背景

TiKV 通过 Raft 一致性协议为所有数据存储了三个副本，因此任意一个 TiKV 进程异常退出后，TiKV 都可以继续提供服务，并且只要异常退出的节点重新启动，又可以正常加入到集群中。但是对于云上 VM 用户而言，任何一个 VM 挂掉后，磁盘上的数据就全部丢失，因此 TiKV 需要为挂掉的节点补齐副本，这一过程需要消耗大量 CPU 和 IO 资源，并且需要恢复很长一段时间，因此我们最好将数据存储在 EBS 或者 S3 这样的由云厂商提供高可用服务的云盘中，由于云盘本身具备了高可用功能，出于节省成本考虑，TiKV 不需要维护三个数据副本，而只需使用一个。我们的 TiKV 现在已经支持了单副本，然而单副本的情况下，用户写入的数据不仅要写入一遍 kvdb，还要写入一遍 Raft 中的日志，这使得 TiKV 相比于其他单副本的数据库的性能下降了不少，本文讨论的就是在单副本情况下，不对 Raft 中的日志进行持久化，以此节省系统资源并且保证数据的正确性以及一致性。

优化思路

对于单个 Raft 节点，我们将其中的数据分为四部分，即：

1. Raft 中尚未 Commit 的日志数据
2. Raft 中已经 Commit 了的日志数据
3. Apply 到了 Raft 状态机中且尚未持久化的数据（truncate_index 之后的数据）
4. Apply 到了状态机中并持久化了的数据（truncate_index 之前的数据，这部分数据在论文中被称作 Snapshot）

其中 2 和 4 的数据集合起来为用户已经写入成功且不会丢失的数据。当用户只有一个副本时，任何写请求都会在 Raft Group 的内部立刻达成一致并且变为已提交（即 1 会立刻转换为 2）。因此我们其实不必把这部分日志落盘，因为只要用户的这部分数据 Apply 到了状态机中并且持久化，同时再推进 truncate_index（即将 3 立刻转变为 4），那么 TiKV 在重启的时候就可以依据 truncate_index 还原出来完整的数据。

调度

TiKV 需要在副本之间搬迁 region 以此实现负载与数据的均衡。在目前的情况下，TiKV 迁移 region 是通过先为该 region 增加一个 learner 副本，然后将 learner 提升为 follower，最后指定当前 leader 让位给新 follower。对于本方案来说，这意味着某些 region 会短暂地拥有两个副本，因此当 leader 监测到自己拥有两个副本（无论另一个副本是 learner 或者 follower），他必须持久化 Raft 中的日志数据，即退化为之前双写 raftdb 和 kvdb 的方案。

后续相关工作

单副本节点在宕机后需要在原来的 EBS 目录上启动新的 TiKV 节点（即新的 AWS Server），这点后续需要 DBaaS 配合，同时 PD 需要支持 TiKV 以相同的 store id 不同的地址启动，这或许需要 PD 为每个 TiKV 节点设置一个租约或者 TiKV 在 EBS 路径下定期写下租约文件，以防止多个 TiKV 节点抢占同一个 store id.

Background

TiKV stores three copies of all data through the Raft consistency protocol. Therefore, after any TiKV process abnormally exits, TiKV can continue to provide services, and as long as the abnormally exited node restarts, it can be added to the cluster normally. However, for VM users on the cloud, after any VM hangs, all data on the disk is lost, so TiKV needs to make up copies for the hung nodes. This process requires a lot of CPU and IO resources, and needs to cost a long time. It is best to store data in cloud disks such as EBS or S3 provided by cloud vendors with high availability services. Since the cloud disk itself has high availability functions, TiKV does not need to be maintained for cost savings. Three copies of data, and only one is needed. Our TiKV has supported single copy. However, in the case of single copy, the data written by the user must be written not only to kvdb, but also to raftdb, which makes the performance of TiKV worse compared to other single copy databases. This article discusses that in the case of a single copy, the logs in Raft do not need to be persisted, so as to save system resources and ensure data accuracy and consistency.

Optimization

We can divide data of a TiKV node into 4 parts as follow:

1. Appended but not committed.
2. Committed but not applied.
3. Applied but not persistent.
4. Persistent.

Among them, the data of 2 and 4 are collected as a complete set of data that the user has written successfully and will not be lost. When the user has only one replica, any write request will be immediately agreed within the Raft Group and become committed (that is, 1 will be converted to 2 immediately). Therefore, we don't actually need to persist this part of the log on disk, because as long as the user's part of the data is applied to the state machine and persists, and then push truncate_index (that is, 3 will immediately change to 4), then TiKV can be based on truncate_index when restart Restore the complete data.

Schedule

TiKV needs to moves regions between nodes to balance load and data. In the current situation, TiKV will add a learner replica , then promote the learner to follower, and finally assign the current leader to give way to this new follower. For this solution, this means that some regions will temporarily have two copies, so when the leader detects that he has two copies (whether the other copy is a learner or follower), he must persist the log entries of Raft , which degenerates into double-writing raftdb and kvdb.

Other Works