MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
DE LA RECHERCHE SCIENTIFIQUE ET DE L'INNOVATION

UNIVERSITÉ SULTAN MOULAY SLIMANE

**ECOLE NATIONALE DES SCIENCES APPLIQUÉS
DE KHOURIBGA**

Mémoire de fin d'études

pour l'obtention du

Diplôme d'Ingénieur d'État

Filière : Informatique et Ingénierie des Données

Présenté par :
Ziyad **RAKIB**

Titre du mémoire

Sous la supervision :
Abdelghani **Ghazdali**
Nidal **Lamghari**

Membres du jury :

Nom et prénoms du président	Entité	Président
Nom et prénoms de l'examinateur	Entité	Examinateur
Nom et prénoms du rapporteur	Entité	Rapporteur
Nom et prénoms du rapporteur	Entité	Rapporteur

Année Académique : **2023**

ehrig2006graph

Sommaire

Dédicace

"

À mLorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus,

À Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus,

À Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus,

À tous ceux qui me sont chers, à vous tous

Merci.

"

- Abdelghani

Remerciements

Nos remerciements

Résumé

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin posuere euismod neque, non semper nibh viverra sed. Praesent ut varius magna. Fusce ipsum ante, semper nec interdum at, semper et lacus. Nulla ultrices magna a fringilla finibus.

ملخص

في هذا التقرير، وردت إشارة إلى تنفيذ نظام كشف الوجه. والهدف من ذلك هو الكشف عن وجوه الناس المهتمين في منشور معين. لقد إستعملنا لهذا الغرض لوحة الكترونية Raspberry pi 2 بالإضافة إلى وحدة الكاميرا الخاصة بها للاحقة الوجوه و العين. يستخدم تطبيق كشف الوجه خوارزمية قوية لفيولا جونز مع تدريب المصنف للكشف عن الوجه الإنساني والعينين، عن طريق اختيار أفضل مصنف ممكن. سيتم حفظ البيانات لكل إعلان في قاعدة بيانات MySQL . وقد تم إنشاء تطبيق ويب لهذا الغرض والذي سيمكن المستخدم من معرفة عدد الأشخاص والمدة الإجمالية للمشاهدة في الفترة (ساعة، يوم، شهر) على شكل رسم بياني.

كلمات مفتاحية : ك ك

Table des matières

Table des figures

Liste des tableaux

TABLE 1 : List of Abbreviations

API	Application programming interface
VSC	Visual Studio Code
DL	Delta Lake
REST	RepresEntational State Transfer
AWS	Amazon Web Services
GCP	Google Cloud Platform
AZR	Microsoft Azure
S3	Simple Storage Service
IAM	Identity and Access Management
MinIO	Minimal Object Storage
SQL	Structured Query Language
DB	Database
ACID	Atomicity, Consistency, Isolation, Durability
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
ReactJS	React JavaScript
SPA	Single Page Application
JS	JavaScript
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
UI	User Interface
API	Application Programming Interface
Spark	Apache Spark
Hadoop	Apache Hadoop
HDFS	Hadoop Distributed File System
YARN	Yet Another Resource Negotiator
MapReduce	MapReduce Programming Model
Metadata	Data about Data
ETL	Extract, Transform, Load
ELT	Extract, Load, Transform
CRM	Customer Relationship Management
RDBMS	Relational Database Management System
SPI	Server Provider Interface

Introduction Générale

bla bla bla Def acronyme (acronyme) puis Acroglo et enfin glossaire

L'entreprise Izicap

Introduction

1.1 About

information about Izicap

1.2 Organigraph

information about Izicap

Conclusion

Chapitre 2

Delta Lake

Introduction

Data warehouses and data lakes are the most common central data repositories employed by most data-driven organizations today, each with its own strengths and tradeoffs. For one, while data warehouses allow businesses to organize historical datasets for use in business intelligence (BI) and analytics, they quickly become more cost-intensive as datasets grow because of the combined use of compute and storage resources. Additionally, data warehouses can't handle the varied nature of data (structured, unstructured, and semi-structured) seen today.

In this chapter, we will explore the key features of Delta Lake, how it works, and why it is a good choice for big data processing. We will also provide examples of how to use Delta Lake with other big data tools, such as Hadoop, Spark, and Trino.

2.1 Definition

Delta Lake is an open-source storage layer built atop a data lake that confers reliability and ACID (Atomicity, Consistency, Isolation, and Durability) transactions. It enables a continuous and simplified data architecture for organizations. A data lake stores data in Parquet formats and enables a lakehouse data architecture, which helps organizations achieve a single, continuous data system that combines the best features of both the data warehouse and data lake while supporting streaming and batch processing.

