外文翻译：

# Impala：一个Hadoop的现代开源Sql引擎

## 摘要：

Cloudera公司的Impala是一个为Hadoop数据处理环境所构建的现代开源MPP SQL引擎。不同于Apache的Hive批处理架构，Impala为在Hadoop上的以BI/分析读操作为主要操作的查询提供了低延迟和高并发。这篇Paper从用户的观点来陈述Impala，并提供了对Impala构成，主要组成部件的概览。通过与在Hadoop上的主流S QL系统的比较，这篇Paper展示了Impala的优秀性能。

## 介绍

Impala是一个开源的，充分整合，代表了先进水平的Mpp Sql查询引擎。它专门被设计来平衡Hadoop的灵活性和可扩展性。Impala的设计目标有三，首先为了实现熟悉的SQL支持。其次为了提供在传统分析型数据库上的多用户性能的基础上实现Apache Hadoop的可扩展性和灵活性。第三是为了支持Cloudera Enterprise上的工业级别的安全性和管理延生。

## 结构

Impala是一个运行在Hadoop簇上成千上百台主机上的高度并行的查询引擎。不同于传统的关系型数据管理系统那样查询处理和存储引擎都是一个紧密协同工作的系统的部件，Impala和存储引擎分离出来。在下图中，表示Impala的高层次架构。

一个完整地Impala系统由三个服务所组成的。首先Impala的守护进行也被称为Impalad服务负责从客户端进程中接收查询请求，并将查询任务分布到簇中。除次之外，Impalad还能够为其它Impalad进程执行单一的查询片段。当一个Impalad作为领导者执行查询操作时，它就被称为那个查询的协调者。然而所有的Impalad进程都是相等地位的，它们能够扮演所有的角色。Impala的这一特点有利于容错和负载均衡。

Impala的守护进程被部署在所有的主机上，而这些主机也同时运行了数据节点进程。Impala的这个特点允许它能够利用数据本地化，即能够读取文件系统的数据而不用使用网络。

Statestored进程是Impala的元数据发布-订购服务，它能够将簇内的元数据发送到所有的Impala进程。在整个Impala系统中一般只会有一个单独的Statestored实例。

最后，Catalogd进程，主要职能是Impala Catalog的持续化存储库和元数据访问路由。通过Catalogd进程，Impalad在执行DDL命令的同时能够将其在外部catalog中进行映射，例如Hive的Matastore。对于提供catalog的改变通过statestore进行广播。

所有这些Impala服务，包括几个设置选项，例如资源池的大小，系统内存等，都被Cloudera Manager控件所管理。Cloudera Manager是一个簇管理应用。它不仅能够管理Impala，并能够管理其他很多的Hadoop部署服务。

#### 状态分布

在很多主机上运行的MPP数据库设计中一个主要的挑战是怎样能够做到簇宽元数据的协调和同步。Impala的均衡节点结构要求所有节点必须能够接受和执行查询操作。因此，所有节点必须拥有实时的系统catalog和最近的Impala簇成员信息以便于查询操作能够被正确地安排。

我们也许能够通过部署一个分布式的簇管理服务和所有簇范围的元数据的标准版本号。Impala的守护进程能够以懒策略来查询这部分数据，这保证了所有的查询都返回了实时的响应。然而，在Impala设计中一个基本原则是避免同步RPC。不管这些小号，我们发现查询延迟通常被建立TCP连接的时间和加载一些远程服务的时间锁拖慢。相应地，我们设计了Impala来推动部分的更新，并设计了一个简单地发布-订阅服务，被叫做Statestore的来传播元数据变化到所有订阅者中。

Statestore保留了一系列的Topics。Topics指的是（键，值，版本）对的数组，这些对也被叫做入口，其中的键和值是字符类型的数组，而版本是64位的整数。Topic被应用定义，所以Statestore并不了解topic入口内容的意义。Topics在Statestore的生命周期中是持续的，但是在服务重启后即丢失了。希望接收到更新的进程被称作订阅者。它们在一开始就向Statestore注册并提供Topics的列表。Statestore进程通过向注册者针对每一个被注册的Topic发送初始化的Topic更新，这个初始化的Topic包含了所有在那个Topic中包含的入口。

