

KV数据库大作业报告

2021210923 顾掀宇

[GitHub - xianyuggg/stupid-kv](#)

总览

环境：Golang1.15, macOS 12.4

特点 & 支持功能

1. 使用Go语言作为整个简易系统实现
2. 支持增删查改(Put/Get/Inc/Dec/Del五种接口)
3. 支持磁盘
 - 目前的实现为将底层数据写入json，因实现有限未能实现增量写入
4. 支持并发
 - 使用锁和并发数据结构保证并发访问，可以支持100+事务启动
5. 支持ACID
 - 分别实现了标准的2PL（Strict 2PL）和简易版的MV2PL
 - 没有支持死锁检测
 - 通过undo log实现回滚
6. 支持MVCC
 - 简易版MV2PL，参考了CMU课程以及课题组论文实现的标准
 - An Empirical Evaluation of In-Memory Multi-Version Concurrency Control
 - <https://15721.courses.cs.cmu.edu/spring2019/slides/03-mvcc1.pdf>
 - 由于Go语言的限制，在当前实现下很难做到记录级别的锁，锁的粒度实现较粗（每个key有各自的版本链，对各个版本链单独加写锁），没有体现性能优势
 - 因时间有限未能实现GC，因此版本空间是无限递增的

实现细节

总共分为两层，存储层(kv)和事务层(txn)，在2PL实现中为了尽可能地减少耦合，整个存储层对于事务是完全不感知的

存储层

底层存储使用了Golang标准库提供的 `sync.Map` 的支持并发的结构，对应的value是ValueSlot，为了支持后续mvcc扩展写成了数组的形式，实际上只需要一个value即可。

存储层支持了Put/Get等接口，方便后续上层调用。

存储层提供了Flush接口，把整个Map转化成json存储持久化到文件

```
// Storage Layer
type ValueSlot struct {
    values []base.ValueT
    tids   []base.Tid    // TODO: mvcc implementation
}

type KvManager struct {
    kv          *sync.Map
    flushGuard *sync.Mutex
}
```

事务层 (Strict 2PL)

事务层做到了和底层之间的完全解耦，TxnManager字段参考下面的代码，其中lockMap是Key -> sync.RWMutex的映射（因为Golang本身的map不支持并发，所以使用sync.Map），而后每个key使用一个读写锁保护起来，Get的时候加读锁并写入readSet，Put/Del等操作加写锁并写入writeSet。writeSet和readSet是tid -> sync.Map的映射，其中作为value的sync.Map存储了一个tid所写入/访问的key

支持了如下的接口

```

// Transaction Layer
type TxnManager struct {
    curTid base.Tid

    writeSet *sync.Map
    //writeSet map[base.Tid]map[base.KeyT]int
    readSet *sync.Map
    //readSet map[base.Tid]map[base.KeyT]int
    lockMap *sync.Map
    //lockMap map[base.KeyT]*sync.RWMutex

    guard *sync.Mutex
}

func (m *Manager) acquireWriteLock(key base.KeyT, tid base.Tid)
func (m *Manager) releaseWriteLock(key base.KeyT, tid base.Tid)
func (m *Manager) releaseReadLock(key base.KeyT, tid base.Tid)
func (m *Manager) acquireReadLock(key base.KeyT, tid base.Tid)
func (m *Manager) AllocateNewTid() base.Tid

func (m *Manager) BeginTxn() base.Tid
func (m *Manager) Put(key base.KeyT, value base.ValueT, tid base.Tid) error
// ...
func (m *Manager) CommitTxn(tid base.Tid) error
func (m *Manager) AbortTxn(tid base.Tid) error

```

Categories of Two Phase Locking (Strict, Rigorous & Conservative) - GeeksforGeeks

在Strict 2PL里，整个事务的执行阶段都要不断上锁，并写入readSet和writeSet，在事务commit或者abort阶段访问writeSet，并通过全局的lockMap解锁。