2.2 How Delta Lake Works

A Delta Lake enables the building of a data lakehouse. Common lakehouses include the Databricks Lakehouse and Azure Databricks. This continuous data architecture allows organizations to harness

the benefits of data warehouses and data lakes with reduced management complexity and cost. Here are some ways Delta Lake improves the use of data warehouses and lakes :

- **Enables a lakehouse architecture :** Delta Lake enables a continuous and simplified data architecture that allows organizations to handle and process massive volumes of streaming and batch data without the management and operational hassles involved in managing streaming, data warehouses, and data lakes separately.

- **Enables intelligent data management for data lakes :** Delta Lake offers efficient and scalable metadata handling, which provides information about the massive data volumes in data lakes. With this information, data governance and management tasks proceed more efficiently.

- **Schema enforcement for improved data quality :** Because data lakes lack a defined schema, it becomes easy for bad/incompatible data to enter data systems. There is improved data quality thanks to automatic schema validation, which validates DataFrame and table compatibility before writes.

- **Enables ACID transactions :** Most organizational data architectures involve a lot of ETL and ELT movement in and out of data storage, which opens it up to more complexity and failure at node entry points. Delta Lake ensures the durability and persistence of data during ETL and other data operations. Delta lake captures all changes made to data during data operations in a transaction log, thereby ensuring data integrity and reliability during data operations.

2.3 Delta Lake Architecture Diagram

Delta Lake is an improvement from the lambda architecture whereby streaming and batch processing occur parallel, and results merge to provide a query response. However, this method means more complexity and difficulty maintaining and operating both the streaming and batch processes. Unlike the lambda architecture, Delta Lake is a continuous data architecture that combines streaming and batch workflows in a shared file store through a connected pipeline.

The stored data file has three layers, with the data getting more refined as it progresses downstream in the dataflow :

- **Bronze tables :** This table contains the raw data ingested from multiple sources like the Internet of Things (IoT) systems, CRM, RDBMS, and JSON files.

- **Silver tables :** This layer contains a more refined view of our data after undergoing transformation and feature engineering processes.

- **Gold tables :** This final layer is often made available for end users in BI reporting and analysis or use in machine learning processes.
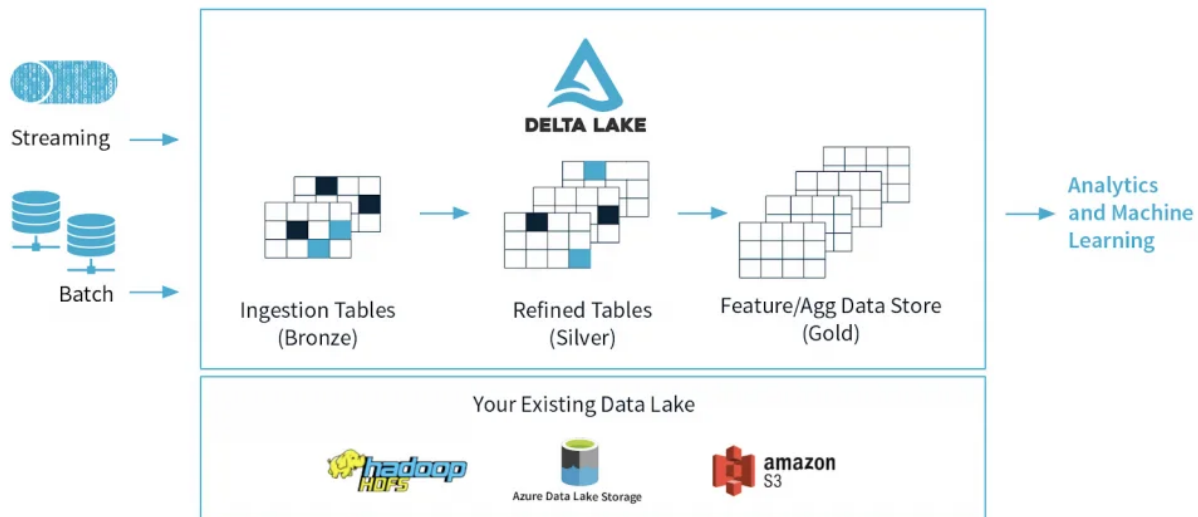
FIGURE 2.1 : Delta Lake multi-hop architecture

2.4 Key benefits and features of Delta Lake

- **Audit trails and history :** In Delta Lake, every write exists as a transaction and is serially recorded in a transaction log. Therefore, any changes or commits made to the transaction log are recorded, leaving a complete trail for use in historical audits, versioning, or for time traveling purposes. This Delta Lake feature helps ensure data integrity and reliability for business data operations.

