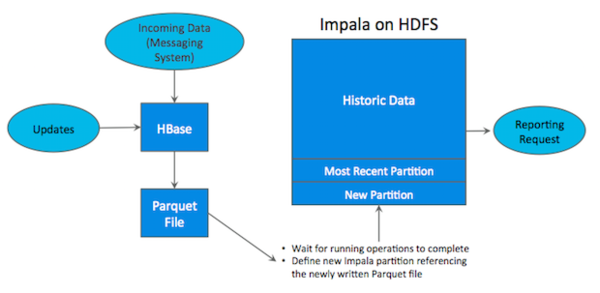
　　Kudu是Cloudera开源的新型列式存储系统，是Apache Hadoop生态圈的新成员之一（incubating），专门为了对快速变化的数据进行快速的分析，填补了以往Hadoop存储层的空缺。本文主要对Kudu的动机、背景，以及架构进行简单介绍。

**背景——功能上的空白**

　　Hadoop生态系统有很多组件，每一个组件有不同的功能。在现实场景中，用户往往需要同时部署很多Hadoop工具来解决同一个问题，这种架构称为混合架构 (hybrid architecture)。比如，用户需要利用Hbase的快速插入、快读random access的特性来导入数据，HBase也允许用户对数据进行修改，HBase对于大量小规模查询也非常迅速。同时，用户使用HDFS/Parquet + Impala/Hive来对超大的数据集进行查询分析，对于这类场景， Parquet这种列式存储文件格式具有极大的优势。

　　很多公司都成功地部署了HDFS/Parquet + HBase混合架构，然而这种架构较为复杂，而且在维护上也十分困难。首先，用户用Flume或Kafka等数据Ingest工具将数据导入HBase，用户可能在HBase上对数据做一些修改。然后每隔一段时间(每天或每周)将数据从Hbase中导入到Parquet文件，作为一个新的partition放在HDFS上，最后使用Impala等计算引擎进行查询，生成最终报表。



　　这样一条工具链繁琐而复杂，而且还存在很多问题，比如：

　　·?如何处理某一过程出现失败？

　　·?从HBase将数据导出到文件，多久的频率比较合适？

　　·?当生成最终报表时，最近的数据并无法体现在最终查询结果上。

　　·?维护集群时，如何保证关键任务不失败？

　　·?Parquet是immutable，因此当HBase中删改某些历史数据时，往往需要人工干预进行同步。

　　这时候，用户就希望能够有一种优雅的存储解决方案，来应付不同类型的工作流，并保持高性能的计算能力。Cloudera很早就意识到这个问题，在2012年就开始计划开发Kudu这个存储系统，终于在2015年发布并开源出来。Kudu是对HDFS和HBase功能上的补充，能提供快速的分析和实时计算能力，并且充分利用CPU和I/O资源，支持数据原地修改，支持简单的、可扩展的数据模型。

**背景——新的硬件设备**

　　RAM的技术发展非常快，它变得越来越便宜，容量也越来越大。Cloudera的客户数据显示，他们的客户所部署的服务器，2012年每个节点仅有32GB RAM，现如今增长到每个节点有128GB或256GB RAM。存储设备上更新也非常快，在很多普通服务器中部署SSD也是屡见不鲜。HBase、HDFS、以及其他的Hadoop工具都在不断自我完善，从而适应硬件上的升级换代。然而，从根本上，HDFS基于03年GFS，HBase基于05年BigTable，在当时系统瓶颈主要取决于底层磁盘速度。当磁盘速度较慢时，CPU利用率不足的根本原因是磁盘速度导致的瓶颈，当磁盘速度提高了之后，CPU利用率提高，这时候CPU往往成为系统的瓶颈。HBase、HDFS由于年代久远，已经很难从基本架构上进行修改，而Kudu是基于全新的设计，因此可以更充分地利用RAM、I/O资源，并优化CPU利用率。我们可以理解为，Kudu相比与以往的系统，CPU使用降低了，I/O的使用提高了，RAM的利用更充分了。

