Lab 2 Description 实验室 2 说明

Lab 2: Raft 实验室 2：筏

Lab Summary 实验室总结

Raft manages a service's state replicas, and in particular it helps the service sort out what the correct state is after failures. Raft implements a replicated state machine. It organizes client requests into a log and ensures that everyone agrees on the ordering of contents in the log. Once log consensus is reached, each replica applies the log entries on its local state, yielding consistent replicated state across all replicas. If a server fails but later recovers, Raft must also ensure that the log of the recovered server is updated. Raft will continue to operate as long as at least a majority of the servers are alive and reachable by each other. If there is no such majority, Raft will stall, but it can restart if a reachable majority is reacquired.

Raft 管理服务的状态副本，特别是它可以帮助服务在发生故障后找出正确的状态。 Raft 实现了复制状态机。它将客户端请求组织到日志中，并确保每个人都同意日志中内容的顺序。一旦达成日志共识，每个副本都会将日志条目应用于其本地状态，从而在所有副本中产生一致的复制状态。如果一台服务器发生故障但后来恢复了，Raft 还必须确保恢复的服务器的日志得到更新。只要至少大多数服务器处于活动状态并且彼此可以访问，Raft 将继续运行。如果没有这样的多数，Raft 将停止，但如果重新获得可达多数，它可以重新启动。

Your job in this Lab is to implement Raft in either Java or Go, including a set of Raft instances that talk to each other using your `remote` library from Lab 1 to maintain replicated logs. Each Raft instance, or *peer*, will store an indefinite sequence of log entries, which are indexed sequentially and will eventually be committed as a replicated log entry. You're responsible for implementing part of the Raft design as described in **the extended Raft paper** (https://raft.github.io/raft.pdf) . You will not need to implement persistence, cluster membership changes, or log compaction/snapshots. In addition, you do not need to manage a state machine, only to maintain the logs that would be applied to a state machine in a full implementation and to track what entries would and would not have been applied to the state machine.

您在本实验中的工作是用 Java 或 Go 实现 Raft，包括一组使用实验 1 中的 `remote` 库相互通信的 Raft 实例，以维护复制的日志。每个 Raft 实例或对等点将存储不确定的日志条目序列，这些日志条目按顺序索引，并最终作为复制的日志条目提交。您负责实现 Raft 设计的一部分，如扩展 Raft 论文中所述。您不需要实现持久性、集群成员身份更改或日志压缩/快照。此外，您不需要管理状态机，只需维护将在完整实现中应用于状态机的日志，并跟踪哪些条目将被应用于状态机，哪些条目将不会被应用于状态机。

Logistics 后勤

You should work with a partner on this lab, ideally the same one you worked with in Lab 1. All of the starter code and test code can be downloaded from our Github Classroom via the links shared on the Canvas assignment. You just need to clone the repository, and you're ready to go. More importantly, you can use this repository to share and maintain your code with your partner as well as to allow the course staff to help when you are running into issues with your implementation. The details of the final submission are in a separate section toward the end of this document.

您应该与本实验室的合作伙伴合作，最好是与您在实验室 1 中合作的合作伙伴。所有起始代码和测试代码都可以通过 Canvas 作业上共享的链接从我们的 Github 课堂下载。您只需要克隆存储库，就可以开始了。更重要的是，您可以使用此存储库与合作伙伴共享和维护您的代码，并允许课程工作人员在您遇到实施问题时提供帮助。最终提交的详细信息位于本文档末尾的单独部分中。

As always, you are expected to submit only code that was typed by your own hands or those of your partner, not anything taken from anywhere online. You are certainly welcome to consult tutorials and references for Go / Java in general, but no copying code.

与往常一样，您应该仅提交由您自己或合作伙伴输入的代码，而不是从网上任何地方获取的任何代码。当然欢迎您查阅一般的 Go / Java 教程和参考资料，但禁止复制代码。

Finally, make sure to start early. Although the amount of code to write isn't huge, getting it to work correctly can be very challenging, as there are many corner cases and timing issues to work out. In addition, if a test fails, it can be very difficult to trace the source of the flaw in your solution and even more difficult to fix it.

