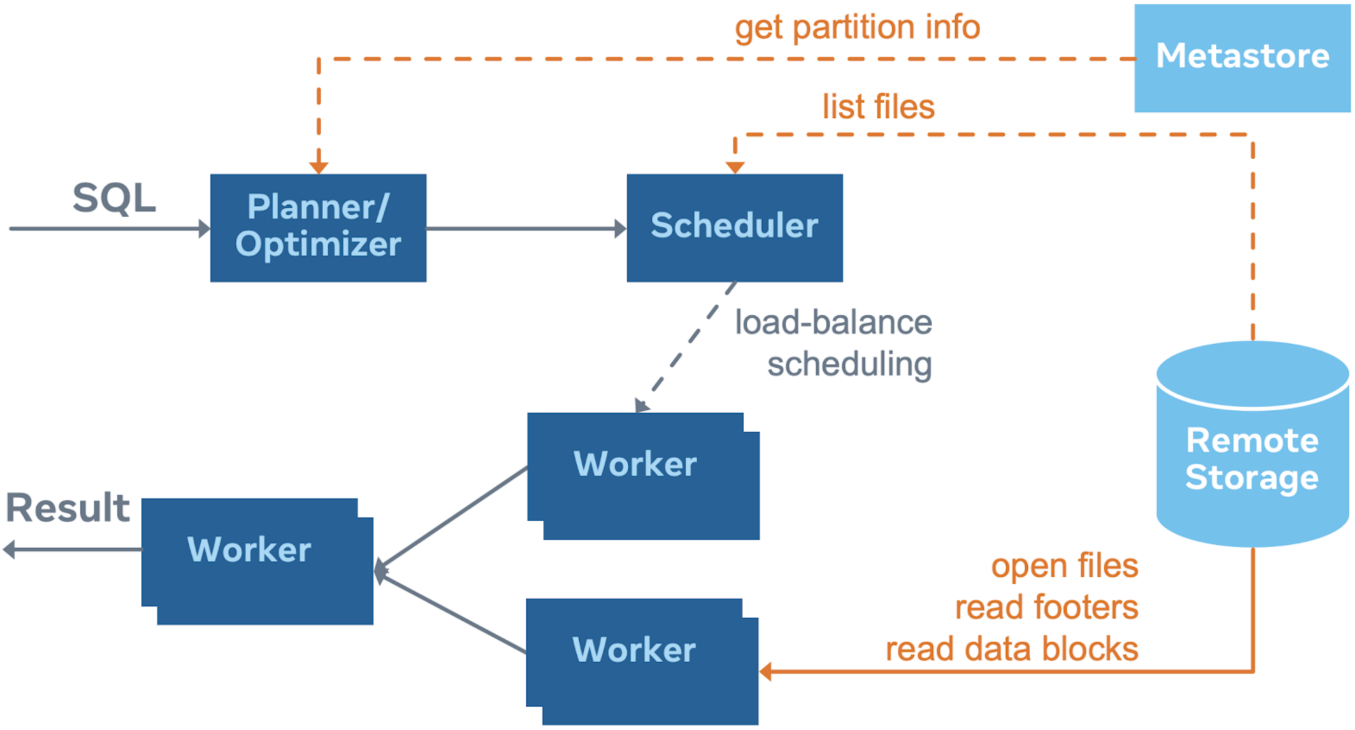
RaptorX: 将 Presto 性能提升十倍

存储计算分离是整个行业的发展趋势，这种架构的存储和计算可以各自独立发展，它帮助云提供商降低成本。Presto 原生就支持这样的架构，数据可以从 Presto 服务器之外的远程存储节点传输过来。

然而，存储计算分解也为查询延迟带来了新的挑战，因为当网络饱和时，通过网络扫描大量数据将受到 IO 限制。 此外，元数据的读取路径也将通过网络来检索数据的位置；元数据 RPC 的几次往返很容易将延迟提高到一秒以上。 下图以橙色显示了 Hive 连接器的 IO 路径，每个路径都可能成为查询性能的瓶颈。



