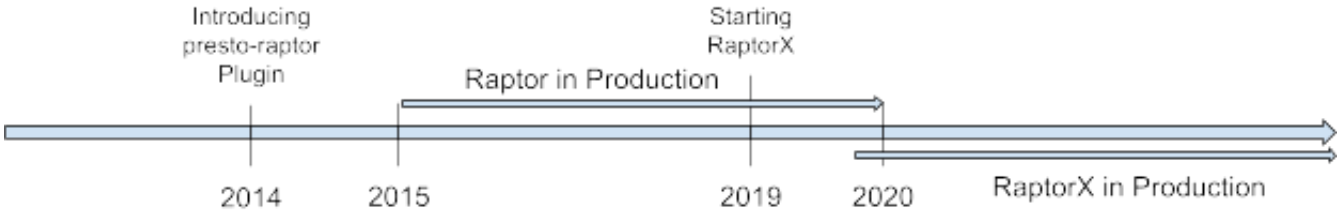
避免 Presto 中的数据孤岛：从 Raptor 到 RaptorX 的旅程

Raptor 是一个 Presto connector (presto-raptor)，用于支持 Meta（以前的 Facebook）中的一些关键的交互式查询工作负载。尽管在 ICDE 2019 年的论文《Presto: SQL on Everything》中提到了这个特性，但对于许多 Presto 用户来说，它仍然有些神秘，因为没有关于这个特性的可用文档。本文将介绍 Raptor 的历史，以及为什么 Meta 最终取代了它，转而支持一种基于本地缓存的新架构，即 RaptorX。



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

Raptor 简介

一般来说，Presto 作为一个查询引擎并不拥有存储空间。相反，开发了 connectors 来查询不同的外部数据源。这个框架非常灵活，但在存储和计算分离的结构中，很难提供低延迟保证。网络和存储延迟难以避免变化。为了解决这个限制，Raptor 被设计成 Presto 的独享存储引擎（shared-nothing storage engine）。

动机——AB 测试框架中的一个初始用例

在 Meta 公司，新产品特性通常要经过 AB 测试，然后才能更广泛地发布。AB 测试框架允许工程师配置实验，向测试组推出新特性，然后监控一些关键指标。该框架为工程师提供了一个 UI 来分析他们的实验统计数据，从而将配置转换为 Presto 查询，查询语句是已知且有限的。查询通常连接多个大型数据集，其中包括用户、设备、测试、事件属性等。这个用例的基本需求是：

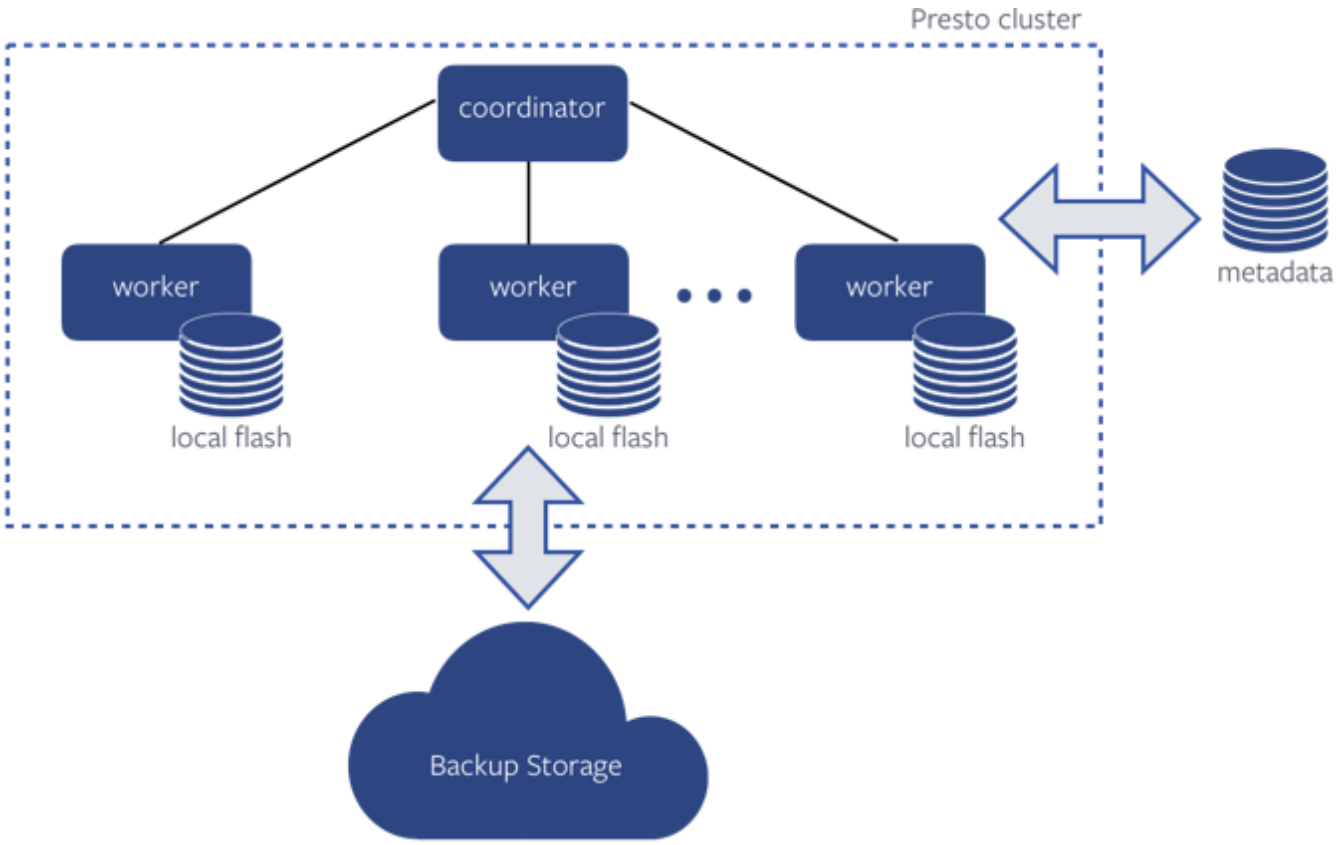
- 准确性：数据必须完整、准确，不能近似；
- 灵活性：用户应该能够任意分割他们的结果；
- 实时性：测试结果应在数小时内得到；
- 交互延迟短：查询需要在几秒钟内返回结果；
- 高可用性：作为产品开发的关键服务，该服务的停机时间应该最小。

