大数据开发之通过 Spark 来扩展 Presto

作者：@零度

2022-01-18 • 本文字数：3218 字 • 阅读完需：约 11 分钟

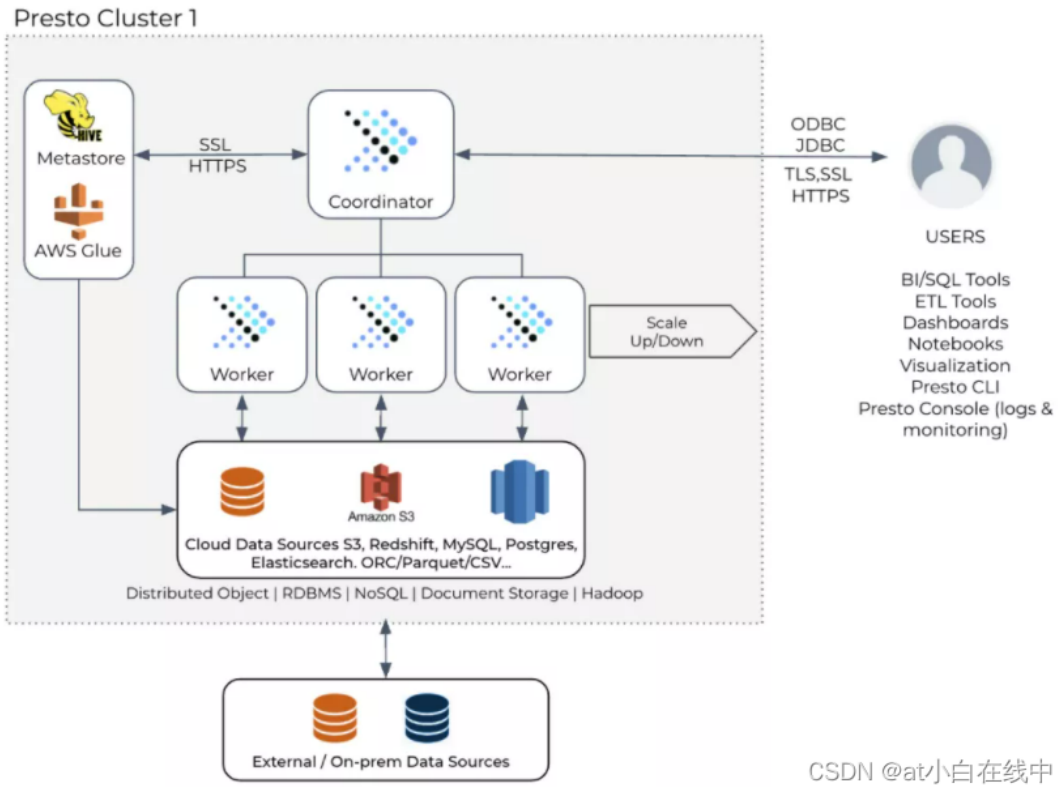
概述

Presto 最初设计是对数据仓库中的数据运行交互式查询，但现在它已经发展成为一个位于开放数据湖分析之上的统一 SQL 引擎，用于交互式和批处理工作负载，数据湖上的流行工作负载包括：

•**报告和仪表盘**：这包括为内部和外部开发人员提供自定义报告以获取业务洞察力，以及许多使用 Presto 进行交互式 A/B 测试分析的组织。这个用例的典型特征是要求低延迟。它在非常高的 QPS 下需要数十到数百毫秒，毫不奇怪，这个场景几乎完全使用 Presto，而这正是 Presto 的设计目的。•**使用 SQL notebooks 的数据科学家**：这个用例是一种 ad-hoc 分析，通常需要从几秒到几分钟的中等延迟。这些是数据科学家和业务分析师的查询，他们希望执行紧凑的临时分析以了解产品使用情况，例如用户趋势和如何改进产品。QPS 相对较低，因为用户必须手动启动这些查询。•**用于大型数据管道的批处理**：这些是每天、每小时或在数据准备就绪时运行的计划作业（scheduled jobs）。它们通常包含对大量数据的查询，延迟可达数十小时，处理的数据从 TB 到 PB 不等。

