White Paper

ENABLING SQL ACCESS TO YOUR DATA LAKE WITH PRESTO, HIVE AND SPARK

A COMPARATIVE APPROACH

by Jorge Villamariona, Qubole Technical Marketing

TABLE OF CONTENTS

INTRODUCTION

Regardless of their role, all data professionals including engineers, scientists, analysts, and administrators use Structured Query Language (SQL) as a common language. The number of data stores and formats has increased dramatically since SQL was first created in the 1970s, yet SQL remains just as relevant — if not more so — than when it was first conceived.

Not long ago, at the dawn of big data, data professionals had to wade through unstructured data using MapReduce, which allowed for filtering and sorting (mapping) as well as summarization (reducing) to gain insights from data. Data professionals soon realized the limitations of this approach, and the open source community quickly started building new SQL engines that today provide a more familiar approach to query various data formats.

This new SQL ecosystem allows a number of data professionals to accelerate development time by applying their knowledge when working with multiple systems. This common language also allows ETL (Extract, Transform, Load) systems to facilitate the creation of data pipelines that carry data between different engines and systems.

All of these factors make SQL the right common tool to enable data lake access for all users within your organization. However, the variety and number of engines by itself represents a challenge when it’s time to decide which engine to use.

The intent of this white paper is to assist data professionals choose the optimal engine for their specific use case. In the last section we document two customer success stories as examples of how your organization can succeed with the right data platform.

COMMON TYPES OF DATA WORKLOADS

The common link between the SQL data lake engines is that they are designed to deal with very large data sets and have a distributed computing architecture that provides scalability. These engines offer different degrees of support for speedy processing, interactivity, fault tolerance, and type of workload.   The most common types of workloads can be classified as: batch, streaming, and interactive. Because of the high impact workload type has on selecting an engine, it's important to explore each type of workload in detail.

Batch: Batch workloads allow processing of large volumes of data that have been aggregated over time. The data is processed in batches that are "well defined" and generally scheduled to run automatically — "well defined" in this instance means the data being processed is the data that was available at the time the process initiated execution. The processing is generally complex and time consuming. With batch processing, achieving a greater throughput is more critical than increasing processing speed.

Streaming: n contrast to batch workloads, streaming workloads process data that is generated continuously. This is the type of data generated by web logs, call detail records from phone companies, social networks, etc. Stream processing allows processing of data as it arrives — and often without the need for persisting it first in a datastore. Stream processing supports real-time analytics that leverage the data as soon as it arrives. With stream processing, both throughput and speed are critical factors. With streaming data loads, the value of the information decreases with the passage of time.

Interactive: Interactive processing is used to explore data sets. As query results come back, new and more in-depth questions are reformulated. Fast processing is critical to preserve the user's train of thought. Data scientists, data analysts, and developers often use interactive queries.

PROLIFERATION OF TECHNOLOGY ALTERNATIVES

Data lake implementations must satisfy the requirements of diverse user groups, which leads to multiple and very diverse approaches to data and tools. A wide range of data integration and quality tools rely on SQL, and data lakes need these tools for data ingestion and data pipelines. Even traditional data warehouse and reporting environments include various SQL-based tools for the same reasons.

This diversity of new data repositories (e.g. Apache Hadoop, Amazon S3, Microsoft Azure Object Storage, and Google Cloud Storage) and data processing methods have precipitated the creation of several SQL engines for data lakes. These data lakes are used to aggregate information from multiple sources and act as a central repository for all data. Businesses then use different data lake engines and frameworks on top of these repositories to create analytics and machine learning models that provide greater insight and business value from the data they contain.

Most data lake tools for self-service analytics, data exploration, preparation, and visualization assume a SQL interface. Without SQL, data lake implementations severely limit the number and types of users that can fully take advantage of them. All of these new engines breathe life into much older BI, reporting, and dashboard technologies. Let's take a technical look at the architecture of three of the most popular SQL engines.

Hive

Hive was created by the Qubole co-founders (Ashish Thusoo and Joydeep Sen Sarma) while they were leading the Facebook data team back in 2008, it was created with the intent of providing a SQL-based declarative interface with the ability to also support programmatic capabilities and to store centralized metadata about all data sets in an organization. Hive is an Apache open-source project built on top of Hadoop for querying, summarizing, and analyzing large data sets using a SQL-like interface. The project is noted for bringing the familiarity of relational technology to data lakes with its Hive Query Language (HQL) as well as structures and operations comparable to those used with relational databases such as tables, JOINs, and partitions.

Apache Hive accepts Hive Query Language (similar to SQL) and converts it to Apache Tez jobs. Apache Tez is an application framework that can run complex pipelines of operators to process data. It replaces the MapReduce engine.

Hive's architecture is optimized for batch processing of large ETL jobs and batch SQL queries on very large data sets. Hive features include:

Metastore: The Hive metastore stores the metadata for Hive tables and partitions in a relational database. The metastore provides client access to the information it contains through the metastore service API.