Presto 在典型的仓库设置中（比如使用 Hive 连接器直接查询仓库数据）可以轻松满足前两个要求，但不能满足其他要求。当时没有近实时的数据摄入，仓库数据大多是 T+1 摄入，不能满足实时性的要求。Meta 的数据中心已经迁移到一个计算/存储分离的架构，当在高 QPS 下扫描大型表时，不能保证延迟。典型的 Presto 部署会停止整个集群，因此不能满足 HA 需求。

为了支持这个关键用例，我们开始了生产 Raptor 的旅程。

Raptor 架构

下面是带有 Raptor 连接器的 Presto 集群高层次的架构：



Raptor 连接器使用 MySQL 作为存储表和文件元数据的 metastore。表数据存储在每个 worker 节点的本地磁盘上，并定期备份到外部存储系统，以便在 worker 节点崩溃时进行恢复。数据以足够小的批量摄入到 Raptor 集群中，以提供极小的延迟，提供数据的实时性。为提供高可用性，需要创建备集群。

局限

通过对计算/存储不分离，Raptor 集群可以支持低延迟、高吞吐量的查询工作负载。然而，这种架构带来了以下几个问题：

- 集群利用率低：Raptor 集群的大小通常取决于需要存储多少数据。随着表的增长，由于计算/存储偶合在一起，需要更多的 worker，这也带来了将这些机器重新用于其他用途的挑战，即使集群空闲。
- Low tail performance：由于数据很难分配给其他 worker 节点，如果一个 worker 节点宕机或变慢，必然会影响查询性能，难以提供稳定的性能。

- High engineering overhead：Raptor 需要很多存储引擎特有的特性和处理，比如数据摄取/删除、数据压缩、数据备份/恢复、数据安全等。对于直接查询 Meta 数据仓库的普通 Presto 集群，所有这些服务都由专门的团队管理，所有用例都能从中受益。而对于 Raptor 来说，情况就不同了，这导致了工程开销。
- High operational overhead：Raptor 集群的额外存储方面也需要额外的操作工作，不同的集群配置和行为意味着需要设置单独的oncall进程。
- 潜在的安全和隐私漏洞
  ：随着安全和隐私需求的增加，安全与隐私策略的统一实现变得更加重要。使用单独的存储引擎使得执行这些策略非常困难和脆弱。

