深入了解 Presto 原生 C++ 查询引擎-Presto 2.0

在过去三年中，来自Meta、Ahana（现为IBM）、Intel和字节跳动的工程师团队联手打造了一款名为Velox的先进执行引擎，它的设计目标是可以在各种计算引擎之间灵活组合使用。在这个过程中，他们开发出了基于C++的Presto
worker，这是一个全新的查询执行引擎，它基于Velox构建，此前被称为Project
Prestissimo，现在则被命名为Presto 2.0。

我们很高兴地宣布，如今已经有像Meta和IBM这样的大公司开始在实际生产环境中使用这款Presto C++引擎了。你可以在我们最新的博客（https://prestodb.io/blog/2024/05/23/embracing-presto-open-source-for-breakthrough-performance/）中了解到IBM的基准测试结果，以及在Meta在VeloxCon演讲中分享的他们如何使用Presto C++提高批处理效率的案例。

这个项目得到了社区的大力支持和积极参与。参与项目的活跃贡献者人数正在迅速增长，现在已经有来自Uber、ByteDance、Pinterest、Intel、Neuroblade等公司的工程师们定期参与到项目的讨论和论坛活动中。我们的进展迅速，每个月都在不断取得进步。这个项目的最终目标是完全取代Presto Java执行引擎。
这篇博客向你全面介绍了Presto原生引擎，包括我们为何要打造它，以及它的架构设计和底层原理。我们还将分享Meta和IBM在生产环境中如何使用它的更多信息。

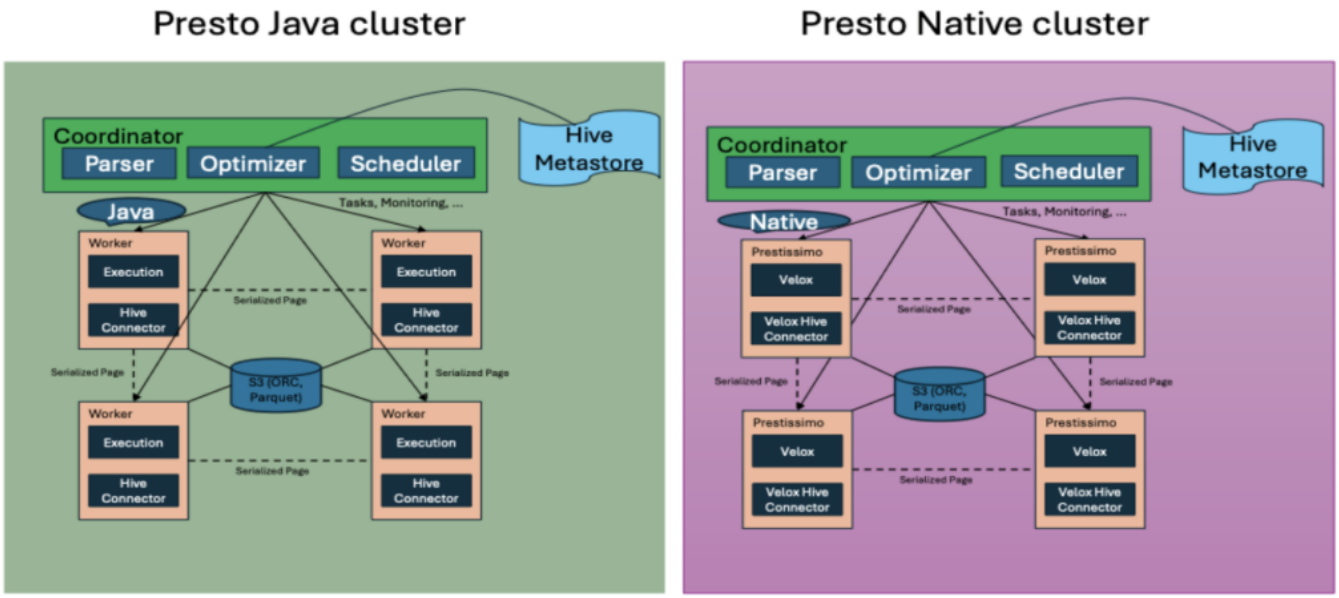
动机

Presto 2.0 完全重写了 Presto 查询执行引擎，也就是我们所说的 Presto worker
进程。我们的目标是，通过从传统的 Java 实现转向现代的 C++ 实现，使其性能和可扩展性提升 3-4 倍。这种迈向向量化执行的改变，与 Databricks Photon、Apache DataFusion
等行业动向相一致。我们对于能把这项技术引入
Presto，让它成为市场上最佳的开放数据湖库引擎，感到非常兴奋。