好处在于2PL避免了级联abort，坏处是会出现死锁，由于时间原因没有实现死锁相关的处理

Abort实现

通过记录undolog实现abort，在事务执行阶段，每当有修改操作发生时，记录一下操作的顺序，然后当事务abort时，通过执行一遍相反的操作还原即可。

如下段代码所示，每个UndoLogger有一个tid -> opList的map，记录每个Tid的操作

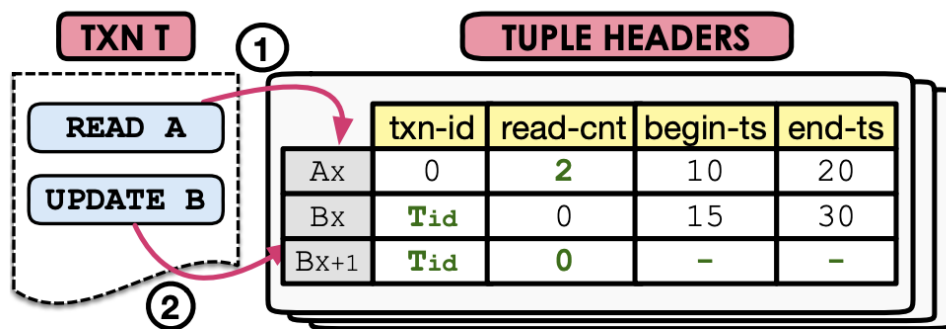
```

type TxnOp struct {
    op    OpType
    key   base.KeyT
    value base.ValueT
}

type UndoLogger struct {
    txnOps map[base.Tid][]TxnOp
    guard  sync.Mutex
}

```

MV2PL实现



(c) Two-phase Locking

标准的MV2PL里，记录多加4个字段，其中使用read-cnt和txn-id一同实现了共享锁和排它锁

这里应该都是原子变量，由于时间原因我只实现了整个版本链的读写锁，也就是继续沿用前一阶段2PL的做法，维护一个writeSet，可以避免写写冲突，此时需要对于存储层做对应的修改，使用列表的方式维护版本链，记录tidBegin和tidEnd

```

// storage layer
type ValueSlot struct {
    values    []base.ValueT
    tidsBegin []base.Tid
    tidsEnd   []base.Tid // support mvcc
}

type Manager struct {
    //kv map[base.KeyT]ValueSlot
    kv          *sync.Map
    slotGuard   map[base.KeyT]*sync.RWMutex
    mapGuard    *sync.Mutex // used for guarding slotguard (concurrent map modify)
    flushGuard  *sync.Mutex
}

```

以下是各个操作的实现逻辑

- Put/Inc/Dec(key, value, tid)
 - 对于key进行加写锁（避免多个事务同时对于一个变量进行修改）
 - 更改版本链最后的一个记录end-ts为当前tid
 - 如果发现tid < end-ts可以直接报错，之后abort事务（我不太确定这条，姑且列为TODO，之后我再理解一下）
 - 向版本链后追加一个新记录(key, value, tid, INF)
- Get(key, tid, activeTids []base.Tid)
 - 从版本链最后一个版本进行查找
 - 如果满足begin-ts <= tid <= end-ts
 - 如果tid != begin-ts && activeTids []base.Tid，说明读到了一个还未提交的数据，需要返回一个VALUE_NOT_COMMIT，等待事务提交之后再读
 - 其余情况下可以直接返回当前的值

TODO：这种方法实现了RepeatableRead，之后等有时间再验证一下

- 其中“读到未提交数据的处理”处理方式如下
 - 使用一个全局的sync.Map记录当前commit的所有事务

返回VALUE_NOT_COMMIT同时返回一个Tid，然后去不断查找该记录是否已经commit

```
[INFO] 2022/06/25 17:22:47.010173 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:30: Transaction manager starts to init
[INFO] 2022/06/25 17:22:47.010467 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 217 start
[INFO] 2022/06/25 17:22:47.010479 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/undo.go:34: undo logger starts to init
[INFO] 2022/06/25 17:22:47.010681 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 218 start
[INFO] 2022/06/25 17:22:47.010693 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:145: tid 218: read uncommitted value and wait 217
[INFO] 2022/06/25 17:22:47.010935 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 219 start
[INFO] 2022/06/25 17:22:47.011116 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 220 start
[INFO] 2022/06/25 17:22:47.011370 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 221 start
[INFO] 2022/06/25 17:22:47.011644 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 222 start
[INFO] 2022/06/25 17:22:47.015779 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 217 commit
[INFO] 2022/06/25 17:22:47.024514 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:148: Txn 218 find 217 find committed in commitTidMap
[INFO] 2022/06/25 17:22:47.025162 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 223 start
[INFO] 2022/06/25 17:22:47.027056 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 221 commit
[INFO] 2022/06/25 17:22:47.027218 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 224 start
[INFO] 2022/06/25 17:22:47.029052 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 219 commit
[INFO] 2022/06/25 17:22:47.029211 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 225 start
[INFO] 2022/06/25 17:22:47.031110 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 222 commit
[INFO] 2022/06/25 17:22:47.031281 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 226 start
[INFO] 2022/06/25 17:22:47.032836 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 220 commit
[INFO] 2022/06/25 17:22:47.033009 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 227 start
[INFO] 2022/06/25 17:22:47.034383 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 218 commit
```