**简介**

　　Kudu设计之初，是为了解决一下问题：

　　·?对数据扫描(scan)和随机访问(random access)同时具有高性能，简化用户复杂的混合架构

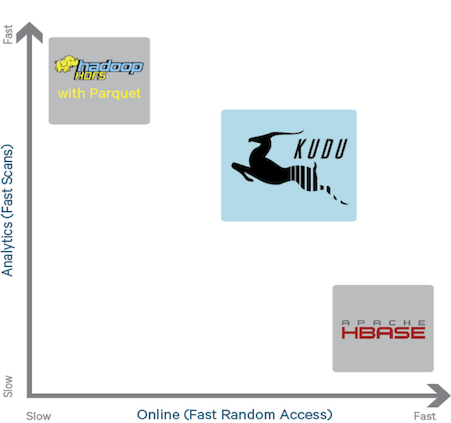
　　·?高CPU效率，使用户购买的先进处理器的的花费得到最大回报

　　·?高IO性能，充分利用先进存储介质

　　·?支持数据的原地更新，避免额外的数据处理、数据移动

　　·?支持跨数据中心replication

　　Kudu的很多特性跟HBase很像，它支持索引键的查询和修改。Cloudera曾经想过基于Hbase进行修改，然而结论是对HBase的改动非常大，Kudu的数据模型和磁盘存储都与Hbase不同。HBase本身成功的适用于大量的其它场景，因此修改HBase很可能吃力不讨好。最后Cloudera决定开发一个全新的存储系统。



　　Kudu的定位是提供”fast analytics on fast data”，也就是在快速更新的数据上进行快速的查询。它定位OLAP和少量的OLTP工作流，如果有大量的random accesses，官方建议还是使用HBase最为合适。

**架构与设计**

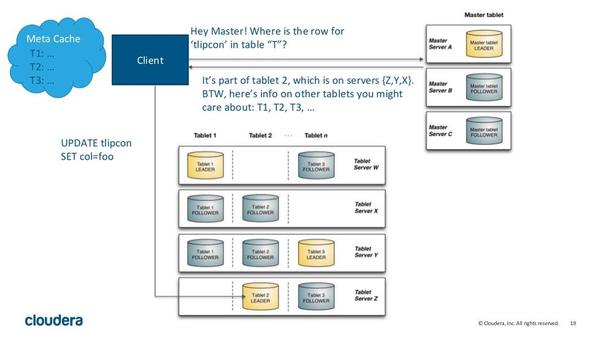
**1.基本框架**

　　Kudu是用于存储结构化（structured）的表（Table）。表有预定义的带类型的列（Columns），每张表有一个主键（primary key）。主键带有唯一性（uniqueness）限制，可作为索引用来支持快速的random access。

　　类似于BigTable，Kudu的表是由很多数据子集构成的，表被水平拆分成多个Tablets. Kudu用以每个tablet为一个单元来实现数据的durability。Tablet有多个副本，同时在多个节点上进行持久化。

　　Kudu有两种类型的组件，Master Server和Tablet Server。Master负责管理元数据。这些元数据包括talbet的基本信息，位置信息。Master还作为负载均衡服务器，监听Tablet Server的健康状态。对于副本数过低的Tablet，Master会在起replication任务来提高其副本数。Master的所有信息都在内存中cache，因此速度非常快。每次查询都在百毫秒级别。Kudu支持多个Master，不过只有一个active Master，其余只是作为灾备，不提供服务。

　　Tablet Server上存了10~100个Tablets，每个Tablet有3（或5）个副本存放在不同的Tablet Server上，每个Tablet同时只有一个leader副本，这个副本对用户提供修改操作，然后将修改结果同步给follower。Follower只提供读服务，不提供修改服务。副本之间使用raft协议来实现High Availability，当leader所在的节点发生故障时，followers会重新选举leader。根据官方的数据，其MTTR约为5秒，对client端几乎没有影响。Raft协议的另一个作用是实现Consistency。Client对leader的修改操作，需要同步到N/2+1个节点上，该操作才算成功。