Hive Client and HiveServer2: Users submit HQL statements to the Hive Client or HiveServer2 (HS2). These function as a controller and manage the query lifecycle. After a query completes, the results are returned to the user. HS2 is a long-running daemon that implements many features to improve the speed of planning and optimization of HQL queries. HS2 also supports sessions, which provide features such as temporary tables, which is useful for ETL jobs.

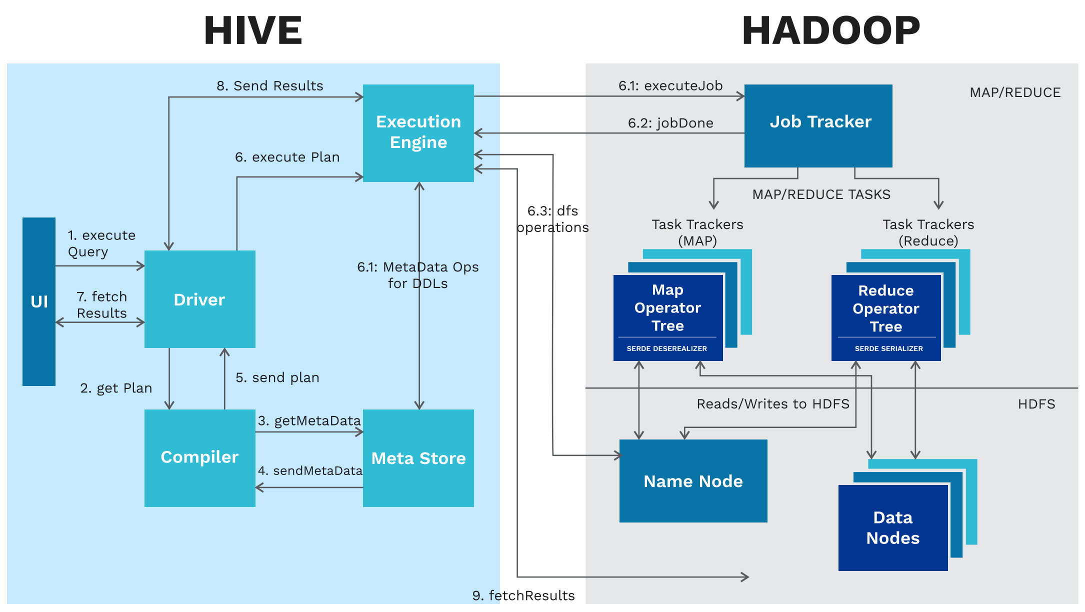
Checkpointing of Intermediate Results: Apache Hive and Apache Tez checkpoint intermediate results. Intermediate results are stored in HDFS (Hadoop File System). Checkpointing allows for fast recovery when tasks fail. Hive can restart tasks from the previous checkpoint.

Speculative Execution: This function helps to improve the speed of queries by redoing work that is lagging due to hardware or networking issues.

Fair Scheduler: Fair scheduling is a method of assigning resources to jobs so that all jobs get, on average, an equal share of resources over time. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, task slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop Scheduler, which forms a queue of jobs, fair scheduling lets short jobs finish in a reasonable time while not starving long jobs. It is also an easy way to share a cluster between multiple users. Fair sharing can also work with job priorities. The priorities are used as weights to determine the fraction of total compute time that each job gets.

Hive Architecture

The diagram below shows the major components of Hive and its interactions with Hadoop:



The diagram above shows how a typical query flows through Hive. The UI calls the Execution Engine through the Driver (step 1 in the diagram). The Driver creates a session handle for the query and sends the query to the Compiler to generate an execution plan (step 2).

UI

The User Interface allows users to submit queries to Hive.

Driver

The Driver receives the queries. The Driver implements session handles and provides execute and fetch APIs modeled after the JDBC/ODBC protocols.

Compiler

The compiler parses queries, performs semantic analysis on different query blocks and query expressions. In turn, the compiler generates an execution plan with the help of the table and partition data from the metastore.

Metastore

The Metastore persists all of the structural information for all tables and partitions in Hive including column and column types, serializers and deserializers necessary to read and write data along with the corresponding HDFS files where the data is stored.

Execution Engine

The Execution Engine carries out the execution plan created by the compiler. The plan is a DAG (Directed Acyclic Graph) of stages. The Execution Engine manages the dependencies between the different steps of the plan and executes these steps on the appropriate system components.

The Compiler gets the necessary metadata from the Metastore (steps 3 and 4). This metadata is used to typecheck the expressions in the query tree as well as to prune partitions based on query predicates. The plan generated by the Compiler (step 5) is a DAG of steps with each step being either a map or reduce job, a metadata operation or an operation on HDFS. The execution engine submits these steps to the appropriate components (steps 6, 6.1, 6.2 and 6.3). In each task (mapper or reducer) the deserializer associated with the table or intermediate outputs is used to read the rows from HDFS files and these are passed through the associated operator tree.

Once the output is generated, it is written to a temporary HDFS file though the serializer (this happens in the mapper in case the operation does not need a reduce). The temporary files are used to provide data to subsequent map/reduce stages of the plan.

