華中科技大學

课 程 实 验 报 告

课程名称： 高级分布式系统

专业班级： 计算机硕 xxxx 班

学　　号： M202xxxxxx

姓　　名： 王牧之

指导教师： 金海、石宣化

报告日期： 202x.xx.xx

计算机科学与技术学院

目录

1. 实验一

1.1 实验目的

本次实验考察学生对 Paxos 算法思想的理解和掌握，并根据注释提示完成相关程序代码设计。

1.2 实验内容

1. 任务描述

根据 Paxos 算法流程完成相关核心成员函数设计。

2. 相关知识

(1). 角色

1) 提议者(proposer)：进行提议的角色；

2) 批准者(acceptor)：通过提议的角色；

3) 学习者(learner)：感知(learn)被选中的提议。

在具体的实现中，一个进程可能同时充当多种角色。比如一个进程可能既是 Proposer 又是 Acceptor 又是 Learner。

(2). 提案

还有一个很重要的概念叫提案(Proposal)。最终要达成一致的 value 就在提案里。Proposer 可以提出(propose)提案；Acceptor 可以接受(accept)提案；如果某个提案被选定(chosen)，那么该提案里的 value 就被选定了。

在何种情况下不同的进程认为提案被选中了？

1) Proposer：只要 Proposer 发的提案被 Acceptor 接受，Proposer 就认为该提案里的 value 被选定了；

2) Acceptor：只要 Acceptor 接受了某个提案，Acceptor 就认为该提案里的 value 被选定了；

3) Learner：Acceptor 告诉 Learner 哪个 value 被选定，Learner 就认为那个 value 被选定。

3. 编程要求

根据提示，补充代码，根据 Paxos 算法流程完成 Proposer.cpp 和 Acceptor.cpp 中 Proposer 和 Acceptor 类的核心成员函数设计。

4. 测试说明

后台会自动检测你的输出结果，当与预期输出一致时，则算通关。

示例 1-1 正确输出示例

```
2019-11-28 18:36:41 Tid:6061 [Info] Proposer4 号的提议被批准,用时 3763MS:最终提议 = [编号:5, 提议:5]
2019-11-28 18:36:52 Tid:6059 [Info] Proposer2 号的提议被批准,用时 15085MS:最终提议 = [编号:13, 提议:5]
2019-11-28 18:36:58 Tid:6058 [Info] Proposer1 号的提议被批准,用时 20606MS:最终提议 = [编号:17, 提议:5]
2019-11-28 18:37:04 Tid:6057 [Info] Proposer0 号的提议被批准,用时 26425MS:最终提议 = [编号:21, 提议:5]
2019-11-28 18:37:09 Tid:6060 [Info] Proposer3 号的提议被批准,用时 31679MS:最终提议 = [编号:24, 提议:5]
2019-11-28 18:37:09 Tid:6060 [Info] Paxos 完成, 用时 31702MS, 最终通过提议值为: 5
```

为了便于评测，后台会对结果进行处理，处理后的预期输出如下：**最终通过提议值为：5。**

1.3 实验方法

本实验模拟了 Paxos 算法达成一致的过程，设计了 5 个 Proposer 进行提案，11 个 Acceptor 对提案进行投票。Proposer 的初始提案被设定为[编号 i+1，提议 i+1]，例如 Proposer0 的初始提案为[编号 1，提议 1]。在 Propose 阶段和 Accept 阶段，每个 Proposer 都需要取得半数以上 Acceptor 的同意，否则就要增大自己的提案编号并重新发起拉票请求，提案编号每次按 Proposer 的数量增加。

