Storage (ADLS)

Microsoft's enterprise-grade storage solution optimized for analytics workloads. Integrates seamlessly with other Azure services and provides hierarchical namespace functionality.

Google Cloud Storage (GCS)

Offers unified object storage with automatic encryption and strong consistency models. Well-suited for multi-regional Starburst deployments with global data access needs.

These object storage systems serve as the foundation for many Starburst implementations, providing elastic capacity for massive datasets. Typical analytics patterns include data lake queries, ETL workflows, and serving as source repositories for materialized views.

What is the Cache Service?

Definition

The Cache Service is a core component of Starburst Enterprise Platform that manages data caching from slower object storage systems to high-performance local or network storage to accelerate query execution.

Core Purpose

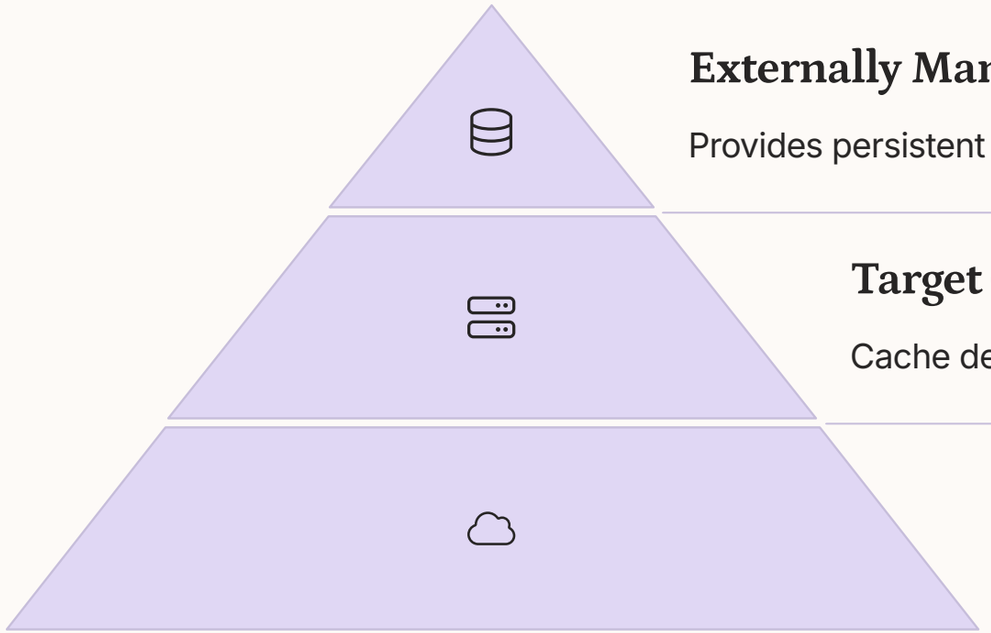
It dramatically improves query performance by reducing I/O bottlenecks associated with object storage, particularly for frequently accessed data that would otherwise require repeated remote access.

Key Functions

The service handles transparent redirection of queries to cached data, automated synchronization to maintain data freshness, and optimized data placement for maximum performance benefit.

By maintaining a local high-performance copy of remote data, the Cache Service creates a performance layer that shields users from the latency and throughput limitations inherent in distributed object storage systems, all while maintaining data consistency.

High-Level Cache Architecture



Externally Managed Database

Provides persistent storage for cached data

Target Catalog

Cache destination with optimized structure

Source Catalog

Original data in object storage

The Starburst cache architecture fundamentally separates compute from storage, allowing each to scale independently. The source catalog connects to your object storage (S3, ADLS, etc.), while the target catalog represents the high-performance cache destination.

This design leverages an externally managed database system that serves as the physical storage for cached data, providing ACID compliance and transactional integrity that basic object storage typically lacks.

How Cache Service Interacts with Object Storage

Read from Source

Query engine scans object storage

Transform & Move

Data is optimized and transferred

3

Write to Cache

Optimized format in target catalog

Redirect Queries

Subsequent queries use cached data

When interacting with object storage, the Cache Service first analyzes the data structure in the source catalog. It then initiates a controlled data movement process, transferring data from object storage to the target catalog while applying optimizations like compression and columnar formatting.

FFor example, when caching an S3 dataset, the service might copy Parquet files, convert them to an optimized ORC format with predicate pushdown capabilities, and store the ORC files in a separate object storage location for accelerated access.