For DML operations the final temporary file is moved to the table's location. For queries, the contents of the temporary file are read by the execution engine directly from HDFS as part of the fetch call from the Driver (steps 7, 8 and 9).

Hive Use Cases

- Business intelligence, reporting, and dashboards due to compatibility with all major BI tool vendors such as Tableau, Looker, and Pentaho.
- Large data set persistence.

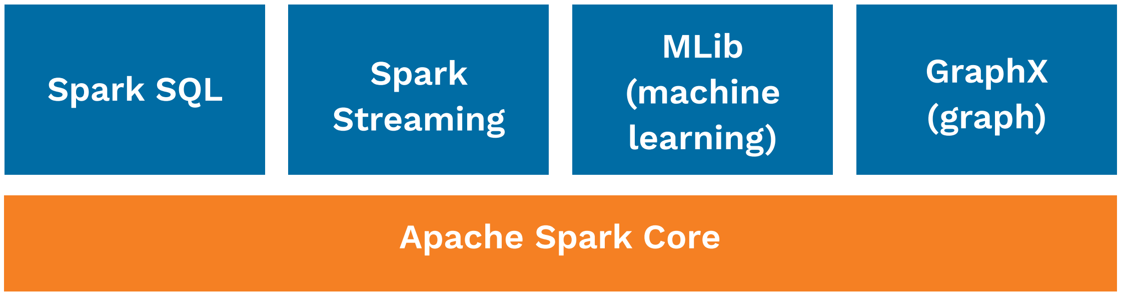
Qubole provides an enhanced implementation of Apache Hive by offering agent technology that augments the original Hive architecture with a self-managing and self-optimizing platform. Additionally, Qubole Hive seamlessly integrates with existing data sources and third-party tools while providing best-in-class security.

Spark

Spark was originally developed at the University of California, Berkeley's AMPLab in 2009 as a fast and general computational engine for Hadoop data. The Spark code was later donated to the Apache Software foundation which maintains it since that time. Spark provides a simple and expressive programming model that supports a wide range of machine learning and data processing applications. Spark's in-memory data model and fast processing makes it particularly suitable for applications such as:

- Machine learning and graph processing
- Stream processing
- Interactive queries against in-memory data
- Batch ETL processing

The following diagram shows the components of Spark:



Apache Spark Core

The Spark Core is the underlying general execution engine for the Spark platform upon which all other functionality is built. It is responsible for I/O functionality, job scheduling and monitoring, task dispatching, networking with different storage systems, fault recovery, and memory management.

Spark SQL

Spark SQL is used to perform SQL queries on distributed data sets. It can be used for data exploration, data pipeline creation, and data transformation. It can read and join heterogeneous data repositories. Spark SQL represents structured data as Spark DataFrames (a DataFrame is a data set organized into named columns; it is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations). These are internally represented as RDDs (resilient distributed data sets) with an associated schema.

When computing a result, the same execution engine is used, independent of which API/ language is used to express the computation. This means that Spark SQL developers can combine SQL queries with code written in Python, Scala, Java, and R. Because it's the same execution engine, developers can switch back and forth to select the optimal way to express a given transformation.

Spark Streaming

Spark Streaming allows developers to create streaming jobs the same way they create batch jobs. Spark Streaming can read data from HDFS, Flume, Kafka, Twitter, and ZeroMQ. Spark Structured Streaming is built to leverage the core Spark engine. It can partition large streaming data sets across multiple worker nodes to be processed in parallel. Spark Structured Streaming expresses SQL queries over streaming data, taking advantage of the Spark SQL component described above. Developers can also define their own custom data sources.

MLlib (Machine Learning Library)

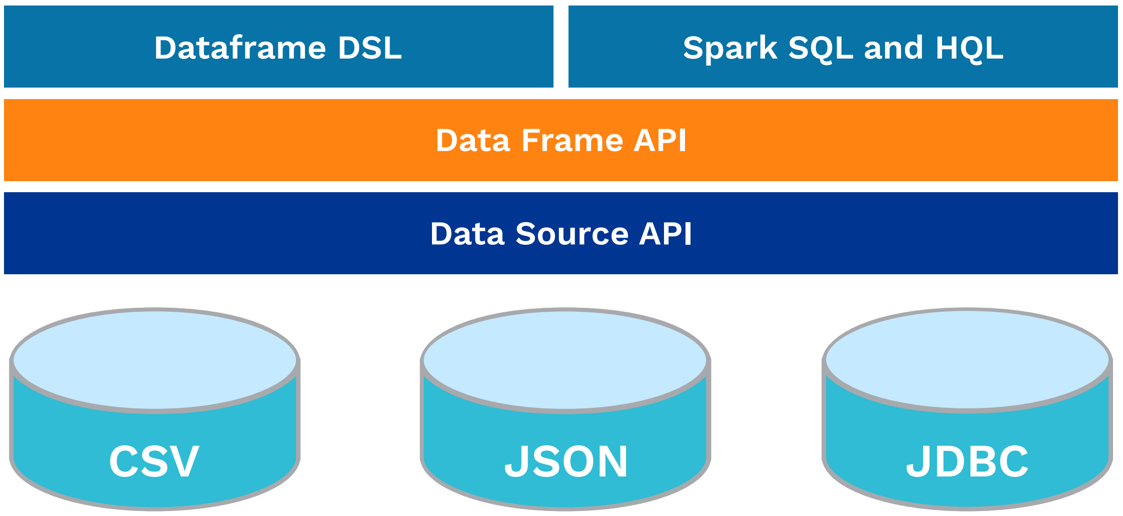
MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, such as classification, regression, clustering, collaborative filtering, and dimensionality reduction. It also includes optimization primitives such as summary statistics.