Paxos 算法达成一致的流程如图 1-1 所示，基于此对代码逻辑进行补全。

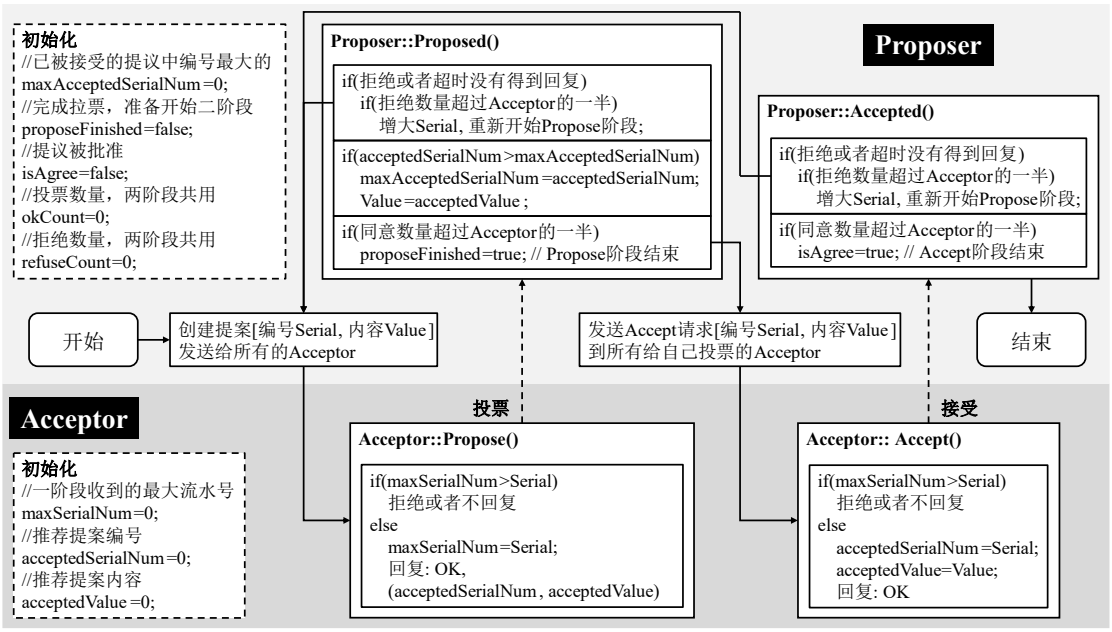

图 1-1 Paxos 算法达成一致流程示意图

1.3.1 Proposer

1. Proposer 在 Propose 阶段的行为如下：

(1). 如果半数以上的 Acceptor 拒绝或者没有回应，则将编号增大并进行新一轮提案；

(2). 如果收到投票，且存在返回的推荐提案，则在 Accept 阶段会改用推荐的提案内容；

(3). 如果累计收到超过半数 Acceptor 的投票，则 Propose 阶段结束，开始 Accept 阶段。

代码 1-1 Proposer::Proposed()代码逻辑补全

```
bool Proposer::Proposed(bool ok, PROPOSAL &lastAcceptValue)
{
    if ( m_proposeFinished ) return true;//可能是一阶段迟到的回应，直接忽略消息
    if ( !ok )
    {
        m_refuseCount++;
        //已有半数拒绝，不需要等待其它 acceptor 投票了，重新开始 Propose 阶段
        //使用 StartPropose(m_value)重置状态
        //请完善下面逻辑
        /**********Begin**********/
        if (m_refuseCount > m_acceptorCount / 2)
        {
            m_value.serialNum += m_proposerCount;
            StartPropose(m_value);
            return false;
        }
        /**********End**********/
        //拒绝数不到一半
        return true;
    }
    m_okCount++;
    /*
    没有必要检查分支：serialNum 为 null
    因为 serialNum>m_maxAcceptedSerialNum，与 serialNum 非 0 互为必要条件
    */
    //如果已经有提议被接受，修改成已被接受的提议
    //请完善下面逻辑
    /**********Begin**********/
    if (lastAcceptValue.serialNum > m_maxAcceptedSerialNum)
    {
        m_maxAcceptedSerialNum = lastAcceptValue.serialNum;
```

```
            m_value.value = lastAcceptValue.value;
        }
        /**********End**********/
        //如果自己的提议被接受
        if ( m_okCount > m_acceptorCount / 2 )
        {
            m_okCount = 0;
            m_proposeFinished = true;
        }
        return true;
}
```

2. Proposer 在 Accept 阶段的行为如下：

(1).与 Propose 阶段相同，如果半数以上的 Acceptor 拒绝或者没有回应，则将编号增大并进行新一轮提案；

(2).如果累计收到超过半数 Acceptor 的批准，则 Accept 阶段结束，达成一致。

代码 1-2 Proposer::Accepted()代码逻辑补全

```
bool Proposer::Accepted(bool ok)
{
    if ( !m_proposeFinished ) return true;//可能是上次第二阶段迟到的回应，直接忽略消息
    if ( !ok )
    {
        m_refuseCount++;
        //已有半数拒绝，不需要等待其它 acceptor 投票了，重新开始 Propose 阶段
        //使用 StartPropose(m_value)重置状态
        //请完善下面逻辑
        /**********Begin**********/
        f (m_refuseCount > m_acceptorCount / 2)
        {
            m_value.serialNum += m_proposerCount;
            StartPropose(m_value);
            return false;
        }
        /**********End**********/
        return true;
    }
    m_okCount++;
    if ( m_okCount > m_acceptorCount / 2 ) m_isAgree = true;
    return true;
    }
```