Presto 对于 ad-hoc 或交互式查询，甚至一些批处理查询都非常有效，其约束条件是整个查询必须适合内存并且运行速度足够快，不需要容错。大多数不适合此框的 ETL 批处理工作负载运行在像 Apache Spark 这种"大数据"计算引擎上。使用不同 SQL 方言和 API 的多个计算引擎使得数据平台团队管理和扩展这些工作负载变得复杂。因此，Facebook 决定使用 Presto on Spark 这种方式来简化和进一步扩展 Presto。在我们介绍 Presto on Spark 之前，我们先来看下这两种流行计算引擎的框架。

Presto 的架构

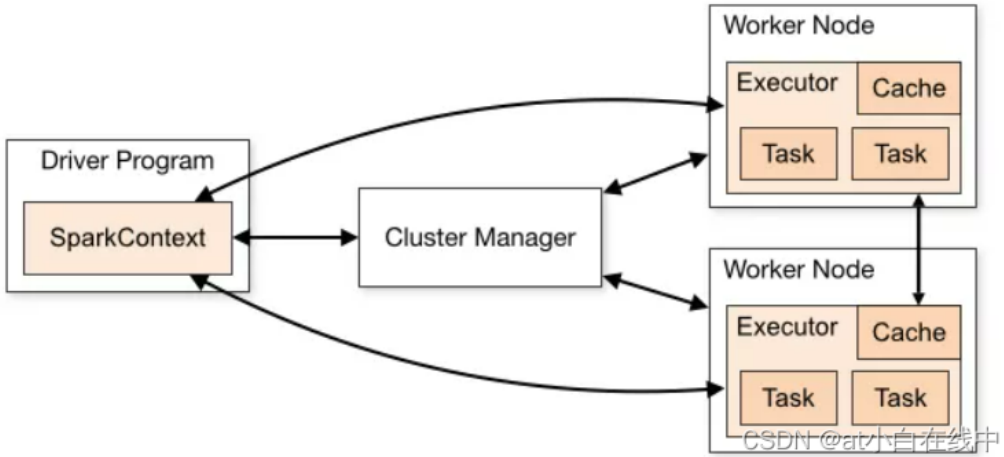


Presto 专为低延迟而设计，遵循经典的 MPP 架构；它使用内存 streaming shuffle 来实现低延迟。Presto 每个集群有一个共享的 coordinator，并带有许多的 workers。Presto 尝试在同一个 Presto worker（共享执行程序）上安排尽可能多的查询，以支持多租户。

这种体系结构提供了非常低的任务延迟调度，并允许对查询的多个 stages 进行并发处理，但其代价是 coordinator 是一个单点故障（SPOF）和瓶颈，查询在整个集群中隔离得很差。