可以通过日志看到事务可以成功提交

- 等到commit和abort的时候释放写锁

分析

- MVCC本质Writers don't block readers.Readers don't block writers.
- 写写冲突：通过设置写锁来实现，这里实现了简易的key级别的锁
 - **TODO：**我暂时未理解事务中写写覆盖的情况锁,暂时通过写锁避免两事务同时修改
- 读写冲突：可以看到写事务加锁不会影响读事务读数据
- 读读冲突：没有这种情况
- MVCC垃圾回收：
 - 有最简单的基于时间戳的算法，通过一个后台vacuum线程遍历版本链，删除小于当前活跃事务的版本，可以作为Future Work
 - Interval-based method，在复杂的系统里面会用到，实现比较复杂
- Read Uncommitted在实际测试中出现了一种边缘情况
 - 当前执行事务Tid3读取A时发现版本A对应的Tid1没有提交，当前Get失败，并查询commitMap判断Tid1是否提交
 - 当Tid1提交之后，Tid3再次查询A，但是这时发现存在Tid2在这个过程中写入了数据，并且Tid2没有提交
 - 这时Tid3再次查询到了uncommitted data，所以需要再次返回等待Tid2成功写数据
 - 这一过程没有违背任何规则，也确保最终读到了来自Tid2的committed data

样例测试

```
func TestCase1() {
    kvManager := kv.GetManagerInstance()
    kvManager.Put( key: "A", value: 3, tid: 1)
    kvManager.Put( key: "B", value: 4, tid: 1)
    kvManager.Inc( key: "A", tid: 1)
    kvManager.Inc( key: "B", tid: 1)
    print( args...: "A ")
    println(kvManager.Get( key: "A", tid: 1, []base.Tid{}))
    print( args...: "B ")
    println(kvManager.Get( key: "B", tid: 1, []base.Tid{}))
    kvManager.Del( key: "A", tid: 1)
    kvManager.Del( key: "B", tid: 1)

    kvManager.Put( key: "A", value: 5, tid: 1)
    print( args...: "A ")
    println(kvManager.Get( key: "A", tid: 1, []base.Tid{}))
    print( args...: "B ")
    kvManager.Put( key: "B", value: 5, tid: 1)
    println(kvManager.Get( key: "B", tid: 1, []base.Tid{}))

    kvManager.Flush()
}

func TestCase2() {
    kvManager := kv.GetManagerInstance()
    println(kvManager.Get( key: "A", tid: 1, []base.Tid{}))
    println(kvManager.Get( key: "B", tid: 1, []base.Tid{}))
}

func TestCase3() {
    kvManager := kv.GetManagerInstance()
    println(kvManager.Get( key: "A", tid: 1, []base.Tid{}))
    println(kvManager.Get( key: "B", tid: 1, []base.Tid{}))
}

package main

func main() {
    println( args...: "VALUE_NOT_VALID : ", 0xdddddd)
    println( args...: "VALUE_NOT_COMMIT: ", 0xeeeeee)
    println( args...: "VALUE_NOT_FOUND : ", 0xfffff)

    TestCase1()
    //TestCase2()
    //TestCase3()

    //TestCase52()

    //TestCase5()
}
```

```
Run: go build stupid-kv x
/usr/local/opt/go/libexec/bin/go build -o /private/var/folders/tx/8Lk15dcj4jq3mtl3c24pv92h0000gn/T/GoLand/___go_build_stupid_kv stupid-kv #gosetup
/private/var/folders/tx/8Lk15dcj4jq3mtl3c24pv92h0000gn/T/GoLand/___go_build_stupid_kv
VALUE_NOT_VALID : 14540253
VALUE_NOT_COMMIT: 15658734
VALUE_NOT_FOUND : 16777215
[INFO] 2022/06/25 20:09:22.559847 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/kv/storage.go:29: KV manager starts to init
A 4 1
B 5 1
A 5 1
B 5 1
Process finished with the exit code 0
```