最后，一定要尽早开始。尽管要编写的代码量并不大，但使其正常工作可能非常具有挑战性，因为有许多极端情况和时序问题需要解决。此外，如果测试失败，则很难追踪解决方案中缺陷的根源，更难以修复它。

Checkpoint and Final Submissions

检查点和最终提交

We will have two deadlines for Lab 2, and you will make two separate submissions to Gradescope, and the results will be graded and posted to the corresponding Canvas assignments (see those assignments for deadlines and submission details). Both of these submissions will be graded on their corresponding components according to the rubric provided below.

我们将为实验 2 设置两个截止日期，您将向 Gradescope 提交两次单独的提交，结果将被评分并发布到相应的 Canvas 作业中（有关截止日期和提交详细信息，请参阅这些作业）。这两项提交的内容都将根据下面提供的评分标准对其相应的部分进行评分。

Checkpoint: This initial submission will show that you are on track to complete the lab by the final deadline. At this point, you should be well on your way toward a functioning implementation of Raft by having a fully functional Raft election mechanism that supports failure and recovery of Raft peers, namely the tests that include "Checkpoint" in the name. Built/rendered documentation is not required for the Checkpoint submission, but the auto-grader should be able to run your code and determine that the checkpoint tests succeed. When you submit your code on Gradescope, you will be able to see which tests pass and how many points the auto-grader has assigned to your submission. *We do not have any hidden tests.*

检查点：初次提交将表明您有望在最终截止日期前完成实验。此时，您应该通过拥有一个功能齐全的 Raft 选举机制来支持 Raft 对等点的故障和恢复，即名称中包含“检查点”的测试，从而顺利实现 Raft 的功能。检查点提交不需要构建/渲染文档，但自动评分器应该能够运行您的代码并确定检查点测试是否成功。当您在 Gradescope 上提交代码时，您将能够查看通过了哪些测试以及自动评分器为您的提交分配了多少分。 *我们没有任何隐藏的测试。 *

Final: The final submission should include your complete Raft implementation that passes all tests (both "Checkpoint" and "Final") along with other required artifacts, including the following:

最终：最终提交应包括通过所有测试（“检查点”和“最终”）的完整 Raft 实现以及其他所需的工件，包括以下内容：

- Completed, well-commented/documented code for your Raft algorithm implementation, with built/rendered documentation output (using `make docs`).

  用于 Raft 算法实现的完整的、经过充分注释/记录的代码，以及构建/渲染的文档输出（使用 `make docs` ）。
- A brief commentary about any potential failure scenarios that you have identified that are not checked by the test code or any limitations on how your Raft implementation could be used.

  关于您已识别出的未通过测试代码检查的任何潜在故障场景的简短评论，或者关于如何使用 Raft 实现的任何限制。

Your completed code will be submitted via Gradescope, similar to the previous labs. The total grade, out of 350 points, will be allocated according to the break-down in Table 1.

您完成的代码将通过 Gradescope 提交，类似于之前的实验。总分为 350 分，将根据表 1 中的细分进行分配。

Table 1: break-down of points by task for Lab 2

表 1：实验室 2 按任务划分的分数

Task 任务	**Points 积分**
Points for passing each test 通过每项测试的分数	
`TestCheckpoint_Setup`	10
`TestCheckpoint_InitialElection`	25
`TestCheckpoint_ReElection`	30
`TestCheckpoint_BasicAgree`	40
`TestFinal_FailAgree`	40
`TestFinal_FailNoAgree`	50
`TestFinal_Rejoin`	50
`TestFinal_Backup`	50
`TestFinal_Count`	25
Documentation (quality, completeness, etc.) 文档（质量、完整性等）	30
Total 全部的	**350**

For both checkpoint and final submissions, the auto-grader on Gradescope will run each test multiple times to account for the possibility of non-deterministic failure cases. The fraction of points earned for each test case will be equal to $\frac{w^{N-p}-w^N}{1-w^N}$, where N is the number of test runs, p is the number of those runs that passed the test, and $w = 0.85$ is a weighting factor.