RaptorX 的启动

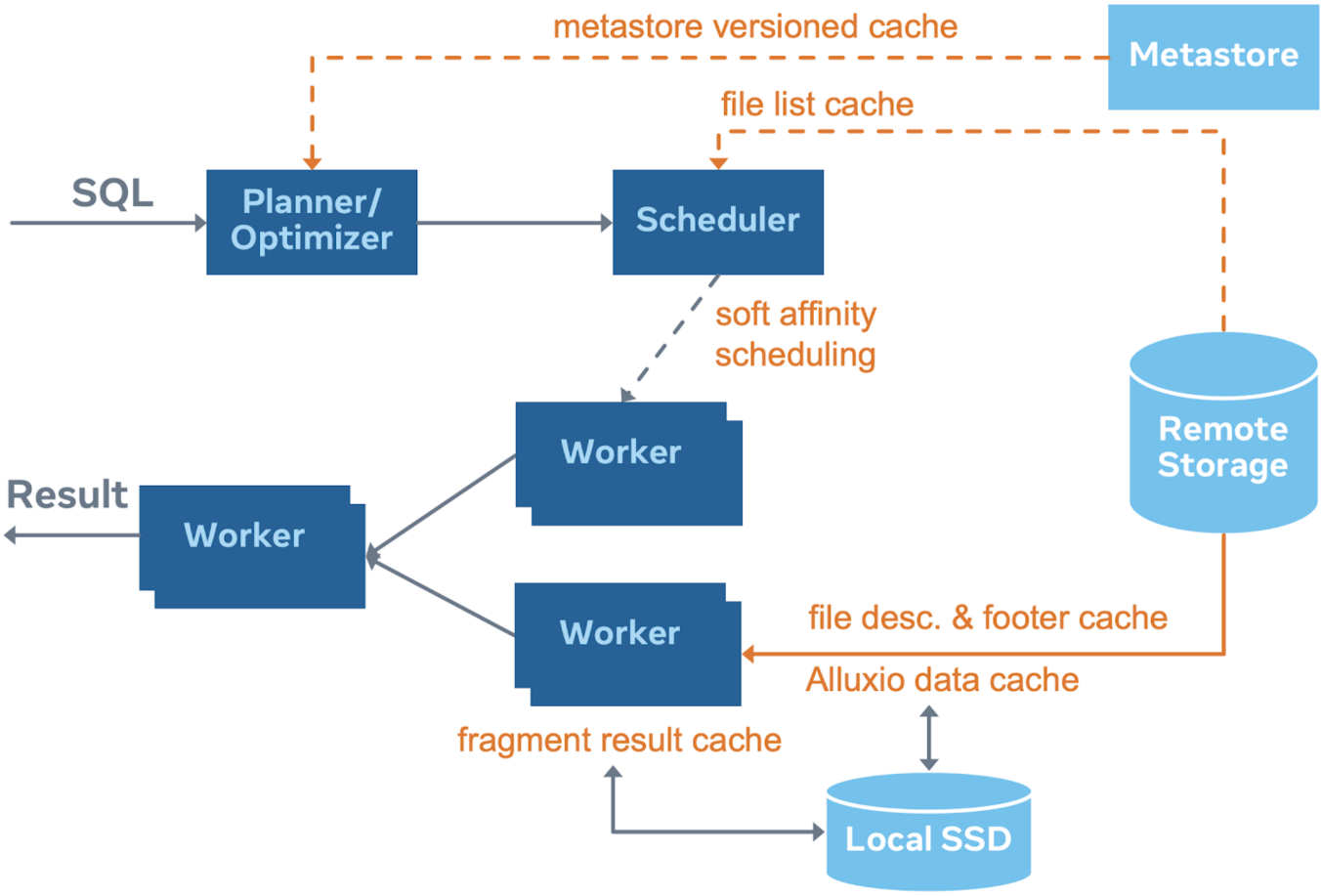
Raptor 有着很多的痛点，Meta 的工程师们在2019年开始重新思考 Raptor 的未来。是否有可能从本地闪存中获得好处，而无需存储和计算绑定在一起？决定的方向是在普通数据仓库之上添加一个新的本地缓存层。这个项目，作为 Presto Raptor 连接器用例的替代品，被命名为 RaptorX。

从技术上讲，RaptorX 项目与 Raptor 无关。我们可以使用相同的闪存驱动器将 Raptor 表作为数据进行缓存，并将热数据保存在计算节点上。使用本地磁盘作为缓存而不是存储引擎的优点是：

