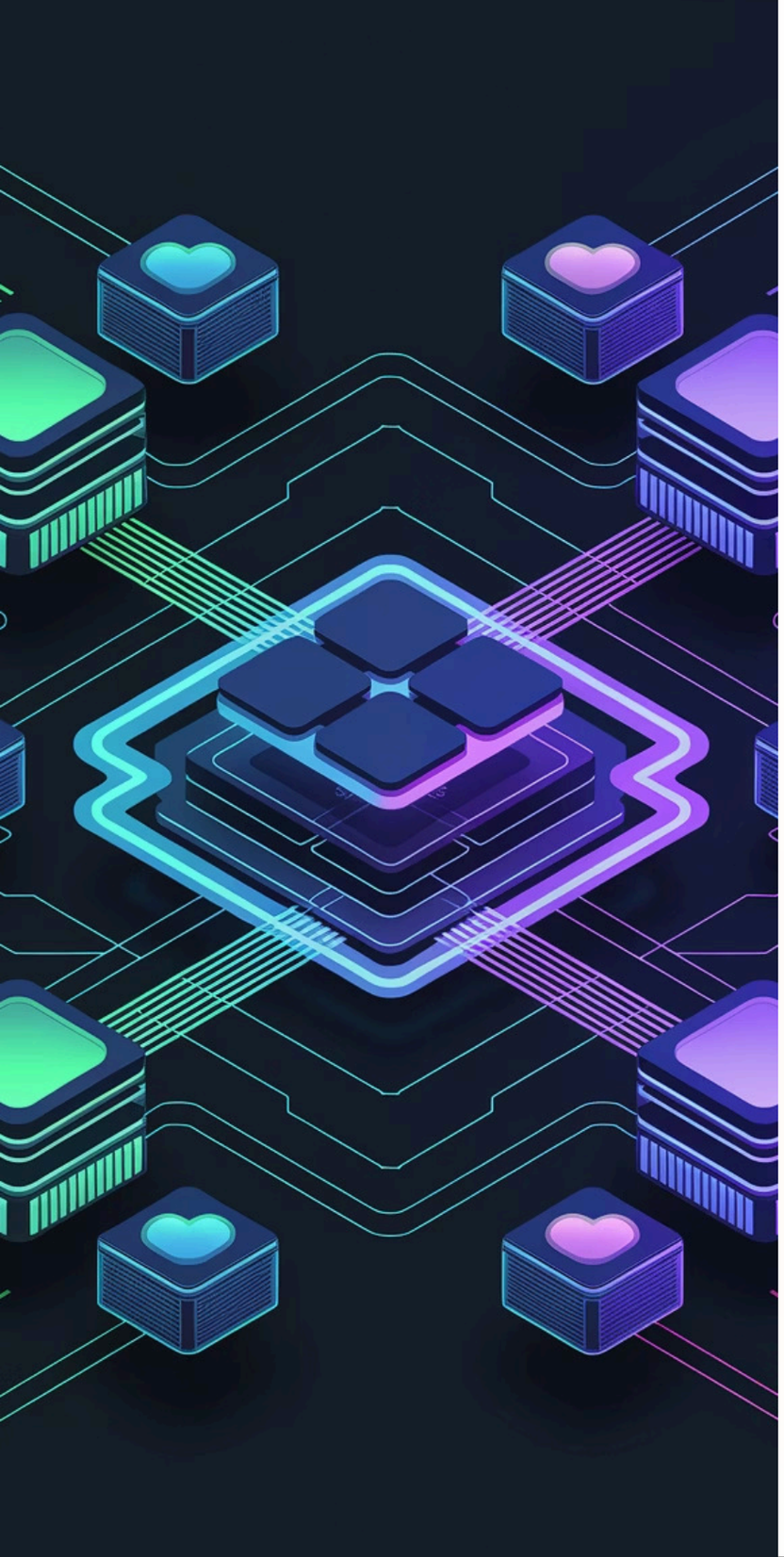


Starburst SEP: Technical Deep Dive for Developers

Welcome to this comprehensive technical deep dive into Starburst Enterprise Platform (SEP). As developers and data engineers, you'll discover how SEP's architecture enables high-performance distributed queries across diverse data sources. We'll explore the technical foundations, optimization techniques, and best practices to help you build robust, efficient data access solutions.

This presentation walks through the key components of Starburst SEP, explaining both the theoretical aspects and practical implementation details you'll need to effectively develop with and operate Starburst in your environment.

