实用 | Apache Kudu读写路径

2017-05-12 Cloudera中国

作者：James Kinley 和 David Alves

James Kinley是Cloudera公司的一名首席解决方案架构师。

David Alves在Cloudera公司的一名软件工程师，并且是Apache Kudu的贡献者，同时也担任PMC。

网址：http://blog.cloudera.com/blog/2017/04/apache-kudu-read-write-paths/

分析和操作访问模式之间存在明显的区别，直到现在，Hadoop生态系统还没有一个可以同时支持这两种模式的独立存储引擎。因此，工程师们为提供这些功能不得不实施复杂的体系架构将多个系统连接在一起。 一方面，HDFS上的不可变数据提供了卓越的分析性能，而Apache HBase中的可变数据最适合用于操作工作负载。但是，Apache Kudu很好地弥补了同时支持这两种模式这一空白。

Kudu的体系架构已经具备了提供良好分析性能的能力，同时还能够接收插入和更新操作的连续流。为了使用户能够专注于其最关心的内容，Kudu提供了简单的API，而封装了后台的复杂性。但是一些高级用户希望了解内部部件，以理解Kudu如何能够快速分析快速数据，以及如何更好地利用其功能。本篇博文旨在向用户介绍向Kudu内写入数据以及从Kudu中读取数据时在其后台会发生什么。本篇博文假设读者对本文中所介绍的Kudu架构已经有一个基本的了解。

多版本并发控制（MVCC）

数据库使用并发控制方法确保用户始终看到一致性的结果，不管是否进行并发写入操作。Kudu使用了一种称为多版本并发控制（MVCC）的方法，该方法可以跟踪正在进行的操作，并通过确保读取操作只能读到已提交的操作来保证一致性。Kudu使用多版本并发控制（MVCC）的主要优点是允许读取者（通常是大型扫描程序）不必获取锁定，分析作业不会阻止同一数据上的并发写入者，这样就可以显著提高其扫描性能。

每个写入都使用系统生成的时间戳进行标记，该时间戳保证在tablet中是唯一的。当用户创建扫描程序从tablet读取数据时，他们可以选择两种读取模式：

- **READ_LATEST**（默认）在tablet中获取多版本并发控制（MVCC）当前状态的快照，无法保证最新版本。也就是说，该功能可读取在副本上提交的任何写入，但是由于写入可以无序执行，所以不保证其一致性。而且由于副本可能落后于同一tablet中的其他副本，因此不能保证读取的是最新版本。

- **READ_AT_SNAPSHOT**可以获取多版本并发控制（MVCC）的快照，其中包括基于特定时间戳的行版本，无论是用户选择还是系统选择（“当前”时间）。tablet等到这个时间戳是“安全的”（即已经完成所有具有较低时间戳的飞行中写入操作），并且由于具有较晚的时间戳，tablet中的进一步写入将被扫描程序忽略。在这种模式下，扫描是一致且可重复的。

