Scaling with Presto on Spark

October 26, 2021

Rohan Pednekar

Co-authors
Shradha Ambekar, Staff Software Engineer at Intuit
Ariel Weisberg, Software Engineer at Facebook

Overview

Presto was originally designed to run interactive queries against data warehouses, but now it has evolved into a unified SQL engine on top of open data lake analytics for both interactive and batch workloads. Popular workloads on data lakes include:

1. Reporting and dashboarding

This includes serving custom reporting for both internal and external developers for business insights and also many organizations using Presto for interactive A/B testing analytics. A defining characteristic of this use case is a requirement for low latency. It requires tens to hundreds of milliseconds at very high QPS, and not surprisingly this use case is almost exclusively using Presto and that's what Presto is designed for.

2. Data science with SQL notebooks

This use case is one of ad hoc analysis and typically needs moderate latency ranging from seconds to minutes. These are the queries of data scientist, and business analysts who want to perform compact ad hoc analysis to understand product usage, for example, user trends and how to improve the product. The QPS is relatively lower because users have to manually initiate these queries.

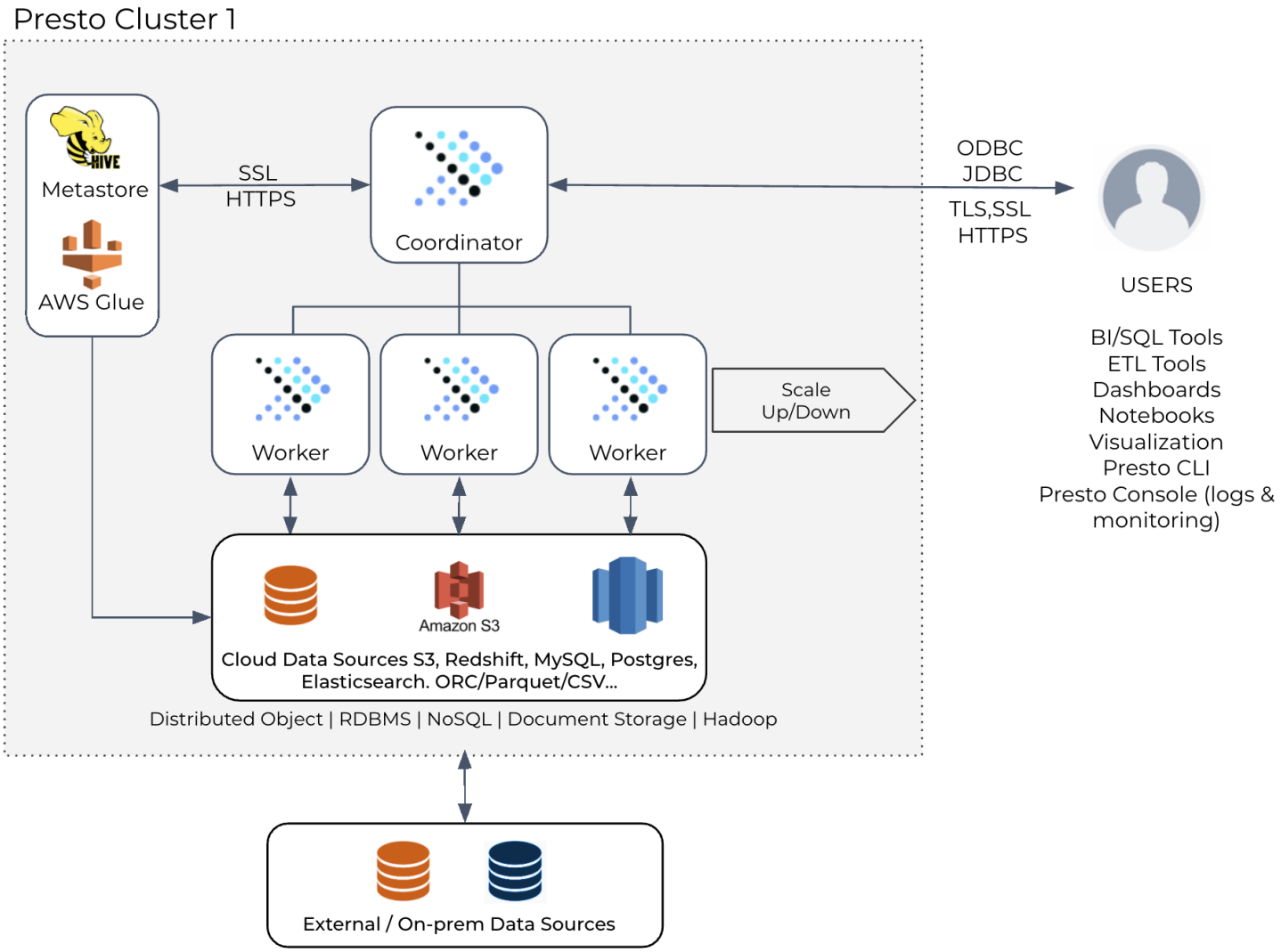
3. Batch processing for large data pipelines