对于检查点和最终提交，Gradescope 上的自动评分器将多次运行每个测试，以考虑非确定性失败情况的可能性。每个测试用例获得的分数将等于 $\frac{w^{N-p}-w^N}{1-w^N}$ ，其中 N 是测试运行的次数， p 是通过测试的运行次数， $w = 0.85$ 是权重因子。

Implementing Raft 实施Raft

Your Raft implementation will focus on only a part of the provided code. In Go, this is all located in `raft.go`, which you can extend as you see fit. In Java, this is in `RaftPeer.java`, which you can populate and create new classes as desired. Nearly all of the "starter code" in the files we're providing is actually comments that provide some details about what needs to be implemented and now it interacts with the test code. To get started, you should first copy your `remote` library from Lab 1 (just the library, not the test code) into the corresponding folder (as explained in the starter code README file), as Lab 2 will use this to enable interactions between Raft peers. In both languages, the test suite uses a `Controller` to create Raft peers, act in the role of clients, and coordinate with the test code. You should not change any part of the `Controller`, but your Raft peer needs to work in the way that the `Controller` expects it to work, or the tests won't work.

您的 Raft 实现将仅关注所提供代码的一部分。在 Go 中，这一切都位于 `raft.go` 中，您可以根据需要进行扩展。在 Java 中，它位于 `RaftPeer.java` 中，您可以根据需要填充和创建新类。我们提供的文件中几乎所有“起始代码”实际上都是注释，它们提供了有关需要实现的内容的一些详细信息，现在它与测试代码进行交互。首先，您应该首先将实验 1 中的 `remote` 库（只是库，而不是测试代码）复制到相应的文件夹中（如起始代码 README 文件中所述），因为实验 2 将使用这是为了实现 Raft 对等点之间的交互。在这两种语言中，测试套件都使用 `Controller` 来创建 Raft 对等点，充当客户端角色，并与测试代码进行协调。您不应更改 `Controller` 的任何部分，但您的 Raft 对等点需要按照 `Controller` 期望的方式工作，否则测试将无法工作。

Tasks 任务

The primary tasks you must complete in Lab 2 include:

您必须在实验 2 中完成的主要任务包括：

- Create a functional Raft peer that implements the methods needed by the `Controller` in order to create each Raft peer, start and stop each Raft peer's underlying `Service` interface, perform the tests, and terminate each Raft peer.

  创建一个功能性 Raft 对等点，实现 `Controller` 所需的方法，以便创建每个 Raft 对等点、启动和停止每个 Raft 对等点的底层 `Service` 接口、执行测试并终止每个 Raft 对等点筏同行。

- Implement all of the needed remote method calls, along with suitable definitions in the `RaftInterface` service interface that is used as input to the `Service` and `StubFactory` components that you created in Lab 1.

  实现所有需要的远程方法调用，以及用作您创建的 `Service` 和 `StubFactory` 组件输入的 `RaftInterface` 服务接口中的适当定义在实验室 1 中。

- Creation of suitable mechanisms / structs / classes to support message exchanges between Raft peers.

  创建合适的机制/结构/类来支持 Raft 对等点之间的消息交换。

- Completion of all of the `TODO` items in the starter code according to the details in **the Raft paper** (https://raft.github.io/raft.pdf)

  根据Raft论文中的详细信息完成起始代码中的所有 `TODO` 项.

