分布式事务: Percolator

概览

在这一章节，我们将实现 Percolator 提交协议。

执行模型介绍

在 TiDB 中，事务的执行过程会被缓存在 buffer 中，在提交时，才会通过 Percolator 提交协议将其完整地写入到分布的 TiKV 存储引擎中。这一调用的入口是 `store/tikv/txn.go` 中的 `tikvTxn.Commit` 函数。

在执行时，一个事务可能会遇到其他执行过程中的事务，此时需要通过 Lock Resolve 组件来查询所遇到的事务状态，并根据查询到的结果执行相应的措施。

Two Phase Commit

Percolator 提交协议的两阶段提交分为 Prewrite 和 Commit，其中 Prewrite 实际写入数据，Commit 让数据对外可见。其中，事务的成功以 Primary Key 为原子性标记，当 Prewrite 失败或是 Primary Key Commit 失败时需要进行垃圾清理，将写入的事务回滚。

一个事务中的 Key 可能会设计到不同的 Region，在对 Key 进行写操作时，需要将其发送到正确的 Region 上才能够处理，`GroupKeysByRegion` 函数根据 region cache 将 Key 按 Region 分成多个 batch，但是可能出现因缓存过期而导致对应的存储节点返回 Region Error，此时需要分割 batch 后重试。

TODO

为了让对于 Key 的操作能够执行，需要实现 `region_cache.go` 中的 `GroupKeysByRegion` 函数。

在执行过程中，会涉及到三类操作，分别是 Prewrite/Commit/Rollback(Cleanup)。这些操作会在同一个流程中被处理。你需要完成 Prewrite 过程中的 `buildPrewriteRequest` 函数，然后仿照 Prewrite 的 `handleSingleBatch` 函数完成 Commit 和 Rollback 的 `handleSingleBatch` 函数。

Lock Resolver

在 Prewrite 阶段，对于一个 Key 的操作会写入两条记录。

- Default CF 中存储了实际的 KV 数据。
- Lock CF 中存储了锁，包括 Key 和时间戳信息，会在 Commit 成功时清理。

Lock Resolver 的职责就是当一个事务在提交过程中遇到 Lock 时，需要如何应对。

当一个事务遇到 Lock 时，可能有几种情况。

- Lock 所属的事务还未提交这个 Key，Lock 尚未被清理；
- Lock 所属的事务遇到了不可恢复的错误，正在回滚中，尚未清理 Key；
- Lock 所属事务的节点发生了意外错误，例如节点 crash，这个 Lock 所属的节点已经不能够更新它。

在 Percolator 协议下，会通过查询 Lock 所属的 Primary Key 来判断事务的状态，但是当读取到一个未完成的事务（Primary Key 的 Lock 尚未被清理）时，我们所期望的，是等待提交中的事物至完成状态，并且清理如 crash 等异常留下的垃圾数据。此时会借助 ttl 来判断事务是否过期，遇到过期事务时则会主动 Rollback 它。

TODO

在 `lock_resolver.go` 中完成 `getTxnStatus` 和 `resolveLock` 函数，使得向外暴露的 `ResolveLocks` 函数能够正常运行。

除了在事务的提交过程中，事务对数据进行读取的时候也可能遇到 Lock，此时也会触发 `ResolveLocks` 函数，完成 `snapshot.go` 中的 `tikvSnapshot.get` 函数，让读请求能够正常运行。

Failpoint

Failpoint 是一种通过内置注入错误测试常规代码路径的手段。Golang 中的 failpoint 使用代码生成实现，因此在使用了 failpoint 的测试中，需要先打开 failpoint。

打开 failpoint，原始代码会被暂时隐藏到 `{filename}__failpoint_stash__`，请勿删除。

```
make failpoint-enable
```

关闭 failpoint 会将 `{filename}__failpoint_stash__` 中的代码恢复到原文件中，并删除暂存用的 stash 文件。

```
make failpoint-disable
```

因为 failpoint 开关状态不同时，使用了 failpoint 的 `.go` 文件代码会有不同，请在关闭 failpoint 的状态下进行代码的提交。

测试

运行 `make lab3`，通过所有测试用例。

可以使用下面的命令测试指定的单个或多个用例。

```
go test {package path} -check.f ^{regex}$
# example
go test -timeout 5s ./store/tikv -check.f ^TestFailAfterPrimary$
```