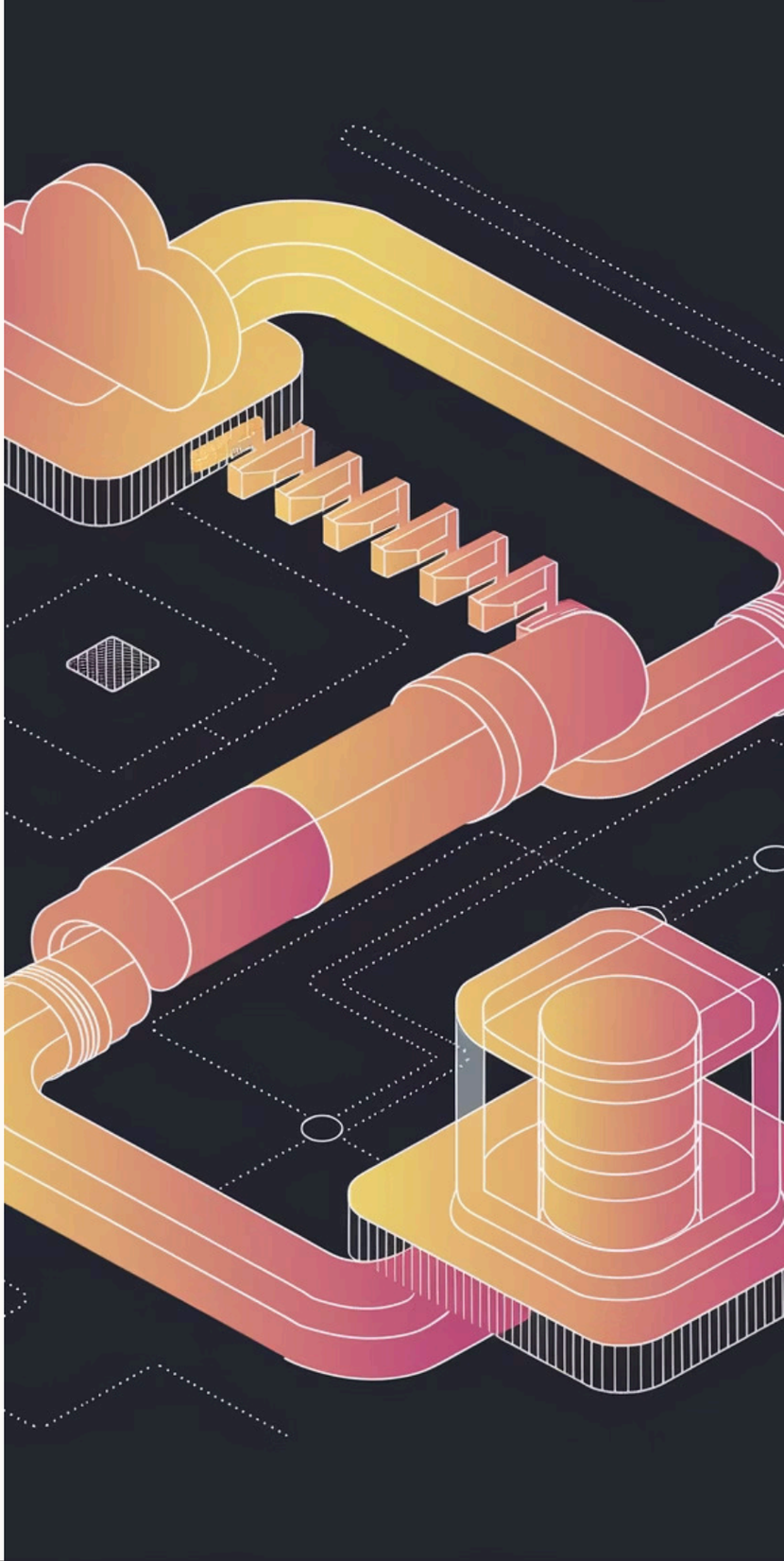


Table Scan Redirection Explained

Query Submission

User submits query referencing tables in object storage catalog without knowing about cache presence.

Redirection Check

Query optimizer automatically detects cached version availability and freshness status.

Transparent Routing

Without modifying the original query, execution is silently redirected to use cached data.

Table scan redirection is the mechanism that makes caching transparent to end users. When a query references a table in object storage, Starburst's query optimizer automatically checks if a valid cached copy exists. If found, the query execution is silently redirected to use the cached version instead.