Requirements for your Implementation

您的实施要求

As given in the starter code, your Raft peer class must implement a constructor / builder method that takes three parameters: the port number that the Raft peer's `Service` interface will operate on, the ID (or index) of the Raft peer within the group, and the number of Raft peers in the group. These parameters are supplied by the `Controller` as part of each test that is run by the test suite, so you don't need to worry about where the numbers come from. However, you need to know that the `Controller` assigns the port numbers sequentially, where the port number p used by Raft peer with ID=0 is randomly chosen, then the other Raft peers will use port numbers $p+1, \ldots, p+num-1$ in order of peer ID. This means that each Raft peer can figure out the Service port numbers used by other Raft peers from the three parameters given. When you create your Raft peer, you will need to carefully consider how the group of Raft peers will coordinate with each other in support of the Raft algorithm. Each peer will expose a single `Service` to other peers, and all other peers will need to access this `Service` using their own

`StubFactory` output; this means that each peer takes the roles of both caller (i.e., client) and callee (i.e., server).

如起始代码中所示，您的 Raft 对等点类必须实现一个构造函数/构建器方法，该方法采用三个参数：Raft 对等点的 `Service` 接口将操作的端口号、Raft 对等点的 ID（或索引）组内的 Raft 对等体，以及组中 Raft 对等体的数量。这些参数由 `Controller` 作为测试套件运行的每个测试的一部分提供，因此您无需担心数字的来源。不过需要注意的是， `Controller` 是按顺序分配端口号的，其中ID=0的Raft Peer使用的端口号 p 是随机选择的，那么其他Raft对等体将按照对等体ID的顺序使用端口号 $p+1,...,p+num-1$ 。这意味着每个 Raft 对等体都可以从给定的三个参数中找出其他 Raft 对等体使用的服务端口号。当您创建 Raft 对等点时，您需要仔细考虑 Raft 对等点组如何相互协调以支持 Raft 算法。每个对等点都会向其他对等点公开一个 `Service` ，并且所有其他对等点将需要使用自己的 `StubFactory` 输出来访问此 `Service` ；这意味着每个对等点都扮演调用者（即客户端）和被调用者（即服务器）的角色。

Since we're building our Raft implementation on top of the remote library created in Lab 1, you'll need to provide a definition of the `RaftInterface` service interface. The Raft algorithm only relies on two remote method calls, but our `Controller` is built around three additional remote method calls, so your service interface needs to support all five of them. The three used by the `Controller` are defined in the `RaftInterface` in the starter code, and you'll need to complete the definition of the two needed for the Raft algorithm before you can use the interface. You will then need to provide complete implementations of all five of the corresponding remote method calls. There are several comments in the starter code to help you through this process, but the main source of information for the two remote methods that Raft relies on will be the Raft paper.

由于我们是在实验 1 中创建的远程库之上构建 Raft 实现，因此您需要提供 `RaftInterface` 服务接口的定义。 Raft 算法仅依赖于两个远程方法调用，但我们的 `Controller` 是围绕三个额外的远程方法调用构建的，因此您的服务接口需要支持所有这五个方法。 `Controller` 使用的三个在起始代码的 `RaftInterface` 中定义，您需要先完成 Raft 算法所需的两个定义，然后才能使用界面。然后，您需要提供所有五个相应远程方法调用的完整实现。起始代码中有一些注释可以帮助您完成此过程，但是 Raft 所依赖的两个远程方法的主要信息来源将是 Raft 论文。

When defining the `RequestVote` and `AppendEntries` remote method calls in the `RaftInterface` definition, you should follow the algorithm definition in [the Raft paper (https://raft.github.io/raft.pdf)](https://raft.github.io/raft.pdf), and the part of the paper that is likely to be the most helpful is the summary of the algorithm provided in Figure 2 of the paper. You'll likely need to define several different data structures, such as messages exchanged between Raft peers, entries to put in each peer's log, and anything else you need. Make sure that anything that needs to be sent over the network is compatible with the underlying remote library, since that is what carries messages between peers and `Controller`. Since our `remote` library only supports synchronous remote method calls, **our implementation of the Raft algorithm will be slightly simplified compared to what is in the Raft paper**; aspects of voting, log replication, and application of committed logs become a lot more clear when all calls are synchronous, and you can take advantage of this simplification.