此外，streaming shuffle 不允许太多的容错，从而进一步影响长时间运行的查询的可靠性。

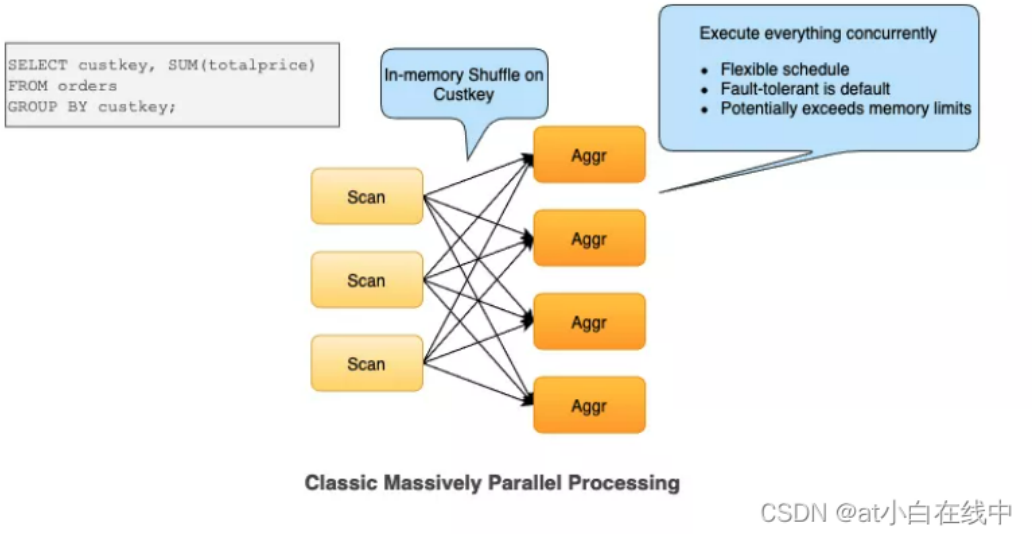
Spark 的架构



另一方面，Apache Spark 从一开始就是为可扩展性而设计的，它实现了 Map-Reduce 架构。Shuffle 在执行阶段之间完全具体化到磁盘，具有抢占或重新启动任何任务的能力。Spark 维护一个独立的 Driver 来协调每个查询，并在按需调度的隔离容器（containers）中运行任务。这些差异提高了可靠性并降低了整体操作开销。

为什么 Presto 不适用于批处理

众所周知，将 MPP 架构的数据库扩展到处理 Internet 规模数据集的批处理是一个极其困难的问题 。为了简化这个问题，让我们看下以下聚合查询。本质上，此查询遍历 TPCH 中的订单表，对 custkey 列进行聚合分组，并对总价求和。Presto 利用内存中的 shuffle 并在每个 worker 上读取数据并为相同的 key 进行聚合。

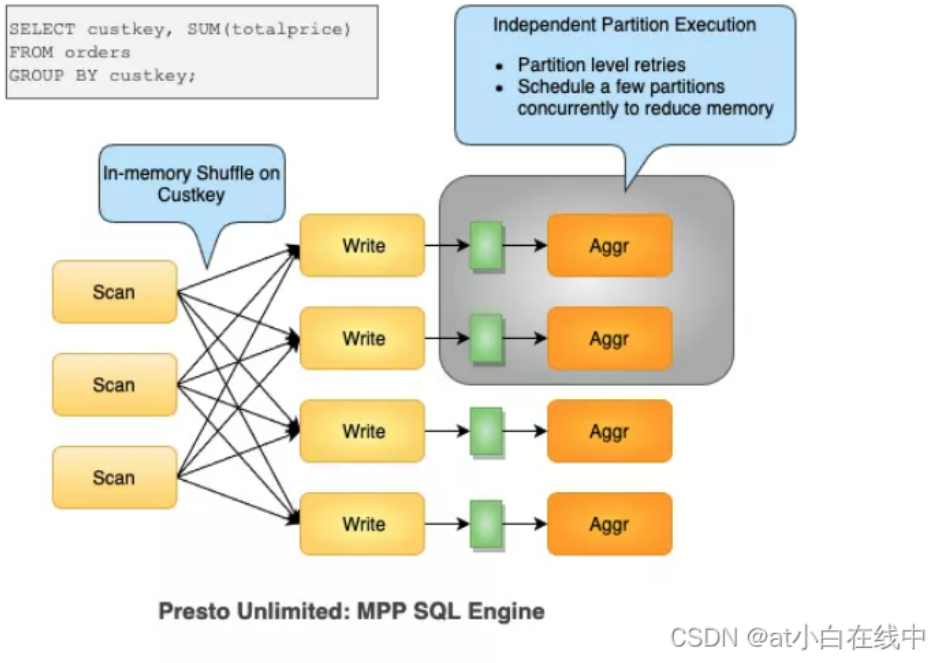


在内存中进行 shuffle 意味着生产者将在内存中缓冲数据，并等待消费者读取数据。我们必须在交换前后同时执行所有任务。在 mapreduce 世界中，所有的 mapper 和 reducer 必须同时运行。这使得 in-memory shuffle 成为一种全有或全无的排除模型（all-or-nothing exclusion model）。

这将导致不灵活的调度和查询大小的扩展变得更加困难，因为所有操作都是并发运行的。在聚合阶段，查询可能会超过内存限制，因为为了跟踪每个组(custkey)，所有内容都必须以散列表的形式保存在内存中。

此外，我们还受到集群大小的限制，即我们可以跨多少个节点对数据进行 hash partition，以避免将所有数据都放入内存中。使用分布式磁盘（Presto-on-Spark、Presto Unlimited)，我们可以进一步对数据进行分区，并且仅受打开文件数量的限制，即使是这个限制，也可以通过 shuffle 服务进行相当大的扩展。