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

RaptorX：构建分层缓存解决方案

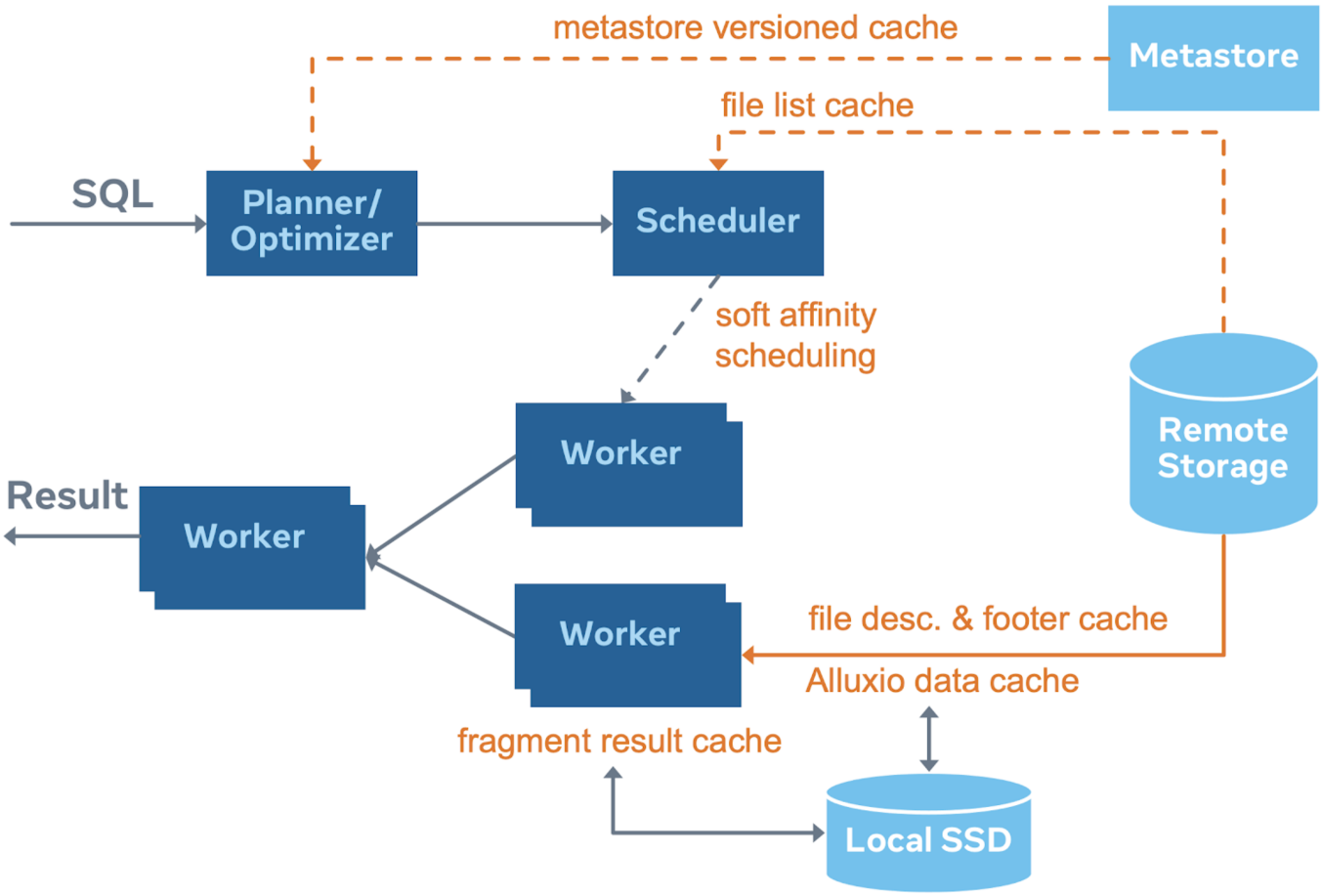
过去，为了解决网络瓶颈问题，Presto 有一个内置的 Raptor 连接器，可以将数据从远程存储加载到本地 SSD，以便快速访问。然而，这个解决方案与共享计算/存储节点没有什么不同，这违背了存储计算分解的思想。其缺点也是显而易见的：要么我们浪费 CPU，因为 Worker 的 SSD 已经满了；要么我们浪费 SSD 容量，如果我们的 CPU 达到瓶颈。为了解决这些问题，我们启动了 RaptorX 项目。