在 `RaftInterface` 定义中定义 `RequestVote` 和 `AppendEntries` 远程方法调用时，应该遵循Raft论文中的算法定义，以及论文中的部分最有帮助的可能是论文图 2 中提供的算法摘要。您可能需要定义几种不同的数据结构，例如 Raft 对等点之间交换的消息、要放入每个对等点日志中的条目以及您需要的任何其他内容。确保需要通过网络发送的任何内容都与底层远程库兼容，因为这是在对等点和 `Controller` 之间传递消息的内容。由于我们的 `remote` 库仅支持同步远程方法调用，因此我们对Raft算法的实现相比Raft论文中的实现会稍微简化；当所有调用都是同步时，投票、日志复制和已提交日志的应用等方面都会变得更加清晰，并且您可以利用这种简化。

Similar to how our remote library simulates message loss and propagation delay, our `Controller` simulates disconnection/failure of Raft peers and further uses this capability to emulate network partitions. The way that our `Controller` does this is by starting/stopping the underlying `Service` interface of a Raft peer. As such, your Raft peer must expose (non-remote) method calls to facilitate this control, namely, functions called `Activate` and `Deactivate` that allow the `Controller` to start and stop the Raft peer's `Service`. These are not a part of the `RaftInterface` that defines remote methods, as the `Controller` must be able to call `Activate` on a Raft peer when its `Service` is disabled, and it must be able to call `Deactivate` on a Raft peer when it is in the midst of handling other remote calls. This also means that your Raft peer implementation must be able to continue to operate when remote calls to another Raft peer fail due to inability to connect to the remote `Service`. You also need to implement the other remote method calls used by the `Controller`, as described previously. All of these methods are straightforward, mainly returning internal Raft peer state to the `Controller` for validation in tests.

与我们的远程库模拟消息丢失和传播延迟的方式类似，我们的 `Controller` 模拟 Raft 对等点的断开/故障，并进一步使用此功能来模拟网络分区。我们的 `Controller` 执行此操作的方式是启动/停止 Raft 对等点的底层 `Service` 接口。因此，您的 Raft 对等方必须公开（非远程）方法调用以促进此控制，即调用 `Activate` 和 `Deactivate` 的函数，它们允许 `Controller` 启动和停止 Raft Peer 的 `Service` 。这些不是定义远程方法的 `RaftInterface` 的一部分，因为 `Controller` 必须能够在其 `Service` /b10> 已禁用，当 Raft 对等点正在处理其他远程调用时，它必须能够调用 Raft 对等点上的 `Deactivate` 。这也意味着当对另一个 Raft 对等点的远程调用由于无法连接到远程 `Service` 而失败时，您的 Raft 对等点实现必须能够继续运行。您还需要实现 `Controller` 使用的其他远程方法调用，如前所述。所有这些方法都很简单，主要是将内部 Raft 对等状态返回到 `Controller` 以在测试中进行验证。

Important notes: 重要笔记:

- Your implementation cannot use any form of communication between Raft peers other than what is provided through the `RaftInterface` (i.e., no files, shared databases, channels, etc.).

  除了通过 `RaftInterface` 提供的通信（即没有文件、共享数据库、通道等）之外，您的实现不能使用 Raft 对等点之间的任何形式的通信。

- The test code for Lab 2 will evaluate a few aspects of algorithm performance, which is why your `Service` from Lab 1 keeps track of how many remote calls it has supported.

  实验 2 的测试代码将评估算法性能的几个方面，这就是为什么实验 1 中的 `Service` 会跟踪它支持的远程调用数量。

- As mentioned, there are a lot of initial comments in the starter code. These are intended to help you, so it is important that you read them early in the process. You can also use the rules in the provided `Makefile` to generate the package documentation from these comments, as with previous labs. You are also expected to remove these comments and replace them with your own comments as part of your overall documentation of your implementation.