- **Time traveling and data versioning :** Because each write creates a new version and stores the older version in the transaction log, users can view/revert to older data versions by providing the timestamp or version number of an existing table or directory to the Sparks read API. Using the version number provided, the Delta Lake then constructs a full snapshot of the version with the information provided by the transaction log. Rollbacks and versioning play a vital role in machine learning experimentation, whereby data scientists iteratively change hyperparameters to train models and can revert to changes if needed.

- **Unifies batch and stream processing :** Every table in a Delta Lake is a batch and streaming sink. With Sparks structured streaming, organizations can efficiently stream and process streaming data. Additionally, with the efficient metadata handling, ease of scale, and ACID quality of each transaction, near-real-time analytics become possible without utilizing a more complicated two-tiered data architecture.

- **Efficient and scalable metadata handling :** Delta Lakes store metadata information in the transaction log and leverages Spark's distributed processing power to quickly process and efficiently read and handle large volumes of data metadata, thus improving data governance.

- **ACID transactions :** Delta Lakes ensure that users always see a consistent data view in a table or directory. It guarantees this by capturing every change made in a transaction log and isolating

it at the strongest isolation level, the serializable level. In the serializable level, every existing operation has and follows a serial sequence that, when executed one by one, provides the same result as seen in the table.

- **Data Manipulation Language operations :** Delta Lakes supports DML operations like updates, deletes, and merges, which play a role in complex data operations like change-data-capture (CDC), streaming upserts, and slowly-changing-dimension (SCD). Operations like CDC ensure data synchronization in all data systems and minimizes the time and resources spent on ELT operations. For instance, using the CDC, instead of ETL-ing all the available data, only the recently updated data since the last operation undergoes a transformation.

- **Schema enforcement :** Delta Lakes perform automatic schema validation by checking against a set of rules to determine the compatibility of a write from a DataFrame to a table. One such rule is the existence of all DataFrame columns in the target table. An occurrence of an extra or missing column in the DataFrame raises an exception error. Another rule is that the DataFrame and target table must contain the same column types, which otherwise will raise an exception. Delta Lake also use DDL (Data Definition Language) to add new columns explicitly. This data lake feature helps prevent the ingestion of incorrect data, thereby ensuring high data quality.

- **Compatibility with Spark's API :** Delta Lake is built on Apache Spark and is fully compatible with Spark API, which helps build efficient and reliable big data pipelines.

- **Flexibility and integration :** Delta lake is an open-source storage layer and utilizes the Parquet format to store data files, which promotes data sharing and makes it easier to integrate with other technologies and drive innovation.

Conclusion

Delta Lake is an important tool for big data processing, providing reliable data management and ensuring data integrity at scale. Its ACID transactions, schema enforcement, and data versioning features make it a popular choice for companies that need to process large amounts of data with high accuracy and reliability.

By using Delta Lake, data engineers and data scientists can easily manage data quality, track data lineage, and collaborate on data analysis projects. With its seamless integration with other big data tools, such as Spark, Hadoop, and Trino, Delta Lake provides a powerful solution for big data processing that can help companies gain insights from their data faster and more efficiently.

Trino

Introduction

3.1 Definition

Trino is an open source distributed SQL query engine. It is a hard fork of the original Presto project created by Facebook. It lets developers run interactive analytics against large volumes of data. With Trino, organizations can easily use their existing SQL skills to query data without having to learn new complex languages. The architecture is quite similar to traditional online analytical processing (OLAP) systems using distributed computing architectures, in which one controller node coordinates multiple worker nodes.

3.2 How it works

Trino is a distributed system that runs on Hadoop, and uses an architecture similar to massively parallel processing (MPP) databases. It has one coordinator node working with multiple worker nodes. Users submit SQL to the coordinator which uses query and execution engine to parse, plan, and schedule a distributed query plan across the worker nodes. It supports standard ANSI SQL, including complex queries, joins aggregations, and outer joins.

Leveraging this architecture, the Trino query engine is able to process SQL queries on large amounts of data in parallel across a cluster of computers, or nodes. Trino runs as a single-server process on each node. Multiple nodes running Trino, which are configured to collaborate with each other, make up a Trino cluster.