These are scheduled jobs that are running every day, hour, or whenever the data is ready. They often contain queries over very large volumes of data and the latency can be up to tens of hours and processing can range from CPU days to years and terabytes to petabytes of data.

Presto works exceptionally effectively for ad-hoc or interactive queries today, and even some batch queries, with the constraint that the entire query must fit in memory and run quickly enough that fault tolerance is not required. Most ETL batch workloads that don’t fit in this box are running on “very big data” compute engines like Apache Spark. Having multiple compute engines with different SQL dialects and

APIs makes managing and scaling these workloads complicated for data platform teams. Hence, Facebook decided to simplify and build Presto on Spark as the path to further scale Presto. Before we get into Presto on Spark, let me explain a bit more about the architecture of each of these two popular engines.

Presto's Architecture

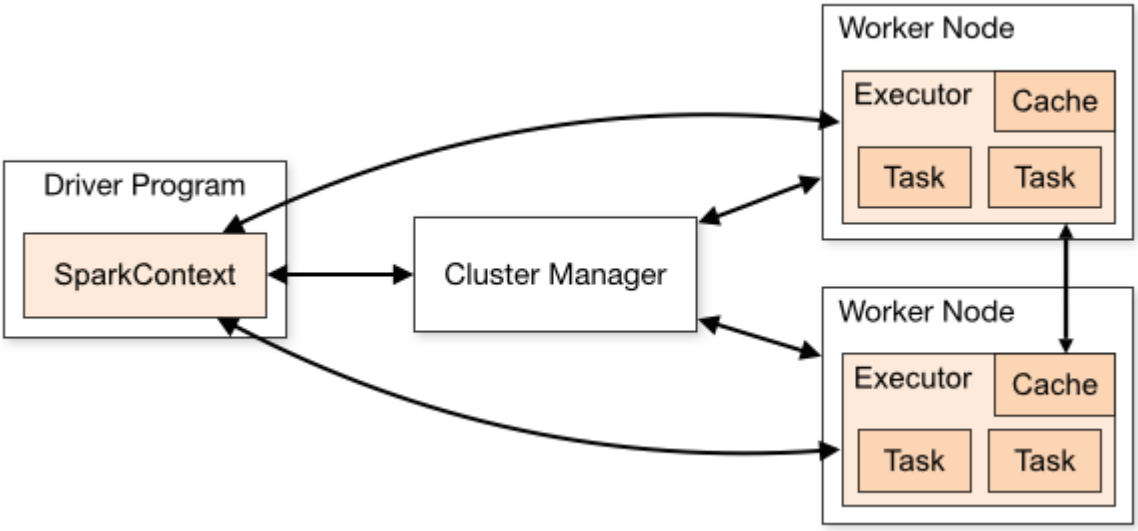


Presto is designed for low latency and follows the classic MPP architecture; it uses in-memory streaming shuffle to achieve low latency. Presto has a single shared coordinator per cluster with an associated pool of workers. Presto tries to schedule as many queries as possible on the same Presto worker (shared executor), in order to support multi-tenancy.

This architecture provides very low latency scheduling of tasks and allows concurrent processing of multiple stages of a query, but the tradeoff is that the coordinator is a SPOF and bottleneck, and queries are poorly isolated across the entire cluster.

Additionally streaming shuffle does not allow for much fault tolerance further impacting the reliability of long running queries.