新的向量化引擎充分利用了查询处理的最新进展，包括矢量化和 SIMD 执行、运行时优化、智能 I/O 预读取和合并等，以提升 CPU 和 I/O
的运行效率。内置的内存管理功能使开发者可以完全控制内存分配，从而消除了 Java 垃圾收集的不确定性和限制，包括复杂的内存仲裁（arbitration）和溢出（spilling）等功能。所有的
CPU、内存、I/O 和网络使用都经过精细调整，以实现理想的性能特性。这一点得到了 IBM
最近在 TPC-DS 基准测试中取得的成绩的证实，我们在上方的博客链接中已经分享了这个信息。

向量化执行框架介绍

Presto C++架构是将Presto Java worker 节点进程替换为原生（C++）worker
节点的一种方式。从下图可以看出，一个Presto集群由一个协调节点、多个 worker 节点，以及一个Hive元数据存储组成。查询任务提交给协调节点，然后在那里进行解析、优化，并安排在各个 worker 节点上分布式执行。你会注意到 worker
节点框中的一些差异，但除此之外，原生Presto C++集群与Java集群几乎完全相同。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：过往记忆大数据

为了实现这种无缝替换，native worker 节点被设计成与Java工作节点具有相同的数据和控制API。然而，实现这个目标并不总是那么顺利。开发过程中的一些增强或限制可能会导致API的变化。例如，交换协议的一些改动，聚合函数的中间类型的变化，或者新的操作符的加入都可能带来期望的性能提升。

这就需要我们对协调节点进行增强，使其能够了解原生执行模式和能力。我们正在协调节点中加入一些锁定功能（lockdown features），以便在使用 worker
节点引擎之间存在差异的功能时抛出错误或警告消息，从而改善Presto C++的用户体验。这些错误可能包括在原生和Java引擎之间缺失的类型，缺失的函数或不支持的计划节点字段。这些功能是 Presto 的原生引擎SPI的先驱。我们也已经开始在 Connector SPI 上进行工作。

向量化引擎的最终目标是完全取代 Java 执行引擎。团队正在全力以赴，以实现与 Java
执行引擎的功能对等。

Velox Library

Presto Native Engine 在很大程度上依赖 Velox 库（https://velox-lib.io/），这是一个来自 Meta 公司的开源项目。

Meta 的数据基础设施团队发起了 Velox 项目，目的是将公司内部使用的各种数据处理引擎的知识整合起来，为这些引擎提供一套可以重复使用的基础组件。这种整合方式简化了工程任务，并在各种引擎中实现了统一的 SQL
体验。这样，无论在哪个引擎中，都可以无缝地使用同一种函数和操作符 API。

现代数据系统越来越倾向于采用可组合的构建方式。Meta 公司强烈支持这种架构，并在 VLDB 2023 上发表了一篇文章，呼吁制定一份关于这种架构的工业宣言 https://research.facebook.com/publications/the-composable-data-management-system-

manifesto/虽然这里的部分观点颇具前瞻性，但 Velox
无疑是朝这个方向迈出的重要一步。它可以与 Substrait 和 Arrow
等其他开放标准进行互操作，构建出一个如本博客所描述的可组合数据系统。

