lab 1

map-reduce 流程

① split： 把 input files 分割为 m 个子片段

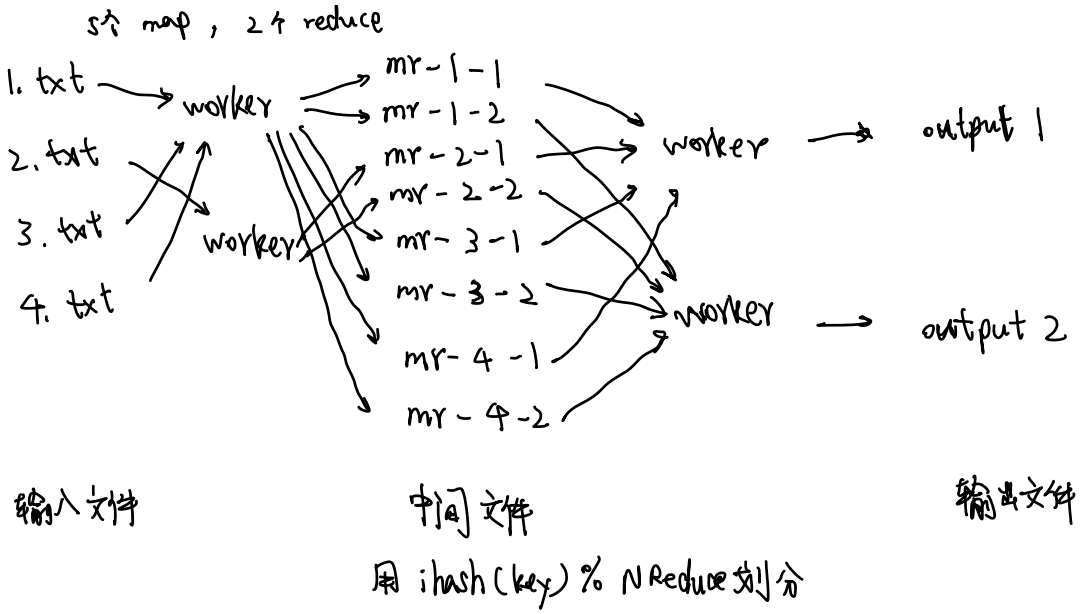
② fork： 创建一个 master 和多个 worker，整个计算过程会执行 m 个 map task 和 r 个 reduce task，master 用于调度 workers

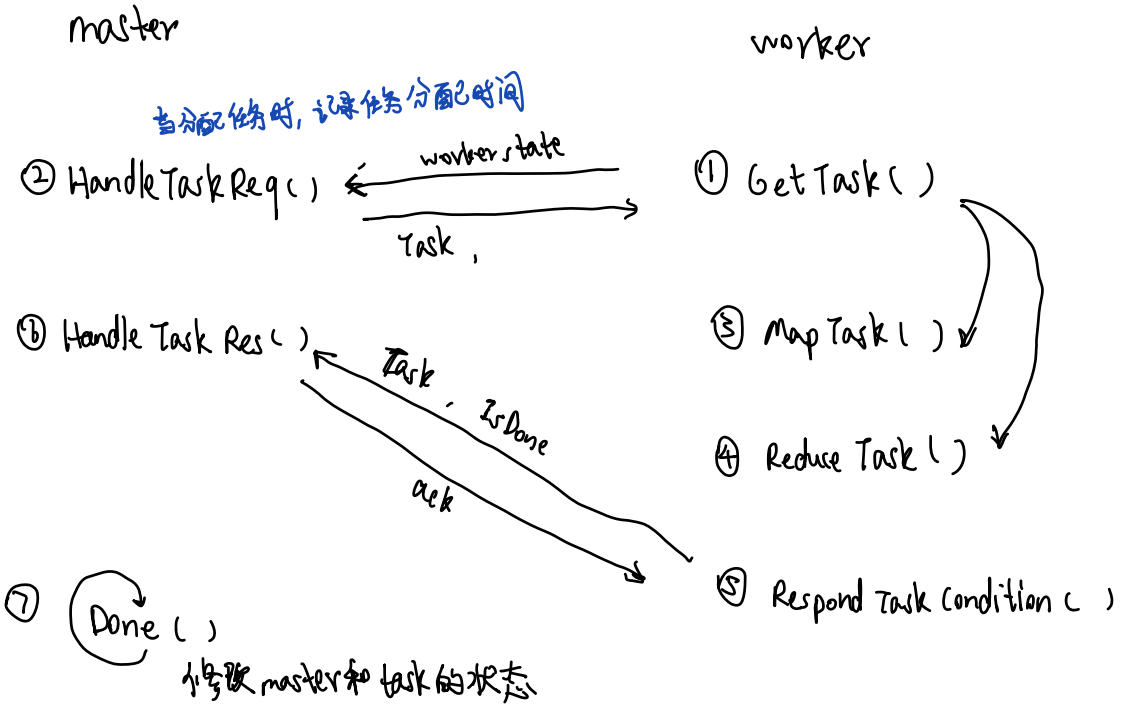
③ map：当 worker 被分配到 map task，它会去读指定的片段，调用 mapf() 处理原生数据，生成 k-v 中间数据，通过 hash 函数把中间数据归类到 r 个文件，写入本地 disk