The following figure displays a high-level overview of a Trino cluster composed of one coordinator and multiple worker nodes. A Trino user connects to the coordinator with a client, such as a tool using the JDBC driver or the Trino CLI. The coordinator then collaborates with the workers, which access the data sources.
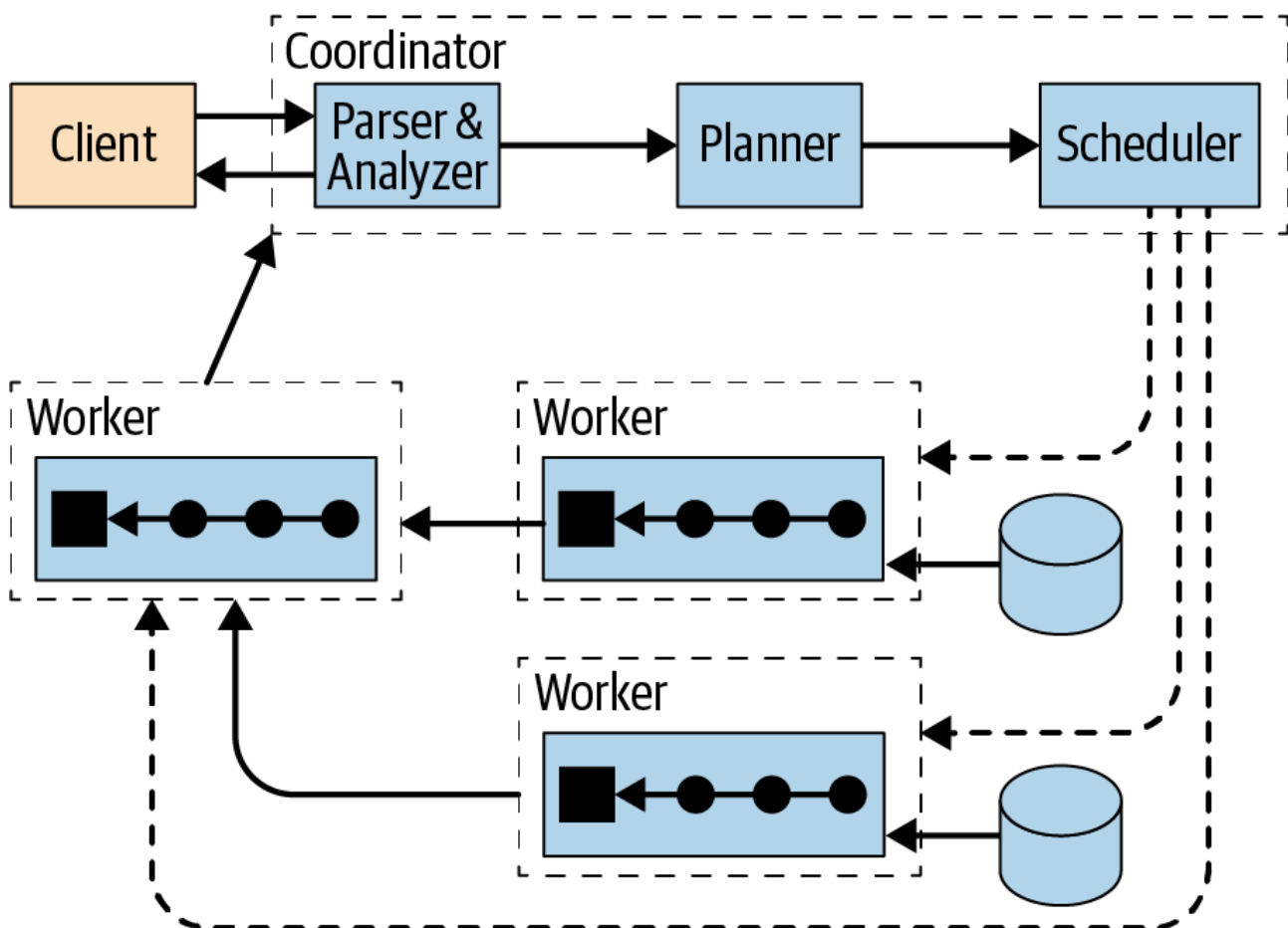
FIGURE 3.1 : Trino architecture overview with coordinator and workers

1. A coordinator is a Trino server that handles incoming queries and manages the workers to execute the queries.

2. A worker is a Trino server responsible for executing tasks and processing data.

3. The discovery service typically runs on the coordinator and allows workers to register to participate in the cluster.

4. All communication and data transfer between clients, coordinator, and workers uses REST-based interactions over HTTP/HTTPS.