除了在 Presto 中使用，Velox 也被应用于 Project Gluten 的 Spark 运行时环境和 Torch Arrow 中。

使用 Velox 进行查询处理

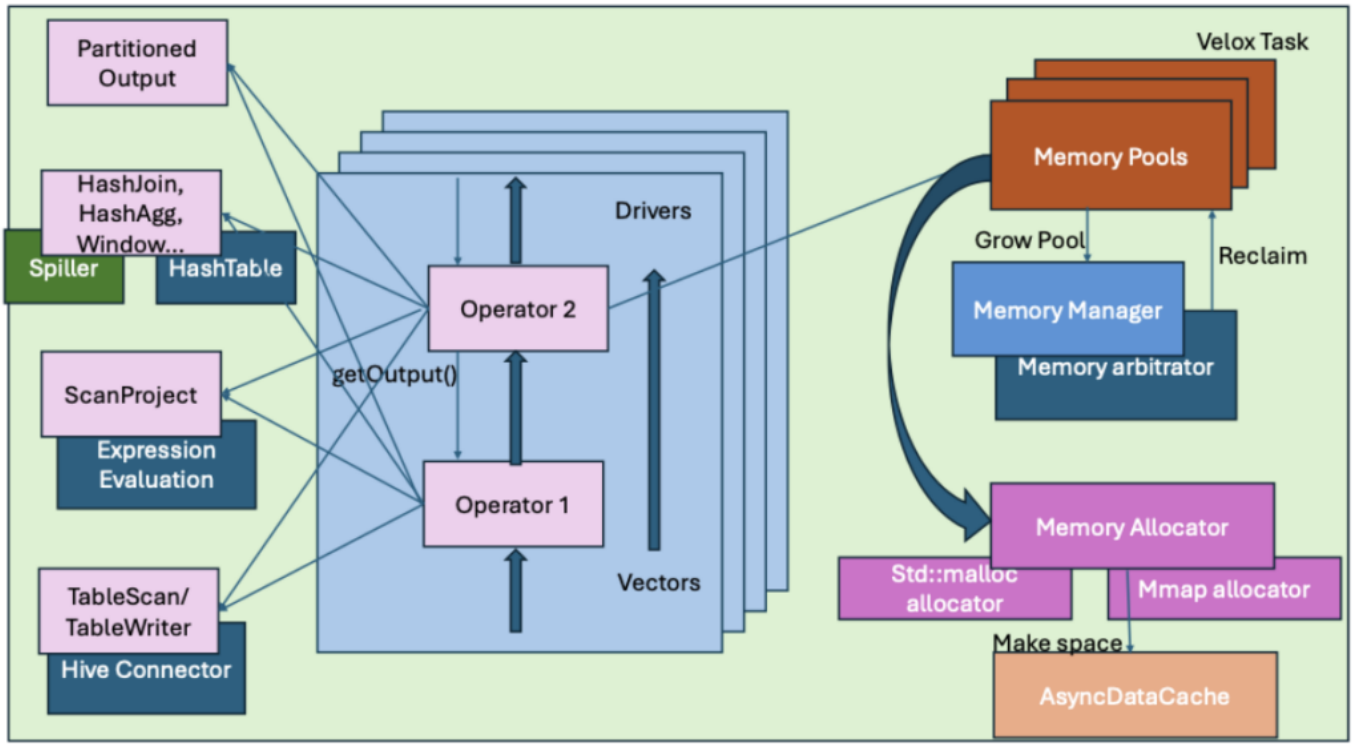
Presto Native 引擎是一个扮演"狭窄接口"的角色，它将绝大部分查询处理任务交给 Velox
处理。如我们之前所述，用户会向协调器提交一个 SQL
查询请求。在协调器那里，这个查询请求会被解析并计划分布式执行。计划的结果是一张由所有 worker 并行执行的计划片段构成的图。每个 Plan fragments 都包含一个查询处理计划节点的有向无环图（DAG），这些节点将会执行管道化的数据（pipelined
data.）。这些管道化的数据源于片段中的表扫描节点，并通过每个计划节点的运算符进行传递。

当 Presto worker 接收到一个计划片段任务时，它会为此任务创建一个相应的 Velox
任务。Presto worker 首先将 Presto 的 plan fragment 转化为 Velox 的 plan node tree。在 Velox 任务开始时，会为 plan nodes 创建多个 operator pipelines 驱动器（Driver）。流水线中的每个 operator 都是系统中的数据处理单元，它们会与 Velox vectors 进行交互。每个 Velox vector 代表数据的一列。这种列向量化（column vectorization）与使用 SIMD
和在紧密循环中进行处理的方式非常适应现代的管线处理器架构。vectors
可以通过各种方式进行编码，如 Flat、Constant 或
Dictionary，以实现在数据处理过程中的最优布局。