GraphX

GraphX is Apache Spark's API for graphs and graph-parallel computation. GraphX unifies ETL, exploratory analysis, and iterative graph computation within a single system.

Spark SQL Architecture

At a very high level, the flexible data access layer is at the bottom, which allows access to multiple data formats. The Data Source API is used to read structured and semi-structured data and converted into a DataFrame API. A DataFrame is equivalent to a relational database table. Once the DataFrame is created, it can be accessed through a programming language (DSL, or domain-specific language) or via SQL or HQL.The following diagram illustrates the Spark SQL architecture:



Spark SQL Use Cases

- Real-time analytics such as stock market analysis, social media sentiment analysis, and credit card fraud.
- Life sciences applications that demand large data sets such as genome sequencing to predict predisposition to disease.

Qubole's Spark implementation greatly improves the performance of Spark workloads with enhancements such as fast storage, distributed caching, advanced indexing, and metadata caching capabilities. Other enhancements include job isolation on multi-tenant clusters and SparkLens, an open source Spark profiler that provides insights into the Spark application.

Presto

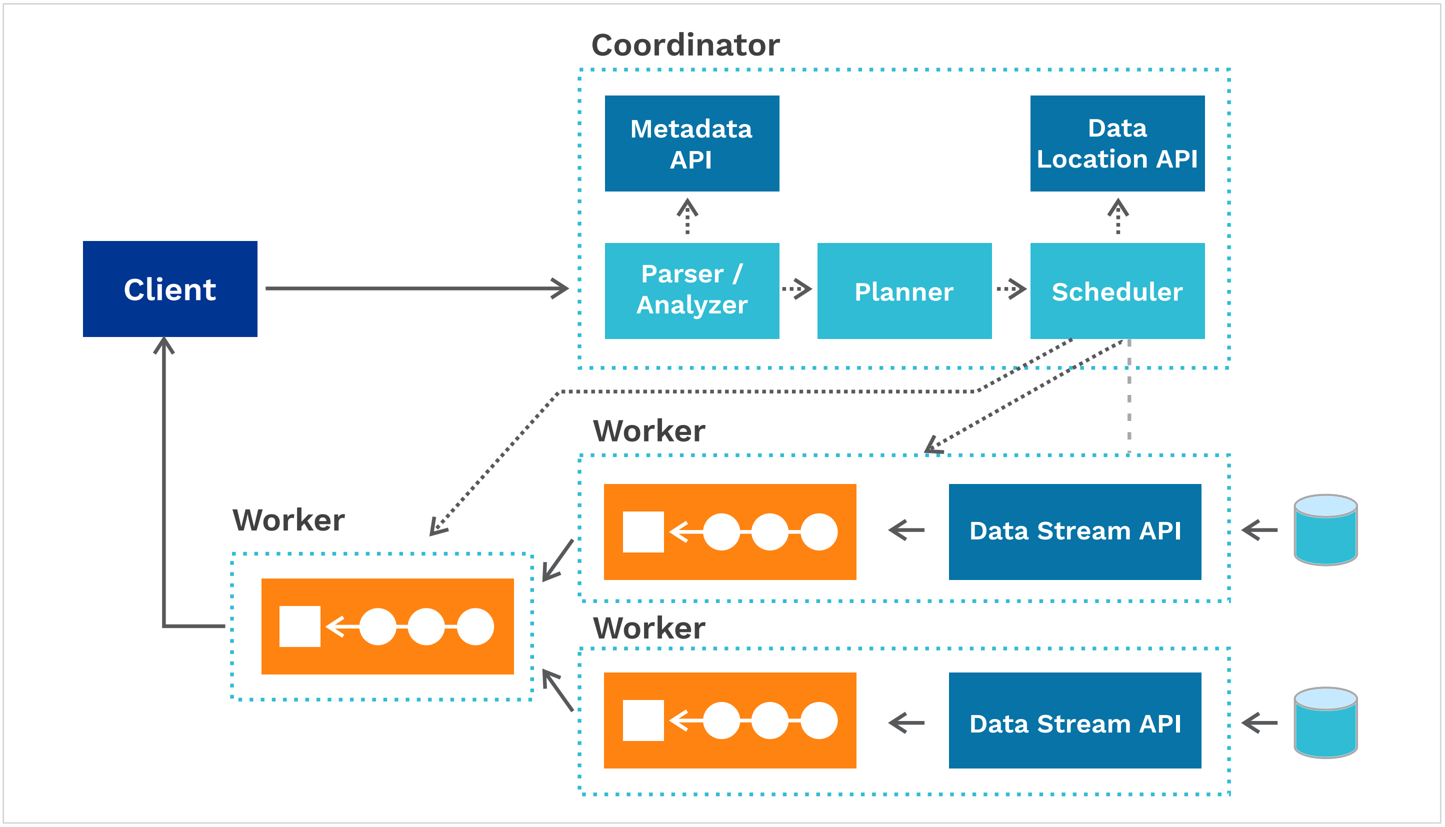
Presto is an open source distributed SQL query engine developed by Facebook. Facebook started development efforts on Presto in 2012, and announced its release as open source for Apache Hadoop in 2013. In 2014, Netflix disclosed they used Presto on 10 petabytes of data stored on the Amazon Simple Storage Service (S3). Presto is used for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