This redirection dramatically reduces both I/O costs and execution time, as the query no longer needs to scan potentially thousands of files across distributed storage. The entire process is configurable through session properties and catalog-level settings.

Materialized Views: Fundamentals

Precomputed Query Results

Materialized views store the results of complex queries as physical data, unlike regular views that recalculate each time.

Automated Query Rewriting

Starburst can automatically rewrite incoming queries to use matching materialized views when applicable.

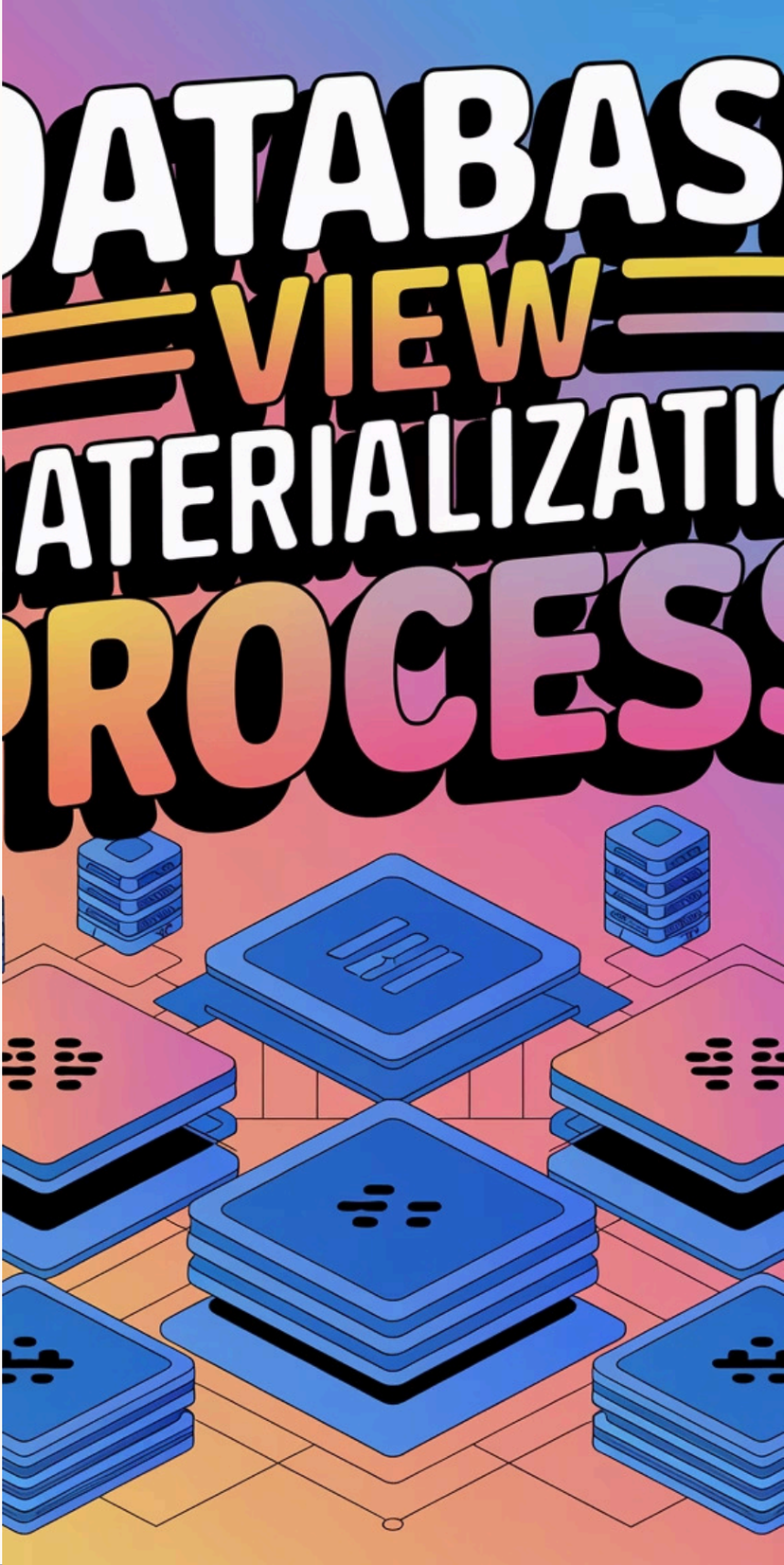
Configurable Refresh Policies

Views can be refreshed on schedule, manually triggered, or based on source data changes to maintain accuracy.