The following figure shows how the communication within the cluster happens between the coordinator and the workers, as well as from one worker to another. The coordinator talks to workers to assign work, update status, and fetch the top-level result set to return to the users. The workers talk to each other to fetch data from upstream tasks, running on other workers. And the workers retrieve result sets from the data source.
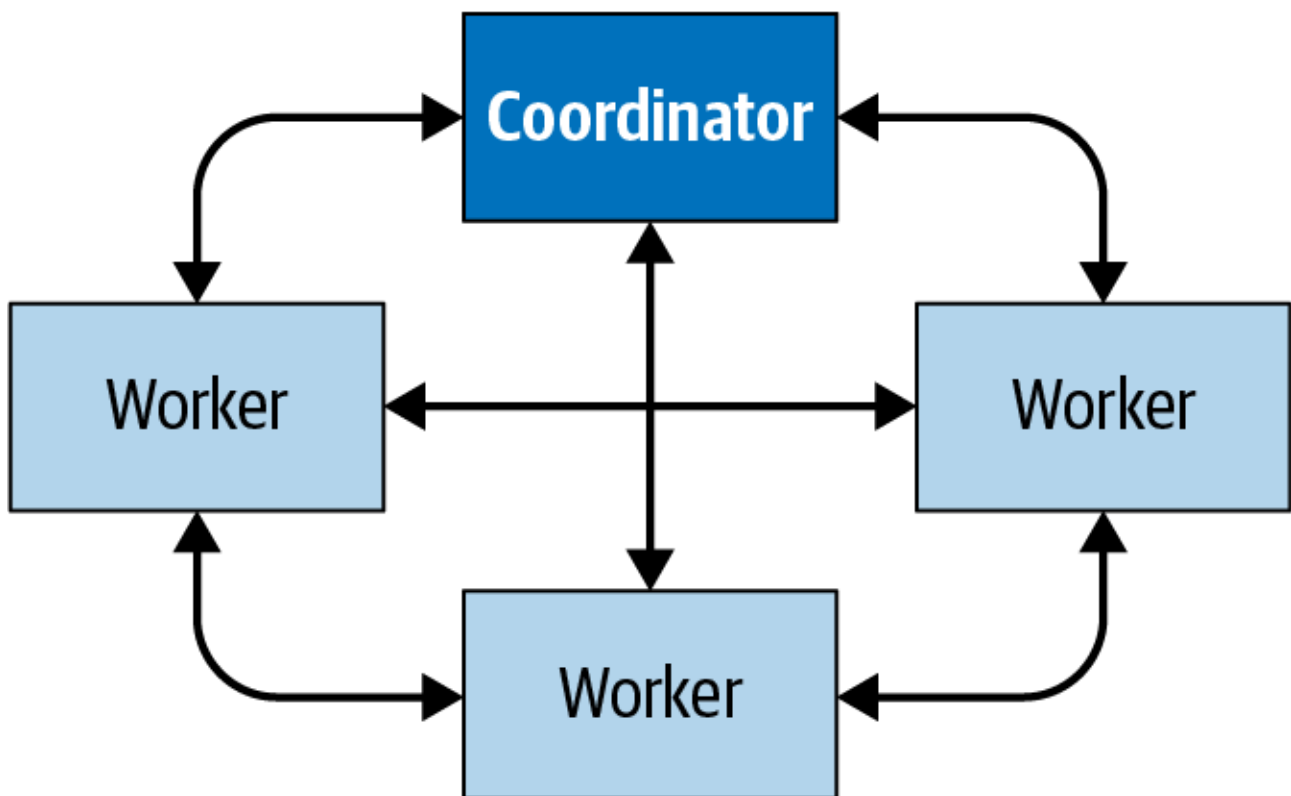
FIGURE 3.2 : Communication between coordinator and workers in a Trino cluster

3.3 Coordinator

The Trino coordinator is the server responsible for receiving SQL statements from the users, parsing these statements, planning queries, and managing worker nodes. It's the brain of a Trino installation and the node to which a client connects. Users interact with the coordinator via the Trino CLI, applications using the JDBC or ODBC drivers, or any other available client libraries for a variety

of languages. The coordinator accepts SQL statements from the client such as SELECT queries for execution.