通过提供您自己的扫描时间戳，第二种快照类型具备发布 “时间行程读取”的能力，这反映了在该时间点数据库的状态。

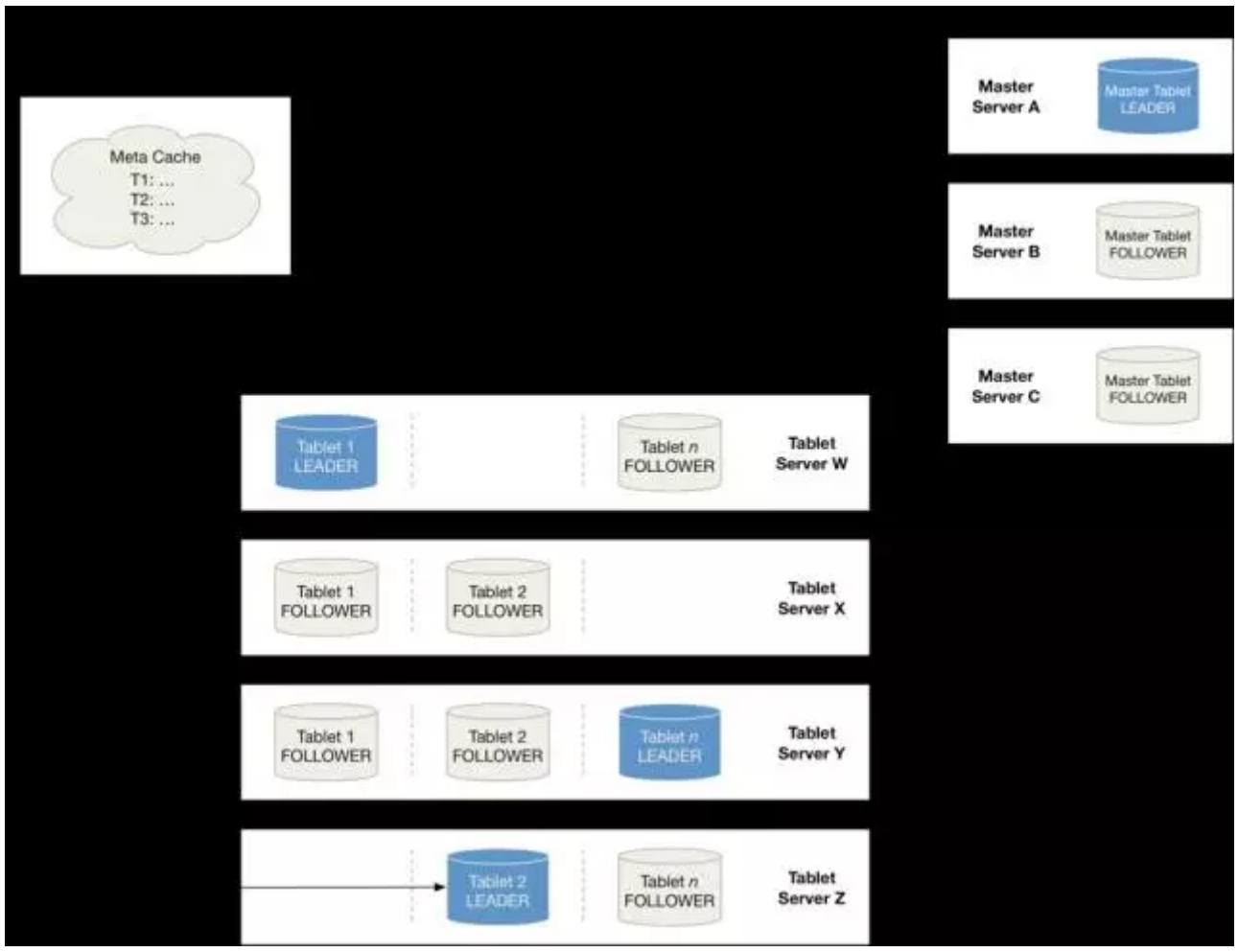
这意味着用户只能看到单个版本的行，但在内部Kudu可以存储行的多个版本以提供多版本并发控制（MVCC）快照功能。

Raft

Kudu中的表被分割成称为tablet的连续片段，并且为了实现容错功能，每个tablet都在多个tablet服务器上进行复制。 Kudu使用Raft一致性算法以保证对tablet进行的更改获得其所有副本的同意。在任何时候，其中一个副本会被选为领导者，而其他则是追随者。任何副本都可以为读取提供服务，但只有领导者可以接受写入。Raft保留了复制的操作日志，只有在大多数副本被持久存储在日志中时才会对其进行确认。复制日志是一个抽象概念，实际上由tablet的写入日志（WAL）表示。具有N个副本（通常为3或5个）的tablet可以继续接受最多（N-1）/ 2个副本的失败。