Optimized Storage Format

Data is stored in performance-optimized formats with statistics and indexes for fastest possible access.

Materialized views in Starburst represent a powerful performance optimization technique. By precomputing and storing the results of expensive queries, they eliminate the need to repeatedly process the same transformations and aggregations, particularly beneficial for complex joins and aggregations over large datasets.



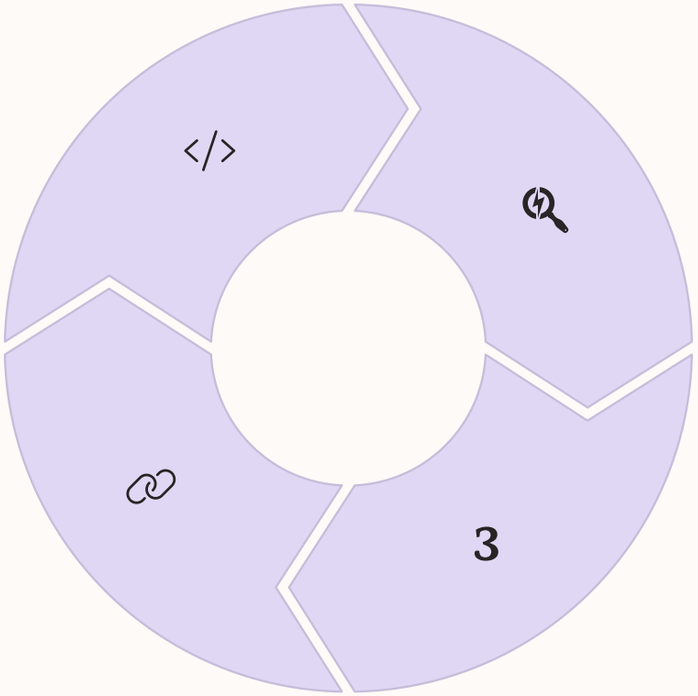
Lifecycle: Creating a Materialized View

Define View

SQL CREATE MATERIALIZED VIEW statement

Register Metadata

View definition stored in system catalog



Initial Population

Cache service executes query against source

Store Results

Data persisted to target catalog

When creating a materialized view, the Cache Service acts as the orchestrator between object storage and the target cache. First, the user defines the view with SQL, typically selecting from tables in object storage. The Cache Service then executes this query against the source, optimizing data retrieval patterns.

As results are generated, they're stored in the target catalog, often in a specialized format optimized for the specific query patterns. Throughout this process, object storage serves as the authoritative data source, while the Cache Service handles data transformation and placement for optimal query performance.

Automatic Synchronization and Refreshing

Schedule-Based Refresh

Configure specific timeframes (hourly, daily, etc.) when materialized views should be updated to reflect the latest source data changes.

Change Detection

When supported by the source catalog, the Cache Service can monitor for data modifications and trigger refreshes automatically when changes are detected.

Incremental Updates

For supported query patterns, only the changed portion of data is processed during refresh operations, significantly reducing resource utilization.

Keeping cached data fresh is critical for maintaining query accuracy. The Cache Service offers multiple synchronization strategies to balance freshness with performance. Scheduled refreshes can be configured using cron-like syntax to update materialized views during specific maintenance windows.

In data-intensive environments, incremental refreshes allow the service to process only the delta changes from object storage, dramatically reducing the refresh overhead. This is particularly valuable for large datasets where complete refreshes would be prohibitively expensive.

Manual Refresh Commands and API Usage

Interface	Command Example	Use Case
SQL	REFRESH MATERIALIZED VIEW schema.view_name	Ad-hoc refreshes from SQL clients
REST API	POST /v1/cache/refresh?name=schema.view_name	Programmatic refresh integration
CLI	starburst-cli cache refresh schema.view_name	Scripted operations and automation

For manual control over materialized view refreshes, Starburst provides multiple interfaces. SQL commands offer the most straightforward approach for database users, allowing refreshes to be triggered directly from query tools. For automation scenarios, the REST API enables programmatic refresh operations that can be integrated with data pipelines and orchestration tools.