Every Trino installation must have a coordinator alongside one or more workers. For development or testing purposes, a single instance of Trino can be configured to perform both roles.

The coordinator keeps track of the activity on each worker and coordinates the execution of a query. The coordinator creates a logical model of a query involving a series of stages.

Once it receives a SQL statement, the coordinator is responsible for parsing, analyzing, planning, and scheduling the query execution across the Trino worker nodes. The statement is translated into a series of connected tasks running on a cluster of workers. As the workers process the data, the results are retrieved by the coordinator and exposed to the clients on an output buffer. Once an output buffer is completely read by the client, the coordinator requests more data from the workers on behalf of the client. The workers, in turn, interact with the data sources to get the data from them. As a result, data is continuously requested by the client and supplied by the workers from the data source until the query execution is completed.

Coordinators communicate with workers and clients by using an HTTP-based protocol.
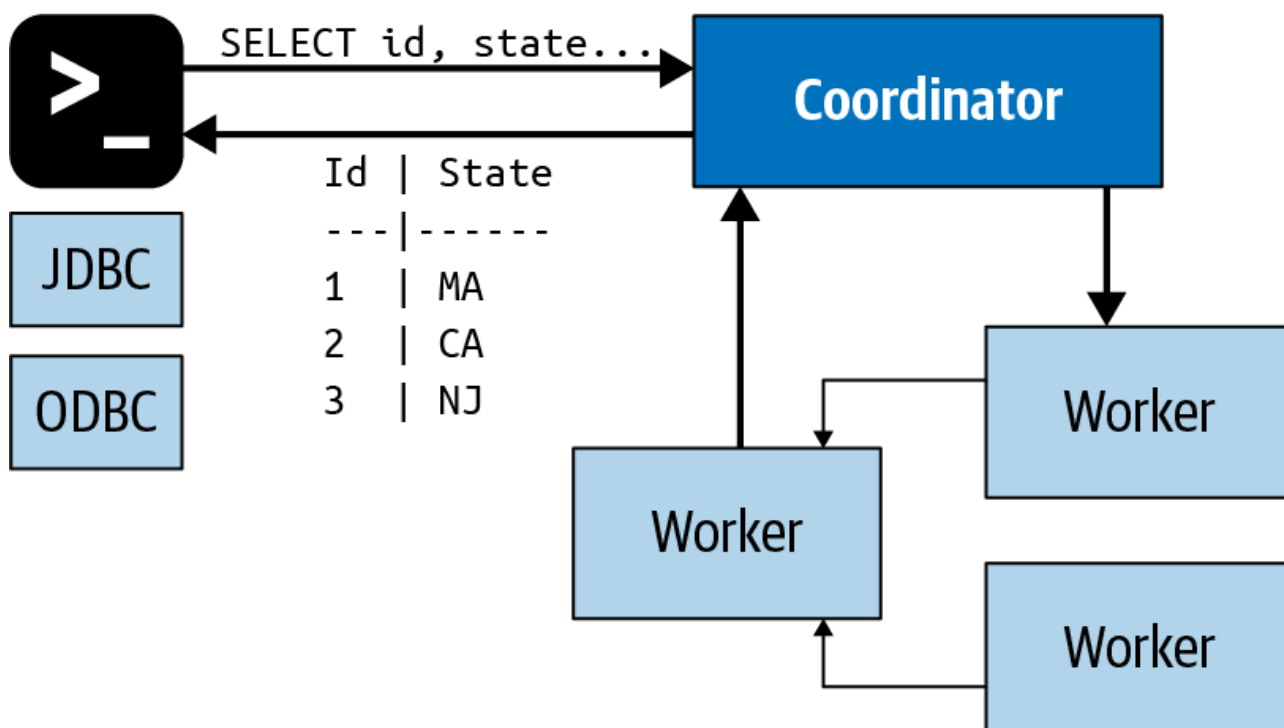

FIGURE 3.3 : Client, coordinator, and worker communication processing a SQL statement

3.4 Workers

A Trino worker is a server in a Trino installation. It is responsible for executing tasks assigned by the coordinator and for processing data. Worker nodes fetch data from data sources by using connectors and then exchange intermediate data with each other. The final resulting data is passed on to the coordinator. The coordinator is responsible for gathering the results from the workers and providing

the final results to the client.

During installation, workers are configured to know the hostname or IP address of the discovery service for the cluster. When a worker starts up, it advertises itself to the discovery service, which makes it available to the coordinator for task execution.

Workers communicate with other workers and the coordinator by using an HTTP-based protocol.