④ reduce：当 worker 被分配到 reduce 任务，它通过 RPC 读取中间文件。由于这些中间文件是无序的（key 与 key 之间未排序），当一个 reduce task 读完它对应的多个中间文件后，需要统一对 key 排序。排序后对每个 key 调用 reducef() 进行聚合，然后把结果输出到对应分区的 output file。

错误处理（lab 中的容错机制比论文简单很多）

① 当 worker 失效（master 发现任务超时），master 把 task 状态改回为 ready，重新分配给可用 worker 即可





1. task 超时 → t.rdy
2. map task 全部完成 → m.MAP_FIN
3. reduce task 全部完成 → m.RED_FIN

master状态：INIT, MAP_FIN, REDUCE_FIN

Task状态：TaskRdy, TaskFIN, TaskRunning

lab 2

① checkElectionLoop() 检查是否需要选举

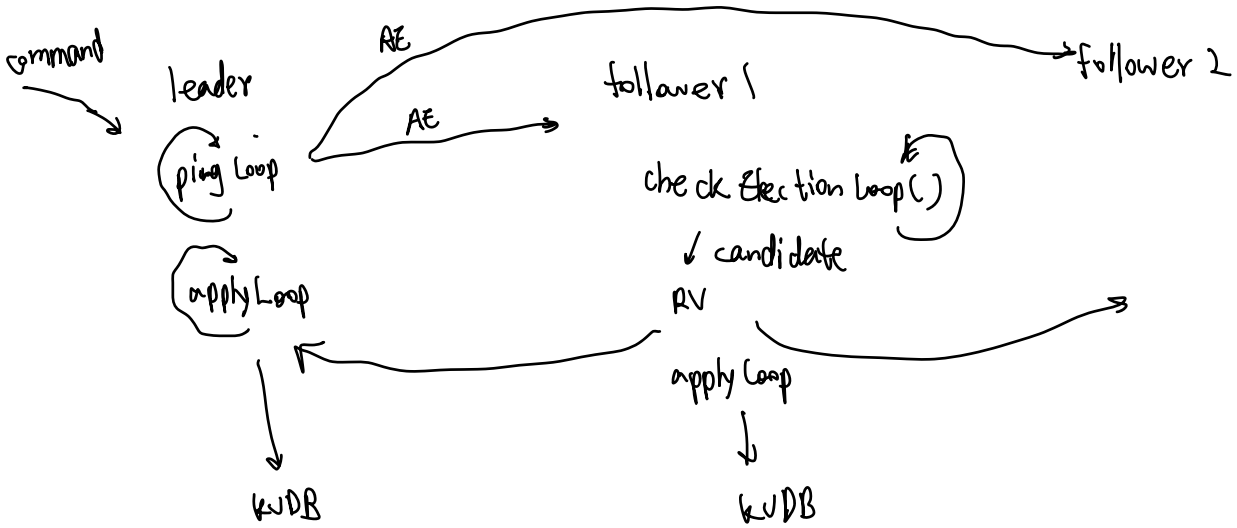
当server不为leader时，不断循环检查heartbeat是否超时，一旦超过随机超时时间就调用sendRequestVote发起选举