每个 operator 都有自己的算法，用于执行诸如表达式求值、过滤、Join、聚合和窗口化等独立处理任务。operator 会使用运行时优化的概念，根据正在处理的 vectors 中的数据特性调整其行为。例如，如果数据中的不同值较少，那么在进行聚合时，可以使用数组索引（array
indexing）作为 group keys，而不是使用昂贵的哈希函数。在进行过滤器的表达式求值时，可以根据选择性对子句进行重新排序。这些策略大大提高了系统的效率和可扩展性。

每个 Velox operator
也有一个与之关联的内存池。这个内存池会记录运算符使用的所有中间结构（如 RowContainer 和 HashTable）所占用的内存。随着查询的进行，Velox 的内存管理器会跟踪不同 operator 的内存池以及系统的内存池。如果 operator 需要更多的内存，它会向内存池发出请求，如果没有可用的空闲内存，这可能会触发内存的仲裁（arbitration）。仲裁可能会导致其他 operator 将他们当前的状态溢出，并为请求内存的 operator 释放内存。这样做可能会减慢系统查询处理的速度，但让查询在有限的内存中进行。可以通过优先级调整仲裁，给用户在他们的工作负载管理中提供一定的控制权。

以下图表展示了在 Presto 查询任务的 Velox 任务中，Velox 组件的构成。



Presto Native 引擎是一个循环，它从 Presto Coordinator
接收带有计划片段的任务，为它们创建一个 Velox 任务，并调用 Velox driver
循环以进行查询执行。

Presto C++ 部署

过去几年，Velox 和 Presto 社区致力于打造一款 Native
的引擎。现在，这款引擎已经可以支持大部分的 Presto SQL
方言。在实际应用或基准测试中，Native 引擎的性能稳定地超过 Presto Java 的 2-3 倍。
接下来，我们将要介绍如何在 Meta 和 IBM 这两大公司中部署 Native 引擎。

Meta 的 Presto 向量化部署

Meta 公司管理着全球最大的数据仓库之一，涵盖了从交互式仪表盘到大规模定时 ETL
任务的各种 SQL 处理任务。Presto 系统支持 Meta 的交互式和短时间运行（~ &gt 20
分钟）的批处理工作。

Meta 的数据基础设施团队开发了 Prestissimo 和
Velox，这两个工具不仅可以加速查询处理，还可以提高执行效率，使我们现有的 Presto
集群能够支持更多的客户使用场景。在这个庞大的数据仓库中部署 Prestissimo
需要多个团队在几年时间内进行深度合作。

尽管 Prestissimo 和 Velox 是顶尖的查询处理引擎，但我们需要进一步利用这个基础设施来支持

我们现有的仓库工作负载。在 Meta 中，Prestissimo 首次被用于一个性能和业务关键的 SQL 使用场景：实验平台。Prestissimo
的优化和部署对实验平台的工作负载产生了巨大的影响，它将硬件使用率降低了 3
倍，即使硬件更少，查询速度也提高了 1.5/2 倍。我们很高兴分享，自从启动以来，Prestissimo 平台运行稳定，几乎没有服务中断，满足了客户的期望。对于一个从零开始开发的全新查询引擎来说，这是我们引以为豪的一个里程碑。

在实验平台取得重大成功之后，Meta 的工程师开始增强
Prestissimo，以适应更多的通用批处理和临时工作负载。虽然实验平台有固定的查询形状，但 Prestissimo 需要适应 Presto 提供的所有功能。我们构建了许多额外的功能，包括表格写入器、内存和溢出管理、功能间隙缩小、安全通信、任务和驱动程序管理 - 几乎 Prestissimo 的所有领域都经过了重大改造，以支持通用工作负载。验证结果和测试全新查询引擎的可靠性至关重要，我们在构建测试基础设施（如模糊测试器、Presto 验证与功能替代以及数据写入器验证）以及广泛的实时工作负载影子测试方面进行了重大投资。我们很高兴分享，这次大规模的改造使得平台变得稳定，墙体时间提高了约 1.5 倍，CPU 时间提高了约 2-3 倍。