  如前所述，起始代码中有很多初始注释。这些内容旨在为您提供帮助，因此在此过程中尽早阅读它们非常重要。您还可以使用提供的 `Makefile` 中的规则从这些注释生成包文档，就像之前的实验一样。您还应该删除这些注释并用您自己的注释替换它们，作为您的实现的整体文档的一部分。

- Section 5.2 of [the Raft paper](https://raft.github.io/raft.pdf) (https://raft.github.io/raft.pdf)

  Raft 论文第 5.2 节 suggests election timeouts in the range of 150-300ms, but this range only makes sense if the leader sends heartbeat

  messages considerably more often than every 150ms, which may not be sustainable in the design of our test suite. You will need to experiment with the timeout parameters to find an election timeout that is larger than what the paper suggests but still small enough to ensure that elections complete within the time allowed.

  建议选举超时在 150-300 毫秒范围内，但只有当领导者发送心跳消息的频率远高于每 150 毫秒时，这个范围才有意义，这在我们的测试套件的设计中可能是不可持续的。您需要对超时参数进行试验，以找到比论文建议的更大但仍然足够小的选举超时，以确保选举在允许的时间内完成。

- Make sure to implement the election restriction mechanism described in Section 5.4.1 of the Raft paper.

  确保实现 Raft 论文第 5.4.1 节中描述的选举限制机制。

Testing 测试

The starter code includes a fairly comprehensive set of tests that evaluate the compliance and performance of your Raft implementation. The vast majority of your grade is directly computed using these tests, and there are no additional "hidden" tests that we will use beyond what is included in the starter code. The commands used to run the tests are included in the `Makefile` and are very similar to those of previous labs.

You are encouraged to test your implementation multiple times, both locally in your own environment and using the auto-graders provided on Gradescope. Due to the additional complexity of debugging a Raft implementation, we are providing an additional auto-grader that will allow you to run subsets of the tests or create much more verbose output logs compared to the Checkpoint and Final auto-graders. This "auxiliary" auto-grader is programmed to run the `make all` rule from the Makefile, running each test one (1) time. The numeric grade computed by the auxiliary auto-grader will not be used in any way toward your checkpoint or final grade on this lab; it is provided purely as an optional testing platform for your use.

起始代码包括一组相当全面的测试，用于评估 Raft 实现的合规性和性能。您的绝大多数成绩都是使用这些测试直接计算的，除了起始代码中包含的内容之外，我们不会使用任何其他“隐藏”测试。用于运行测试的命令包含在 `Makefile` 中，并且与之前实验的命令非常相似。我们鼓励您多次测试您的实施，无论是在您自己的本地环境中还是使用 Gradescope 上提供的自动评分器。由于调试 Raft 实现的额外复杂性，我们提供了一个额外的自动评分器，与 Checkpoint 和 Final 自动评分器相比，它允许您运行测试的子集或创建更详细的输出日志。这个“辅助”自动评分器被编程为运行 Makefile 中的 `make all` 规则，运行每个测试一 (1) 次。辅助自动评分器计算出的数字成绩不会以任何方式用于您在本实验中的检查点或最终成绩；它纯粹作为一个可选的测试平台提供给您使用。

Submission of Deliverables

提交成果

There are two different Gradescope assignments / auto-graders for this lab that you are required to submit to, one for the Checkpoint and one for the Final submission. You can submit as many times as you like to these assignments, and we will use the "Active" score on each submission for the corresponding assignment grades. The auto-grader will over-write the test suite in your submitted archive with the test suite from the starter-code repository (subject to any changes announced on Piazza, such as bug fixes), to ensure that no changes were made.