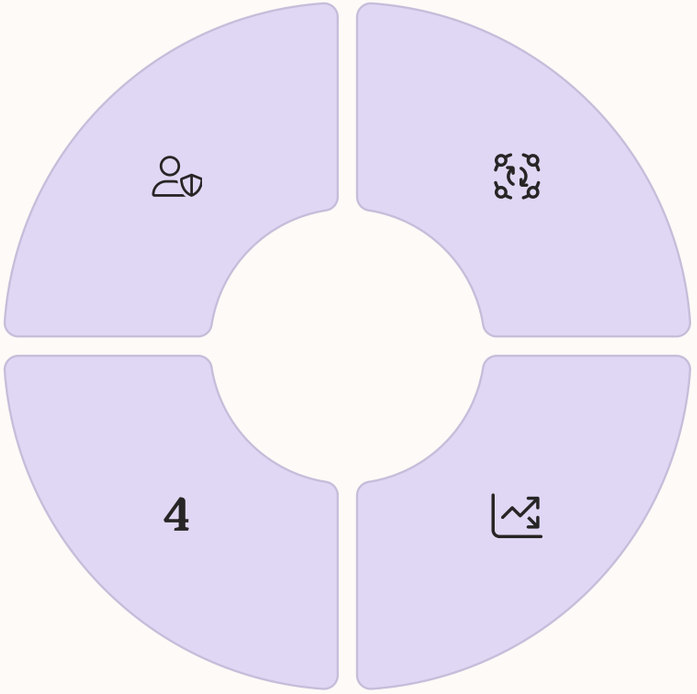
Command-line options provide a middle ground, suitable for scripting and scheduled jobs. Each approach offers parameters to control refresh behavior, such as forcing full refreshes or setting timeout limits for long-running operations.

To perform incremental refreshes, you can use the `REFRESH MATERIALIZED VIEW INCREMENTAL` SQL command or the `POST /v1/cache/refresh?name=schema.view_name&incremental=true` REST API endpoint. This will only update the materialized view with the changes since the last refresh, rather than a full rebuild.

The Cache Service User Role

View Definition Management

Create, alter, and drop materialized view definitions



Refresh Operations

Execute manual refreshes and monitor automatic refresh jobs

Storage Management

Monitor and manage cache storage utilization

Performance Monitoring

Track cache hit rates and query acceleration metrics

The Cache Service user role requires specific permissions spanning both source and target catalogs. This specialized role typically needs read access to source data in object storage, write privileges on the target cache storage, and administrative permissions to manage view definitions and metadata.

Organizations often designate dedicated cache administrators who optimize cache performance, manage storage allocation, and ensure refresh operations complete successfully. These users monitor cache hit ratios, identify optimization opportunities, and collaborate with data engineers to maximize the value of the caching infrastructure.

End-to-End Flow: User Refreshes a Materialized View

Initiate Refresh

User executes refresh command via SQL, API, or CLI, authenticating with appropriate credentials.

Source Data Analysis

Cache Service identifies source tables, partitions, and columns that need scanning.

3 Data Transfer

Query executes against object storage, with results streamed to target cache storage.

Metadata Update

System catalogs updated with new refresh timestamp and data statistics.

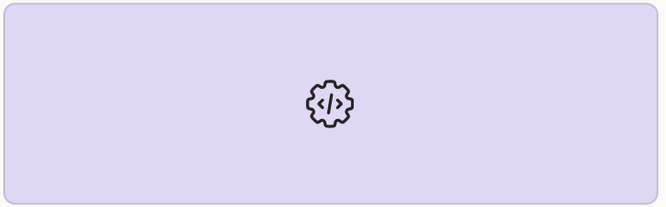
When a user initiates a materialized view refresh, the Cache Service coordinates a complex workflow across multiple systems. The process begins with authenticating the user's permissions and validating the view definition. The service then analyzes what data needs to be retrieved from object storage, potentially using file listing operations to identify specific objects or partitions.

Throughout the refresh operation, the user receives status updates and can monitor progress through system tables. Upon completion, the service updates metadata to reflect the new data freshness timestamp, ensuring subsequent queries leverage the updated cache.

Embedded Cache: What Is It?

Integrated Component

Part of the SEP coordinator process



Shared Resources

Uses same JVM memory and threads

Simplified Setup

Minimal configuration required

The embedded cache runs within the same process as the Starburst Enterprise Platform coordinator, sharing its Java Virtual Machine (JVM) and resources. This tight integration means the cache service starts and stops alongside the main Starburst process, simplifying operational management but potentially impacting performance under heavy load.