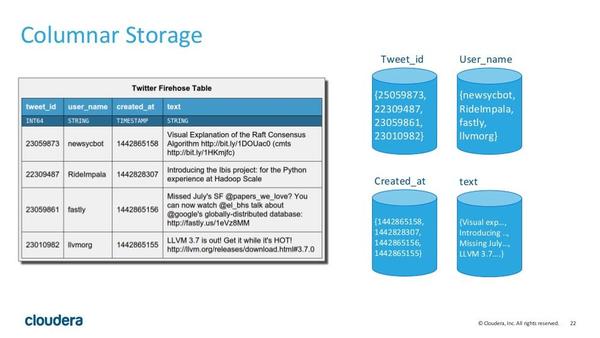
　　Kudu采用了类似log-structured存储系统的方式，增删改操作都放在内存中的buffer，然后才merge到持久化的列式存储中。Kudu还是用了WALs来对内存中的buffer进行灾备。

**2.列式存储**

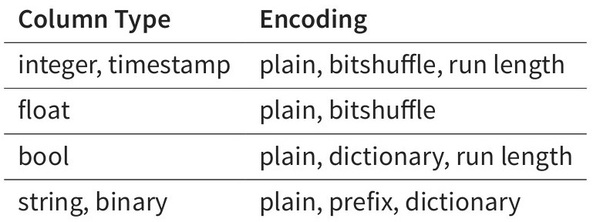
　　持久化的列式存储存储，与HBase完全不同，而是使用了类似Parquet的方式，同一个列在磁盘上是作为一个连续的块进行存放的。例如，图中左边是twitter保存推文的一张表，而图中的右边表示了表在磁盘中的的存储方式，也就是将同一个列放在一起存放。这样做的第一个好处是，对于一些聚合和join语句，我们可以尽可能地减少磁盘的访问。例如，我们要用户名为newsycbot

　　的推文数量，使用查询语句：

　　SELECT COUNT(\*) FROM tweets WHERE user\_name = ‘newsycbot’;



　　我们只需要查询User\_name这个block即可。同一个列的数据是集中的，而且是相同格式的，Kudu可以对数据进行编码，例如字典编码，行长编码，bitshuffle等。通过这种方式可以很大的减少数据在磁盘上的大小，提高吞吐率。除此之外，用户可以选择使用通用的压缩格式对数据进行压缩，如LZ4, gzip, 或bzip2。这是可选的，用户可以根据业务场景，在数据大小和CPU效率上进行权衡。这一部分的实现上，Kudu很大部分借鉴了Parquet的代码。



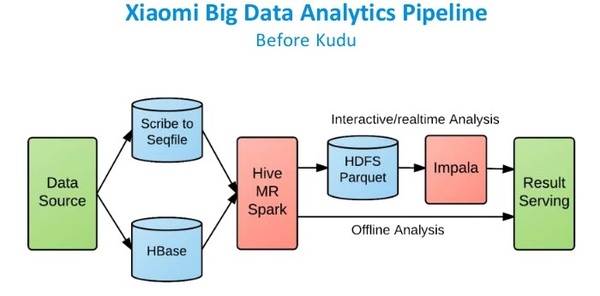
　　HBase支持snappy存储，然而因为它的LSM的数据存储方式，使得它很难对数据进行特殊编码，这也是Kudu声称具有很快的scan速度的一个很重要的原因。不过，因为列式编码后的数据很难再进行修改，因此当这写数据写入磁盘后，是不可变的，这部分数据称之为base数据。Kudu用MVCC（多版本并发控制）来实现数据的删改功能。更新、删除操作需要记录到特殊的数据结构里，保存在内存中的DeltaMemStore或磁盘上的DeltaFIle里面。DeltaMemStore是B-Tree实现的，因此速度快，而且可修改。磁盘上的DeltaFIle是二进制的列式的块，和base数据一样都是不可修改的。因此当数据频繁删改的时候，磁盘上会有大量的DeltaFiles文件，Kudu借鉴了Hbase的方式，会定期对这些文件进行合并。

