知乎 首发于 数据库前沿技术

[Paper Review] SIGMOD'18 Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits


陈宗志
研发工程师.

关注他

89 人赞同了该文章

这个是Amazon Aurora 发的第二篇文章, 发在2018 年SIGMOD上, 题目很吸引人避免在I/O, commit, 成员变更的过程使用一致性协议. 在大家都在使用一致性协议(raft, multi-paxos)的今天, Aurora 又提出来了不用一致性协议来做, 主要观点是现有这些协议太重, 而且会带来额外的网络开销, 也可以理解, 毕竟Aurora 是6副本, 主要的瓶颈是在网络上. 那么他是怎么做的?

因为Aurora 很多细节还是没有揭露, 所以很多内容是我自己的解读, 以及问的作者, 如果错误, 欢迎探讨

这篇文章也主要回答这个问题.

Aurora is able to avoid distributed consensus during writes and commits by managing consistency points in the database instance rather than establishing consistency across multiple storage nodes.

在Aurora 中, storage tier 没有权限决定是否接受write, 而是必须去接受database 传过来的write. 然后都是由database tier 去决定是否这个 SCL, PGCL, VCL 是否可以往前推进, 也就是说 storage tier 本身并不是一个强一致.

出底层的系统只需要是一个quorum 的系统, storage tier + database tier 实现一个强一致的方案.

比如像Spanner 里面, 每一个spanservers 本身是多副本, 多副本之间通过multi-paxos 来保证数据的一致性, 然后上层的F1 这一层主要做的协议转换, 把SQL 协议转换成kv 请求去请求spanserver.

我们的PolarDB 也是这样的一套系统, 底层的存储节点 polarstore 是一个稳定可靠的强一致系统, 上层的计算节点PolarDB 是一个无状态的节点.

接下来具体的 Aurora 是如何实现的呢?

Term:

- LSN: log sequence number
  每一条redo log 有一个唯一的单调递增的 Log Sequence Number(LSN), 这个LSN 是由database 来生成, 由于Aurora 是一写多读的结构, 很容易满足单调递增
- SCL: segment complete LSN
  SCL(segment complete LSN) 表示的是当前这个segment 所知道的最大的LSN, 在这个SCL 之前的所有记录当前这个节点已经收到, 到SCL 位置的数据都是连续的. **这里与VCL 的区别是, VCL 是所有节点确认的已经提交的LSN, 而SCL 是自己认为确认已经提交的LSN, VCL 可以认为是storage node 的commit index, 而SCL只是记录当前节点的LastLogIndex** Aurora 也会使用这个SCL来进行节点间交互去补齐log.
- VCL: volume complete LSN
  这个VCL 就是storage node 认为已经提交的LSN, 也就是storage node 保证小于等于这个VCL 的数据都已经确认提交了, 一旦确认提交, 下次recovery 的时候, 这些数据是保证有的. 如果在storage node recovery 阶段的时候, 比VCL 大于的数据就必须要删除, VCL 相当于commit Index. 这个VCL 只是在storage node 层面保证的, 有可能后续database 会让VCL 把某一段开始的 log 又都删掉.
  这里VCL 只是storage node 向database 保证说, 在我storage node 这一层多个节点已经同步, 并且保证一致性了.这个VCL 由storage node 提供.
- PGCL: Protection Group Complete LSN
  每一个分片都有自己的SCL, 这个SCL 就叫做PGCL. 等于说SCL 是database 总的SCL, 每一个分片有各自的PGCL, 然后这个database 的SCL 就等于最大的这个PGCL
- CPL: consistency point LSN
  CPL 是由database 提供, 用来告诉storage node 层哪些日志可以持久化了, 其实这个和文件系统里面保证多个操作的原子性是一样的方法.
  为什么需要CPL, 可
  有些日志我已经提

