KV实现报告

June 25, 2023

罗境佳
jingjialuo26@gmail.com

1 功能介绍

DemoKV 是一个使用 Rust 实现的分布式的 key-value 存储系统，它支持以下功能：

- 支持增删改查接口，通过 `PUT`, `GET`, `DEL` 等语句修改 key-value，另支持简单的表达式如 `PUT A (A+1)` 或 `PUT B (C-1)` 等。
- 完成 ACID 语义，提供 repeatable read 的隔离级别。
- 支持磁盘，节点宕机从 snapshot 和 log 恢复数据。
- 客户端和服务端通过 RPC 沟通，支持多个客户端并发操作。
- 使用 MVCC 并发控制协议。
- 分布式架构，支持键值对的多副本存储。

2 实现细节

2.1 内存结构

使用 lock-free 的 B-tree 存储键值对。其中键为 `(key, timestamp)`，值为 `Option<value>`。使用 B 树是为了更快地利用使用时间戳定位到具体的版本。其中，`(key, usize::MAX)` 作为该 key 的 lock 存在。

2.2 MVCC 实现

模仿 percolator 实现基于时间戳的 MVCC 协议。在客户端保存 read set 和 write set，`get`, `put`, `commit` 的伪代码如下：

```
struct TxnContext {
  start_ts: usize,
  read_set: HashMap<Key, Value>,
  write_set: HashMap<Key, Value>,
}

fn get(key: Key) -> Option<Value> {
  let value = server.read_at(key, start_ts);
  read_set.update(key, value);
}
fn put(key: Key, value: Value) {
  read_set.update(key, value);
```

```
  write_set.update(key, value);
}
fn commit() {
  for (key, value) in write_set {
    server.lock(key)?;
    server.validate(key, start_ts)?;
  }
  let commit_ts = new_timestamp();
  for (key, value) in write_set {
    server.update(key, value, commit_ts);
    server.unlock(key);
  }
}
```

该过程中，所有读视作发生在 `start_ts`，写视作发生在 `commit_ts`。Validation 过程是为了避免 WAW 冲突。

2.3 日志

采用一个 write-ahead log 将操作持久化，在 log 刷盘后回复客户端。

2.4 分布式实现

采用主从架构，使用 etcd 存储 cluster membership 信息，leader 将收到的更新发给 follower。使用 etcd 提供的功能进行 leader 的选举，leader 在 etcd 中维护 lease 以避免无 leader 造成的 liveness 问题等。

对于新加入集群的节点，leader 会将自己的快照发给他，以便新的 follower 可以快速跟上。

leader	follower
读写	只读

3 性能测评

测评方式：

```
cargo build --release
python3 bench.py --all
```

将测例 4 和 5 运行 60 秒，得到 A 和 B 的值结果如下：

	A	B
testcase 4	101500	101500

testcase 5	297263	297264

4 正确性测评

具体测试见 tests 目录底下的 single.rs 和 distributed.rs 两个文件，包括所有作业文档中所提到的测试。

测评方式：

```
etcd --data-dir data/
cargo test
killall etcd
```

4.1 测例结果

4.1.1 testcase 1

`A=4, B=5, A=5, B=None, B=5.`

4.1.2 testcase 2

Server 进程重启后可读到：

`A=5, B=5.`

4.1.3 testcase 3

读到的值均为 2。

4.1.4 testcase 4

`A=B.`

4.1.5 testcase 5

`A1=A2, B1=B2.`

4.1.6 testcase 6

从所有节点处均可读到 `A=2, B=1`，增加节点后从所有节点（包括新节点）可读到 `A=2, B=1`。再更改 A, B 的值后从所有节点读到 `A=4, B=2`，减少一个节点后从剩下节点读到 `A=4, B=2`。

5 改进方案

本项目仅仅是一个 prototype，还有着许多的改进空间：

- 陈旧版本数据的 garbage collection。
- 支持数据 sharding。
- 节点宕机自动检测（目前仅 leader 节点支持 lease）。
- 支持更丰富的表达式。
- 目前的多副本还有许多的一致性问题，也许使用 Raft 等共识协议是比较完善的解决方法。