**3.对外接口**

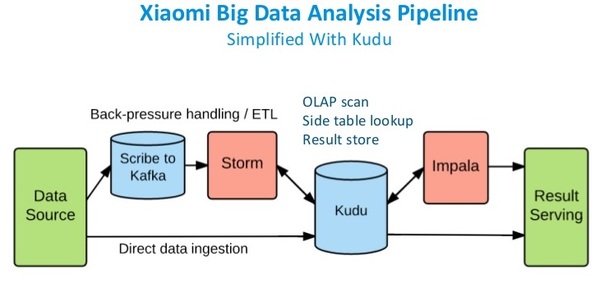
　　Kudu提供C++和JAVA API，可以进行单条或批量的数据读写，schema的创建修改。除此之外，Kudu还将与hadoop生态圈的其它工具进行整合。目前，kudu beta版本对Impala支持较为完善，支持用Impala进行创建表、删改数据等大部分操作。Kudu还实现了KuduTableInputFormat和KuduTableOutputFormat，从而支持Mapreduce的读写操作。同时支持数据的locality。目前对spark的支持还不够完善，spark只能进行数据的读操作。

**使用案例——小米**

　　小米是Hbase的重度用户，他们每天有约50亿条用户记录。小米目前使用的也是HDFS + HBase这样的混合架构。可见该流水线相对比较复杂，其数据存储分为SequenceFile，Hbase和Parquet。

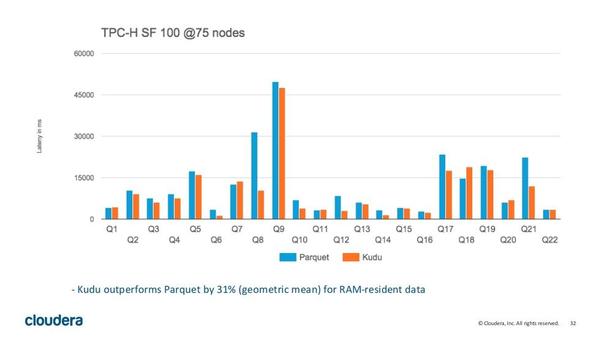


　　在使用Kudu以后，Kudu作为统一的数据仓库，可以同时支持离线分析和实时交互分析。



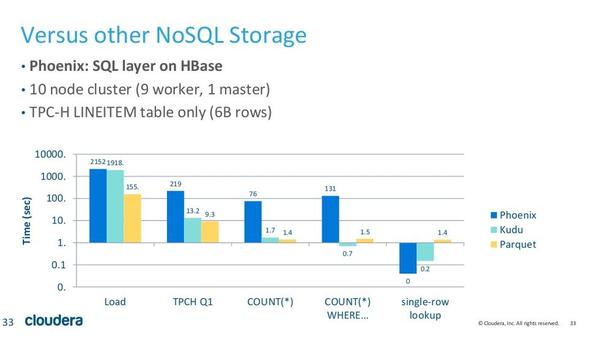
**性能测试**

**1. 和parquet的比较**



　　图是官方给出的用Impala跑TPC-H的测试，对比Parquet和Kudu的计算速度。从图中我们可以发现，Kudu的速度和parquet的速度差距不大，甚至有些Query比parquet还快。然而，由于这些数据都是在内存缓存过的，因此该测试结果不具备参考价值。

**2.和Hbase的比较**



　　图是官方给出的另一组测试结果，从图中我们可以看出，在scan和range查询上，kudu和parquet比HBase快很多，而random access则比HBase稍慢。然而数据集只有60亿行数据，所以很可能这些数据也是可以全部缓存在内存的。对于从内存查询，除了random access比HBase慢之外，kudu的速度基本要优于HBase。

**3.超大数据集的查询性能**

　　Kudu的定位不是in-memory database。因为它希望HDFS/Parquet这种存储，因此大量的数据都是存储在磁盘上。如果我们想要拿它代替HDFS/Parquet + HBase，那么超大数据集的查询性能就至关重要，这也是Kudu的最初目的。然而，官方没有给出这方面的相关数据。由于条件限制，网易暂时未能完成该测试。下一步，我们将计划搭建10台Kudu + Impala服务器，并用tpc-ds生成超大数据，来完成该对比测验。