除了性能和可靠性，确保 Prestissimo 查询结果的正确性并与 Presto
中的现有行为匹配也很重要；这是最具挑战性的任务之一，考虑到 Meta
工作负载的规模。虽然已知的正确性错误已在 Prestissimo 中修复，但在 Presto
中也发现了多个问题，也正在修复，以使两个平台保持一致。另一个挑战是 Prestissimo
使用的库与 Presto Java（例如 RE2，SimdJson）相比有所不同，这导致了与现有 Presto 平台的行为不同。这对于工作负载迁移提出了独特的挑战，我们正在积极寻找创新的解决方案来解决这些问题。

我们很高兴看到客户正在接受这项新技术及其带来的好处。例如，实验平台也开始将其批处理工作负载从 Spark 迁移到
Prestissimo；大部分已经在生产中。我们正在将仓库的批处理工作负载从旧的 Java 堆栈迁移到 Prestissimo，计划到 2025 年第一季度将我们现有的大部分工作负载投入生产。

IBM 的 Presto 向量化部署

不同于大数据厂商，IBM 是一家专注于平台的公司。在 IBM，我们致力于打造一款名为 watsonx.data 的全面开放数据湖仓库平台 https://www.ibm.com/products/watsonx-data。各大企业使用 watsonx.data 来创建开放且可管控的数据湖仓库。数据以 Parquet 和 Iceberg 等开放格式存储，因此可以由多种引擎和工具进行查询。Watsonx.data
自带了多种数据引擎，如 Presto、Spark 或像 Milvus
这样的向量数据库，可以根据不同的查询负载进行优化。Watsonx.data 支持智能化的目录管理，可以高效地在湖仓库中搜索各类数据。它支持混合云部署，并可以在几分钟内配置为多云或本地环境。

现在，Watsonx.data 还提供了 Presto Native 引擎作为查询引擎，与 Presto Java/Spark
等一同使用。Presto C++ 是处理开放数据湖仓库工作负载的主要查询引擎。IBM 的前 Ahana 团队在 Velox 和 Presto C++ 中开发了多项功能以支持此类应用。除了在 Velox 中增强了 SQL 覆盖范围，团队还开发了一个高性能的 Parquet 和 Iceberg 读取器，支持 S3
存储系统，实现了基于 JWT 和 TLS 的认证，以及与 Watson 知识目录的集成。

该团队一直在通过追求 TPC-DS 基准来持续提升 Native
引擎的性能。各大数据基础设施供应商都使用 TPC-DS 基准作为公正比较他们性能的手段。TPC-DS 通过执行多个高强度和高复杂度的决策支持查询流，包括中间的数据维护任务，来检验平台的性能极限。虽然我们还有很多工作要做，但在 1、10 和 100 TB
的规模因子上的基准测试的早期结果非常令人鼓舞。更多的测试结果可以参考我们的博客文章。

总结

Presto C++ 向量化引擎在 Presto 领域的发展中起到了重要的推动作用。通过发挥向量化执行的优势，并利用强大的Velox库，它为Presto的数据处理开辟了新的可能。我们非常高兴能够将这些成果带给整个行业，看到我们的工作带来的影响和增长，我们总是感到惊奇。

这个项目得到了一支强大的工程师社区的支持。除了Meta和IBM，还有来自Uber、字节跳动、Pinterest、Intel、阿里巴巴和Neuroblade的工程师参与到了项目的开发中。这个项目的特别兴趣小组 https://lists.prestodb.io/g/native-worker-wg 每两周的周四美国太平洋时间上午11点会进行一次会议，讨论项目中的设计和问题。参与公司的大多数资深工程师都会出席这个论坛。所有的问题都会在社区成员中进行讨论并找到解决方案。

欢迎你加入我们的社区。如果你有任何问题或者想要了解更多信息，随时通过我们的社区渠道与我们取得联系。

请保持关注，向量化引擎团队将会带来更多令人兴奋的更新！

本文翻译自：https://prestodb.io/blog/2024/06/24/diving-into-the-presto-native-c-query-engine-presto-2-0/