出于这个原因，Presto 很难扩展到非常大和复杂的批处理管道。这样的管道会持续运行数小时，所有这些都是为了 join 和聚合大量数据。这推动了 Presto Unlimited 的开发，它使 Presto 的 MPP 设计适应大型 ETL 工作负载，并大规模改善用户体验。



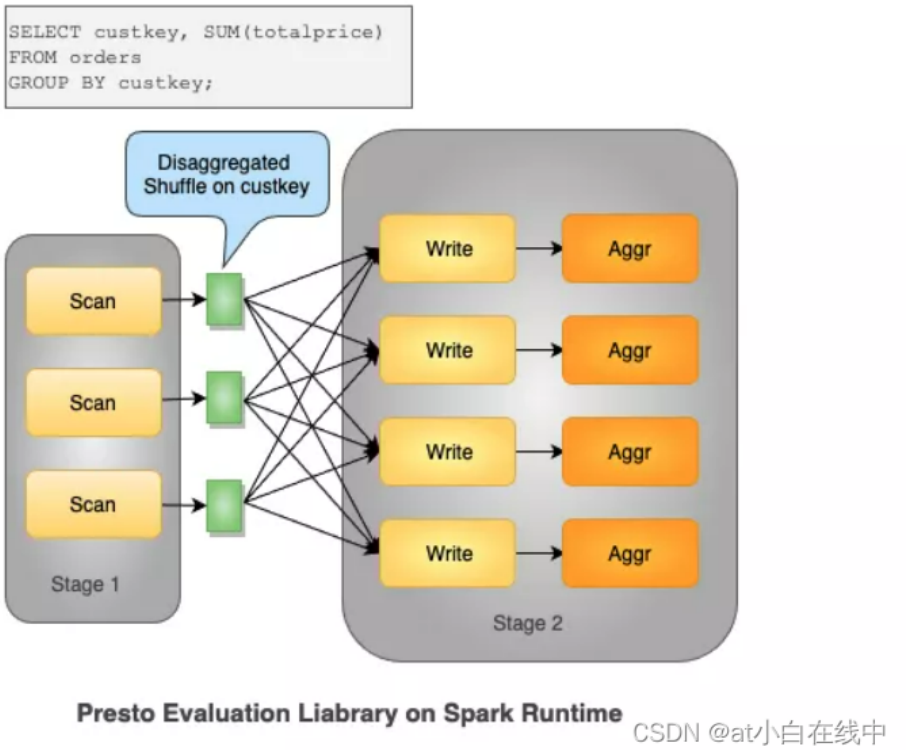
虽然 Presto Unlimited 通过允许在分布式磁盘上对分区进行 shuffle 解决了部分问题，但它并没有完全解决容错问题，也没有改善隔离和资源管理。

Presto on Spark

Presto on Spark 是 Presto 和 Spark 之间的集成，它利用 Presto 的 compiler/evaluation 作为类库，并使用 Spark 的 RDD API 来管理 Presto embedded evaluation 的执行。这类似于 Google 选择将 F1 Query 嵌入其 MapReduce 框架的方式。

高级目标是为 Presto 的 MPP 运行时带来完全分解的 shuffle，我们通过在 shuffle 后立即添加一个 materialization 步骤来实现这一点。物化后的 shuffle 被建模为临时分区表，在 shuffle 后带来更灵活的执行，并允许分区级别重试。

使用 Presto on Spark，我们可以在 mapper 和 reducer 端对上述查询的 custkey 进行完全分解的 shuffle，这意味着所有 mapper 和 reducer 都可以独立调度并且可以独立重试。



Presto Evaluation Liabrary on Spark Runtime

Presto On Spark 在 Intuit 的使用

Superglue 是 Intuit 自主研发的工具，可以帮助用户构建、管理和监控数据管道。Superglue 的建立是为了让分析师和数据科学家的数据民主化（democratize data）。Superglue 最大限度地减少了开发和调试数据管道的时间，并最大限度地增加了构建业务洞察力和 AI/ML 的时间。