A **by Atish Kumar Sinha**

Agenda

Architecture Overview

Coordinator-worker model, query lifecycle, and distributed execution

Object Storage Integration

Connecting to S3, ADLS, GCS, and optimizing performance

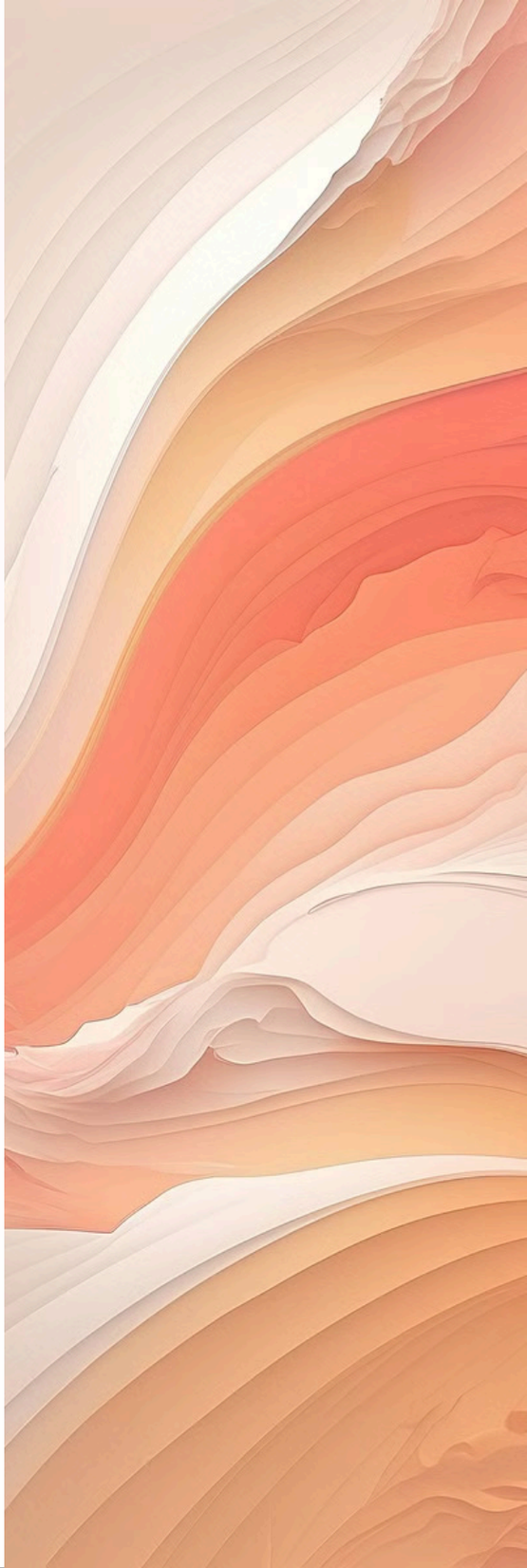
Performance Optimizations

Materialized views, caching strategies, and metadata management

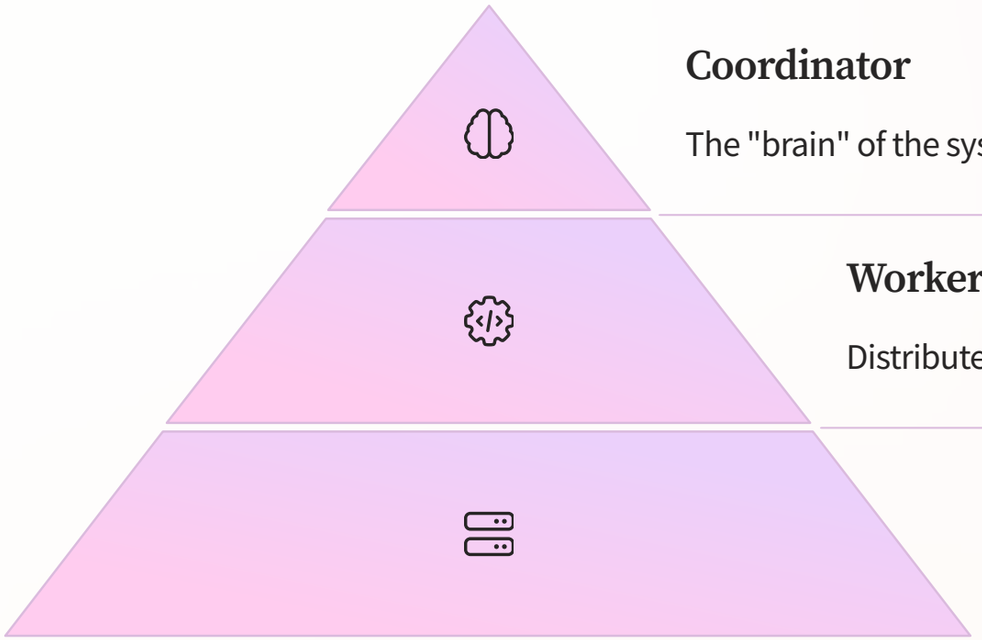
Networking & Extensions

Communication patterns and custom connector development

This technical session is designed for software developers and data engineers with experience in data warehousing. We'll focus on implementation details and architectural concepts rather than high-level business benefits, providing you with the knowledge needed to effectively implement and optimize Starburst SEP in your environment.