Presto was designed and written completely for interactive analytics and approaches the speed of commercial data warehouses. Facebook uses Presto for interactive queries against several internal data stores including its 300-petabyte data warehouse. Over 1,000 Facebook employees use Presto everyday to run more than 30,000 queries that in total scan over a petabyte each per day.

The execution model of Presto is fundamentally different from Hive or MapReduce. Hive translates queries into multiple stages of MapReduce tasks that execute one after the other. Each task reads inputs from disk storage and writes intermediate output back to disk. In contrast, the Presto engine does not use MapReduce. It employs a custom query and execution engine with operators designed to support SQL semantics. In addition to improved scheduling, processing is in memory and pipelined across the network between stages. This avoids unnecessary I/O and associated latency overhead. The pipelined execution model runs multiple stages at once and streams data from one stage to the next as it becomes available. This significantly reduces end-to-end latency for many types of queries.

Presto Architecture

Presto runs on one or more machines that form a cluster. Presto instances include one coordinator and a number of workers. The Presto coordinator is the machine to which clients submit their queries. The coordinator is responsible for parsing, planning, and scheduling queries among the workers. Adding workers increases parallelism and yields faster query processing. The following figure illustrates the key components of Presto.



Clients

Clients submit queries to Presto. Clients use JDBC/ODBC/REST protocols to talk to the Presto coordinator.

Coordinator

The Presto coordinator manages worker nodes, parses the queries, generates execution plans, and manages the execution of queries. It also manages the delivery of the data between tasks during query execution. The coordinator translates queries into a logical plan. The logical plan has a series of stages and each stage is then executed in a distributed manner using several tasks across workers. This is similar to other distributed query execution engines like Hive and Spark.

Workers

The Presto workers execute tasks and process data. This is where the actual data processing happens.

Communication

Presto workers announce themselves to the coordinator through the discovery server. All of the communication in Presto between the coordinator, the workers, and the client occurs through REST APIs.

Connectors

Presto has a federated query model where each data source is a connector. Presto connectors are similar to database drivers. Some of the available connectors on the Presto project are Kafka, Cassandra, Hive, Accumulo, MongoDB, MySQL, PostgreSQL, Redis, etc.

Catalog

Catalogs are associated with a specific connector. The Presto Catalog Manager decides how to query a particular data source. When writing a query in Presto, you can use the fully qualified name that contains connector.schemaname.tablename. For instance, if you have a Hive table called customer in a database called sales, you can refer it as hive.sales.customer.

Presto Use Cases

- Interactive (ad hoc) queries
- Reporting
- Supporting analytical tools such as dashboards
- Self-service BI

Qubole has optimized Presto for the cloud. Qubole's enhancements allow for dynamic cluster sizing based on workload and termination of idle clusters — ensuring high reliability while reducing compute costs. Qubole's Presto clusters support multi-tenancy and provide logs and metrics to track performance of queries.

COMPARISON AND SUITABILITY OF EACH ENGINE

You should evaluate several factors when considering the optimal cloud SQL engine and cluster pairing. The tables below highlight the reasons to choose one engine over another. They also identify each item as either a fact or a suggestion. Facts always hold true, but suggestions should be weighed against your particular use case.

Architecture Breakdown

		HIVE using MapReduce	HIVE using Tez	Presto	Spark
Fact	Write to disk between consecutive jobs	✓			
Fact	Streams data in memory between tasks		✓	✓	✓
Fact	ANSI SQL compliant			✓	✓

Usage Breakdown

		HIVE using MapReduce	HIVE using Tez	Presto	Spark
Suggestion	Appropriate for Data Analysts	✓	✓	✓	
Suggestion	Ad-Hoc use case appropriate	✓	✓	✓	
Suggestion	Appropriate for Data Engineers	✓	✓		✓
Fact	Data volume above 100 TB	✓	✓		✓
Fact	Dynamic Partitioning support	✓	✓		
Suggestion	Batch Workload / ETL use case appropriate	✓	✓		✓
Fact	Notebook support available	✓	✓	✓	✓
Suggestion	Appropriate for Data Scientists				✓