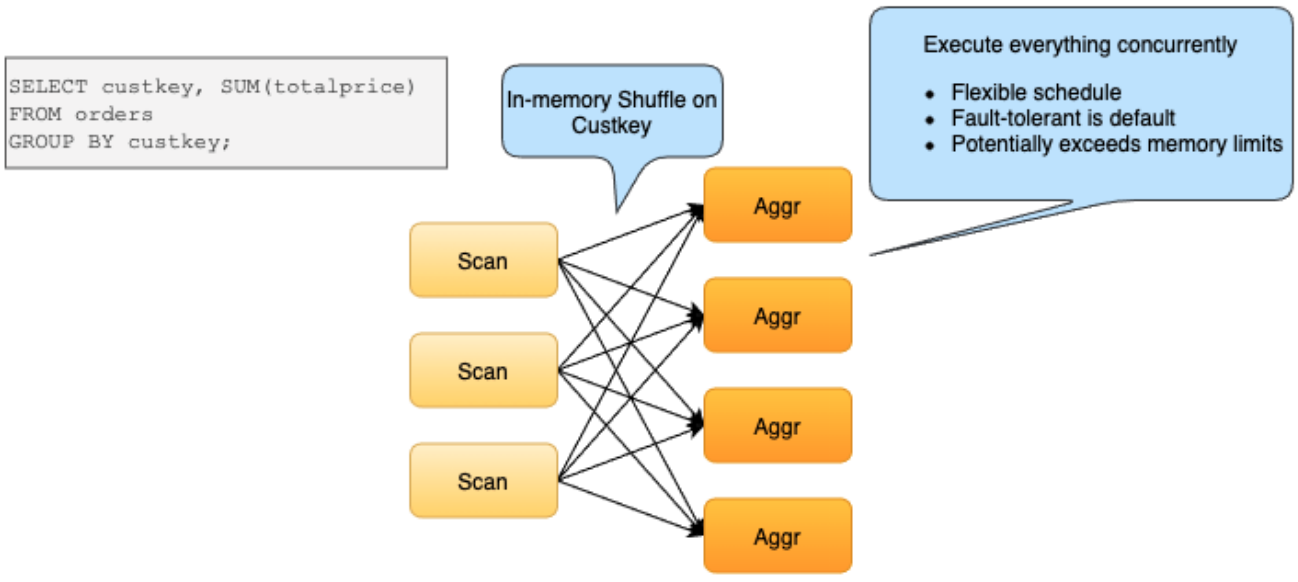
Spark's Architecture



On other hand, Apache Spark is designed for scalability from the very beginning and it implements a Map-Reduce architecture. Shuffle is materialized to disk fully between stages of execution with the capability to preempt or restart any task. Spark maintains an isolated Driver to coordinate each query and runs tasks in isolated containers scheduled on demand. These differences improve reliability and reduce overall operational overhead.

Why Presto alone isn’t a good fit for batch workloads?

Scaling an MPP architecture database to batch data processing over Internet-scale datasets is known to be an extremely difficult problem [1]. To simplify this let's examine the below aggregation query. Essentially this query goes over the orders table in TPCH and does aggregation grouping on custom keys, and summing the total price. Presto leverages in-memory shuffle and executes shuffle on the custom key, after reading the data and doing aggregation for the same key, on each worker.