测例1~3已经写到testcase.go文件里，可以按预期执行（截图略）

测例4，使用2pl执行，可以看到30s内执行10w+条事务，数据如下

```
{"A":"376691 113004","B":"376691 113004"}
```

The screenshot shows an IDE with the following components:

- Project Explorer:** Lists files like `util.go`, `kv`, `flush.go`, `storage.go`, `logutil`, `log.go`, `logger.go`, `variable.go`, `txn`, `transaction.go`, `undo.go`, `.gitignore`, `DATA.json`, `go.mod`, `main.go`, `README.md`, `STATE.txt`, and `testcase.go`.
- Editor:** Displays `testcase.go` with the following code:

```
91
92     wg.Done()
93 }
94
95 func TestCase4() {
96     kvManager := kv.GetManagerInstance()
97
98     println(kvManager.Get( key: "A", tid: 200000))
99     println(kvManager.Get( key: "B", tid: 200000))
100    kvManager.Put( key: "A", value: 1, tid: 0)
101    kvManager.Put( key: "B", value: 1, tid: 0)
102
103    wg := sync.WaitGroup{}
104    wg.Add( delta: 2000000)
105    go TestCase41(&wg)
106    go TestCase41(&wg)
107    go TestCase41(&wg)
108    time.Sleep(30 * time.Second)
109    kvManager.Flush()
110    //wg.Wait()
111 }
```
- Run Output:** Shows the execution of `go build stupid-kv` and subsequent log messages:

```
Run: go build stupid-kv x
[INFO] 2022/06/25 20:18:21.374616 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 112991 commit
[INFO] 2022/06/25 20:18:21.374641 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 112998 start
[INFO] 2022/06/25 20:18:21.374860 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 112992 commit
[INFO] 2022/06/25 20:18:21.374875 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 112999 start
[INFO] 2022/06/25 20:18:21.375120 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:254: txn 112997 abort
[INFO] 2022/06/25 20:18:21.375141 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113000 start
[INFO] 2022/06/25 20:18:21.375595 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113000 commit
[INFO] 2022/06/25 20:18:21.375623 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113001 start
[INFO] 2022/06/25 20:18:21.375798 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 112998 commit
[INFO] 2022/06/25 20:18:21.375813 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113002 start
[INFO] 2022/06/25 20:18:21.376164 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:254: txn 113002 abort
[INFO] 2022/06/25 20:18:21.376224 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113003 start
[INFO] 2022/06/25 20:18:21.377166 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:254: txn 112999 abort
[INFO] 2022/06/25 20:18:21.377244 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113004 start
[INFO] 2022/06/25 20:18:21.377984 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113001 commit
[INFO] 2022/06/25 20:18:21.378036 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113005 start

Process finished with the exit code 0
```

测例5, 30s内执行10w+条事务, 数据如下

{ "A": "187230 113724", "B": "187230 113724", "C": "57556 113722" }


```
stupid-kv > main.go
GO BUILD STUPID-KV
config.go x testcase.go x README.md x proc.go x chan.go x main.go x storage.go x
Project
  base
    config.go
    const.go
    util.go
  kv
    flush.go
    storage.go
  logutil
    log.go
    logger.go
    variable.go
  txn
    transaction.go
    undo.go
  .gitignore
  DATA.json
  go.mod
  main.go
  README.md
  STATE.txt
  testcase.go
Pull Requests
Database
Notifications
Run: go build stupid-kv x
[INFO] 2022/06/25 20:23:04.874717 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113712 commit
[INFO] 2022/06/25 20:23:04.874730 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113719 start
[INFO] 2022/06/25 20:23:04.875002 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113713 commit
[INFO] 2022/06/25 20:23:04.875032 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113720 start
[INFO] 2022/06/25 20:23:04.875266 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113714 commit
[INFO] 2022/06/25 20:23:04.875285 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113721 start
[INFO] 2022/06/25 20:23:04.875497 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113716 commit
[INFO] 2022/06/25 20:23:04.875530 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113722 start
[INFO] 2022/06/25 20:23:04.875782 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113717 commit
[INFO] 2022/06/25 20:23:04.875795 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113723 start
[INFO] 2022/06/25 20:23:04.876068 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113718 commit
[INFO] 2022/06/25 20:23:04.876089 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:96: txn 113724 start
[INFO] 2022/06/25 20:23:04.876681 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:204: txn 113719 commit
[INFO] 2022/06/25 20:23:04.876705 /Users/xianyu
Process finished with the exit code 0
```