SQL Breakdown

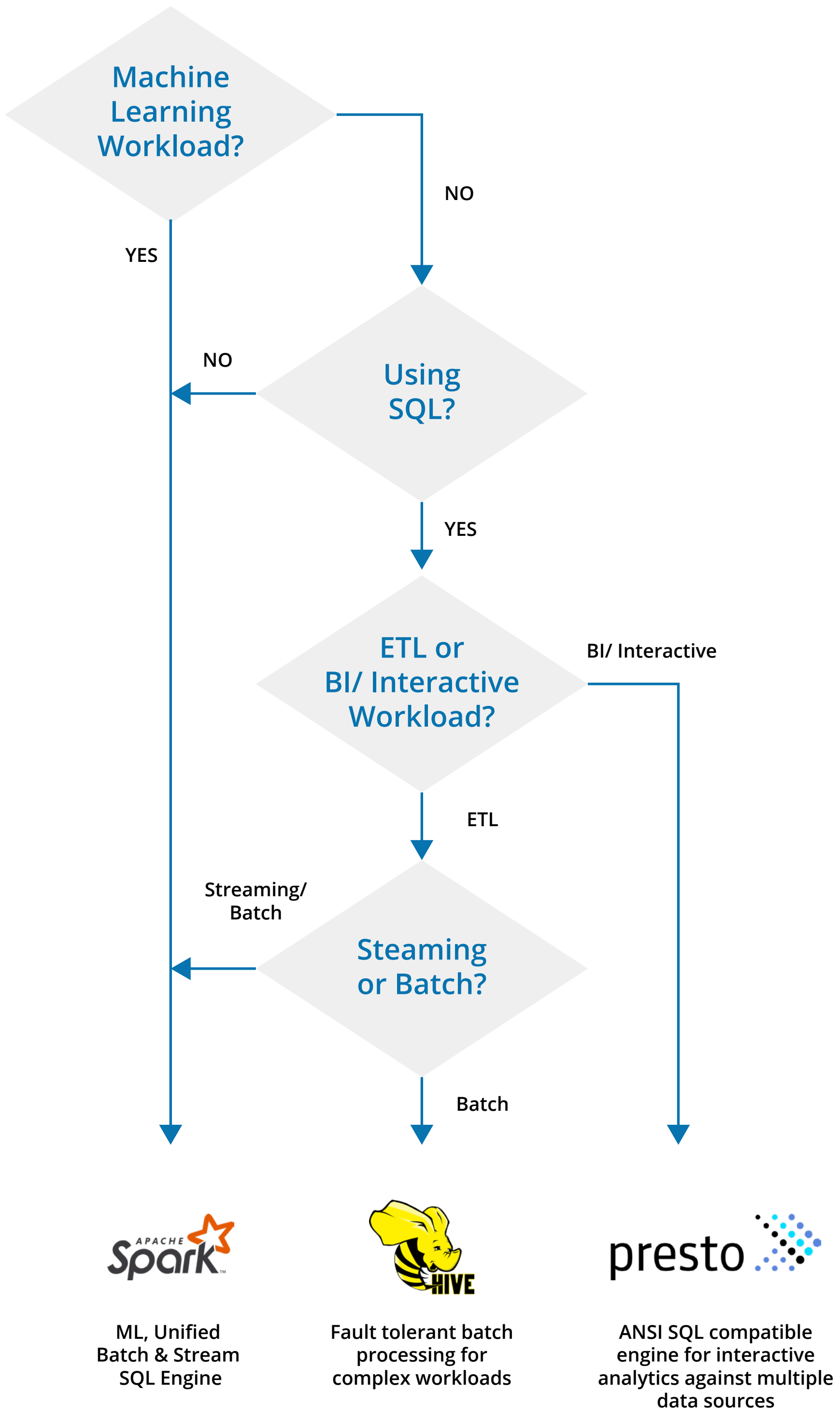
		HIVE using MapReduce	HIVE using Tez	Presto	Spark
Fact	In Memory join option		✓	(default behavior)	✓
Suggestion	Appropriate for simple joins	✓	✓	✓	✓
Suggestion	Appropriate for complex joins	✓	✓	✓	
Suggestion	4 or more tables in join	✓	✓	✓	✓

File Format Breakdown

		HIVE using MapReduce	HIVE using Tez	Presto	Spark
Suggestion	Prefers the Parquet format				✓
Suggestion	Prefers the ORC format	✓	✓	✓	✓

WHICH ENGINE SHOULD YOU USE (CONCLUSION /RECOMMENDATIONS)

SQL engines for data lakes are part of a quickly evolving field with a lot of activity and fast innovation. Engines can be differentiated on several characteristics including: type of workload, and processing speed. The decision tree below follows a functional and use case–based approach to arrive at a SQL engine recommendation.


Machine Learning Workload?
NO
YES
NO
Using SQL?
YES
ETL or BI/ Interactive Workload?
BI/ Interactive
ETL
Streaming/ Batch
Steaming or Batch?
Batch
Apache Spark
ML, Unified Batch & Stream SQL Engine
HIVE
Fault tolerant batch processing for complex workloads
presto
ANSI SQL compatible engine for interactive analytics against multiple data sources

When selecting an engine, it is critical to keep in mind that every SQL engine was built to deal with a particular problem that may include one or more types of workload (batch, streaming, and interactive). Most importantly, you should keep in mind that there is no one-size-fits-all SQL engine that can fulfill all possible use cases. For this reason, it is important to select a platform that offers multiple SQL engines.