The following figure shows how multiple workers retrieve data from the data sources and collaborate to process the data, until one worker can provide the data to the coordinator.
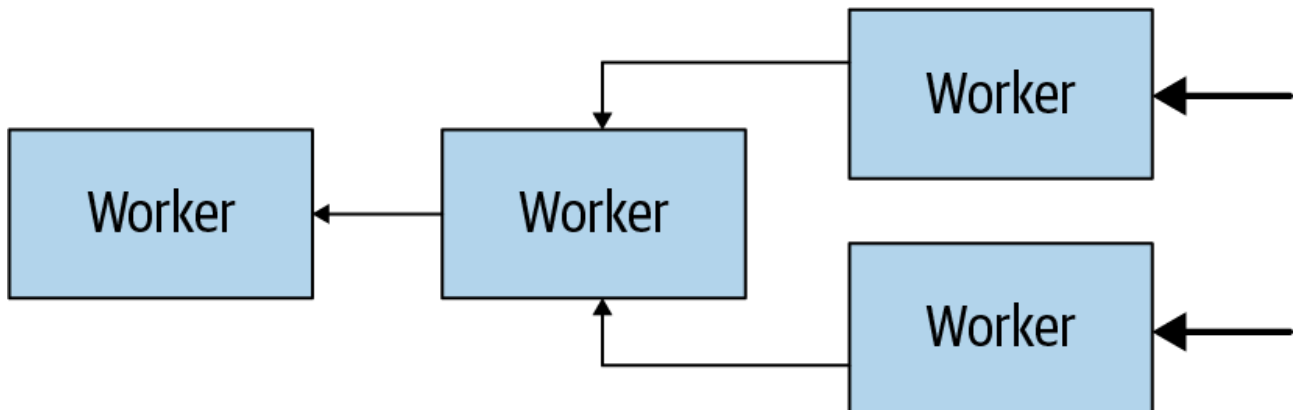
FIGURE 3.4 : Workers in a cluster collaborate to process SQL statements and data

3.5 Connector-Based Architecture

At the heart of the separation of storage and compute in Trino is the connector-based architecture. A connector provides Trino an interface to access an arbitrary data source.

Each connector provides a table-based abstraction over the underlying data source. As long as data can be expressed in terms of tables, columns, and rows by using the data types available to Trino, a connector can be created and the query engine can use the data for query processing.

Trino provides a service provider interface (SPI), which is a type of API used to implement a connector. By implementing the SPI in a connector, Trino can use standard operations internally to connect to any data source and perform operations on any data source. The connector takes care of the details relevant to the specific data source.

At the heart of the separation of storage and compute in Trino is the connector-based architecture. A connector provides Trino an interface to access an arbitrary data source.

Each connector provides a table-based abstraction over the underlying data source. As long as data can be expressed in terms of tables, columns, and rows by using the data types available to Trino, a connector can be created and the query engine can use the data for query processing.

Trino provides a service provider interface (SPI), which is a type of API used to implement a connector. By implementing the SPI in a connector, Trino can use standard operations internally to connect to any data source and perform operations on any data source. The connector takes care of the details relevant to the specific data source.

- Operations to fetch table/view/schema metadata

- Operations to produce logical units of data partitioning, so that Trino can parallelize reads and writes

- Data sources and sinks that convert the source data to/from the in-memory format expected by the query engine

Trino provides many connectors to systems, here are the list of connects as of the writing of this report :