RaptorX 是一个内部项目，旨在将 Presto 的查询性能提高至少10倍。分层缓存是 RaptorX 成功的关键。当存储节点从计算节点分解时，缓存特别有用。RaptorX 的目标不是创建一个新的连接器或产品，而是一个内置的解决方案，以便现有的工作负载无需迁移就可以无缝地受益于它

。我们特别针对现有的 Hive 连接器，因为它被许多工作负载所使用。

下图显示了缓存解决方案的架构。 我们有分层的缓存层，将在本文的其余部分详细介绍：

- Metastore 版本缓存（Metastore versioned cache）：我们在协调器中缓存表/分区信息。鉴于元数据是可变的，如 Iceberg 或 Delta Lake，所以存储的信息是版本化的。 我们仅将版本与 Metastore 同步，并在版本过期时获取更新的元数据。
- 文件列表缓存：缓存远程存储分区目录的文件列表。
- Fragment 结果缓存：在 leaf worker 的本地 SSD 上缓存部分计算结果。需要修剪技术来简化查询计划，因为它们一直在变化。
- 文件句柄和页脚缓存：将打开的文件描述符和 stripe/file 页脚信息缓存到 leaf worker 内存中，这些数据块在读取文件时经常被访问。
- Alluxio data cache：在 leaf worker 的本地 SSD 上缓存带有1MB对齐块的文件段，这个功能是由 Alluxio 的缓存服务构建的。
- 亲和力调度器（Affinity scheduler）：它根据文件路径向特定 worker 发送请求的一种调度程序，以最大限度地提高缓存命中率。



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

Metastore 版本缓存

Presto 协调器缓存表元数据（模式、分区列表和分区信息）以避免对 Hive Metastore 的长时间 getPartitions 调用。 但是，Hive 表元数据是可变的。需要版本控制来确定缓存的元数据是否有效。为此，协调器将版本号附加到每个缓存键值对中。 当读取请求到来时，协调器要求 Hive Metastore 获取分区信息（如果它根本没有缓存）或检查 Hive Metastore 以确认缓存的信息是最新的。虽然无法避免到 Hive Metastore 的往返，但与获取整个分区信息相比，版本匹配相对便宜。

文件列表缓存

Presto 协调器在内存中缓存文件列表，以避免对远程存储的长 listFile 调用。 这只能应用于密封目录（sealed directories）。 对于开放分区（open partitions），Presto 将跳过缓存这些目录以保证数据的新鲜度。 开放分区的一个主要用例是支持近实时摄取和服务的需求。 在这种情况下，摄取引擎（例如，微批处理）将不断将新文件写入开放分区，以便 Presto 可以读取近乎实时的数据。 压缩、元存储更新或用于近实时摄取的复制等详细信息将不在本文的讨论范围之内。

Fragment 结果缓存

运行 leaf stage 的 Presto worker 可以决定将部分计算的结果缓存在本地 SSD 上。这是为了防止对多个查询进行重复计算。最典型的用例是使用扫描（scan）、过滤（filter）、投影（project），and/or 聚合等在 leaf stage 缓存的 plan fragments。

例如，假设用户发送了以下查询，其中 ds 是一个分区列：

SELECT SUM(col) FROM T WHERE ds BETWEEN '2021-01-01' AND '2021-01-03'