Intuit 的许多分析师使用 Presto (AWS Athena) 来探索 Data Lake/S3 中的数据。这些分析师会花几个小时将这些为 Presto 编写的探索 SQL 转换为 Spark SQL，以便将它们作为 Superglue 中的数据管道进行操作/调度。为了最大限度地减少 SQL 方言转换问题和分析师的相关生产力损失，Intuit 团队开始探索各种选项，包括查询翻译、查询虚拟化（query virtualization）和 presto on spark。在快速 POC 之后，Intuit 决定使用 presto on spark 模式，因为它利用 Presto 的编译器/评估作为库（不需要查询转换）和 Spark 的可扩展数据处理能力。

Presto on Spark 现已在 Intuit 投入生产。在三个月内，有数百个关键管道通过 Superglue 在 Presto On Spark 上运行了数千个作业。

Presto on Spark 作为库运行，该库通过 Spark 集群上的 spark-submit 或 Jar Task 方式提交。在临时集群上启动预定的批处理数据管道，以利用资源隔离、管理成本并最大限度地减少运营开销。DDL 语句在 Hive 上执行，DML 语句在 Presto 上执行。这使分析师能够编写与 Hive 兼容的 DDL，并且用户体验保持不变。

该解决方案帮助实现了一个具有无缝端到端体验的高性能和可扩展平台，供分析师探索和处理数据。因此，它提高了分析师的工作效率，并使他们能够高速提供洞察力。

何时在 Presto 中使用 Spark 的执行引擎

Spark 是整个行业运行大规模复杂批处理 ETL 管道的首选工具。Presto on Spark 极大地受益于用 Presto 编写的管道，这些管道可以处理 TB/PB 的数据，因为它利用了 Spark 的大规模处理能力。这里最大的好处是不需要查询转换，在以下的场景中你可以利用 Spark：

•处理更大范围的数据•将 Presto 的资源管理扩展到更大的集群•提高 Presto 作为计算引擎的可靠性和弹性

为什么 Presto on Spark 很重要

我们尝试实现以下目标，以使 Presto on Spark 适应互联网规模的批处理工作负载：

•完全分解的 shuffle（disaggregated shuffles）•Isolated executors•Presto 资源管理、不同调度器、推测执行等。

批量数据和 ad hoc 处理的统一对于创建可扩展的查询体验非常重要，其不需要在不同的 SQL 方言之间重写。我们相信，这只是 Spark 和 Presto 社区进一步融合的第一步，也是实现交互用例和批处理用例之间统一 SQL 体验的重要一步。今天，许多互联网巨头，如 Facebook 等，都已转向 Presto on Spark，我们已经看到许多组织，包括 Intuit，开始使用 Presto on Spark 运行他们复杂的数据管道。

作者：过往记忆大数据

发布于: 2022-01-18 | 阅读数: 111

大数据 spark

评论

Presto设计之初一切都是为了速度，能驱动FB公司内部很大一部分数据分析queries达到interactive的运行速度(sub-minute latency)。这个目标Presto目前做得很好，相比较起来SQL-on-MapReduce类型的系统(e.g., SparkSQL)，Presto本身的执行核心是更偏MPP的，更加高效，优化的很好。
而接下来Presto希望进一步支持ETL类型的queries，ETL型的动辄需要run几个小时读几百T数据的pipelines。Presto的全内存型pull-based inter-stage data exchange显然不适用这个scenario，Presto Unlimited是Presto team内部尝试做的一些基本的scalability方面的尝试，包括materialized exchange到hive tables，grouped execution，operator spilling等等技术。但是有两个design decisions成为了瓶颈（共享JVM container而非每个query独立的JVM container；全局单一coordinator）导致Presto在fault-tolerance和resource management这两点有短板。改进这两点基本等同于在Presto内部内置一个Spark core类似的设计了。恰好FB内部也用Spark，于是索性借用spark core作为资源调度和分布式运行平台，而与此同时保留Presto的运算内核和上层的query parsing & planning 部分即可，从而在user-facing的层面做到semantic的严格统一。
类似的做法Google的F1也差不多，建议读一下那篇论文。理论上不存在一个单一的系统设计fits all scenarios。通常的方法是找到最核心的问题，然后切换对应的组件来适应不同的需求。