# 概述

Yugabyte DB是一个全球部署的分布式数据库，和国内的TiDB和国外的CockroachDB类似，也是受到Spanner论文启发，所以在很多地方这几个数据库存在不少相似之处。

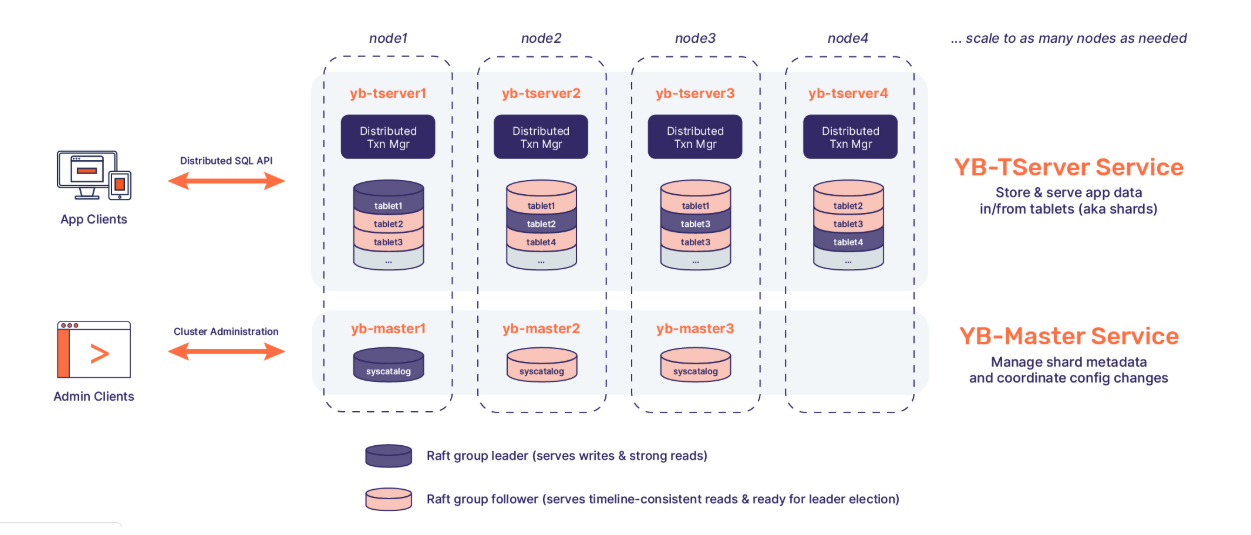
与Cockroach类似，Yugabyte也主打全球分布式的事务数据库——不仅能把节点部署到全球各地，还能完整支持ACID事务，这是他最大的卖点。除此以外还有一些独特的特性，比如支持文档数据库接口。

# 架构

逻辑上，Yugabyte采用两层架构：查询层和存储层。不过这个架构仅仅是逻辑上的，部署结构中，这两层都位于TServer进程中。这一点和TiDB不同。

Yugabyte的查询层支持同时SQL和CQL（Cloud Query Language）两种API，其中CQL是兼容Cassandra的一种方言语法，对应于文档数据库的存储模型；而 SQL API是直接基于PostgresQL魔改的，能比较好地兼容PG语法，据官方说这样可以更方便地跟随PG新特性。

Yugabyte的存储层才是重头戏。其中TServer负责存储tablet，每个tablet对应一个Raft Group，分布在三个不同的节点上，以此保证高可用性。Master负责元数据管理，除了tablet的位置信息，还包括表结构等信息。Master本身也依靠Raft实现高可用。