- VDL: volume durable LSN
  因为database 会标记多个CPL, 这些CPL 里面最大的并且比VCL小的CPL叫做VDL(Volume Durable LSNs). 因为VCL表示的是storage node 认为已经确认提交的LSN, 比VCL小, 说明这些日志已经全部都在storage node 这一层确认提交了, CPL 是database 层面告诉storage node 哪些日志可以持久化了, 那么VDL 表示的就是已经经过database 层确认, 并且storage node层面也确认已经持久化的Log, 那么就是目前database 确认已经提交的位置点了.
  所以VDL 是database 这一层已经确认提交的位置点了, 一般来说VCL 都会比VDL 要来的大, 这个VDL 是由database 来提供的, 一般来说VDL 也才是database 层面关心的, 因为VCL 中可能包含一个事务中未提交的部分.
- MTR: mini transaction
  那么事务commit 的过程就是这样, 每一个事务都有一个对应"commit LSN", 那么这个事务提交以后就去做其他的事情, 什么时候通知这个事务已经提交成功呢? 就是当VDL(VDL 由databse 来发送, storage service来确认更新) 大于等于"commit LSN" 以后, 就会有一个专门的线程去通知这个等待的client, 你这个事务已经提交完成了.
  如果这个事务提交失败, 那么接下来的Recovery 是怎么处理的呢?
  首先这个Recovery 是由storage node 来处理的, 是以每一个PG 为维度进行处理, 在database 起来的时候通过 quorum 读取足够多的副本, 然后根据副本里面的内容得到VDL, 因为每一个时候最后一条记录是一个CPL, 这些CPL 里面最大的就是VDL, 然后把这个VDL 发送给其他的副本, 把大于VDL 的redo log 清除掉, 然后恢复这个PG的数据
- SCN: commit redo record for the transaction
  也就是一个transaction 的 commit redo record, 每一个transaction 生成的redo record 里面最大commit LSN. 主要用于检查这个事务是否已经被持久化了
  这里就是通过保证SCN 肯定小于VCL 来进行保证提交的事务是一定能够持久化的, 所以Aurora 一定是将底下的VCL 大于当前这个transaction 的SCN 以后才会对客户端进行返回
- PGM-RPL: Protection Group Minimum Read Point LSN
  这个LSN 主要是为了回收垃圾使用, 表示的是这个database 上面读取的时候最低的LSN, 低于这个LSN 的数据就可以进行清理了. 所以storage node 只接受的是PGMRPL -> SCL 之间的数据的读请求

那么写入流程是怎样?

在database tier 有一个事务需要提交, 那么这个事务可能涉及多个分片(protection group), 那么就会生成多个MTRs, 然后这些MTRs 按照顺序提交log records 给storage tier. 其中每一个MTR 中有可能包含多条log records, 那么这多条log records 中最后一条的LSN, 也称作CPL. storage tier 把本地的SCL 往前移. database tier 在接收到超过大多数的storage node 确认以后, 就把自己的VCL 也往前移. 下一次database tier 发送过来请求的时候, 就会带上这个新的VCL 信息, 那么其他的storage node 节点就会去更新自己的VCL 信息.