1.3.2 Acceptor

1. Acceptor 在 Propose 阶段的行为如下：

(1). 当 m_maxSerialNum>serialNum，即当前提案的编号小于记录的最大编号时，拒绝该提案或者不回应。在代码中使用 return false 显式地通知给 Proposer 拒绝信息，也可以选择不做回应，Proposer 通过超时推算出自己的提案没有通过，从而进如新一轮提案。

(2). 提案通过(m_maxSerialNum<=serialNum)，会执行两个动作：

1) 更新记录的最大编号 m_maxSerialNum=serialNum，从而 Acceptor 作出承诺保证后面不会再投票给编号小于 serialNum 提案；

2) 如果存在同意过的其他提案，则推荐最后一次同意的提案内容 m_lastAcceptValue，从而使 Proposer 在 Accept 阶段使用 lastAcceptValue 达成一致。

代码 1-3 Acceptor::Propose()代码逻辑补全

```
bool Acceptor::Propose(unsigned int serialNum, PROPOSAL &lastAcceptValue)
{
    if ( 0 == serialNum ) return false;
    //提议不通过
    if ( m_maxSerialNum > serialNum ) return false;
    //接受提议
    //请完善下面逻辑
    /**********Begin**********/
    m_maxSerialNum = serialNum;
    lastAcceptValue = m_lastAcceptValue;
    /**********End**********/
    return true;
}
```

2. Acceptor 在 Accept 阶段的行为如下：

(1). 如果 m_maxSerialNum>value.serialNum，则说明 Acceptor 又同意了其他提案，对该次的 accept 请求回复拒绝或者不回复，Proposer 收到反馈会重新进入 Propose 阶段；

(2). 批准提案通过。

代码 1-4 Acceptor::Accept()代码逻辑补全

```
bool Acceptor::Accept(PROPOSAL &value)
{
    if ( 0 == value.serialNum ) return false;
    //Acceptor 又重新答应了其他提议
    //请完善下面逻辑
```

```
    /**********Begin**********/
    if (m_maxSerialNum > value.serialNum) return false;
    /**********End**********/
    //批准提议通过
    //请完善下面逻辑
    /**********Begin**********/
    m_lastAcceptValue = value;
    /**********End**********/
    return true;
}
```

1.4 实验结果

educoder 平台中实验结果如图 1-2 所示，顺利完成了该实验。

图 1-2 educoder 平台实验通过结果截图

Paxos 达成一致结果输出如图 1-3 所示。

```
2023-02-05 11:09:33 Tid:62 [Info] 5个Proposer, 11个Acceptor准备进行Paxos
每个Proposer独立线程, Acceptor不需要线程
Proposer编号从0-10,编号为i的Proposer初始提议编号和提议值是 (i+1, i+1)
Proposer每次重新提议会将提议编号增加5
Proposer被批准后结束线程,其它线程继续投票最终, 全部批准相同的值, 达成一致。

2023-02-05 11:09:33 Tid:62 [Info] Paxos开始

2023-02-05 11:09:38 Tid:67 [Info] Proposer4号的提议被批准,用时4945MS:最终提议 = [编号:5, 提议:5]

2023-02-05 11:09:55 Tid:64 [Info] Proposer1号的提议被批准,用时21842MS:最终提议 = [编号:17, 提议:5]

2023-02-05 11:10:01 Tid:63 [Info] Proposer0号的提议被批准,用时27817MS:最终提议 = [编号:21, 提议:5]

2023-02-05 11:10:06 Tid:66 [Info] Proposer3号的提议被批准,用时33494MS:最终提议 = [编号:24, 提议:5]

2023-02-05 11:10:10 Tid:65 [Info] Proposer2号的提议被批准,用时37424MS:最终提议 = [编号:28, 提议:5]

2023-02-05 11:10:10 Tid:65 [Info] Paxos完成, 用时37425MS, 最终通过提议值为: 5
```