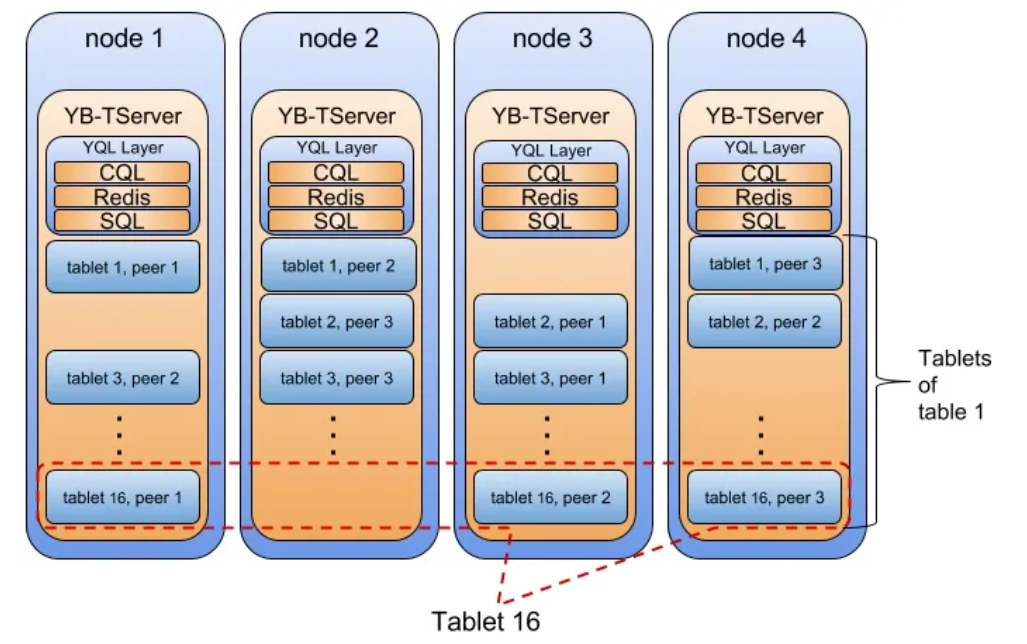


# 基于Tablet的分布式存储

这一部分是HBase/Spanner精髓部分，Cockroach/TiDB的做法几乎也是一模一样的。

如下图所示，每张表被分成很多个tablet，tablet是数据分布的最小单元，通过在节点间搬运tablet以及tablet的分裂与合并，就可以实现几乎无上限的scale out。每个tablet有多个副本，形成一个Raft Group，通过Raft协议保证数据的高可用和持久性，Group Leader负责处理所有的写入负载，其他Follower作为备份。

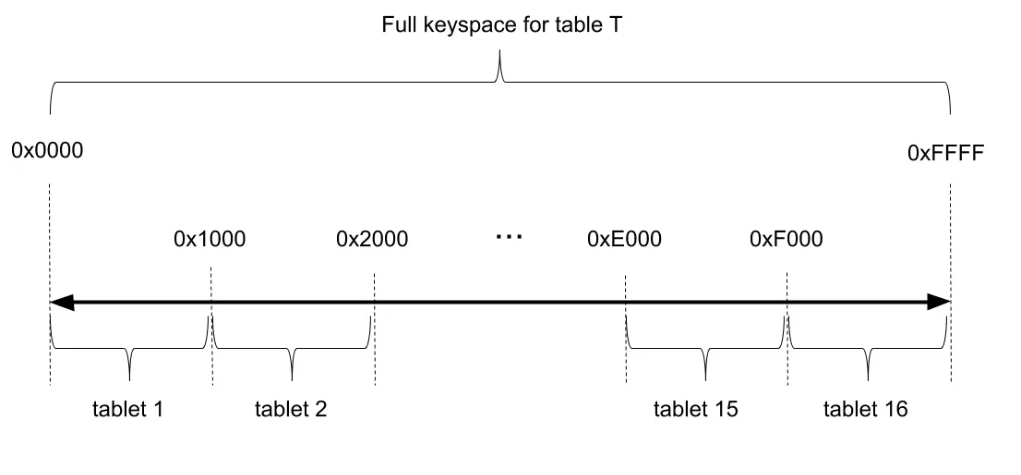
下图是一个例子：一张表被分成16个tablet，tablet的副本和Raft Group leader均匀分布在各个节点上，分别保证了数据的均衡和负载的均衡。



和其他产品一样，Master节点会负责协调tablet的搬运、分裂等操作，保证集群的负载均衡。这些操作是直接基于Raft Group实现的。

有趣的是，Yugabyte采用哈希和范围结合的分区方式：可以只有哈希分区、也可以只有范围分区、也可以先按哈希再按范围分区。之所以这么设计，猜测也是因为Cassandra的影响。相比之下，TiDB和Cockroach都只支持范围分区。