Coordinator-Worker Architecture



Coordinator

The "brain" of the system

Workers

Distributed processing nodes

Cluster

Full execution environment

Starburst SEP employs a single-coordinator, multi-worker architecture. The coordinator serves as the central control point, handling query parsing, planning, and task distribution. It accepts all client connections and orchestrates the execution process.

Workers are homogeneous processing nodes that execute the tasks assigned by the coordinator. They perform the heavy computational work in parallel, reading from data sources and processing data. All nodes run in JVMs, typically deployed in the same network for optimal performance.

Coordinator Functions

Client Interface

Accepts connections via JDBC, ODBC, and HTTP. Provides web UI for monitoring and management.

Query Processing

Parses SQL, validates semantics, and generates optimized execution plans.

Task Management

Breaks plans into stages, creates tasks, and distributes work to available workers.

Cluster Oversight

Monitors worker health, tracks task completion, and handles fault recovery.

The coordinator is the entry point for all client interactions. It handles authentication, processes incoming queries, and delivers results. For high availability deployments, Starburst supports standby coordinators that can take over within seconds if the primary fails, ensuring minimal disruption to operations.

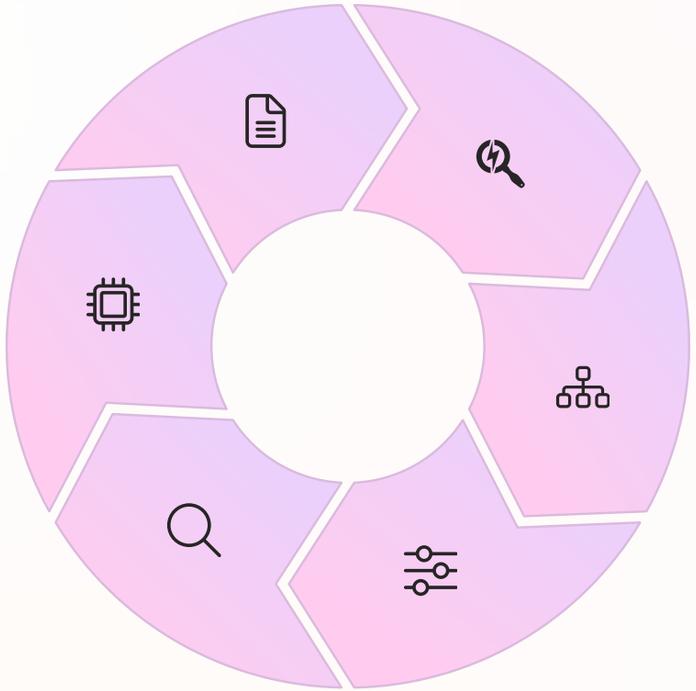
Life of a Query in Starburst

Parsing

SQL to syntax tree conversion

Analysis

Semantic validation



Execution

Parallel processing

Planning

Execution plan creation

Scheduling

Task distribution

Optimization

Plan rewriting and tuning

When a query is submitted, it undergoes transformation from SQL text to a distributed execution plan. The coordinator parses the SQL into a syntax tree, validates table and column references, then creates a logical plan determining necessary operations (scans, filters, joins).

This plan is optimized and converted into a physical execution plan with stages, which are distributed across workers for parallel processing. Results flow between stages, ultimately returning to the client.

Query Execution Example

Stage 1: Table Scan & Filter

Workers scan partitions and apply filters

Stage 2: Partial Aggregation

Local grouping and aggregation

Stage 3: Shuffle Exchange

Data redistribution by keys

Stage 4: Final Aggregation

Complete aggregation of shuffled data

Consider a query that counts orders by region with a date filter. Workers first scan table partitions in parallel, applying the date filter immediately. Each worker computes a partial count by region from its portion of data.

To get accurate global counts, Starburst shuffles data between workers based on region values. Stage 3 workers partition and send data so all records for a specific region arrive at the same destination. Finally, workers combine the partial aggregations into final results.