在注册之后，Statestore向订阅者发送两种类型的消息。第一种消息是Topic的更新，并且包含了在上一次成功向订阅者发送Topic更新后所有的变化。

每一个订阅者都维护了一个最近版本号，能够允许Statestore能够只发送更新之间的差值。响应与一个topic的更新，每一个订阅者发送变化列表。这些改变被确保在下次更新被收到之前已经被应用。

第二种类型的Statestore消息是Keepalive，即存活消息。Statestore使用Keepalive消息来维持与每一个Subscriber之间的联系，否则订阅者将会认为订阅超时并试图重新订阅。之前的Statestore版本使用Topic更新消息来达到Keepalive的目的，但是由于topic更新数量的增长，定时地向每一台订阅者发送更新变得困难，导致了订阅者失败检测进程错误地响应。

如果Statestore检测到了一个失败的发布者（例如，重复发送Keepalive信号失败），这将会终止发送更新。一些Topic入口也许会被标记为短暂的，表示如果他们拥有的订阅者发生错误，他们也就会被移除。

Statestore提供非常弱的语义：Subscribers也许会被以不同的速率更新（虽然Statestore进程总是试图将更新公平地分布到各个节点），因此可能会产生多个不同的topic内容的版本。然而，Impala仅仅使用topic的元数据在本地做出决策，而不是通过簇内节点的协同合作。例如，查询计划根据catalog的元数据Topic在单一节点上得到执行。一旦一个完整的计划被计算，所有那个计划执行所需要的信息被直接分布到各个执行节点。一个执行节点不需要获知元数据主题的相同版本。

虽然在整个Impala部署中只有一个Statestore进程，我们发现它能够很好地适应于中等规模的部署。通过相同的配置，也能够很好地服务于大型的部署。Statestore不将任何元数据存到磁盘，所有当前的元数据被各个活动Subscribers推送到Statestore，例如载入信息。因此，如果一个Statestore被重启，它的状态也能够通过初始化的Subscriber注册过程被恢复。或者如果一台主机上，Statestore进程运行失败了，一个新的Statesotre进程能够在任何节点被重启，并且一个Subscribers也可能订阅失败。在Impala中没有内嵌的失败克服机制，取而代之，部署的时候使用了一个重启DNS入口来强迫Subscriber去自动移动到新的Statestore进程实例。

#### Catalog 服务

Impala的catalog服务通过Statestore的广播机制为Impala的守护进程提供元数据。另外，它还为Impala守护进程执行DDL操作。Catalog服务从第三方元数据存储中获得信息，例如Hive的元数据获得HDFS的名字节点，并且将那些信息转换到Impala兼容的catalog结构。这个结构允许Impala对于它所依赖的元数据仓库引擎是关系不可知的，而这一特点允许我们将快速地将新的元数据仓库添加到Impala关系中，例如HBase支持。所有系统Catalog的变化通过Statestore分发。

Catalog服务也允许我们使用Impala定义的信息来反对系统的Catalog。例如，我们仅仅使用Catalog服务来注册用户定义功能信息（不将这些发布到Hive的元数据中）因为它们是专门用于Impala的。

因为Catalogs通常很大，并且访问的表很少是均匀的，所以Catalog服务仅仅为每张表载入了一个框架性的入口。更加详细的信息能够在后台慢慢从第三方数据仓库中被载入。如果一张表在它完全载入之前被请求到，一个Impala的进程将会检测并且将向catalog服务的请求置于优先地位。这一点将会阻塞请求直到整张表被载入。

## 前端

Impala的前端负责编译SQL文本，将其转换为能够被后端执行的查询计划。前端通过Java语言写成，包含了一个SQL语句分析器和基于成本的查询优化器，而这些都是从头开始写的。在基本的SQL特征的基础上，包括了select，project，join，group by，order by， limit， Impala支持Inline views、不相关和相关子查询（被作为joins操作重写），所有的外连接的变种包括明确左/右半和反连接，和窗口分析功能。