您需要向该实验室提交两种不同的 Gradescope 作业/自动评分器，一种用于检查点，一种用于最终提交。您可以多次提交这些作业，我们将在每次提交时使用“有效”分数作为相应的作业成绩。自动评分器将使用起始代码存储库中的测试套件覆盖您提交的存档中的测试套件（以 Piazza 上宣布的任何更改为准，例如错误修复），以确保不进行任何更改。

For each submission, you will upload your code as a `.zip` file on Gradescope. Gradescope will automatically run our tests and display your auto-graded score. **All documentation for the lab must be built/rendered and included in the `.zip` file (e.g., `javadoc` or `go doc` outputs)**

for manual grading, as we do not have the ability to build the documentation outputs from within the Gradescope backend (and downloading / building offline is not always possible due to build environment differences). Make sure to include all of your project artifacts in the `.zip` file, even if it is only needed for the manually graded part of the assignment.

对于每次提交，您都将代码作为 `.zip` 文件上传到 Gradescope 上。 Gradescope 将自动运行我们的测试并显示您的自动评分分数。实验室的所有文档都必须构建/渲染并包含在 `.zip` 文件中（例如 `javadoc` 或 `go doc` 输出）以进行手动评分，因为我们不这样做能够从 Gradescope 后端构建文档输出（由于构建环境差异，离线下载/构建并不总是可行）。确保将所有项目工件包含在 `.zip` 文件中，即使仅作业的手动评分部分需要它。

If you get an HTTP 500 error when uploading to Gradescope, try the following steps (one at a time, or all at once):

如果您在上传到 Gradescope 时收到 HTTP 500 错误，请尝试以下步骤（一次一个或一次全部）：

- Remove executables (e.g., `.class`, `.o`) from the `.zip` archive.

  从 `.zip` 存档中删除可执行文件（例如 `.class` 、 `.o` ）。
- Remove metadata (e.g., `.git`, `__MACOSX`) from the `.zip` archive.

  从 `.zip` 存档中删除元数据（例如 `.git` 、 `__MACOSX` ）。
- For the Java version, you can also remove the `test/` directory from the `.zip` archive.

  对于 Java 版本，您还可以从 `.zip` 存档中删除 `test/` 目录。

Important note: the auto-grader expects your `.zip` file to contain a top-level directory called either `lab2-go` or `lab2-java`, depending on which language you are using, described in the `README.md` file in the starter-code repository -- please make sure your submission complies with this expectation, or the auto-grader will assign a score of zero. For the Go version of the lab, we will attempt to retain certain information from your `raft_test.go` file, so make sure it is included in your submission.

重要提示：自动评分器期望您的 `.zip` 文件包含一个名为 `lab2-go` 或 `lab2-java` 的顶级目录，具体取决于您使用的语言，起始代码存储库中的 `README.md` 文件中描述 - 请确保您的提交符合此期望，否则自动评分器将分配零分。对于 Go 版本的实验，我们将尝试保留您的 `raft_test.go` 文件中的某些信息，因此请确保将其包含在您提交的内容中。

For the Checkpoint submission, your submitted code should be capable of passing the initial `TestCheckpoint` tests. For the Final submission, your code should include all functionality required to pass all tests. If only a subset of tests are satisfied, the grade will be assigned according to the point break-down given earlier in this document.

对于检查点提交，您提交的代码应该能够通过初始 `TestCheckpoint` 测试。对于最终提交，您的代码应包含通过所有测试所需的所有功能。如果仅满足一部分测试，则将根据本文档前面给出的分值细目分配等级。

You are free to modify the `Makefile`, but it must correctly support the `checkpoint` and `final` rules for whichever language the repository was cloned from. You are free to add files as long as the `make` rules perform as needed. Each of our auto-graders is written as a bash script; for full transparency, we will share the bash script with anyone who wants to see it.