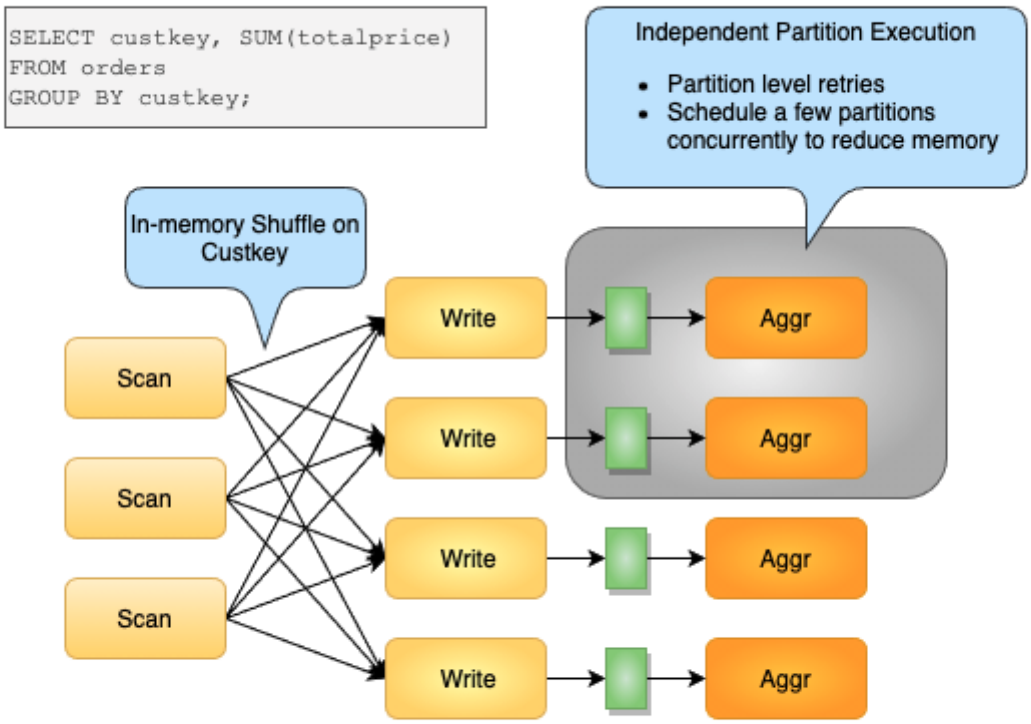
Classic Massively Parallel Processing

Doing in-memory shuffle means the producer will buffer data in memory and wait for the data to be fetched by the consumer as a result. We have to execute all the tasks, before and after the exchange at the same time. So thinking about in the mapreduce world all the mappers and the reducer have to be run concurrently. This makes in-memory shuffle an all-or-nothing exclusion model.

This causes inflexible scheduling and scaling query size becomes more difficult because everything is running concurrently. In the aggregation phase the query may exceed the memory limit because everything has to be held in the memory in hash tables in order to track each group (custkey).

Additionally we are limited by the size of a cluster in how many nodes we can hash partition the data across to avoid having to fit it all in memory. Using distributed disk (Presto-on-Spark, Presto Unlimited) we can partition the data further and are only limited by the number of open files and even that is a limit that can be scaled quite a bit by a shuffle service.

For that reason it makes Presto difficult to scale to very large and complex batch pipelines. Such pipelines remain running for hours, all to join and aggregate over a huge amount of data. This motivated the development of Presto Unlimited which adapts Presto's MPP design to large ETL workloads, and improves user experience at scale.



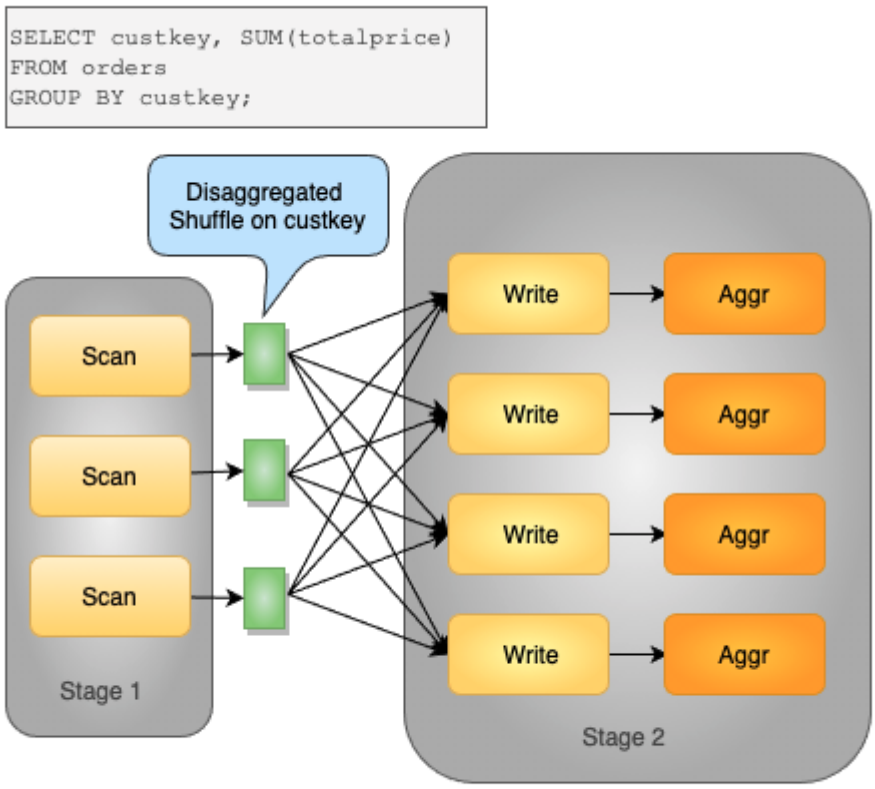
Presto Unlimited: MPP SQL Engine

While Presto Unlimited solved part of the problem by allowing shuffle to be partitioned over distributed disk, it didn't fully solve fault tolerance, and did nothing to improve isolation and resource management.