Object Storage Integration

Supported Object Stores

- Amazon S3
- Azure Data Lake Storage (ADLS)
- Google Cloud Storage (GCS)
- HDFS
- S3-compatible services (MinIO)

Supported File Formats

- Parquet (recommended)
- ORC
- Avro
- CSV
- JSON

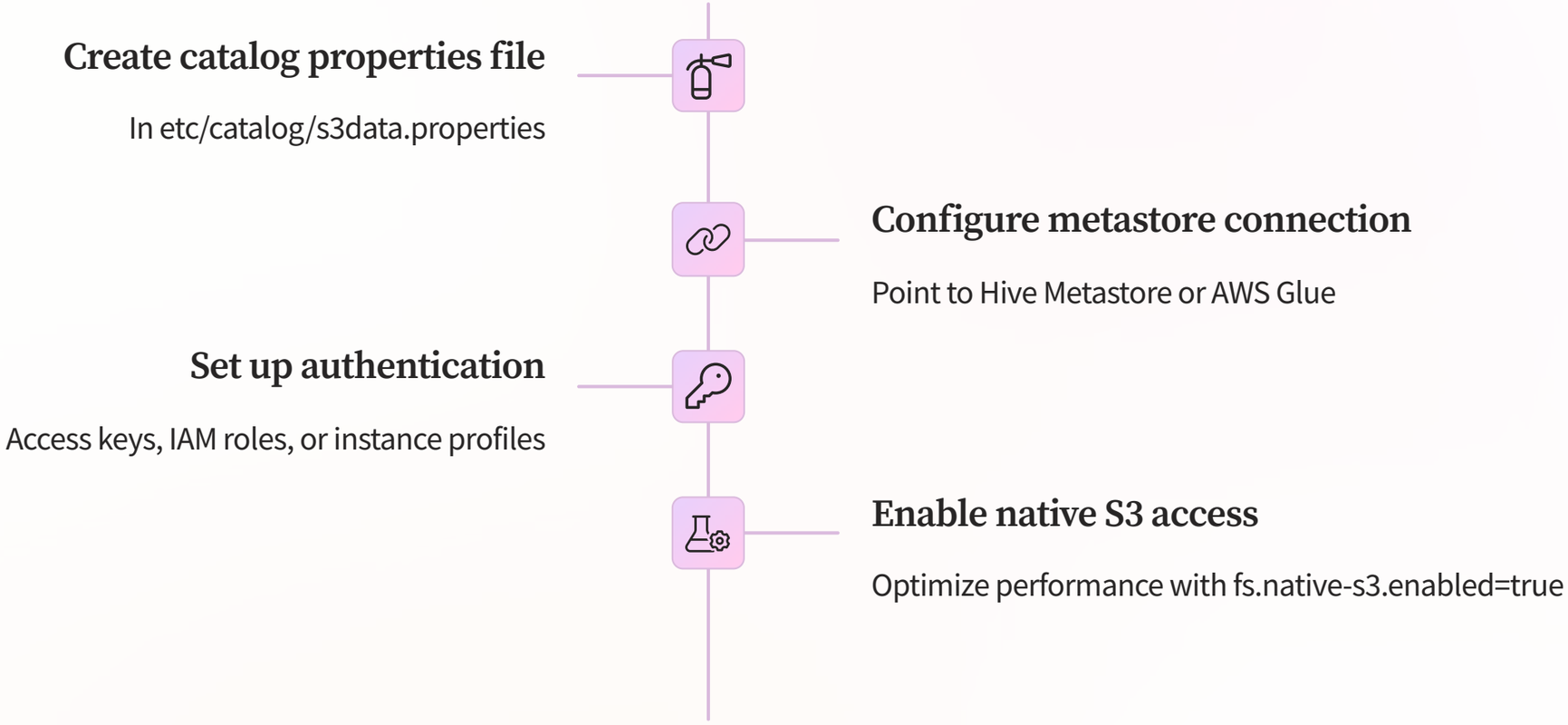
Metastore Requirements

- Hive Metastore
- AWS Glue Data Catalog
- Maps files to logical tables
- Stores schema definitions

Starburst SEP connects to object storage systems through specialized connectors, primarily the Hive connector for traditional data lakes. Unlike relational databases, object stores contain raw files without inherent table structures, so Starburst relies on external metastore services to map files to logical tables.

Using columnar formats like Parquet with compression significantly improves performance by reducing scan time and network I/O. Partitioning data by relevant dimensions (date, region) enables Starburst to skip unnecessary files during queries.