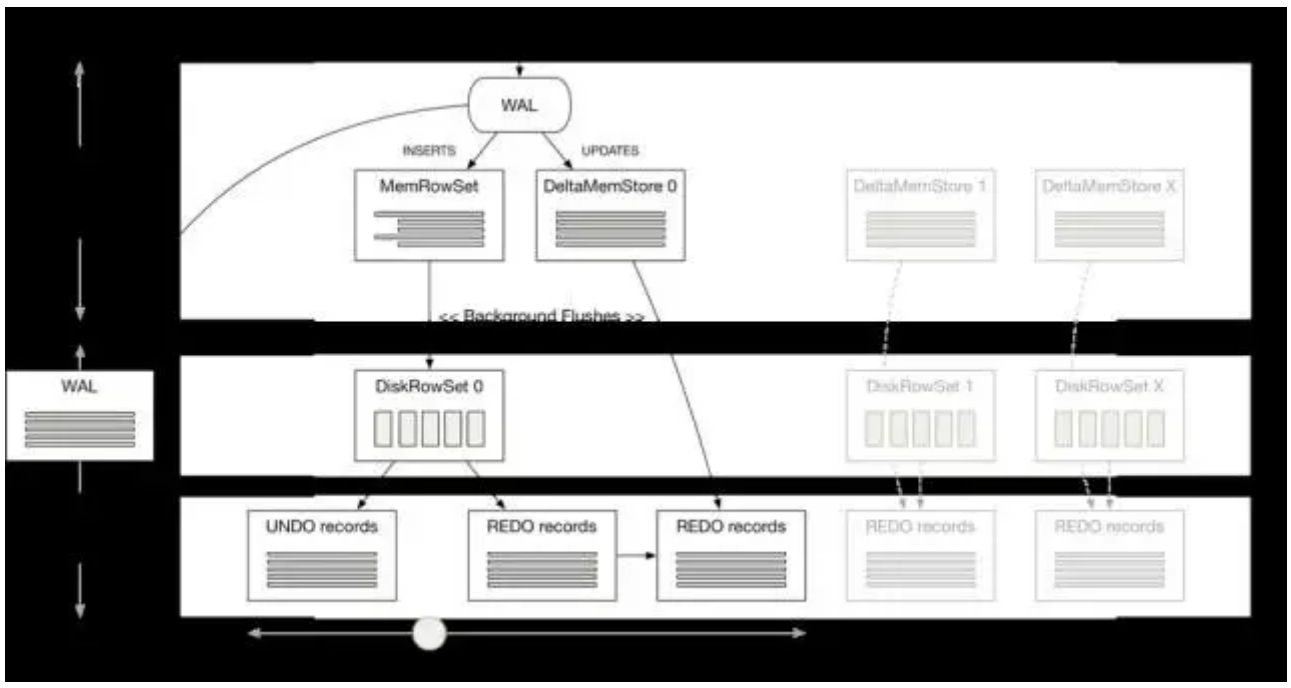
Tablet发现

当创建Kudu客户端时，其会从主服务器上获取tablet位置信息，然后直接与服务于该tablet的服务器进行交谈。为了优化读取和写入路径，客户端将保留该信息的本地缓存，以防止他们在每个请求时需要查询主机的tablet位置信息。随着时间的推移，客户端的缓存可能会变得过时，并且当写入被发送到不再是tablet领导者的tablet服务器时，则将被拒绝。然后，客户端将通过查询主服务器发现新领导者的位置来更新其缓存。



写入路径

写入操作是指需进行插入、更新或删除操作的一组行。需要注意的事项是Kudu强制执行主关键字的唯一性，主关键字是可以更改行的唯一标识符。为了强制执行此约束条件，Kudu必须以不同的方式处理插入和更新操作，并且这会影响tablet服务器如何处理写入。



Kudu中的每个tablet包含预写式日志（WAL）和多个行集合（RowSet），它们是保存在存储器和磁盘上（被刷新时）的不相交的行集合。写入操作先被提交到tablet的预写式日志（WAL），并根据Raft 一致性算法取得追随节点的同意，然后才会被添加到其中一个tablet的内存中：

插入会被添加到tablet的MemRowSet中。为了在MemRowSet中支持多版本并发控制（MVCC），对最近插入的行（即尚未刷新到磁盘的新的行）的更新和删除操作将被追加到MemRowSet中的原始行之后以生成重做（REDO）记录的列表。读取者需要应用相关的重做（REDO）记录，根据扫描程序给定的时间戳构建行的正确快照。

当MemRowSet填满时，则被刷新到磁盘并成为一个DiskRowSet。为了支持磁盘中存储数据的多版本并发控制（MVCC）功能，DiskRowSets被分为两种不同的文件类型。 MemRowSet中行的最新版本（即应用了其所有重做（REDO）记录的原始插入）被写入到基础数据文件。这是一种柱状文件格式（非常像Parquet），用于快速、高效的读取访问，其赋予了Kudu支持分析访问模式的能力。MemRowSet中存在的行的先前版本（即重做（REDO）记录的倒序）作为一组撤销（UNDO）记录写入增量文件。时间行程读取可以应用相关的撤销（UNDO）记录，从早期的时间点构建行的正确快照。