Presto on Spark

Presto on Spark is an integration between Presto and Spark that leverages Presto's compiler/evaluation as a library with Spark's RDD API used to manage execution of Presto's embedded evaluation. This is similar to how Google chose to embed F1 Query inside their MapReduce framework.

The high level goal is to bring a fully disaggregated shuffle to Presto's MPP run time and we achieved this by adding a materialization step right after the shuffle. The materialized shuffle is modeled as a temporary partition table, which brings more flexible execution after shuffle and allows to partition level retries. With Presto on Spark, we can do a fully disaggregated shuffle on custom keys for the above query both on mapper and reducer side, this means all mappers and reducers can be independently scheduled and are independently retriable.



Presto Evaluation Liabrary on Spark Runtime

Presto On Spark at Intuit

Superglue is a homegrown tool at Intuit that helps users build, manage and monitor data pipelines. Superglue was built to democratize data for analysts and data scientists. Superglue minimizes time spent developing and debugging data pipelines, and maximizes time spent on building business insights and AI/ML.

Many analysts at Intuit use Presto (AWS Athena) to explore data in the Data Lake/S3. These analysts would spend several hours converting these exploration SQLs written for Presto to Spark SQL to operationalize/schedule them as data pipelines in Superglue. To minimize SQL dialect conversion issues and associated productivity loss for analysts, the Intuit team started to explore various options including query translation, query virtualization, and presto on spark. After a quick POC, Intuit decided to go with Presto on Spark as it leverages Presto's compiler/evaluation as a library (no query conversion is required) and Spark's scalable data processing capabilities.

Presto on Spark is now in production at Intuit. In three months, there are hundreds of critical pipelines that have thousands of jobs running on Presto On Spark via Superglue.

Presto on Spark runs as a library that is submitted with spark-submit or Jar Task on the Spark cluster. Scheduled batch data pipelines are launched on ephemeral clusters to take advantage of resource isolation, manage cost, and minimize operational overhead. DDL statements are executed against Hive and DML statements are executed against Presto. This enables analysts to write Hive-compatible DDL and the user experience remains unchanged.

This solution helped enable a performant and scalable platform with **seamless end-to-end experience** for analysts to explore and process data. It thereby improved analysts' productivity and empowered them to **deliver insights at high speed**.

When To Use Spark's Execution Engine With Presto

Spark is the tool of choice across the industry for running large scale complex batch ETL pipelines. Presto on Spark heavily benefits pipelines written in Presto that operate on terabytes/petabytes of data, as it takes advantage of Spark's large scale processing capabilities. The biggest win here is that no query conversion is required and you can leverage Spark for

- Scaling to larger data volumes
- Scaling Presto's resource management to larger clusters
- Increase reliability and elasticity of Presto as a compute engine

Why 'Presto on Spark' matters

We tried to achieve the following to adapt 'Presto on Spark' to Internet-scale batch workloads [2]:

- Fully disaggregated shuffles
- Isolated executors
- Presto resource management, Different Scheduler, Speculative Execution, etc.

A unified option for batch data processing and ad hoc is very important for creating the experience of queries that scale instead of fail without requiring rewrites between different SQL dialects. We believe this is only **a first step towards more confluence between the Spark and the Presto communities, and a major step towards enabling unified SQL experience between interactive and batch use cases**. Today many internet giants like Facebook, etc. have moved over to Presto on Spark and we have seen many organizations including Intuit started running their complex data pipelines in production with Presto on Spark.

"Presto on Spark" is one of the most active development areas in Presto, feel free check it out and please give it a star! If you have any questions, feel free to ask in the PrestoDB Slack Channel.

Reference

[1] MapReduce: Simplified Data Processing on Large Clusters [2] Presto-on-Spark: A Tale of Two Computation Engines