对于 2021-01-01、2021-01-02 和 2021-01-03 分区（或者更准确地说，对应的文件），每个分区的部分计算的总和将缓存在 leaf worker 上，形成一个“fragment result”。假设用户发送了另一个查询:

SELECT sum(col) FROM T WHERE ds BETWEEN '2021-01-01' AND '2021-01-05'

然后，leaf worker 将直接从缓存中获取 2021-01-01、2021-01-02 和 2021-01-03 的片段结果（fragment result），只计算 2021-01-04 和 2021-01-05 的部分和。

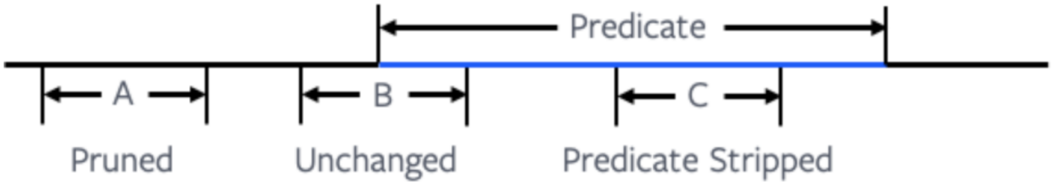
注意，fragment result 是基于 leaf query fragment 的，这可能非常灵活，因为用户可以添加或删除过滤器或投影（projections）。上面的示例表明，我们可以很容易地只使用分区列来当作过滤器。为了避免频繁更改非分区列过滤器造成的缓存丢失，我们引入了基于分区统计的剪枝。考

虑以下查询，其中 time 是非分区列:

```
SELECT SUM(col) FROM T
WHERE ds BETWEEN '2021-01-01' AND '2021-01-05'
AND time > now() - INTERVAL '3' DAY
```

请注意， now() 是一个值一直在变化的函数。如果 leaf worker 根据 now()
的绝对值缓存计划片段，则几乎没有机会获得缓存命中。 但是，如果谓词 time > now() -
INTERVAL '3' DAY 是一个“松散”条件，对于大多数分区都将成立，我们可以在调度期间从计划中
删除谓词。例如，如果今天是 2021-01-04，我们知道对于分区 ds = 2021-01-04，time > now() -
INTERVAL '3' DAY 过滤条件总是为真。

更一般地，考虑下面的图，它包含一个谓词和 3 个分区（A、B、C），箭头两边是 min 和
max。当分区统计域与谓词域（例如分区
A）没有任何重叠时，我们可以直接删除该分区，而不向 worker 发送 splits。如果分区统计信息
完全包含在谓词域（例如分区C）中，那么我们就不需要这个谓词，因为它对于这个特定的分区
总是成立的，并且我们可以在进行计划比较时去除该谓词。对于与谓词有一些重叠的其他分区，
我们必须使用给定的过滤器扫描该分区。



File Descriptor 和 Footer 缓存

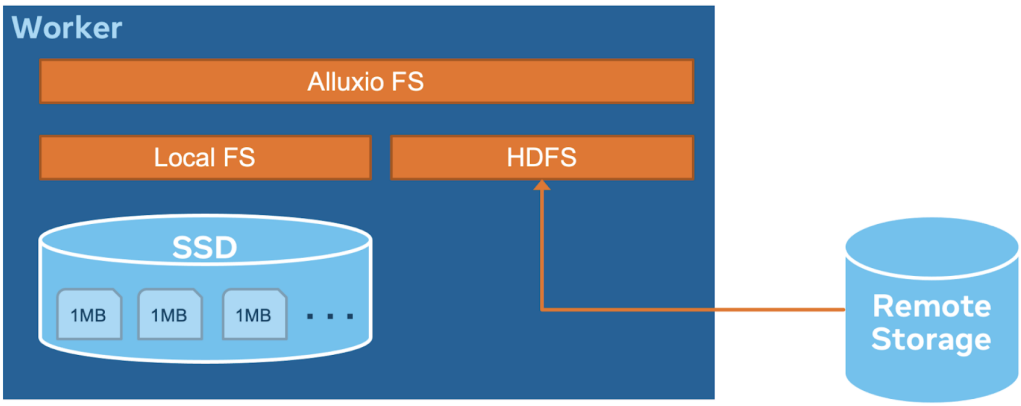
Presto worker 将文件描述符缓存在内存中，以避免对远程存储进行长时间的 openFile
调用。此外，一个 worker 会在内存中缓存普通的列式文件和 stripe
footers。目前支持的文件格式有 ORC、DWRF 和
Parquet。将这类信息缓存到内存中的原因是页脚的高命中率，因为它们是数据本身的索引。