Connecting to Amazon S3



To connect Starburst to S3, create a catalog configuration file specifying the connector type, metastore connection, and authentication details. Modern SEP deployments should use the native S3 filesystem integration (fs.native-s3.enabled=true) rather than the legacy Hadoop-based connector for better performance.

Authentication options include direct AWS credentials, assuming IAM roles for cross-account access, or using the instance profile when running on EC2. For optimal performance, deploy Starburst in the same region as your S3 buckets and configure appropriate connection pooling (s3.max-connections).

S3 Configuration Example

```
# File: etc/catalog/hive.properties

connector.name=hive
hive.metastore.uri=thrift://hive-metastore:9083
hive.metastore=glue                    # For AWS Glue

# S3 Authentication
s3.aws-access-key=AKIAIOSFODNN7EXAMPLE
s3.aws-secret-key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
# OR use IAM Role
s3.iam-role=arn:aws:iam::123456789012:role/starburst-role

# Performance Optimization
fs.native-s3.enabled=true
s3.region=us-west-2
s3.max-connections=500

# Optional Security
s3.sse.type=KMS
s3.sse.kms-key-id=arn:aws:kms:us-west-2:123456789012:key/...
```

This example configuration shows how to set up a Hive catalog connecting to AWS S3 storage. The configuration specifies the metastore connection, authentication credentials, and performance optimizations. The native S3 filesystem integration provides better throughput with features like multi-part fetch and parallel reads.

For security-conscious deployments, you can enforce server-side encryption (SSE) by configuring the encryption type and KMS key. Remember that all SEP workers must have network access to both the S3 endpoints and the metastore service.

Metadata Management with Catalogs

Catalog

Represents a data source or service (e.g., hive, postgres)

Schema

Namespace within a catalog (e.g., sales, marketing)

Table

Actual data structure referenced via catalog.schema.table

Starburst organizes all data access through a three-level hierarchy: catalogs, schemas, and tables. A catalog represents a data source (like Hive or PostgreSQL), a schema corresponds to a database or namespace within that source, and tables contain the actual data.

For data lakes, Starburst relies on external metadata services like Hive Metastore or AWS Glue to map files in object storage to logical tables. These services store schema definitions, partition information, and file locations. Without a metastore, Starburst cannot interpret raw files as structured tables.

Metastore Integration

Connect to Metastore

Starburst connects to Hive Metastore via Thrift (port 9083) or AWS Glue via REST APIs. This connection is configured in the catalog properties file.

Query Table Metadata

When a query references a table, Starburst asks the metastore for its schema, partitions, and file locations. The metastore returns this information for query planning.

Cache Metadata

To improve performance, Starburst caches metadata information using configurable TTLs. This reduces repeated calls to the metastore for unchanged information.

The Hive Metastore (HMS) or AWS Glue acts as the bridge between Starburst and object storage data. It stores critical metadata about tables, including their schemas, partitioning keys, file formats, and physical locations. When planning queries, Starburst first consults this metadata to understand what data to access and how.

For optimal performance, deploy your metastore on a robust database with low latency access from Starburst. Consider high availability configurations for the metastore to prevent it from becoming a single point of failure.

Materialized Views for Performance

What are Materialized Views?

Materialized views persist the results of a query as a physical table, dramatically speeding up subsequent queries that use those results. They act as a form of pre-computed cache, trading storage for query speed.

Starburst supports materialized views through the Hive and Iceberg connectors, storing the view data in the data lake (typically on object storage) and tracking it in the metastore.

Key Benefits

- Dramatically accelerate common query patterns
- Pre-aggregate large fact tables
- Cache joined data from slow or remote sources
- Store complex calculations for reuse
- Reduce pressure on source systems

Materialized views function like regular tables but store pre-computed query results. When properly applied, they provide massive performance improvements for analytical workloads by avoiding expensive scans and calculations repeatedly. The view data is physically stored using the underlying connector's format (e.g., Parquet files) and can be as large as needed.