更新已被编码和压缩的柱状格式化数据文件需要重写整个文件，因此基础数据文件一旦被刷新则被认为是不可变的。此外，行关键字唯一性约束意味着基本记录的更新和删除不能被添加到tablet的MemRowSet中，而是被添加到名为DeltaMemStore的单独的内存存储器中。像MemRowSet一样，所有的变化都将被添加到DeltaMemStore中作为一组重做（REDO）记录；当DeltaMemStore填满时，重做（REDO）记录将被刷新到磁盘上存储的增量文件中。每个DiskRowSet都存在一个单独的DeltaMemStore。如需构建行的正确快照，读取者必须在应用相关撤销（UNDO）或重做（REDO）记录之前首先找到行的基本记录。

压缩

随着时间的推移，tablet会积累许多DiskRowSets，并且会在行更新时累积很多增量重做（REDO）文件。当插入一个关键字时，为了强制执行主关键字唯一性，Kudu会针对RowSets查询一组布隆过滤器，来找到可能包含该关键字的Rowset。越多的布隆过滤器检查及随后的DiskRowSet搜索，写入操作就会变得越慢。随着更多DiskRowSets的积累，必须采取措施确保写入性能不会下降。

此外，随着每个RowSet累积更多的重做（REDO）增量文件，为了将基础数据转换为行的最新版本，需要执行更多的工作扫描。这意味着如果不采取任何行动，读取性能也会随时间而下降。Kudu可以通过执行压缩功能来处理这些问题，其中包括三种类型的压缩：

- **微小增量压缩（Minor delta compaction）**可以在不会触及基础数据的前提下，通过将增量文件合并在一起以减少文件的数量。其结果是读取操作必须寻求更少的增量文件来生成当前版本的行。

- **重大增量压缩（Major delta compaction）**将重做（REDO）记录迁移到撤销（UNDO）记录中，并更新基础数据。考虑到大多数读取将读取最近的快照（从基础数据的时间向前计算），并且基础数据将以最有效的方式进行存储（被编码和压缩），最小化重做（REDO）记录存储的数量可以获得更有效的读取。在重大压缩期间，行的重做（REDO）记录可以合并到基础数据中，并替换为包含先前版本行的等效的撤销（UNDO）记录集。可以对列的任何子集执行重大压缩，因此如果某列与其他列相比收到更多的更新，就可以在单个列上执行压缩，通过避免重写未更改的数据以减少重大增量压缩的I / O。

- **行集合压缩（RowSet compaction）**可以将范围内重叠的行集合（RowSets）组合在一起，从而生成更少的行集合（RowSets），从而加快写入速度。

这是针对压缩过程的简要介绍，旨在方便读者理解Kudu执行的后台工作以管理和优化其物理存储。在博客中我们将发表后续文章更详细地描述该压缩过程。

读取路径

当客户端从Kudu的表中读取数据时，必须首先建立需要连接的系列tablet服务器。通过执行tablet发现过程（如上所述）来确定包含要读取的主关键字范围的tablet的位置（读取不必在领导者tablet上发生，除非用户明确选择该选项）。tablet随后使用扫描程序基于行集合（RowSets）和相关撤销（UNDO）或重做（REDO）增量记录生成最终行。

首先，tablet必须确定包含基本记录的行集合（RowSet）。然后扫描MemRowSet以及一个或多个DiskRowSets。最后，tablet将遍历所选的行集合（RowSet），匹配该扫描的谓词并生成最终记录。

在MemRowSet中，扫描程序将实现每一行的完整投影并应用任何增量记录。由于所有操作都发生在内存中因此可以非常快速地完成。对于每一个DiskRowSet而言，扫描程序将每次实现一列，并且在转到下一列前应用对当前列任何增量记录和谓词。 只有与扫描的谓词匹配的列才可以从磁盘读取，这使得磁盘I / O非常高效，也赋予了Kudu快速的分析性能。Kudu将优先扫描已定义谓词的列。如果谓词不满足的话，Kudu可以完全避免扫描其他列，从而避免不必要的I / O。

我们希望本篇博文能够对Kudu的读写路径进行清晰简明的概述，并且使读者理解Kudu如何能够在不断变化的数据上支持快速的分析访问模式。