Designed primarily for development environments and small-scale deployments, the embedded cache offers a convenient way to experience caching benefits without additional infrastructure. Configuration is straightforward, requiring only minimal property settings in the SEP configuration files.

Standalone Cache: What Is It?

Independent Service

Runs as a separate process with its own dedicated resources, allowing for separate scaling and management.

Fault Isolation

Issues with the cache service won't directly impact query processing, and vice versa, improving overall system reliability.

Dedicated Resources

Allocates its own memory, CPU, and network resources without competing with the query engine.

Independent Lifecycle

Can be upgraded, restarted, or maintained separately from the main Starburst cluster.

The standalone cache operates as an independent service with its own process and resource allocation. This separation creates a clear boundary between query processing and cache management, allowing each to scale independently according to workload demands. For production environments, this isolation is crucial for maintaining service level agreements.

Communication between Starburst and the standalone cache typically occurs via a service discovery mechanism, enabling flexible deployment across different hosts or containers while maintaining connectivity.

Comparison: Embedded vs Standalone

Feature	Embedded Cache	Standalone Cache
Deployment Complexity	Low - integrated with SEP	Higher - requires separate service
Resource Isolation	None - shares with coordinator	Complete - dedicated resources
Fault Tolerance	Low - tied to coordinator uptime	High - independent availability
Scaling Flexibility	Limited - scales with coordinator	High - can scale independently
Maintenance Impact	High - requires SEP restart	Low - can be updated separately

The choice between embedded and standalone cache configurations represents a fundamental architectural decision. Embedded caches offer simplicity and quick setup but come with significant limitations in resource allocation and fault isolation. They share memory and CPU with the query engine, potentially causing resource contention under heavy load.

Standalone caches provide superior isolation and scalability at the cost of increased operational complexity. They enable independent resource allocation, separate maintenance windows, and greater fault tolerance, making them the preferred choice for production deployments.

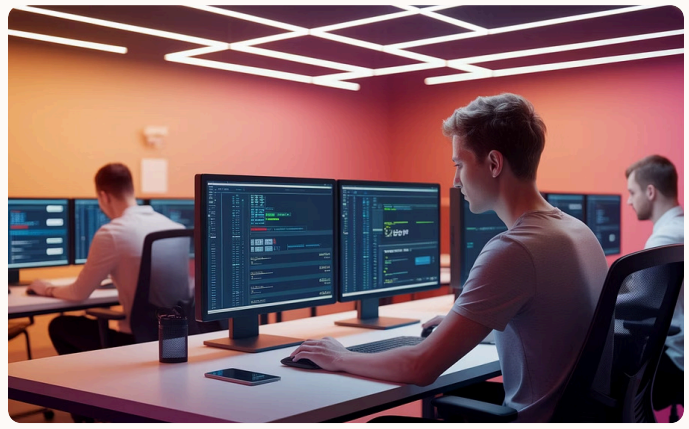
Choosing Embedded vs Standalone: Key Criteria

Embedded Cache: Development Scenarios

Choose embedded cache for development environments, proof-of-concept deployments, and small-scale testing. This configuration minimizes infrastructure requirements and simplifies setup, making it ideal for individual developers or small teams exploring Starburst's capabilities.

Standalone Cache: Production Workloads

Production environments benefit from standalone cache deployments, particularly when supporting multiple users, processing large datasets, or requiring high availability. The independent scaling and fault isolation capabilities ensure consistent performance even under variable workloads.

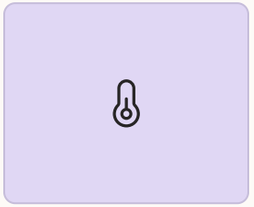


Migration Considerations

When transitioning from development to production, plan for migration from embedded to standalone cache. This shift requires additional configuration but provides the reliability and performance characteristics necessary for business-critical applications.

The decision between embedded and standalone cache should align with your deployment's criticality and scale. For production workloads processing terabytes of data or supporting dozens of concurrent users, standalone cache provides the necessary performance isolation and reliability.