Creating and Refreshing Materialized Views

Create View

Define the view with a SQL query that computes the desired result

Refresh

Update the view's content manually or automatically on a schedule

Query

Access pre-computed results directly via SQL

Configure

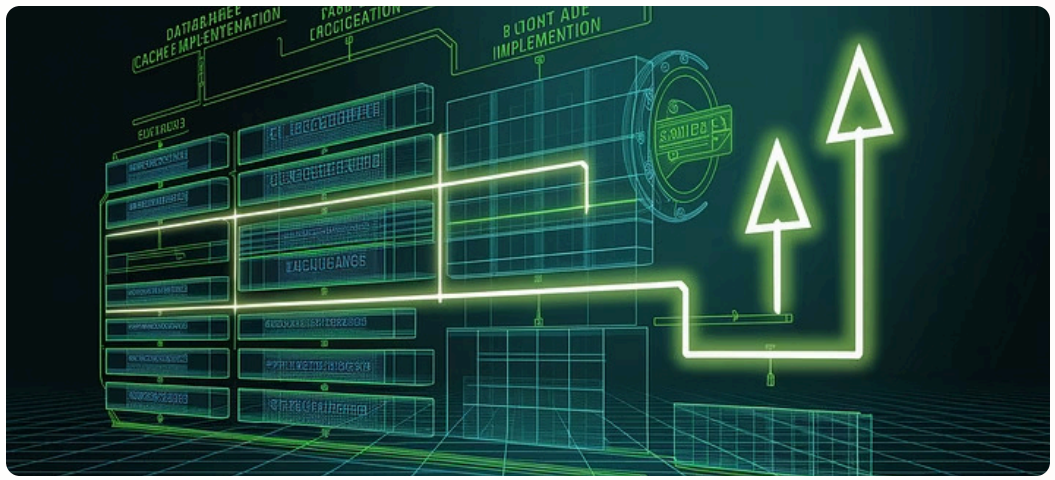
Set refresh policies and access parameters

```
-- Creating a materialized view
CREATE MATERIALIZED VIEW sales.customer_totals
WITH (
  refresh_interval = '24h',
  grace_period = '5m',
  max_import_duration = '30m'
)
AS SELECT
  customer_id,
  SUM(amount) AS total_spend
FROM sales.transactions
WHERE year = 2024
GROUP BY customer_id;

-- Manual refresh
REFRESH MATERIALIZED VIEW sales.customer_totals;
```

Materialized views need periodic refreshing to remain current with source data. Starburst supports both manual refreshes via SQL and automatic scheduled refreshes. For time-series data, incremental refreshes can be configured to only process new records since the last refresh.

Starburst Cache Service



Starburst's Cache Service, part of "Starburst Cached Views," delivers advanced functionality beyond basic materialized views. It can automatically redirect queries to use materialized views without explicitly referencing them, functioning as a transparent query acceleration layer.

When enabled, the optimizer can detect when a query's requirements could be satisfied by an existing materialized view and automatically use it instead of scanning base tables. The service also handles view refresh scheduling and synchronization with source data, providing a hands-off performance boost for your most important queries.

Caching Strategies in Starburst

Query Result Cache

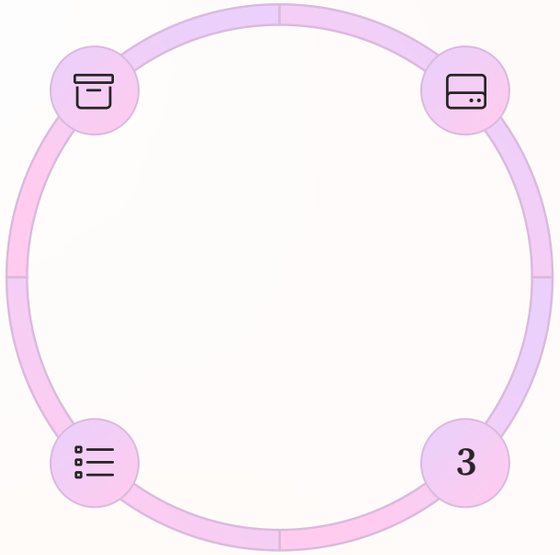
Stores complete query results for identical reuse

- Configurable TTL
- Stored in S3 or compatible storage

Metadata Cache

Caches table and partition metadata

- Reduces metastore calls
- In-memory with configurable TTL



Alluxio File Cache

Caches object storage data blocks locally

- Reduces S3/cloud storage latency
- Local SSD or memory storage

Materialized Views

Pre-computed query results stored as tables

- Scheduled refreshes
- Query redirection via Cache Service

Starburst implements multiple caching layers to optimize performance. Query Result Caching stores complete query outputs for identical reuse, while the Alluxio file cache brings frequently accessed data blocks closer to compute, reducing object storage latency.

Metadata caching minimizes repeated calls to the Hive Metastore, and materialized views act as semantic caches for complex queries. Each cache layer addresses different performance aspects, from reducing I/O to eliminating computation.

Configuring Query Result Cache

Implementation

The query result cache stores the complete output of queries in an S3 bucket or compatible storage. When an identical query runs again, Starburst retrieves the cached result instead of re-executing the query.