哈希分区的方式是将key哈希映射到2字节的空间中（即 0x0000 到 0xFFFF），这个空间又被划分成多个范围，比如下图的例子中被划分为16个范围，每个范围的key落在一个tablet中。理论上说最多可能有64K个tablet，这对实际使用足够了。



哈希分区的好处是插入数据（尤其是从尾部 append 数据）时不会出现热点；坏处是对于小范围的范围扫描（例如 pk BETWEEN 1 AND 10）性能会比较吃亏。

# 基于RocksDB的本地存储

每个TServer节点上的本地存储称为DocDB。和TiDB/Cockroach一样，Yugabyte也用RocksDB来做本地存储。这一层需要将关系型tuple以及文档编码为key-value保存到RocksDB 中，下图是对文档数据的编码方式，其中有不少是为了兼容Cassandra设计的，我们忽略这些，主要关注以下几个部分：

1、key中包含

16-bit hash：依靠这个值才能做到哈希分区

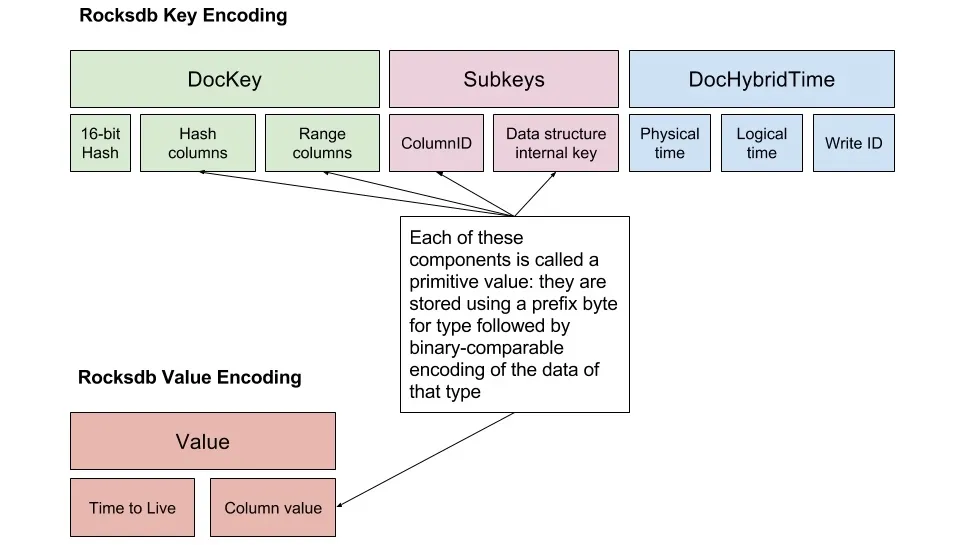
主键数据（对应图中hash/range columns）

column ID：因为每个tuple有多个列，每个列在这里需要用一个key-value来表示

hybrid timestamp：用于MVCC的时间戳

2、value中包含

column的值



如果撇开文档模型，key-value的设计很像 Cockroach：每个cell（一行中的一列数据）对应一个key-value。而TiDB是每个tuple 打包成一个key-value。个人比较偏好TiDB的做法。

# 事务

和TiDB/Cockroach一样，Yugabyte也采用了MVCC结合2PC的事务实现。

## 时间戳

时间戳是分布式事务的关键选型之一。Yugabyte和Cockroach一样选择的是Hybrid Logical Clock (HLC)。

HLC将时间戳分成物理（高位）和逻辑（低位）两部分，物理部分对应UNIX时间戳，逻辑部分对应Lamport时钟。在同一毫秒以内，物理时钟不变，而逻辑时钟就和Lamport时钟一样处理——每当发生信息交换（RPC）就需要更新时间戳，从而确保操作与操作之间能够形成一个偏序关系；当下一个毫秒到来时，逻辑时钟部分归零。

不难看出，HLC的正确性其实是由Logical Clock来保证的：它相比Logical Clock只是在每个毫秒引入了一个额外的增量，显然这不会破坏Logical Clock的正确性。但是，物理部分的存在将原本无意义的时间戳赋予了物理意义，提高了实用性。