Caching Strategies for Object Storage

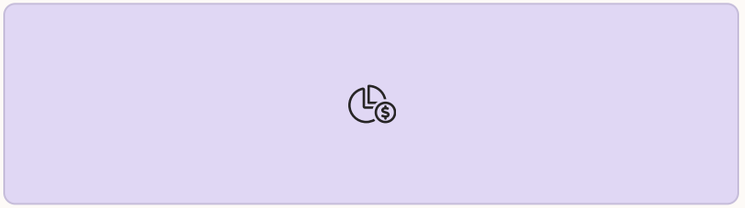


Hot Path Caching

Prioritize frequently accessed data

Selective Column Caching

Cache only the most queried columns



Aggregate-Level Caching

Store pre-aggregated summaries

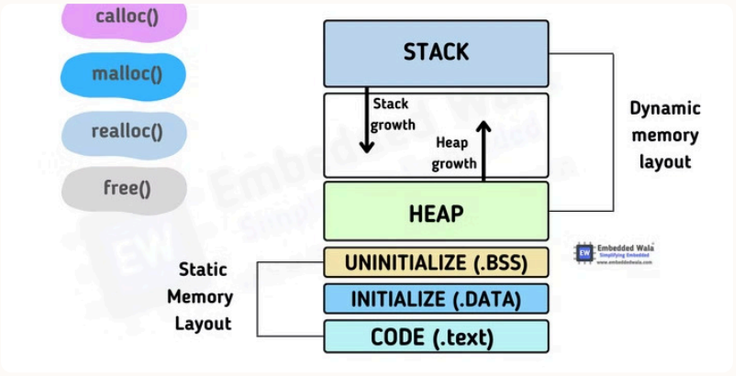
Time-Based Strategies

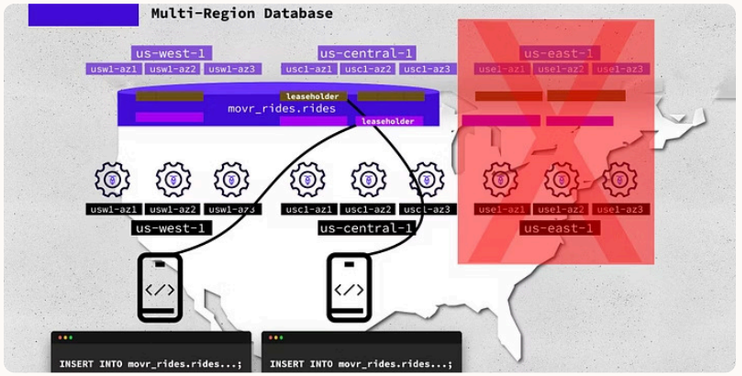
Focus on recent or temporal data

Effective object storage caching requires strategic approaches tailored to access patterns. Hot path caching identifies frequently queried data paths and prioritizes them for caching, dramatically improving performance for common queries while minimizing cache storage requirements. This approach works particularly well with partitioned data in object storage.

Column-level strategies optimize for analytical workloads that typically focus on specific columns. By caching only the most frequently accessed columns rather than entire tables, this approach maximizes the performance impact per gigabyte of cache storage, especially valuable when working with wide tables containing hundreds of columns.

Best Practices for Cache Deployment





Object Storage for Caching

Use object storage as the primary storage layer for caching data. Object storage provides a scalable, durable, and cost-effective solution for storing frequently accessed data.

Memory Allocation

Size the memory allocation for the cache database based on your workload patterns. For read-heavy analytics, prioritize memory allocation to maximize the in-memory cache hit rate and minimize object storage access latency.

Cache Refresh Scheduling

Schedule cache refreshes during off-peak hours to minimize impact on query performance. Align refresh schedules with source data update patterns to maintain optimal data freshness.

Implementing these best practices ensures optimal cache performance while minimizing operational overhead. By leveraging object storage for caching and carefully managing memory allocation, you can maximize the benefit of Starburst's caching capabilities.