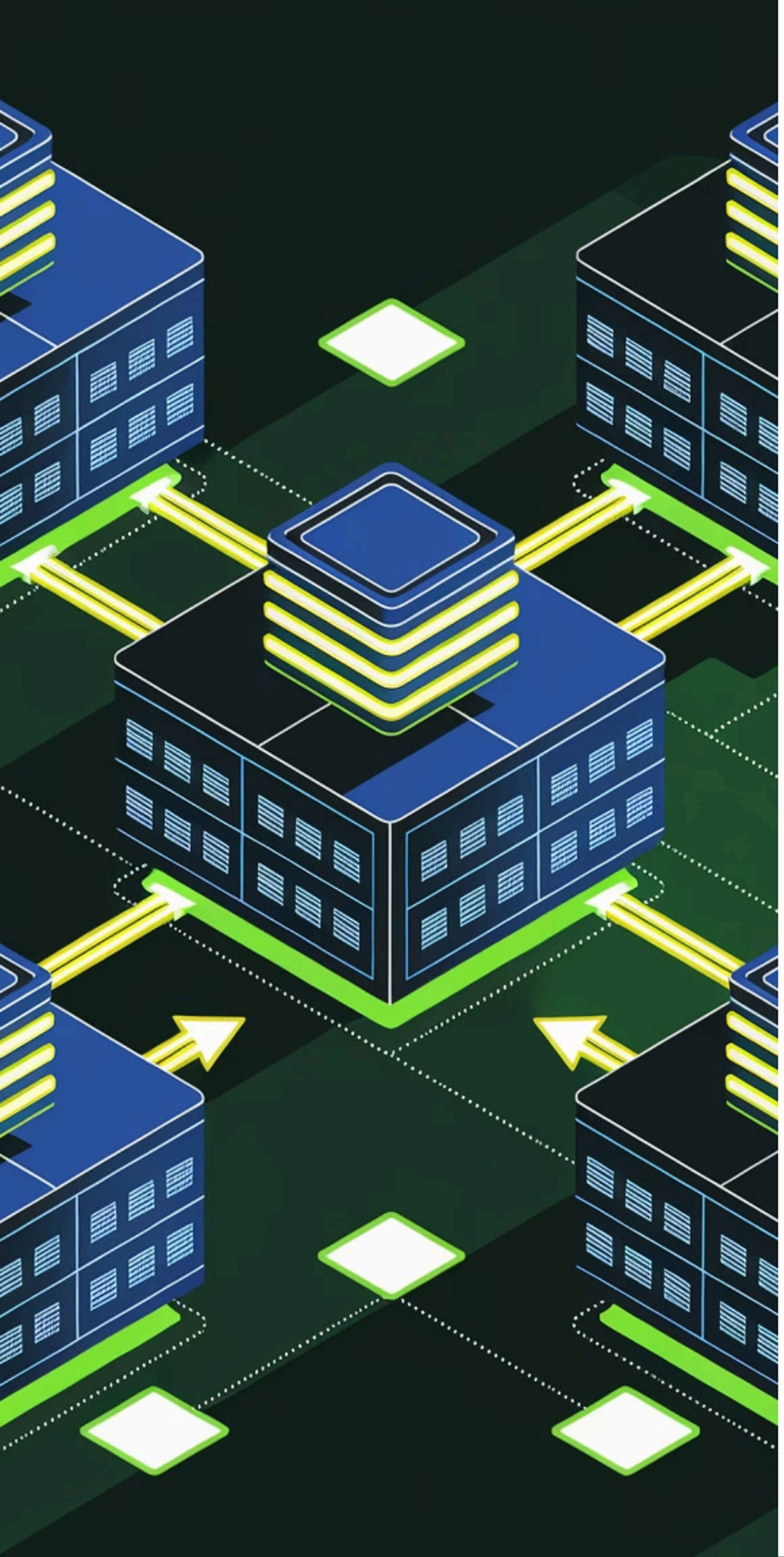
This cache is particularly effective for dashboards or reports that repeatedly run the same queries on slowly changing data. Configuration is straightforward, requiring just a few settings in the coordinator's properties file.

```
# In etc/config.properties
results-cache.enabled=true
results-cache.s3.bucket=my-sep-cache
results-cache.s3.prefix=results/
results-cache.expiration=1h

# Optional authenticaton if needed
results-cache.s3.aws-access-key=AKIA...
results-cache.s3.aws-secret-key=****
```

The query result cache can deliver dramatic performance improvements for identical repeated queries, as results are retrieved directly from storage instead of recomputing. Cache entries expire after the configured TTL (time-to-live), ensuring data eventually becomes consistent.