您可以随意修改 `Makefile` ，但它必须正确支持克隆存储库的语言的 `checkpoint` 和 `final` 规则。只要 `make` 规则根据需要执行，您就可以自由添加文件。我们的每个自动评分器都是作为 bash 脚本编写的；为了完全透明，我们将与任何想要查看它的人共享 bash 脚本。

Please add names, AndrewIDs, and GitHub team name to the `README.md` file in the archive that you submit.

请将姓名、AndrewID 和 GitHub 团队名称添加到您提交的存档中的 `README.md` 文件中。

Helpful Reminders 有用的提醒

In general, the goal of this lab is get a firm understanding of how the Raft consensus protocol works at a fairly deep level. The goal isn't to make you struggle with programming, but rather to leverage advanced programming capabilities to implement a very useful algorithm. If there's ever anything you don't know how to do in a particular programming language, please ask! Even if we don't know off hand, we'll help you work through the details. Aside from programming itself, if there is any aspect of the Raft algorithm that you'd like to discuss, please ask! Remember, the first step to designing a large application should never be writing code. If anything else comes up, please ask! Never hesitate to reach out to the course staff via Piazza, Slack, or office hours. Helping you learn is our job.

总的来说，这个实验室的目标是深入了解 Raft 共识协议的工作原理。目标不是让您在编程方面遇到困难，而是利用高级编程功能来实现非常有用的算法。如果您有任何不知道如何使用特定编程语言做的事情，请询问！即使我们不知道，我们也会帮助您解决细节问题。除了编程本身之外，如果您想讨论 Raft 算法的任何方面，请询问！请记住，设计大型应用程序的第一步永远不应该是编写代码。如果还有什么问题请追问！请随时通过 Piazza、Slack 或办公时间联系课程工作人员。帮助您学习是我们的工作。

One of the challenges that you will probably face in this lab is creating useful log/debug output to help you debug and streamline your Raft implementation. Since the tests involve spawning multiple Raft peers that run asynchronously, debug output is not always easy to obtain and parse, since the different peers may either output their logs to different files (which makes it difficult to correlate them to each other) or merge them into a single stream (which makes it difficult to differentiate peer output). There are many useful things that you can do, including using extremely verbose logging (every print statement has a timestamp, peer ID, and other relevant state) and the use of third-party log generation / visualization tools. You are certainly allowed to use such approaches and tools for this lab, as long as they do not provide specific contributions toward your Raft implementation (other than helping you debug).

您在本实验中可能面临的挑战之一是创建有用的日志/调试输出来帮助您调试和简化 Raft 实现。由于测试涉及生成多个异步运行的 Raft 对等点，因此调试输出并不总是易于获取和解析，因为不同的对等点可能会将其日志输出到不同的文件（这使得很难将它们相互关联）或合并它们到单个流中（这使得很难区分对等输出）。您可以做许多有用的事情，包括使用极其详细的日志记录（每个打印语句都有时间戳、对等 ID 和其他相关状态）以及使用第三方日志生成/可视化工具。当然，您可以在本实验中使用此类方法和工具，只要它们不为您的 Raft 实现提供特定贡献（除了帮助您调试）。

If You Get Stuck

Please reach out to us! We are here to help you. There are several difficult parts of this lab, and there are sure to be bumps along the way. However, we would very much appreciate a few things before asking for coding or debugging help, and these will help you more than us.

- Start early! The more stressed you get, the harder it will be to work out coding issues.
- When asking for help, be as descriptive as possible. What have you tried? What do you think may be happening? Can you come up with scenarios that do/don't exhibit buggy behavior? Have you traced through the code to find the precise thing that is going wrong (not just a line of code in an error message)?
- Make frequent commits / branches to your Github repo! You can always roll back to your last commit, especially if something goes very wrong.
- Use Piazza and Slack! Since OH time is limited, take advantage of the asynchronous resources. We're often willing to have a quick chat outside of OHs.
- A private post is a great way to ask questions about your code, all we need is a link to a specific git commit or branch.

Lastly, don't forget to consult **the Raft paper** (https://raft.github.io/raft.pdf) . ;)