原文戳这里：<http://blog.cloudera.com/blog/2016/04/how-to-use-impala-and-kudu-together-for-analytic-workloads/>

**操作方法（How-to）：针对分析性工作负荷，同时使用Impala和Kudu**

2016年04月20日/作者：David Alves

**同时使用Apache Kudu（正在孵化中）和Apache Impala （正在孵化中）可以带来显著的性能优势。**

[Apache Kudu](http://getkudu.io/)（正在孵化中）是与Apache Hadoop生态系统集成在一起的一系列存储引擎的最新成员。Kudu的目标是为客户提供更高的扫描性能，更有针对性的分析工作负荷，以及允许用户可以同时插入、更新和删除记录。由于具有这些特性，Kudu成为HDFS和/或Apache HBase现有组合的一个可行的替代解决方案，可以通过不太复杂的ETL管道来实现类似的结果，并且具有相当的性能和较轻的维护负担。

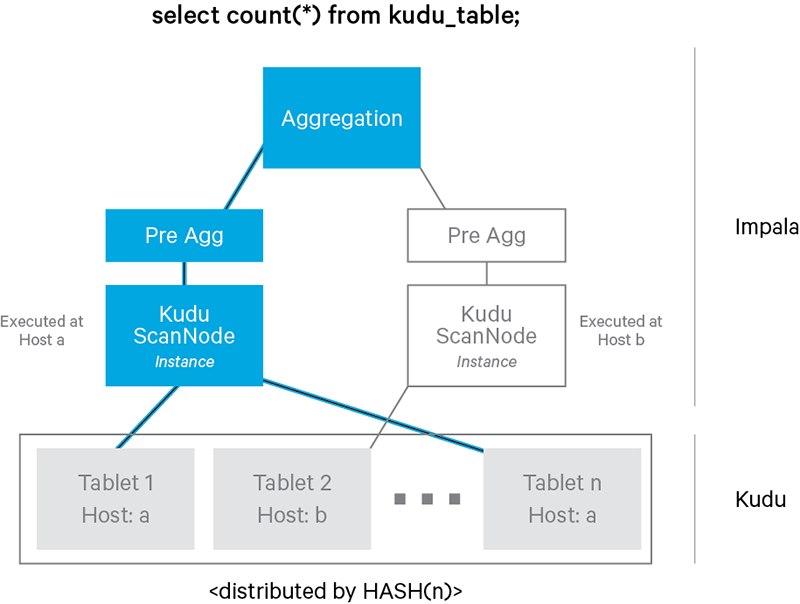
在这篇博客中，我们想介绍[Apache Impala](http://impala.io/)（正在孵化中）是如何与Kudu集成在一起来实现Hadoop的分析SQL查询，以及其是如何充分利用Kudu的独特属性的。

**从Kudu中扫描数据**

当从Kudu中扫描数据时，与扫描HDFS中存储的数据相比，Impala的操作方式无异。但是，与从HDFS中扫描数据并送入不同的算子（如连接或聚合）不同，Impala直接从Kudu中扫描数据。当将Kudu部署到一个集群中，Impala假定遵循与HDFS相同的局部性原理，这意味着通常希望一个Kudu子表服务器与一个Impala守护进程处于相同的服务器，以减少网络传输。

在执行SQL查询之前，Impala的查询前端将对该查询进行分析和规划。在查询规划期间，Impala将从Kudu中获取表元数据，其中包含与每一个子表相关的物理位置和关键字范围的信息。扫描范围定义了Impala的扫描节点的工作单元，并且目前每一个子表将与某一个确定的扫描范围相关。

在查询计划中，现已推出一种新型的扫描节点（Kudu*扫描节点*），其与HDFS和HBase扫描节点相类似。在该Kudu扫描节点内部，Impala使用公共客户端API与Kudu进行通讯。值得注意的是，Impala不会使用Kudu的原始表格格式；相反，其实例化了从客户端的扫描，然后由Kudu守护进程执行扫描。Impala将启动一个线程来执行每一个所分配的扫描范围，目前与Kudu子表存在一对一的映射关系。如果存在多个子表将本地数据扫描到单个节点中，Impala将启动多个线程，同时并行进行扫描。



与Apache Parquet类似，Kudu以列操作的方式将数据存储在磁盘上，并且可以视情况进行编码和压缩，从而生成非常有效的扫描投影，构成表格中总列数的子集。然而简单地扫描并返回所有的行，以在Impala中执行过滤可能无法得到所需的性能，与此相反，Impala充分利用Kudu关联谓语与由Kudu所执行的扫描的能力。如果该谓语涉及的是某个表的主关键字，则可能完全跳过整个扫描范围；如果该谓语涉及的是一个非关键字列，则其将在Kudu侧进行评估，并且仅返回满足谓语条件的行。（目前，对于涉及一个表的主关键字且该主关键字将导致跳过整个扫描范围[Kudu子表]的谓语，Impala仍然将发起扫描；然而，它们将在Kudu侧被转换成停止操作指令（no-ops），并且扫描将立即返回。今后，对于这种情况，将根本不可能在Impala侧创建扫描。）

由于Kudu尚且还不支持所有的列类型或谓语表达式，在查询规划过程中，需要额外加一个步骤来提取所有那些可以通过Kudu执行的谓语。Kudu目前支持以下类型<=、=和=>的谓语评估：整数（Integer）、字符串（String）、浮点（Float）和双精度（Double）列。此外，在整数类型列的情况下，Impala将执行常数转换，以允许叠加< and >类型谓语。