② pingLoop() leader定期发送AE

一旦server成为leader就会调用pingLoop()直到发现自己不再是leader

③ applyLoop()

每个server都通过这个循环检查是否有新的committed日志，把新日志通过chan ApplyMsg应用到状态机（kvDB）



持久化

对于Raft节点，需要持久的状态分别是currentTerm，voteFor，log[]

因此，每当3个变量发生变更就调用一次persist()

触发时机可以归为3个：

① 投票时 ③ 升term

② AE追加新日志

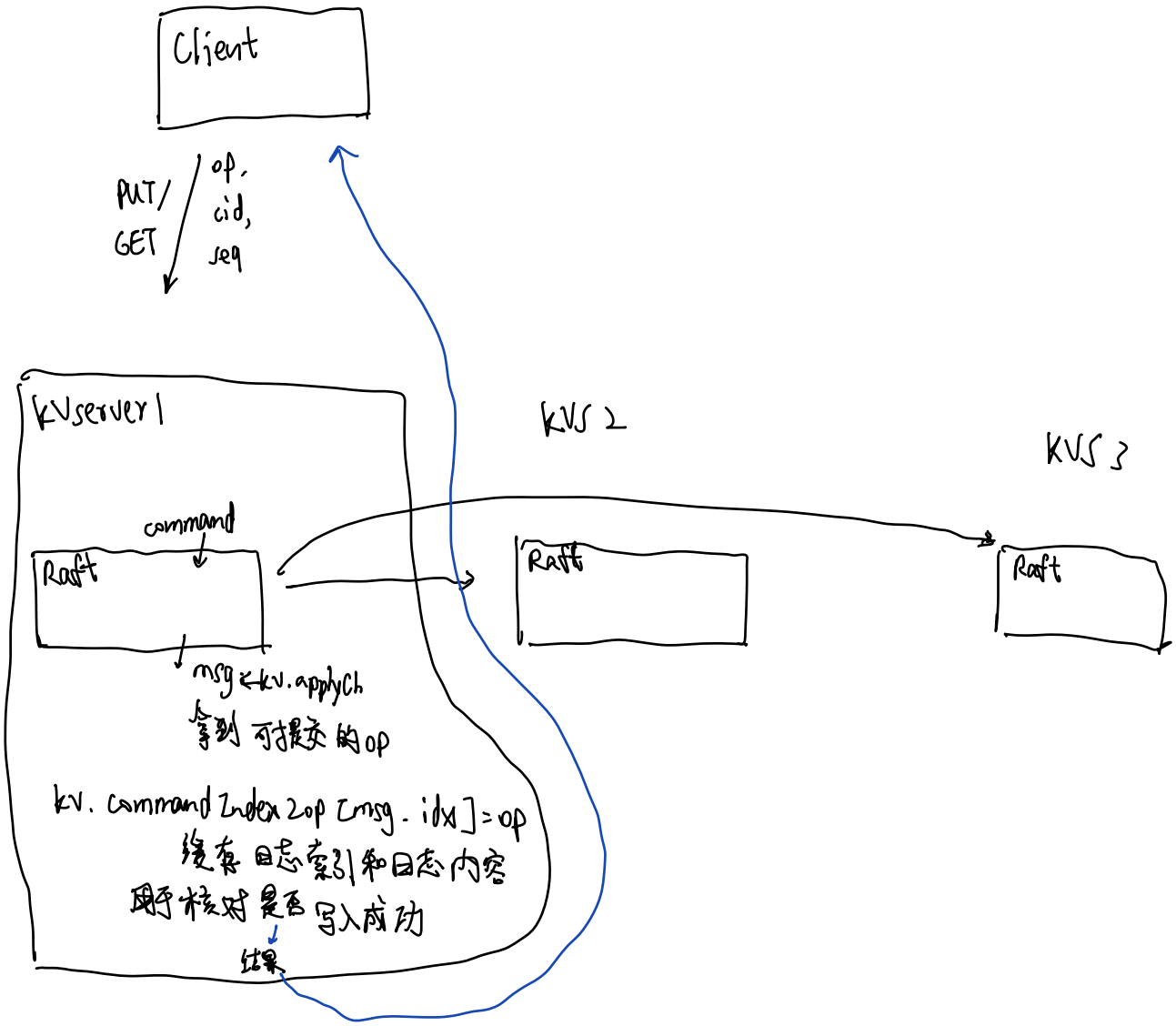
应该重置 timeout 的3个时机

① 收到当前 leader 的 AE

② server 开启一次 选举

③ 向一个 peer 投票

lab 3



日志压缩与快照（注意lab的持久化都由Raft模块完成）

所有KVServer在收到来自Raft的command后，会检查raft的log[]长度。超过阈值则生成快照[包括KVDB, cid2seq, lastIncludedIndex, lastIncludedTerm, currentTerm, VoteFor, log[]，其中后5个属于RaftState]