Alluxio Data Cache

Alluxio 数据缓存是弃用 Raptor 连接器的主要特性。Presto worker
在读取远程存储数据时，将其原始形式（压缩并可能加密）缓存到本地 SSD
上。如果将来有一个读请求可以在本地 SSD 上找到的范围，该请求将直接从本地 SSD
返回结果。这个缓存库是由 Alluxio 和 Presto 开源社区共同构建的。

缓存机制是将每个读取切分为 1MB 的块，其中 1MB

是可配置的，以适应不同的存储能力。例如，假设 Presto 发出一个从偏移量 0 开始的长度为 3MB 的读取，那么 Alluxio 缓存会检查 0 - 1MB、1 - 2MB 和 2 - 3MB 的块是否已经在磁盘上，并只读取那些没有缓存的块。清除策略基于 LRU，它从长时间没有被访问的磁盘中移除块。Alluxio 数据缓存向 Hive 连接器提供了一个标准的 Hadoop 文件系统接口，透明地将请求的块存储在一个高性能、高并发和容错的存储引擎中，该引擎旨在为 Facebook 规模的工作负载提供服务。

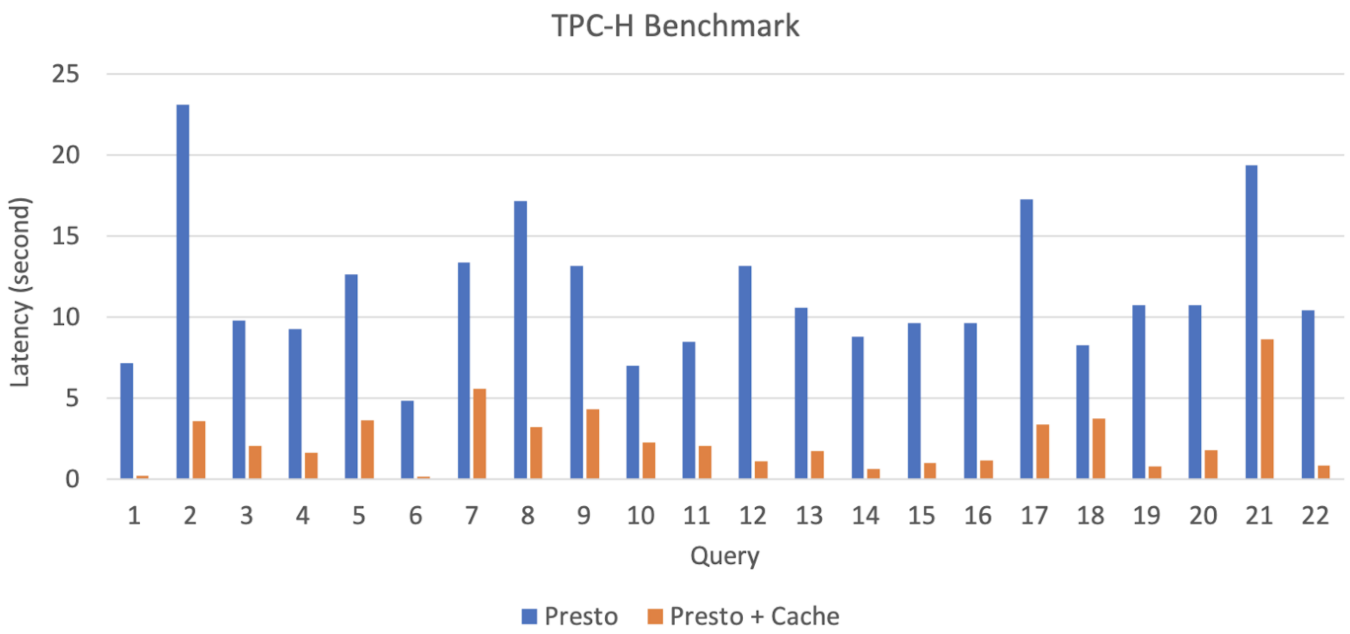


软亲和力调度器（Soft Affinity Scheduling）

为了最大化 workers 的缓存命中率，协调器需要将相同文件的请求调度给相同的 workers。因为文件的一部分很有可能已经缓存到那个特定的 worker 上了。调度策略是“软”的，意思是如果目标 worker 太忙或不可用，调度程序将回退到它的第二选择 worker 进行缓存或在必要时跳过缓存。这种调度策略保证缓存不在关键路径上，但仍然可以提高性能。

性能测试

RaptorX 缓存已经在 Facebook 内部进行了全面部署和测试。为了比较与普通 Presto 的性能，我们在一个114个节点的集群上运行 TPC-H 基准测试。每个 worker 有一个1TB的本地 SSD，每个任务配置4个线程。我们在远程存储中准备了比例系数为100（scale factor=100）的 TPC-H 表。下面的图表展示了原生 Presto 和 Presto 层次缓存的比较。



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

在基准测试中，像 Q1、Q6、Q12 - Q16、Q19 和 Q22
这样扫描量大或聚合量大的查询都有超过10倍的延迟改进。即使像 Q2、Q5、Q10 或 Q17
这样需要大量 JOIN 的查询也有 3X - 5X 的延迟改进。

使用指南

为了完全启用该特性，需要在 worker 上提供本地 ssd
磁盘。为了打开不同的缓存层，请相应地调整以下配置。

调度（ /catalog/hive.properties ）

hive.node-selection-strategy=SOFT_AFFINITY

Metastore versioned cache (/catalog/hive.properties):

hive.partition-versioning-enabled=true
hive.metastore-cache-scope=PARTITION