Monitoring and Troubleshooting Caching

99.9%

Cache Availability

Target uptime percentage for production cache services

<100ms

Response Time

Ideal cache lookup latency for optimal performance

4x

Query Speedup

Typical performance improvement with properly configured caching

95%

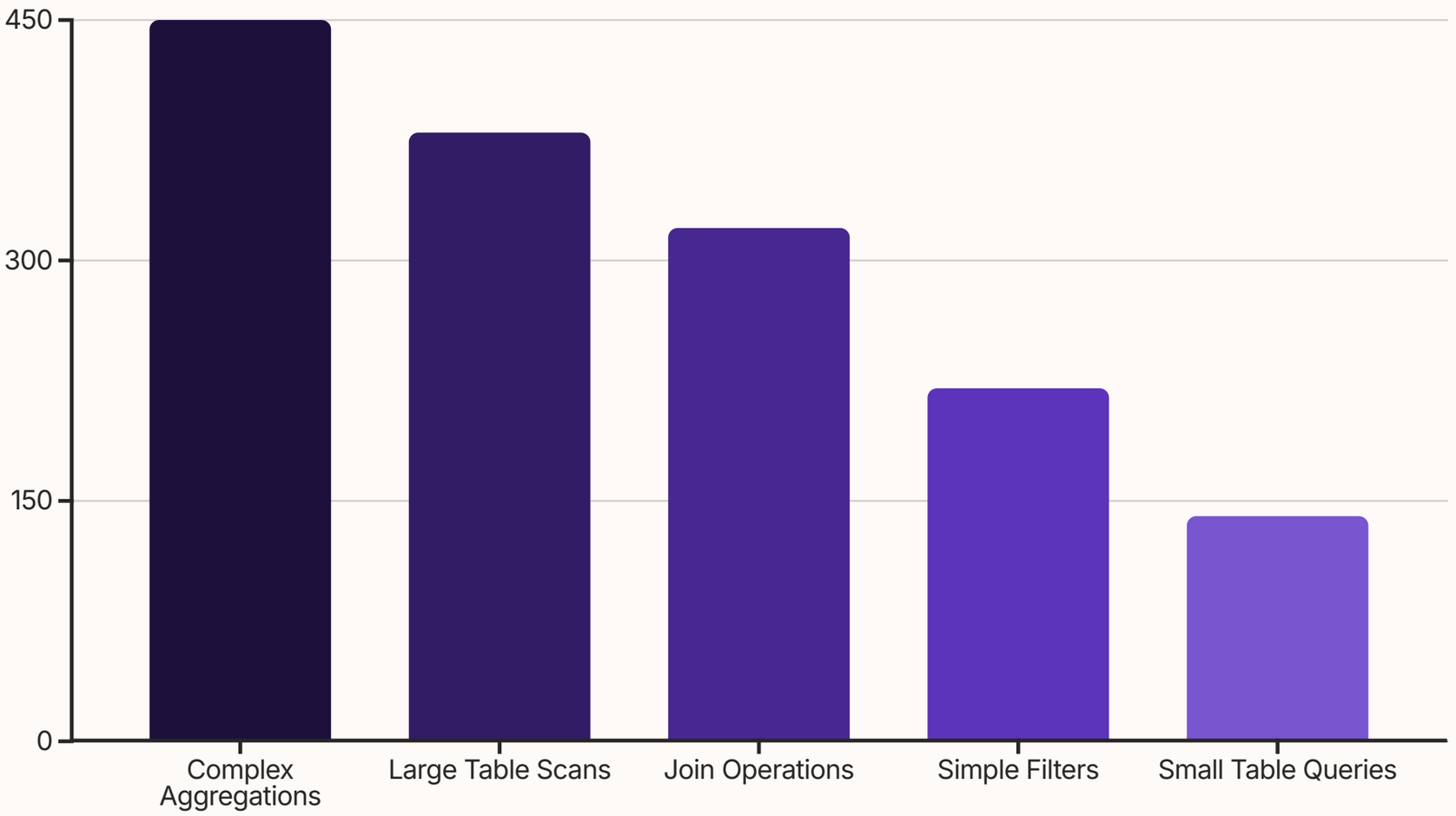
Hit Rate

Target cache hit percentage for frequently used queries

Effective monitoring is essential for maintaining optimal cache performance. Key metrics to track include cache hit rates, which indicate how often queries benefit from cached data, and refresh completion times, which help identify potential bottlenecks in the data synchronization process.

Common troubleshooting scenarios include stale cache data, which often results from failed refresh operations, and cache miss patterns, which may indicate opportunities to expand cache coverage. System logs provide valuable diagnostics, with cache service logs containing detailed information about refresh operations and query redirections.

Conclusion & Recommendations



The Starburst Cache Service bridges the performance gap between high-latency object storage and the speed requirements of interactive analytics. By implementing the appropriate cache architecture—embedded for development or standalone for production—organizations can dramatically accelerate query performance while maintaining data freshness.

Our key recommendations include starting with the most frequently accessed data for initial caching efforts, implementing proactive monitoring of cache performance metrics, and carefully planning cache refresh schedules to align with business needs. By following these guidelines, you can maximize your return on investment in Starburst Enterprise Platform and deliver exceptional analytics performance to your organization.