查询编译进程遵守传统的工作分工：查询分析，语义分析，和查询计划/优化。我们将会在后面的篇幅中关注在查询编译中最具挑战性的部分。对于查询计划的执行中包括了两个阶段：单点查询和查询并行和分片。

在第一阶段，查询树被转换为一个不可执行的单点计划树，包含了下面的查询节点：HDFS/HBase扫描，哈希连接，交叉连接，组合，哈希聚合，排序，和分析估计。这一步骤负责为在低层次可能的计划节点负责断言。成本分析被寄予table/分块基数加上每一列的不同值的总数。

计划的第二个阶段将单节点计划作为输入并输出一个分布式的执行计划。总体的目标是数据迁移最小化，将扫描区域最大化。在HDFS中，远程读取要比本地读取慢得多。计划通过在必要的时候在计划节点之间增加交换节点来分布。在必要地时候通过增加额外的非交换计划节点来最小化网络数据迁移。在这第二个阶段，我们为每一个连接节点（在这一点中连接顺序是固定的）决定连接策略。所支持的连接策略是广播和分片。Impala选择能够最小化网络数据交换的策略，并利用存在的数据分片作为连接输入。

## 后端

Impala的后端从前端接收查询片段并负责它们的执行。它被设计于能够利用现代硬件。后端使用C++写成并在执行阶段使用代码生成来产生高效率的代码路径和最小化的存储开销。

Impala促使数十年在并行化数据库上的研究发生改变。执行模型是传统的Volcano-style加上交换执行器。处理以批处理的方式被执行，每个GetNext的调用对行执行了批处理。得益于stop-and-go异常执行器，执行能够被流水化，这使存储最小化来存储立即结果。

那些可能会消耗大量数据的执行器被设计成能够将它们的工作的一部分数据保存到磁盘在需要的情况下。能够被泄露的操作包括了哈希连接，基于哈希的断言，排序和分析功能估值。

Impala采用了一种分片的方法来执行哈希连接和断言操作。换言之，每一个元组的哈希值的一些位决定了目标，而剩下的位则决定了哈希表的探查。当所有的哈希表能够适应于内存时，分片步骤的开销是最小的，能够比非泄露，非基于分片实现降低10%的开销。当有内存压力时，一个受害者分片可能会被移到磁盘中，通过这种方法来为其它分片腾出空间而让它们能够完成他们的执行过程。当监理一个哈希表来做哈希连接操作，我们创建一个Bloom过滤器，这个过滤器接下来被传递到探索阶段扫描器，实现了一个简单版本的半连接。

## 结论

在本论文中我们介绍了Cloudera Impala，这是一个开源的SQL引擎，能够将并行DBMS技术带到Hadoop环境。我们的性能结果表明除开Hadoop作为一个批处理环境的事实，监理一个分析型的DBMS在Hadoop的顶层是可行的。并且它在性能上能够表现地和目前的商业处理方法一样好。但在同时，它也保留了Hadoop的灵活性和成本优势。

目前，Impala已经能够在很多的工作领域代替传统的单片分析性关系型数据库系统。我们能够预测随着时间的推移，在这些系统间在SQL功能上的差距将会渐渐消失。然而，我们认为Hadoop环境中的模块化现状有一些优势是传统的大片分析型关系型数据库管理系统所不能实现的。特别的，将文件形式和处理框架混合的能力代表了很多的任务能够被单一的系统执行，而不需要数据迁移。

在Hadoop生态圈中得数据管理缺乏了一些功能，而这些功能已经在过去的年代中被传统的关系型数据库管理系统所实现了。除开这些，我们希望上文中提到的这两种数据库之间的性能差距能够快速缩小，然后这种开放式模块化环境能够允许它在不久的将来成为主要的数据管理结构。