这样我们就可以清空原来的log[]，新log[]的真正index为i+lastIncludedIndex

处理流程

① 每个 Raft 节点检查自己的日志大小，当超过阈值，需要将 kvDB 和 cid2seq 序列化并发送到 Raft 层。

② Raft 收到上述调用会生成快照（上述两部分 + RaftState），落盘。这两步每个节点独立完成，快照都是写入状态机的 committed 日志，不违反一致性。

③ 在 lab 中，leader 一旦发现 follower 打算接收的下一条日志 nextIndex[i] 比 log[] 中的日志都小，说明 follower 的快照落后太多，把准备发送的 AE 改为 InstallSnapshot

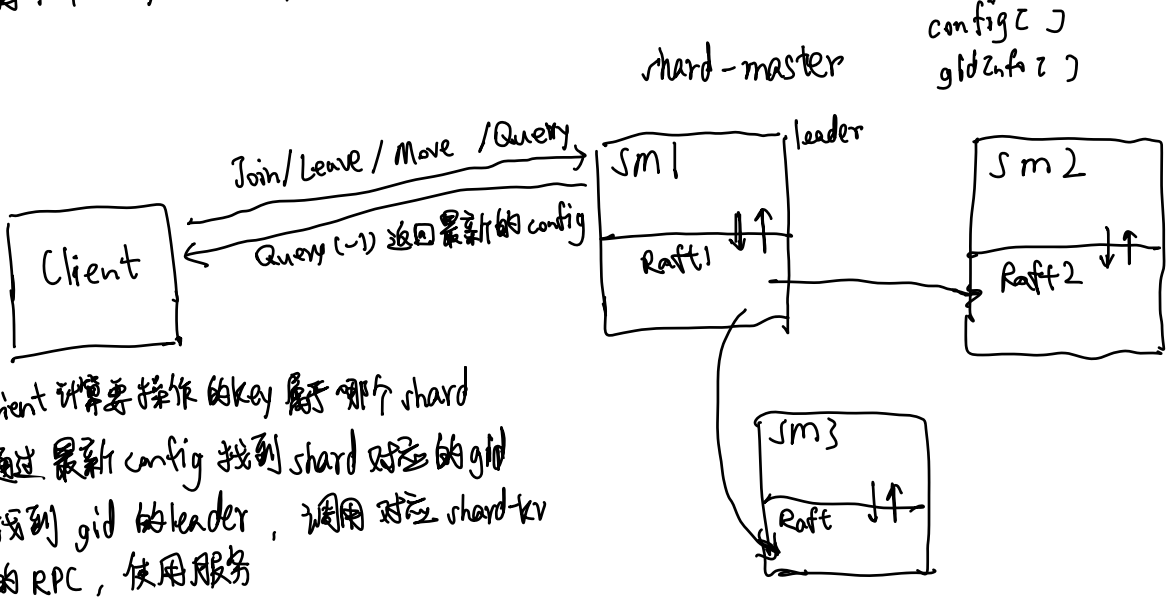
④ follower 用 leader 的 snapshot 覆盖自己的 snapshot 并同步 lastIncludedIndex 和 lastIncludedTerm 使数据与 leader 一致

