CSE5020 Assignment 4

本报告记录了作业 4 中 Raft 共识算法的实现过程。本次作业的目标是构建一个具备容错能力的分布式系统，能够实现领导者选举（Leader Election）、日志复制（Log Replication）以及持久化状态管理（Persistent State Management）。该实现能够处理网络故障、分区以及不可靠通信信道等情况。

1 实现逻辑

1.1 Raft 状态与持久化

`Raft` 结构体维护了 Raft 论文中定义的状态。

- **持久化状态**：`currentTerm`（当前任期）、`votedFor`（投给谁）和 `log`（日志）通过 `gob` 编码持久化到稳定存储中。每当这些变量发生变化时，都会调用 `persist()` 函数以确保在崩溃后能够恢复。
- **易失性状态**：`commitIndex`、`lastApplied`、`nextIndex` 和 `matchIndex` 维护在内存中，用于跟踪日志复制的进度。

1.2 领导者选举 (Part 1)

领导者选举由一个随机化的选举定时器驱动。

- **定时器**：选举超时时间在 150ms 到 300ms 之间随机化，以防止选票瓜分（Split Vote）。
- **RequestVote**：当定时器超时，服务器变为候选人（Candidate），增加任期，给自己投票，并广播 `RequestVote` RPC。
- **投票逻辑**：服务器仅在当前任期内未投票，且候选人的日志至少与自己一样新（比较任期和索引）时，才投赞成票。

1.3 日志复制 (Part 2)

一旦当选，领导者（Leader）负责管理复制日志。

- **AppendEntries**：领导者周期性地广播 `AppendEntries` RPC。这些 RPC 既作为心跳（空条目），也用于携带新的日志条目。
- **一致性检查**：RPC 包含 `PrevLogIndex` 和 `PrevLogTerm`。跟随者（Follower）将这些值与自己的日志进行验证以确保连续性。如果发现不匹配，请求将被拒绝。

- **提交**：领导者跟踪每个对等节点的 `matchIndex`。当一条条目被复制到大多数服务器时，`commitIndex` 会推进，并将该条目应用到状态机。

2 挑战与解决方案

2.1 处理日志不一致（快速回退优化）

挑战：在不可靠网络（例如 `TestFigure8Unreliable`）中，如果发生日志不匹配，简单地将 `nextIndex` 减 1 的回退方式太慢，导致测试超时。

解决方案：我实现了“快速回退”（Fast Backup）优化。

1. 修改 `AppendEntriesReply` 结构体，增加 `ConflictTerm` 和 `ConflictIndex` 字段。
2. 当跟随者因日志不匹配拒绝请求时：
   - 如果它没有该日志条目，返回其日志长度。
   - 如果它有冲突的任期，返回该任期以及该任期的第一个索引。
3. 领导者利用这些信息一次性跳过所有冲突条目，显著加快了收敛速度。

2.2 并发与数据竞争

挑战：在多个 goroutine（RPC 处理器、定时器、后台发送者）之间管理共享状态经常导致竞争条件（Race Conditions）。

解决方案：通过 `rf.mu.Lock()` 强制执行严格的锁机制。所有对共享变量（状态、日志、任期）的访问都受到保护。耗时操作（如发送 RPC）在临界区之外执行，以避免死锁，并在重新获取锁后对状态进行重新验证。

3 测试结果

该实现成功通过了所有提供的测试用例，包括基本的一致性测试、持久化测试以及具有挑战性的不可靠网络场景。

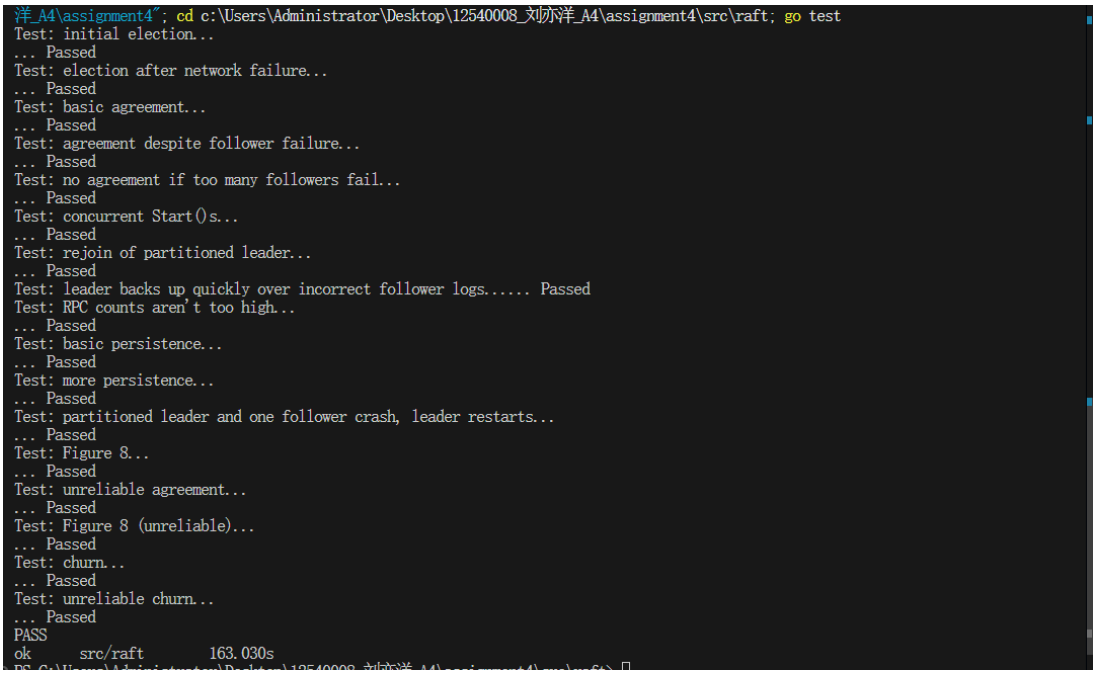

图 1: 所有 Raft 测试成功通过的截图。