实验指导手册

准备开发环境

理论上知识准备

实验过程通过相关 lab 代码，从存储、日志事务引擎出发逐步完善，支持 SQL 引擎，最终实现一个完整的支持分布式事务的分布式数据库内核。需要对于数据库理论、分布式系统有相关基础知识了解，在每个 lab 对应
参考文档中，也有对应参考资料链接。lab 设计和代码实现参考 TiDB 架构，可通过 TiDB Book 了解相关内容。

机器资源

实验过程需要在本地开发测试、调试，推荐本地开发机器有较好的处理器、内存、硬盘资源配置：

- 处理器资源配置没有特殊要求，更好的配置利于更快的运行测试程序
- 运行部分实验需要约 8GB 内存，如果运行测试时遇到内存不足的相关错误，可以尝试调低 `GOGC` 环境变量，详见*本地编码和测试*一节
- 硬盘推荐使用固态硬盘，预留至少 50GB 空间

实验准备

vldb summer school 2021 的所有实验相关组件均使用 `golang` 实现，需要在开发机配置 golang 相关开发环境。

安装 golang

参考 golang get started 根据开发机平台进行安装，推荐安装 v1.16 golang 版本。
安装完成后可以本地编译 golang 代码生成可执行程序，例如

验证方法：

```
bash> go version
go version go1.16.4 linux/amd64
```

理解 golang 包管理

`go module` 为 golang 推荐的依赖包管理方式，需要注意的是使用 go mod 方式需要确保环境变量 `GO111MODULE` 处于 `ON` 状态。vldb summer school 2021 的所有实验相关组件
均使用 go mod 进行包管理，一般情况下无需手动更改 mod 管理相关内容。

验证方法：
参考 tutorial 练习使用 go mod 进行开发，管理外部依赖。

如果您的网络环境难以下载实验所需的依赖，推荐使用 GOPROXY 代理所有的依赖包下载。

```
export GOPROXY=https://goproxy.io,direct
```

实验过程

加入 Github Classrom

参考 classroom doc 进入 github classroom 之后，将会自动生成 `vldb-2021-labs` 对应个人 private repo，尝试将该 repo clone 到本地

```
git clone git@github.com/vldbss-2021/vldb-labs-yourid.git .
```

即可进行本地开发调试，测试过程可参考每个 lab 具体说明文档进行。在该 private repo 提交修改到 master 分支之后，github classroom 也将自动执行评分任务。推荐本地测试通过之后再提交 master 分支修改进行后台评分。

我们采用了 GitHub Actions 来自动化测试和评分，在第一次向 GitHub 提交代码前，需要执行：

```
cp scripts/classroom.yml .github/workflows/classroom.yml
```

来覆盖 classroom 默认生成的 `classroom.yml` 文件。

参考每个 lab 具体文档（lab1, lab2, lab3, lab4）分别进行即可，其中 lab1 和 lab2 在 `tinykv` repo 中进行，有前后依赖关系。lab3 和 lab4 在 `tinysql` repo 中进行，有前后依赖关系。

本地编码和测试

编码可使用任意熟悉的文本编辑器或者集成开发环境，编码完成之后可在本地通过 `make xxx` 命令触发测试任务，具体可参考每个 lab 的说明文档。如果希望了解具体测试命令执行方法或者执行某个特定的测试用例可
参考 `Makefile` 中测试命令的具体实现，大部分测试的底层均使用 `go test` 进行。

本地测试过程中一些常见的系统参数调整：

```
1. 如果内存资源不足，尝试逐个运行单个测试用例，方法为
go test -v ./kv/test_raftstore -run test_name

2. 测试失败部分临时目录占用空间可能无法清理，需要手工清理，可尝试
rm -rf /tmp/test-raftstore-xxx
清理临时数据

3. 尝试配置更大的 open file limit，避免 too many open files 报错如
ulimit -n 8192

4. 如果机器内存资源不足，可设置更小的 [GOGC](https://pkg.go.dev/runtime) 环境变量加速内存回收减少物理内存
export GOGC=1; make lab1P1b
export GOGC=1; go test -v --count=1 --parallel=1 -p=1 ./kv/test_raftstore -run TestBasic2BLab1P1a
```

上传 GitHub 自动评分

完成实验后，可以将本地代码上传到 GitHub，并触发 GitHub Actions 自动评分。

查看 GitHub Actions 运行日志中 `Run you06/autograding@go` 这一步骤的输出即可检查通过了哪些测试。

注意做完全部 4 个 lab 之后才会得到全部的分数并显示通过。

问题排查方法

问题排查的思路为：

```
1. 问题是什么？即预期的输入和输出是什么？
2. 问题的范围是什么？即怀疑的点是什么？
3. 如何缩小问题的范围？即通过分析手段逐步缩小问题范围找到根源。
```

测试用例是了解模块或者代码功能的一个很好的入口，在测试过程中如果遇到问题，常用的分析手段包括：

- 通过代码分析上下文，找到不符合预期的问题点
- 通过加 log 获得更多的执行过程信息，log 可使用代码中常用的 log 方式，良好的日志输出可帮助理解执行流程和诊断问题
- 通过 dlv 调试 golang 代码

一些注意事项：

```
1. 尽量使用提供的辅助函数获取类的成员
2. 一些问题的细节或许可以参考参考 [TiDB](https://github.com/pingcap/tidb) 或者 [TiKV](https://github.
3. 如果没有较大把握，尽量不要改动原有代码
```

部分问题也可通过 classroom 同学相互讨论或者与讲师讨论尝试解决。