⑤ 通知 kvserver 拉取新的 kvDB 和 cid2seq，使状态机包与 leader 一致

③.④.⑤ 为 leader 与 follower 交互实现

lab 4

有10个分片 shard 1 - shard 10



① Client 计算要操作的key属于哪个shard

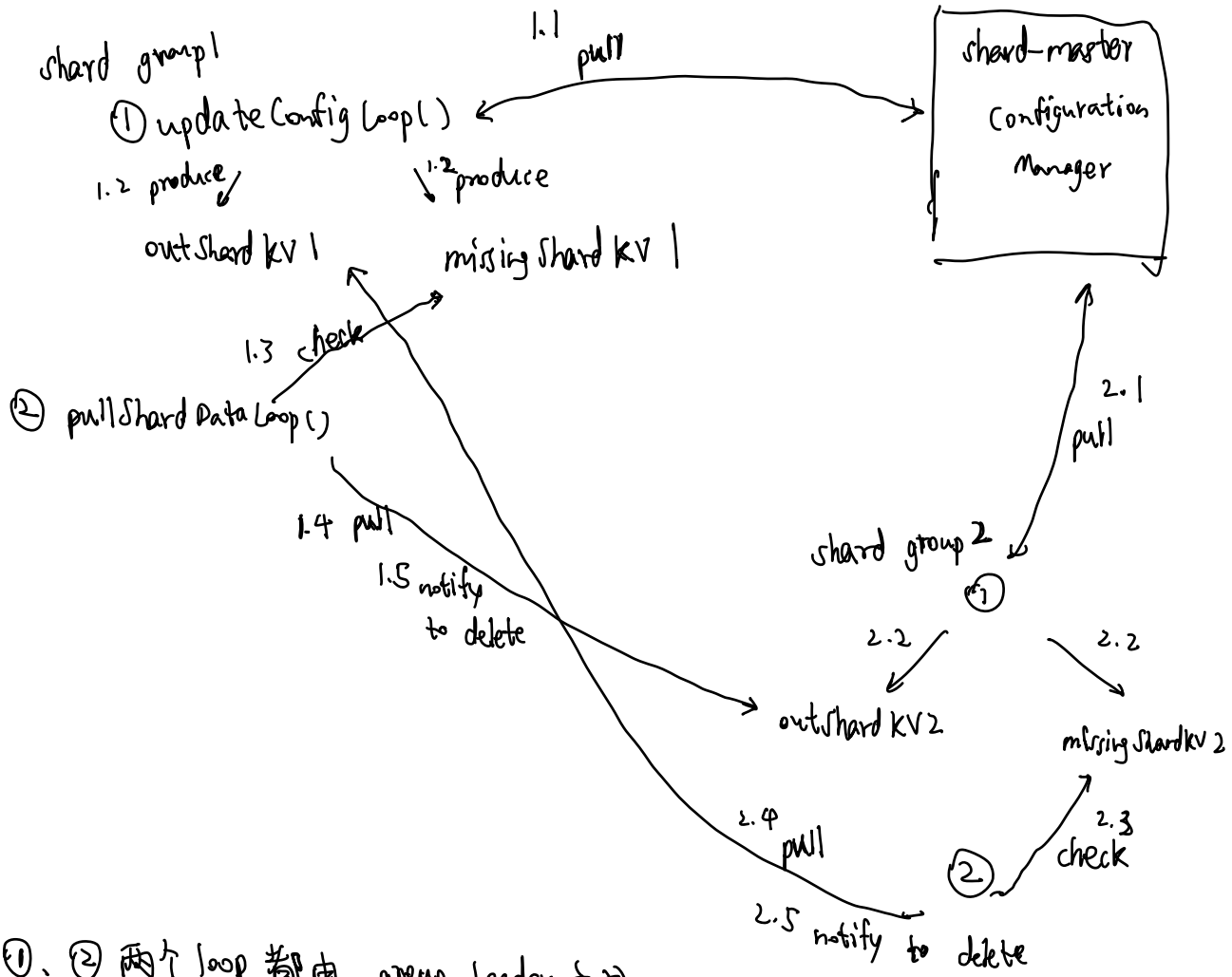
② 通过最新config找到shard对应的gid

③ 找到gid的leader，调用对应shard-kv的RPC，使用服务

GET/Append/PUT

shard-kv

① 通过 updateConfigLoop() 不断从 sm 拉取新配置，根据 shard 变化，记录要迁出的数据与迁入数据：outShardKV，missingShardKV



①、② 两个 loop 都由 group leader 完成，通过 Raft 把 config 和迁入数据以日志方式同步到 group 中的 follower