MVCC测例5测试

```
[INFO] 2022/06/25 20:39:36.164578 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/kv/storage.go:29: KV manager starts to init
[INFO] 2022/06/25 20:39:36.165636 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:30: Transaction manager starts to init
[INFO] 2022/06/25 20:39:36.166053 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 537 start
[INFO] 2022/06/25 20:39:36.166070 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/undo.go:34: undo logger starts to init
[INFO] 2022/06/25 20:39:36.167382 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 538 start
[INFO] 2022/06/25 20:39:36.167652 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 542 start
[INFO] 2022/06/25 20:39:36.167718 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:145: tid 542: read uncommitted value and wait 538
[INFO] 2022/06/25 20:39:36.168315 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 540 start
[INFO] 2022/06/25 20:39:36.169024 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 539 start
[INFO] 2022/06/25 20:39:36.169440 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 541 start
[INFO] 2022/06/25 20:39:36.182676 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 537 commit
[INFO] 2022/06/25 20:39:36.182904 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 543 start
[INFO] 2022/06/25 20:39:36.199816 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 538 commit
[INFO] 2022/06/25 20:39:36.199902 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:148: txn 542 find 538 committed in commitTidMap
[INFO] 2022/06/25 20:39:36.199923 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:150: tid 542: read uncommitted value and wait 540
[INFO] 2022/06/25 20:39:36.201151 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 544 start
[INFO] 2022/06/25 20:39:36.217463 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 541 commit
[INFO] 2022/06/25 20:39:36.218568 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 545 start
[INFO] 2022/06/25 20:39:36.233824 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 540 commit
[INFO] 2022/06/25 20:39:36.233929 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:148: txn 542 find 540 committed in commitTidMap
[INFO] 2022/06/25 20:39:36.233952 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:150: tid 542: read uncommitted value and wait 539
[INFO] 2022/06/25 20:39:36.235021 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 546 start
[INFO] 2022/06/25 20:39:36.246721 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 543 commit
[INFO] 2022/06/25 20:39:36.247243 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:67: txn 547 start
[INFO] 2022/06/25 20:39:36.255382 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:86: txn 539 commit
[INFO] 2022/06/25 20:39:36.255420 /Users/xianyu/Desktop/SourceCode/Database/stupid-kv/txn/transaction.go:148: txn 542 find 539 committed in commitTidMap
```

MVCC可以通过测例5简单测试正确性如上，性能因为GC和持久化问题和2pl相比存在差距

存在的问题 & 改进方案

因为时间有限，一些部分只能用粗糙的方法来实现，也会造成一些性能问题，列举如下

- 存储层使用了Map + List的方式存储key和版本链，Golang的List实际上是线程不安全的，在实现MVCC的时候修改List的时候需要对整个List加锁，这样会低效一些
 - TODO：调研并发list的实现，可以考虑提前先分配一个空间，等到需要的时候扩容，让写操作/读操作互不影响
- 持久化的过程比较低效，在一次commit之后将整个map写入的json文件里
 - TODO：调研增量的方法，mmap应该也可以（但是CMU的某篇文章抨击了mmap）
 - 在容忍数据丢失的情况下可以定期刷盘
- Undolog还没有做持久化处理，目前undolog可以用来处理abort
- MV2PL目前实现的锁的级别粒度比较粗
- MVCC过程中遇到了一些事务本身的问题，在上一节分析里面有写
 - 如果发现tid < end-ts可以直接报错，之后直接abort事务是否合理

- 如果tid != begin-ts && activeTids []base.Tid, 说明读到了一个还未提交的数据, 需要返回一个VALUE_NOT_COMMIT, 等待事务提交之后重新读, 可能会出现读多次的情况
- 需要通过一些测例证明实现了RepeatableRead/Snapshot Isolation