Qubole is an open, simple, and secure data lake platform for machine learning, streaming analytics, data exploration, and ad-hoc analytics. No other platform radically simplifies data management, data engineering and run-time services like Qubole. Qubole provides multiple SQL engines and enables reliable, secure data access and collaboration among users while reducing time to value, improving productivity, and lowering cloud data lake costs from day one. The Qubole Open Data Lake Platform:

- Provides a unified environment for creation and orchestration of multiple data pipelines to dynamically build trusted datasets for ad-hoc analytics, streaming analytics, and machine learning.
- Optimizes and rebalances underlying multi-cloud infrastructure for the best financial outcome while supporting the unmatched levels of concurrent users, and workloads at any point in time.
- Enables collaboration through workbench between data scientists, data engineers, and data analysts with shareable notebooks, queries, and dashboards.
- Improves the security, governance, reliability, and accessibility of data residing in data lakes.
- Provides APIs, and connectors to 3rd party tools such as Tableau, Looker for analytics, RStudio, H2O.ai for machine learning use cases.

CUSTOMER SUCCESS STORIES

In a not so distant past the following organizations faced the challenge of democratizing access to their data lake without compromising security while staying within budget. They were able to achieve this by taking advantage of Qubole's open data lake platform which allowed them to enable self-service while successfully addressing multiple use cases.

Ibotta

Ibotta is a mobile technology company transforming the traditional rebates industry by providing in-app cashback rewards on receipts and online purchases for groceries, electronics, clothing, gifts, supplies, restaurant dining, and more for anyone with a smartphone. Today, Ibotta is one of the most used shopping apps in the United States, driving over $5 billion in purchases per year to companies like Target, Costco and Walmart. Ibotta has over 23 million total downloads and has paid out more than $250 million to users since its founding in 2012.

Business Need

Ibotta's ability to track consumer engagement at the point of sale allows them to provide a 360 degree view of analytics on purchase attribution back to their partners. Their app provides Business Analytics that allows retailers and brands to make more informed buying decisions in-store and online. This information helps retailers and brands engage with their customers at a very personal level, as well as optimize future investments in new products and marketing campaigns.

The insights they provided were so valuable, their partners kept requesting more detailed information and features to better engage with existing and new customers. As a result, the company decided to focus on expanding advertising and eCommerce segments in the product. This, in turn, has created new revenue streams that they can reinvest back into the business and increase our consumer's savings. It was at this point Ibotta began to see a huge growth in the data. They needed a solution to help scale the company's goals.

While the growth in data has helped improve insight, it also had a significant impact on the data infrastructure and was a key driver for them to change technology operations.

Since March of 2017, Ibotta's data has grown by over 70x, to nearly 1PB, with over 20 TB of new data coming in daily. The biggest driver of data growth has come from generating first-party data features in order to improve their users' personalized experiences.

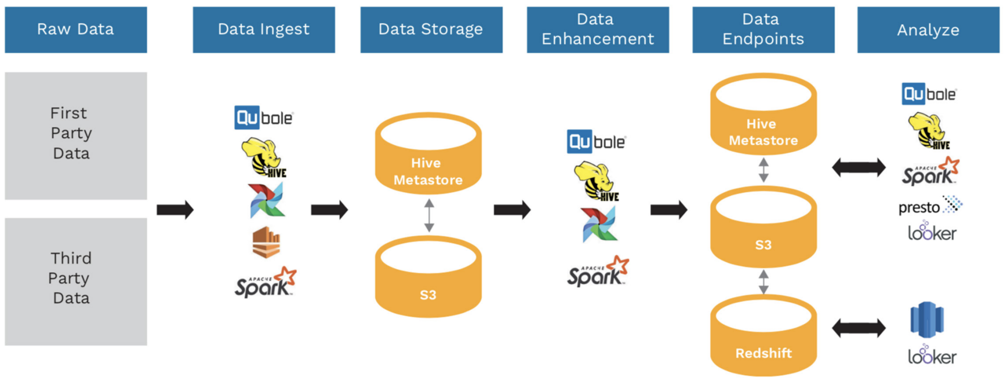
Solution

To address the problems faced by the data teams, they built a cost-efficient self-service data platform. Ibotta needed a way for every user -- particularly Data Science and Engineering -- to have self-service access to the data; and to be able to use the right tools for their use cases with big data engines like Apache Spark, Hive, and Presto. The data team needed to be able to complete the tasks of preparing data for those Data Science and Engineering Teams at the same time. Qubole simultaneously provided an answer to the demands of both teams, those perfecting operations as well as analyzing the data.