尽可能叠加更多的谓语将显著提高扫描性能。如果有任何范围谓语涉及到关键字列，Kudu将跳过那些确信不能满足谓语要求的所有数据，同时由于得益于Kudu的主关键字索引，可以有效地完成这一功能。在找到满足主关键字谓语要求的数据后，Kudu将会具体化那些也被分配了谓语的非关键字列，并选择满足所有谓语要求的行，然后Kudu才将读取剩余列。相比之下，通过Impala和Parquet存储，所有列将被具体化为适用于非关键字列的谓语。

**插入/更新/删除**

Impala主要侧重于批量加载数据，以及针对这些数据进行快速分析。尽管通过Impala在一般情况下也可以完成单独的插入，但是这样的用法是强烈不被推荐的，因为每一个单独的插入最终会被存储在一个单独的文件中。由于使用Kudu备份的表没有这个限制，因此同时使用Impala和Kudu备份的表相对于经过优化的分析数据仓库而言，表现得更像一个传统的数据库系统。然而，如果Impala希望最大限度地利用这些额外的功能，就有必要在Impala中扩展SQL语言对于UPDATE（更新）和单行DELETE（删除）操作的支持。

针对Kudu表的插入操作通常是通过传入从另一个表的扫描结果或通过单独的插入，因此他们不会拥有相对于源表的先决条件。对于UPDATE（更新）和DELETE（删除）操作而言，情况略有不同。针对Kudu的UPDATEs（更新）和DELETEs（删除）存在一个以关键字为中心的API，可以修改一条记录，至少除了被修改的列之外整个主关键字都需被指定。在一次操作过程中，主关键字本身并不能进行更新，只有同时经过一次DELETE（删除）和一次UPDATE（更新）操作方可更新主关键字。但是，由于这两个操作是独立地执行的，因此无法一致性地保证这两个操作都是成功完成的。在Impala中，DELETE（删除）和UPDATE（更新）操作是以非常相似的方式实现的，其首先发出一个扫描至Kudu，来检索匹配的行并写回修改。在DELETE（删除）操作的情况下，只有主关键字被发送回Kudu，以确定将被删除的记录。

虽然Kudu提供了单记录操作的事务属性，但是执行复杂的将涉及更多行的DELETE（删除）和UPDATE（更新）操作，评估复杂的非主关键字谓语以及与其他表进行潜在的连接将不会在事务环境中执行。为了避免潜在的并发性UPDATEs（更新）或DELETEs（删除），我们增加了一个额外的IGNORE（忽略）关键字：INSERT IGNORE（忽略插入），UPDATE IGNORE（忽略更新）和DELETE IGNORE（忽略删除）是专门针对忽略某些问题而设计的，例如一条记录已经存在相同的主关键字（INSERT IGNORE（忽略插入）），或该记录已经不再存在（UPDATE IGNORE（忽略更新）和DELETE IGNORE（忽略删除））。

**性能注意事项**

当分析管道已经设置好且性能需要优化时，请注意以下事项。首先，虽然Kudu提供了高效的单行插入和更新功能，但是Impala未针对这些类型的查询进行优化。如果您确认您的工作负荷会受到这样非常小型操作很大的影响，可能最好还是直接使用Kudu API而不是借助Impala。此外，您应该总是想方设法批量完成尽可能多的插入，以达到最佳性能。对于此等插入，这种批处理方式可通过使用VALUES子句（[values clause](http://www.cloudera.com/content/www/en-us/documentation/archive/impala/2-x/2-0-x/topics/impala_insert.html)）或INSERT INTO表的SELECT符号来一次性地插入多行予以实现。

当使用INSERT INTO表的SELECT符号在Kudu中插入数据时，有可能通过在Impala中增大批处理大小为较高数值以提高吞吐量。需要注意的是，较高的批处理大小对Impala的内存消耗有直接影响。此外，Kudu可在单个RPC发送的数据量存在上限，大约为8MB。如果批处理大小调整得太大，因为上述限制属性写入操作将出现错误。

Impala使用扫描范围来执行特定主机上读取操作的线程调度，目前单一子表是最小的工作单元，所以子表数量太少将会导致不够好的性能。另一方面，子表数量太多且平均每个字表的行数太少将导致较高的协调和网络开销，因此可能会降低查询的速度。当同时使用Impala与Kudu时，我们发现一条一般的经验法则：让每一台主机拥有尽可能多的子表，以并行扫描。例如，假设您拥有10台服务器，每台服务器包含16核，每台服务器负责至少16个子表，总计160个子表。（请注意，如果每个子表拥有足够的数据的话，则这个简单的经验法则才有效。）

**未来工作（Future Work）**

随着Impala-Kudu一体化方案最近被合并到Impala的主要产品线中，Impala团队目前迫切关注的是其稳定性、可扩展性和高性能。特别是，我们一直致力于解决包含非常大的数据集的查询以及因大型合并造成的一些遗留问题。

Kudu团队一直努力地致力于分区修剪性能改进，在其准备就绪后就将尽快被整合进Impala集成解决方案中。在运行那些可能会影响一个小数目范围的查询时，通过显着地减少需进行扫描的分区数量，这些性能改进应该有所帮助。再来，我们打算实施Apache Arrow内存中数据格式以及共享Kudu与Impala之间的内存，预计将有助于我们提升性能和资源使用率。

针对上述这两个项目，我们都非常欢迎您在这些方面（以及其他所有方面）做出贡献；所以请伸出您的代码之手吧！

David Alves是Cloudera公司的一名软件工程师，并且是Kudu的一名贡献者。