hive.metastore-cache-ttl=2d
hive.metastore-refresh-interval=3d
hive.metastore-cache-maximum-size=10000000

List files cache (/catalog/hive.properties):

```
hive.file-status-cache-expire-time=24h
hive.file-status-cache-size=100000000
hive.file-status-cache-tables=*
```

Data cache (/catalog/hive.properties):

```
cache.enabled=true
cache.base-directory=file:///mnt/flash/data
cache.type=ALLUXIO
cache.alluxio.max-cache-size=1600GB
```

Fragment result cache (/config.properties 和 /catalog/hive.properties):

```
fragment-result-cache.enabled=true
fragment-result-cache.max-cached-entries=1000000
fragment-result-cache.base-directory=file:///mnt/flash/fragment
fragment-result-cache.cache-ttl=24h
hive.partition-statistics-based-optimization-enabled=true
```

File and stripe footer cache (/catalog/hive.properties):
对于 ORC 或 DWRF 格式的文件

```
hive.orc.file-tail-cache-enabled=true
hive.orc.file-tail-cache-size=100MB
hive.orc.file-tail-cache-ttl-since-last-access=6h
hive.orc.stripe-metadata-cache-enabled=true
hive.orc.stripe-footer-cache-size=100MB
hive.orc.stripe-footer-cache-ttl-since-last-access=6h
hive.orc.stripe-stream-cache-size=300MB
hive.orc.stripe-stream-cache-ttl-since-last-access=6h
```

对于 Parquet 文件

```
hive.parquet.metadata-cache-enabled=true
hive.parquet.metadata-cache-size=100MB
hive.parquet.metadata-cache-ttl-since-last-access=6h
```

本文翻译自RaptorX: Building a 10X Faster Presto