个人认为，**HLC是除了TrueTime以外最好的时间戳实现了，唯一的缺点是不能提供真正意义上的外部一致性，仅仅能保证相关事务之间的“外部一致性”**。 另一种方案是引入中心授时节点（TSO），也就是TiDB使用的方案。TSO方案要求所有事务必须从TSO获取时间戳，实现相对简单，但引入了更多的网络RPC，而且TSO过于关键——短时间的不可用也是极为危险的。

HLC的实现中有一些很tricky的地方，比如文档中提到的Safe timestamp assignment for a read request。对于同一事务中的多次read，问题还要更复杂，有兴趣的读者可以看Cockroach团队的这篇博客 Living Without Atomic Clocks。

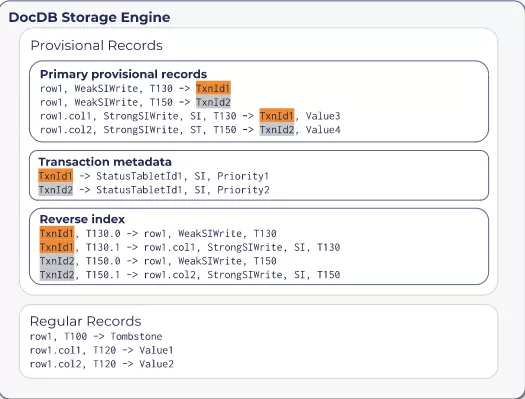
## 事务提交

毫不惊奇，Yugabyte的分布式事务同样是基于2PC的。他的做法接近Cockroach。事务提交过程中，他会在DocDB存储里面写入一些临时的记录（provisional records），包括以下三种类型：

Primary provisional records：还未提交完成的数据，多了一个事务ID，也扮演锁的角色

Transaction metadata：事务状态所在的tablet ID。因为事务状态表很特殊，不是按照hash key分片的，所以需要在这里记录一下它的位置。

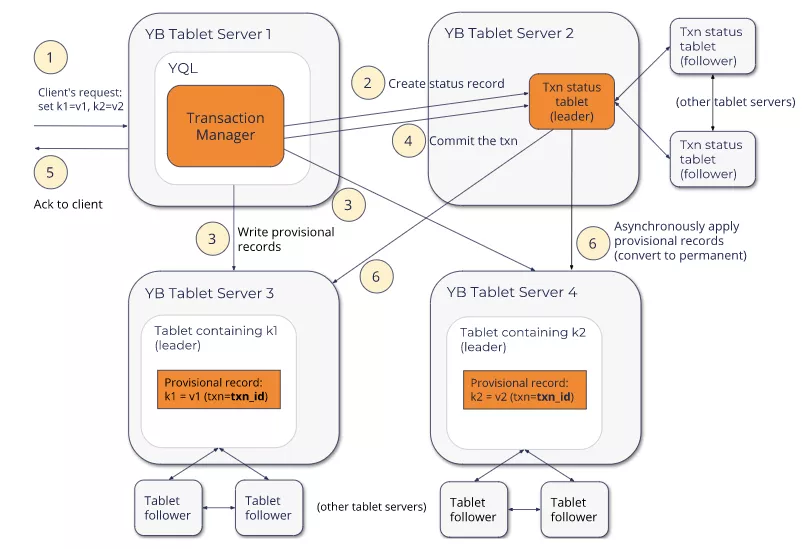
Reverse Index：所有本事务中的primary provisional records，便于恢复使用



事务的状态信息保存在另一个tablet上，包括三种可能的状态：Pending、Committed或Aborted。事务从Pending状态开始，终结于Committed或Aborted。

事务状态就是Commit Point的那个“开关”，当事务状态切换到Commited的一瞬间，就意味着事务的成功提交。这是保证整个事务原子性的关键。

完整的提交流程如下图所示：



另外，Yugabyte文档中提到它除了Snapshot Isolation还支持Serializable隔离级别，但是似乎没有看到他是如何规避Write Skew问题的。从Release Notes看来这应该是2.0 GA中新增加的功能！