Connector Name	Documentation Link
Accumulo	https://trino.io/docs/current/connector/accumulo.html
Atop	https://trino.io/docs/current/connector/atop.html
BigQuery	https://trino.io/docs/current/connector/bigquery.html
Black Hole	https://trino.io/docs/current/connector/blackhole.html
Cassandra	https://trino.io/docs/current/connector/cassandra.html
ClickHouse	https://trino.io/docs/current/connector/clickhouse.html
Delta Lake	https://trino.io/docs/current/connector/delta-lake.html
Druid	https://trino.io/docs/current/connector/druid.html
Elasticsearch	https://trino.io/docs/current/connector/elasticsearch.html
Google Sheets	https://trino.io/docs/current/connector/googlesheets.html
Hive	https://trino.io/docs/current/connector/hive.html
Hudi	https://trino.io/docs/current/connector/hudi.html
Iceberg	https://trino.io/docs/current/connector/iceberg.html
Ignite	https://trino.io/docs/current/connector/ignite.html
JMX	https://trino.io/docs/current/connector/jmx.html
Kafka	https://trino.io/docs/current/connector/kafka.html
Kinesis	https://trino.io/docs/current/connector/kinesis.html
Kudu	https://trino.io/docs/current/connector/kudu.html
Local File	https://trino.io/docs/current/connector/localfile.html
MariaDB	https://trino.io/docs/current/connector/mariadb.html
Memory	https://trino.io/docs/current/connector/memory.html
MongoDB	https://trino.io/docs/current/connector/mongodb.html
MySQL	https://trino.io/docs/current/connector/mysql.html
Oracle	https://trino.io/docs/current/connector/oracle.html
Phoenix	https://trino.io/docs/current/connector/phoenix.html
Pinot	https://trino.io/docs/current/connector/pinot.html
PostgreSQL	https://trino.io/docs/current/connector/postgresql.html
Prometheus	https://trino.io/docs/current/connector/prometheus.html
Redis	https://trino.io/docs/current/connector/redis.html
Redshift	https://trino.io/docs/current/connector/redshift.html
SingleStore	https://trino.io/docs/current/connector/singlestore.html
SQL Server	https://trino.io/docs/current/connector/sqlserver.html
System	https://trino.io/docs/current/connector/system.html
Thrift	https://trino.io/docs/current/connector/thrift.html
TPCDS	https://trino.io/docs/current/connector/tpcds.html
TPCH	https://trino.io/docs/current/connector/tpch.html

TABLE 3.1 : List of Trino Connectors and their documentation

Trino's SPI also gives you the ability to create your own custom connectors. This may be needed if you need to access a data source without a compatible connector. If you end up creating a connector, we strongly encourage you to learn more about the Trino open source community, use our help, and

contribute your connector. Check out "Trino Resources" for more information. A custom connector may also be needed if you have a unique or proprietary data source within your organization. This is what allows Trino users to query any data source by using SQL—truly SQL-on-Anything.

The following figure shows how the Trino SPI includes separate interfaces for metadata, data statistics, and data location used by the coordinator, and for data streaming used by the workers
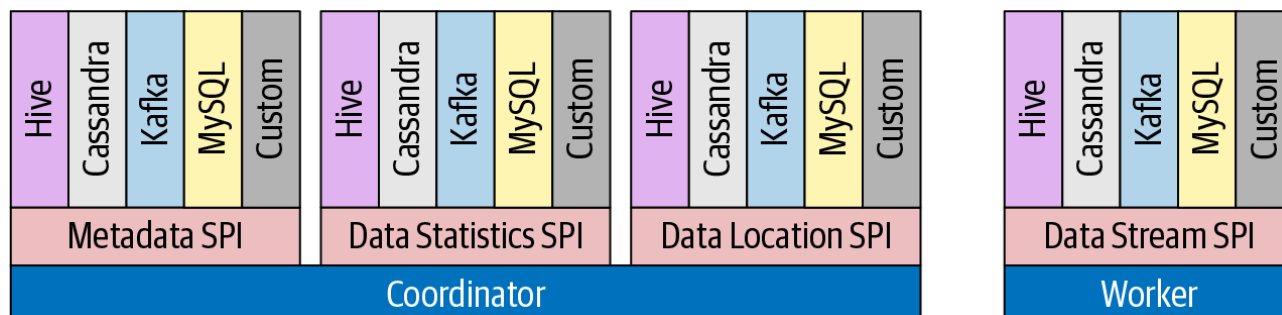

FIGURE 3.5 : Overview of the Trino service provider interface (SPI)

Trino connectors are plug-ins loaded by each server at startup. They are configured by specific parameters in the catalog properties files and loaded from the plug-ins directory.

3.6 Catalogs, Schemas, and Tables

The Trino cluster processes all queries by using the connector-based architecture described earlier. Each catalog configuration uses a connector to access a specific data source. The data source exposes one or more schemas in the catalog. Each schema contains tables that provide the data in table rows with columns using different data types. You can find out more about catalogs, schemas, tables. Specifically in "Catalogs", "Schemas", and "Tables".

Conclusion

The Trino architecture has a coordinator receiving user requests and then using workers to assemble all the data from the data sources. Each query is translated into a distributed query plan of tasks in numerous stages. The data is returned by the connectors in splits and processed in multiple stages until the final result is available and provided to the user by the coordinator.

Conclusion Générale

Bla bla bla [1]

Bibliographie

[1] A. C. H. Ehrig, U. M. L. Ribeiro, and G. Rozenberg. Graph transformations. 2006.