For best results, consider the update patterns of your data when setting the cache expiration. Frequently changing operational data might need shorter TTLs, while stable historical data can use longer expirations for maximum performance benefit.



Alluxio File Caching

10-100x

Latency Reduction

Compared to direct object storage access

443-e

SEP Version

Introduction of Alluxio caching

30%

Throughput Gain

For frequently accessed data

Starting with SEP version 443-e, Starburst reintroduced advanced file caching via Alluxio, replacing the older Rubix system. Alluxio acts as a distributed caching layer between Starburst workers and object storage, storing frequently accessed data blocks on local SSDs or in memory.

When a query requests data from object storage, Alluxio first checks if the blocks are already cached locally. Cache hits deliver dramatically lower latency compared to remote storage access. The cache operates at the block level, so even partial file overlaps between queries can benefit from caching.

Starburst Network Protocols

Starburst SEP (Starburst Execution Platform) relies on a robust network architecture to enable efficient data processing and querying. The platform utilizes a variety of network protocols to facilitate communication between its various components, ensuring seamless integration and high-performance data delivery.

1 HTTP/HTTPS

Starburst's primary internal communication protocol. This is used for efficient data exchange between the coordinator node and worker nodes, as well as between worker nodes themselves. The use of HTTP/HTTPS ensures secure and reliable data transmission during query execution.

2 Thrift

Starburst leverages the Thrift protocol to integrate with the Hive Metastore, a critical component for managing metadata about data sources. This allows Starburst to seamlessly access and utilize the rich metadata stored in the Hive Metastore.

3 REST/HTTP

When connecting to external services like AWS Glue and object storage, Starburst utilizes the REST/HTTP protocol. This provides a standardized and widely-adopted way to access these systems, enabling Starburst to integrate with a variety of data sources and catalogs.

4 Port Requirements

All nodes in a Starburst cluster must be able to freely communicate over the network, typically using ports 8080 (unencrypted) or 8443 (HTTPS secured). This ensures the necessary connectivity for efficient data processing and query execution.

By leveraging these network protocols, Starburst SEP is able to provide a highly scalable and performant data processing platform, seamlessly integrating with a wide range of data sources and services.

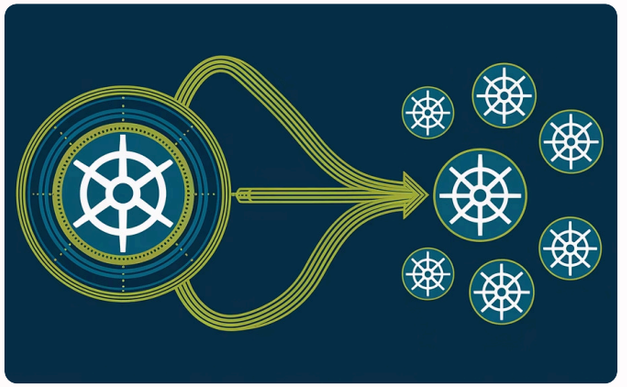
High Availability Networking

Elastic Network Interface (AWS)

In AWS deployments, Starburst can use an Elastic Network Interface (ENI) to maintain a consistent coordinator IP address. If the primary coordinator fails, the ENI is quickly remapped to a standby coordinator, usually within seconds.

Load Balancer Approach

Alternatively, a load balancer can direct traffic to the active coordinator. Health checks detect coordinator failures and automatically reroute traffic to a healthy standby node.



Kubernetes Services

In Kubernetes deployments, services provide stable endpoints that automatically route to healthy pods. If a coordinator pod fails, the service redirects traffic to the standby coordinator.

Starburst SEP supports Coordinator High Availability to eliminate the coordinator as a single point of failure. When properly configured, the system can recover from coordinator failures with minimal disruption, typically within seconds.

Custom Connector Development

Implement Plugin Interface

Create a class that extends io.trino.spi.Plugin

Develop Connector Components

Implement ConnectorFactory, ConnectorMetadata, ConnectorPageSource

Package as JAR

Include service provider configuration in META-INF/services

Deploy to Starburst

Place in plugins directory and configure in catalog properties

Starburst's Service Provider Interface (SPI) allows developers to extend functionality by creating custom plugins. These can include new connectors for unsupported data sources, user-defined functions, custom authentication handlers, or event listeners.

Developing a connector requires implementing several interfaces that handle metadata retrieval, data splitting, and reading. The connector translates Starburst's internal requests into source-specific operations. Once packaged as a JAR and placed in the plugins directory, the new connector becomes available through a catalog configuration file.