- Presto 不再需要管理数据生命周期；
- 单个 worker 故障导致的数据丢失对查询性能的影响较小；
- 缓存作为文件系统层的一个特性，是 presto-hive 连接器的一部分，因此 RaptorX 集群的架构类似于其他 presto 集群，减少了操作开销。

RaptorX 架构

下面是 RaptorX 的架构。



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

Raptor 和 RaptorX 之间的根本区别是如何使用 workers 上的本地 SSD。在 RaptorX 中，Presto Worker 使用 Alluxio 在本地缓存文件数据。很容易理解，不同表列的访问模式可能非常不同，像 ORC 和 Parquet 这样的列式文件格式通常用于数据存储，以增加文件中的数据局域性。通过在列式文件上以较小的页面大小缓存文件片段，只有频繁访问的数据才会被保存在接近计算的地方。Presto coordinator 尝试将处理相同数据的计算调度到相同的 worker
节点，以提高缓存效率。RaptorX
还实现了文件页脚和元数据缓存，以及其他进一步提高性能的智能缓存策略。

更多关于 RaptorX 的信息可以参见 《RaptorX: 将 Presto 性能提升十倍》。

RaptorX 和 Raptor 性能基准测试

社区对 RaptorX 和 Raptor 进行了基准性能测试。基准测试运行在一个有大约 1000 个 Worker 节点和一个 coordinator 的集群上。Rapto r和 RaptorX 使用相同的硬件，整个数据集在 RaptorX 中都客户缓存到本地 SSD 中，因此缓存命中率接近100%。

Architecture	Latency (sec)		
	Avg	P75	P90
raptor	12.7	12.7	19.4
raptorX	8.41	9.90	10.9

从基准测试结果中可以看到，RaptorX 的 P90 延迟几乎是 Raptor 的两倍。RaptorX 中的平均查询延迟和 P90 查询延迟之间的差异要比 Raptor 小得多。这是因为在 Raptor 中，数据被物理地绑定到计算它的 worker 点上，因此一个慢的节点将不可避免地影响查询延迟。在 RaptorX 中，我们在调度时使用 soft affinity 调度。soft affinity 调度将选择两个 worker 节点作为处理 split 的候选节点。如果第一选择 worker 节点运行正常，则将选择该节点，否则将选择辅助 worker 节点。数据可以在多个节点上缓存，调度可以优化整体工作负载的 CPU 从而达到负载均衡。

从 Raptor 迁移到 RaptorX

Meta 中所有以前的 Raptor 用例都迁移到 RaptorX, RaptorX 提供了更好的用户体验，并且易于扩展。

A/B 测试框架

在前一节中，我们提到了 A/B 测试框架的要求是：准确性、灵活性、实时性、交互延迟短和高可用性。因为 RaptorX 是 Hive 原始数据的缓存层，所以 Hive 保证了数据的准确性。它享受所有来自核心Presto 引擎的查询优化，以及 Hive 连接器中的许多特定优化。Benchmark 测试表明，P90 和 P90 的平均查询延迟都优于 Raptor。对于实时性要求，我们能够从 Meta 的近实时仓库数据摄取框架改进中受益，它提高了所有 Hive 数据的实时性。与 Raptor 一样，备用集群保证了高可用性。

在迁移过程中，由于良好的用户体验，测试框架的流量增长了2倍。RaptorX 集群能够以与迁移前的 Raptor 集群相同的容量支持额外的流量。集群的 CPU 容量被充分利用，而无需担心存储限制。
Dashboard
在 Meta 中 Raptor 的另一个典型用例是改进仪表板体验。Presto 用于支持 Meta 中的许多仪表板用例，一些数据工程团队选择将他们的预聚合表接入到专门的 Raptor 集群中以获得更好的性能。通过迁移到 RaptorX，数据工程师可以删除摄取步骤，不再需要担心基表和预聚合表之间的数据一致性，同时还可以在 P50 以上的大多数百分比中享受约 30% 的查询延迟减少。

Raptor 范围之外

由于 RaptorX 在正常的 Hive connector 工作负载下可以很容易地作为一个增强器使用，我们也为 Meta 的交互工作负载启用了 RaptorX。这些是多租户集群，通过 Presto 处理几乎所有非 ETL 查询到 Hive 数据，包括 Tableau、内部仪表板、各种自动生成的 UI 分析查询、各种内部工具生成的工作负载、管道原型（pipeline prototyping）、调试、数据探索等。RaptorX 为这些集群提供了支持，可以为访问相同数据集的查询提供性能提升。

本文翻译自：Avoid Data Silos in Presto in Meta: the journey from Raptor to RaptorX