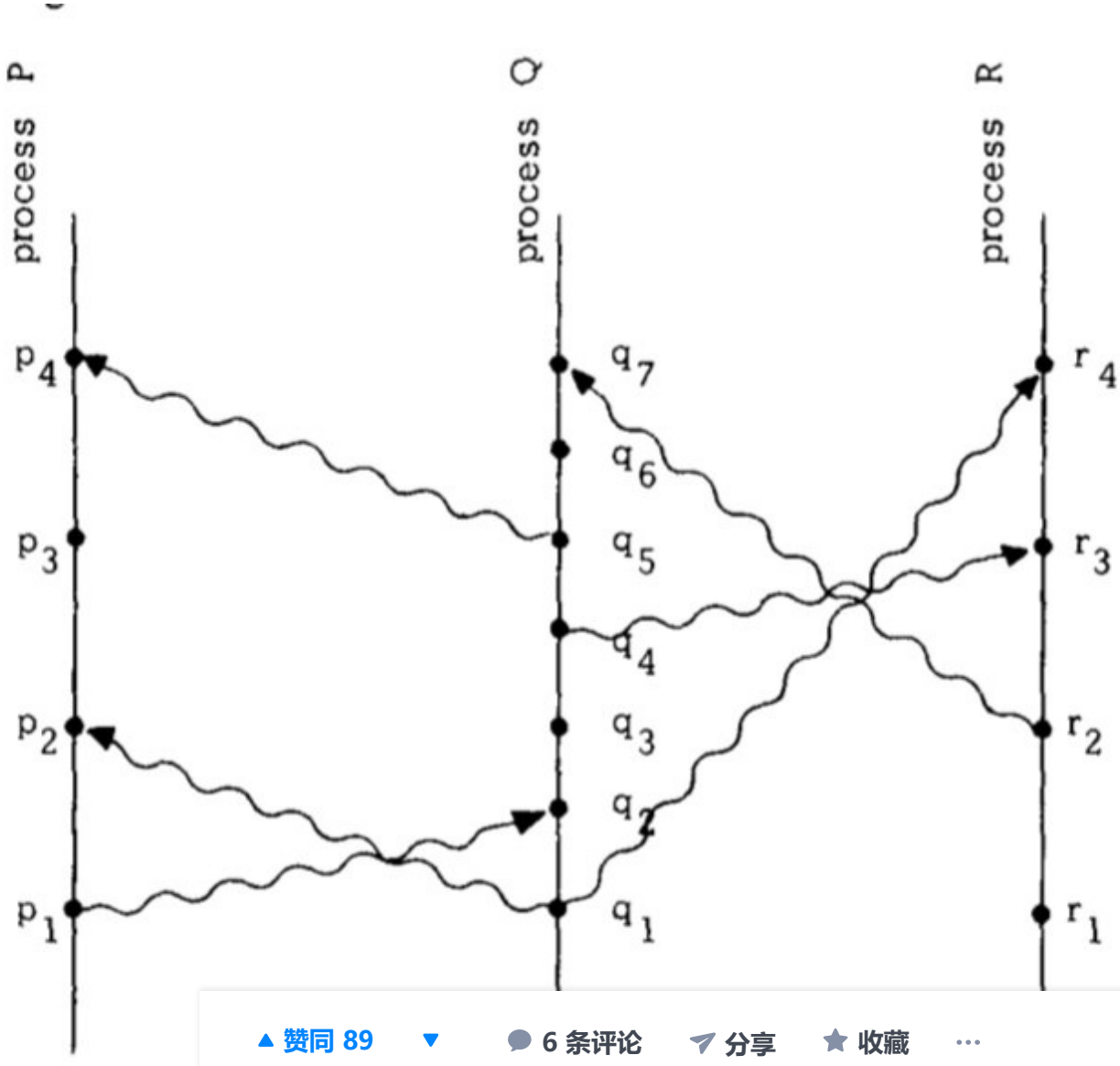
在aurora 的quorum 做法中, 读取的时候并没有走quorum.

从master 节点来说, master 在进行quorum 写入的时候是能够获得并记录每一个storage node 当前的VDL, 所以读取的时候直接去拥有最新的VDL 的storage node 去读取即可.

对于slave 节点, master 在向storage node 写redo record 的同时, 也异步同步redo log给slave 节点, 同时也会更新VDL, VCL, SCL 等等这些信息给从节点, 从节点本身构造本地的local cache. 并且slave 节点也拥有全局的每一个storage node 的VDL 信息, 因此也可以直接访问到拥有最新的 storage node 的节点.