Impact

Ibotta's Data Science and Engineering teams were immediately empowered once Qubole was in place. They achieved the goal of self-service access to the data and efficient scale of compute resources in AWS EC2 for big data workloads. The following diagram depicts how the Qubole Platform enables their Data Science and Analytics teams to focus less on mundane tasks and more on what matters, becoming the foundation for their data-driven culture.

The following diagram depicts how the Qubole Platform enables their Data Science and Analytics teams to focus less on mundane tasks and more on what matters, becoming the foundation for their data-driven culture.



Ibotta is well on its way to building the world's starting point for rewarded shopping by partnering with Qubole and building out their cloud data lake. More than ever, they are focusing on delivering next generation ecommerce features and products that help drive both a better user experience and partner monetization. Qubole allows them to spend time developing and productionalizing scalable data products, more importantly concentrating on bringing value back to their users and company.

TiVo

TiVo Corporation (NASDAQ: TIVO) is a global leader in entertainment technology and audience insights. From the interactive program guide to the DVR, TiVo delivers innovative products and licensable technologies that revolutionize how people find content across a changing media landscape. TiVo enables the world's leading media and entertainment providers to deliver the ultimate entertainment experience.

Business Need

TiVo's entertainment platform consolidates terabytes of data every month: raw viewership data from cable boxes in millions of homes, purchasing data from first and third parties, and location-based consumer data. TiVo's network and advertising partners need reports based on this data to better understand the viewing and purchasing behaviors of various customer demographics.

Because TiVo's partners often have drastically different reporting needs, all of this data needs to be transformed, segmented, and packaged in several different ways to satisfy their requirements. TiVo's data engineering team needed a way to do this efficiently, affordably, and at scale.

Solution

To more readily make its data available for analytics operations, TiVo deployed a data lake on Amazon S3. The data lake allows the company to store any data type in a single convenient repository. Data can be collected from multiple sources and moved into the data lake in its original format. This allows TiVo to scale to data of any size, while saving time by eliminating the need to define data structures, schema, and transformations.

TiVo's data engineering team chose Presto as its query engine based on its flexibility and efficiency. The team then decided to use Qubole, which allows it to easily scale and manage its Presto clusters and more easily audit queries and debug commands. The Activation Platform provided out-of-the box functionality that Tivo would have needed to create from scratch if it had chosen to deploy Presto on top of AWS EC2 without Qubole. TiVo's data engineers found Qubole simple to deploy: after configuring permissions for AWS and the Qubole website, they were ready to run queries.

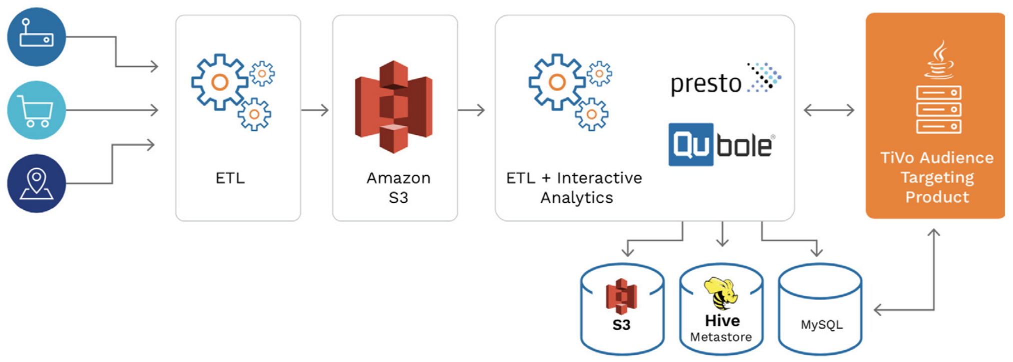
Qubole templates automate every element of TiVo's queries, including activating Presto clusters and scaling the clusters based on usage. This eliminates the need to manually write scripts to tell Presto how to behave. The query results are then saved in Amazon S3 buckets for later auditing. Through its service administration portal, TiVo can track its queries and view and download intermediate queries and results.

Qubole's rich feature set includes the ability to label individual clusters according to their workload. TiVo labels clusters (e.g. "ETL," "Reporting," and "Interactive") to help its team of developers stay organized. Qubole's notebook feature provides a convenient way to save, share, and re-run a set of queries on a data source – for example, to track changes in the underlying data over time, or to provide different views using different parameters.

Impact

Qubole streamlines the process of generating reports for TiVo's partners, whose needs are constantly changing week-to-week in terms of scope, data type, and time (weekly, monthly, yearly). The financial and human resources required to run data science operations depend heavily on the complexity of the reports being run. Today, TiVo can do more with fewer resources by automating its reporting with Qubole.

Qubole provides a simple, intuitive way for TiVo's partners to set up and schedule reports tailored for their specific requirements. This self-service feature provides TiVo's network and advertising partners with the business intelligence tools they need to interpret data from highly targeted demographics at a cadence that works best for them. Having access to any kind of viewership, and purchasing only the reporting they need, allows networks and advertisers to more easily customize and scale new media products to thrive in a highly competitive space.



469 El Camino Real, Suite 201
Santa Clara, CA 95050
(855) 423-6674 | info@qubole.com

WWW.QUBOLE.COM