个人观点:

这篇文章开头讲的是通过 quorum I/O, locally observable state, monotonically increasing log ordering 三个性质来实现Aurora, 而不需要通过一致性协议. 那我们一一解读



clock(因为这里只有一个节点是写入节点, 并且如果写入节点挂了以后有一个恢复的过程, 因此可以很容易的保证这个LSN 是递增的)

locally observable state 表示当前节点看到的状态, 也就是每一个节点看到的状态是不一样的, 每一个节点(包含database node 和 storage node) 都有自己认为的 SCL, VCL, VDL 等等信息, 这些信息表示的是当前这个节点的状态, 那么就像Lamport logic clock 文章中所说的一样, 在分布式系统中没有办法判断两个不同节点中的状态的先后顺序, 只有当这两个状态发生消息传递时, 才可以确定偏序关系. 那么在这里就是通过quorum I/O 确定这个偏序关系

quorum I/O 在每一次的quorum IO达成确认以后, 就相当于确认一次偏序关系. 比如在一次写入成功以后, 那么我就可以确定当前的data node 的状态一定是在其他的storage node 之前. 在一次gossip 节点间互相确认信息以后, 主动发起确认信息的节点的状态也一定在其他节点之前. 所以整个系统在写入之后或者在重启之后的gossip 一定能够存在一个在所有节点最前面的状态的节点, 那么这个节点的状态就是当前这个系统的状态, 这个状态所包含的SCL, VCL, VDL 信息就是一致性信息

在每一次读取的时候, 他所在的节点一定是当前这个偏序关系最前面的位置, 因为读取操作之前一定是write 或者 recovery 操作, 在write, recovery 操作的过程中, 就与超过半数的节点达成一致, 走在了偏序关系的最前面. 获得了当前节点的local observable state. 所以读到的一定是最新的内容

Aurora 系统很容易实现的一个重要原因是他是只有一个writer 存在, 这样保证了同一时刻只可能在发生一个事件

刚开始看会觉得这个系统比较琐碎, 不像Paxos/raft 那样有一个比较完备的理论证明, 不过问过作者, 实现这一套过程也是经过TLA+的证明

编辑于 2018-06-22

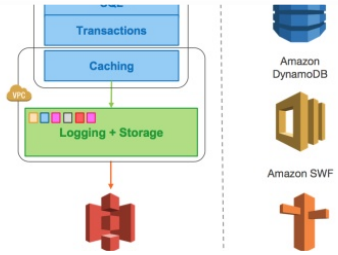
分布式存储 数据库 分布式一致性

文章被以下专栏收录

数据库前沿技

知乎 首发于 **数据库前沿技术**

推荐阅读



Amazon Aurora 读后感

杨东东

CAP，ACID，我们能做什么

本文从CAP理论和ACID性质为切入点，讨论分布式（存储）系统的设计。分布式系统，尤其是分布式存储系统，在进行设计考虑时，首先需要想到的就是CAP问题，即在C（Consistency）和...

张帅

各大公司分布式存储领
文

各大公司系统实现
Google[SOSP’03] The
File System[OSDI’06] B
A Distributed Storage Sy
Structured Data[OSDI’
Chubby lock service for

张帅

6 条评论

⇌ 切换为时间排序

写下你的评论...

科技大昊哥 2018-06-22

去年第一篇的补充。大部分内容在gupta的四篇博客中已经讲过了。

赞

陈宗志 (作者) 回复 科技大昊哥 2018-06-22

嗯, 是这样的. 写的比gupta 的四篇会深入一点

赞

朱一聪 2018-06-22

多写都没讲，又能发一篇。哪天它有兴趣谈谈多写了，又是一篇

赞

lambda喵 2018-07-11

multi-master 后多点写入会怎样

赞

阿莱克西斯

▲ 赞同 89 ▼ 6 条评论 分享 收藏 ···

知乎 首发于 **数据库前沿技术**

无名 2019-09-26

有个问题请教，如果每条事务只涉及一条log record append，那么SCN或者VDL就等于VCL了?
对论文里提到的 达成ack的时机是VCL超过SCN这一点有疑惑。
上述情况，岂不是没法向客户端ack

赞