图 1-3 Paxos 达成一致过程

本实验中，一共有 5 个 Proposer 和 11 个 Acceptor 进行 Paxos 算法过程模

拟，最终通过的提议值为 5。下面以 Proposer0 的行为为例，展开分析。Proposer0 的提议-接受过程如图 1-4 所示。Proposer0 的初始提案为[编号: 1,提议: 1]，得到了半数以上 Acceptor 投票进入 Accept 阶段，但在 Accept 阶段由于编号较小被拒绝，于是增大提案编号（1→6）重新提案；在第二次 Propose 阶段由于被拒绝，继续增大提案编号（6→11）并重新提案；在第三次 Propose 阶段收到投票和推荐提案，则修改提案内容（1→5）并重新提案；在第四次 Propose 阶段得到了半数以上 Acceptor 投票进入 Accept 阶段，同样地，在 Accept 阶段由于编号较小被拒绝，于是增大提案编号（11→16）重新提案；如此循环往复，直到编号增大到 21 时，提案得到半数以上 Acceptor 批准。

```
2023-02-05 11:09:33 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:1, 提议:1]
2023-02-05 11:09:37 Tid:63 [Info] Proposer0号开始(Accept阶段):提议=[编号:1, 提议:1]
2023-02-05 11:09:39 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:6, 提议:1]
2023-02-05 11:09:43 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:11, 提议:1]
2023-02-05 11:09:44 Tid:63 [Info] Proposer0号修改了提议:提议=[编号:11, 提议:5]
2023-02-05 11:09:44 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:11, 提议:5]
2023-02-05 11:09:49 Tid:63 [Info] Proposer0号开始(Accept阶段):提议=[编号:11, 提议:5]
2023-02-05 11:09:50 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:16, 提议:5]
2023-02-05 11:09:55 Tid:63 [Info] Proposer0号开始(Accept阶段):提议=[编号:16, 提议:5]
2023-02-05 11:09:56 Tid:63 [Info] Proposer0号开始(Propose阶段):提议=[编号:21, 提议:5]
2023-02-05 11:10:00 Tid:63 [Info] Proposer0号开始(Accept阶段):提议=[编号:21, 提议:5]
2023-02-05 11:10:01 Tid:63 [Info] Proposer0号的提议被批准,用时27817MS:最终提议 = [编号:21, 提议:5]
```

图 1-4 Proposer0 的提议-接受过程

类似地，结合图 1-3 可知，各个 Proposer 编号增长过程如下：

- Propose0：1→6→11→16→21
- Propose1：2→7→12→17
- Propose2：3→8→13→18→23→28
- Propose3：4→9→14→19→24
- Propose4：5

这说明在 Paxos 算法中当提案编号增长到大于上一次批准的提案编号时，提案才能被批准。

2. 实验二

2.1 实验目的

本次实验考察学生对写穿算法思想的理解和掌握，并根据注释提示完成相关程序代码设计补全。

2.2 实验内容

1. 任务描述

补充程序实现写穿算法。

2. 相关知识

为了完成本实验，需要掌握：分布式文件系统基础概念，写穿算法。

(1). 分布式文件系统

1) 从用户的使用角度来看，分布式文件系统是一个标准的文件系统，提供了一系列 API，由此进行文件或目录的创建、移动、删除，以及对文件的读写等操作。

2) 从内部实现角度来看，分布式文件系统还要通过网络管理存储在多个节点上的文件和目录。并且，同一文件不只是存储在一个节点上，而是按规则分布存储在一簇节点上，协同提供服务。

(2). 写穿算法

当用户在修改高速缓存项(文件或块)时，新的值保存在高速缓存中，并立即写回到服务器，当用户读取速缓存项(文件或块)时，需要先和服务端进行文件的 version 号比对，如果一致，直接从高速缓存项 cache 中读取文件，如果不一致则从服务器中读取文件并将文件缓存在本地高速 cache 中。

3. 编程要求

根据提示，补充代码。根据 cache.py 中的提示完成 cache 类的读和写程序。

4. 编程提示

类与函数解释

(1). sim_file 模拟文件条目类

1) sim_file.write(data,verion)向文件条目写入 data 和 version。

2) sim_file.get_data()获取文件条目数据，返回 file_name,data,version。

3) sim_file.get_version()获取文件 version。

(2). server 模拟服务器类，在 cache 类初始化时绑定为 self._target_server，请使用该变量调用

1) server.write(target_file,data,version)尝试向服务器写入目标文件，如果目标文件已经存在，则会覆盖写入。

2) server.read(target_file)尝试向服务器读取目标文件，如果服务器中存在该文件，则会返回 file_name,data,version；如果不存在，则会返回 None。

3) server.get_version(target_file)尝试向服务器获取目标文件 version。

(3). cache 模拟高速缓存类(在实现的函数中时候 self 调用下列函数)

1) cache._get_new_version()返回一个时间戳作为 version。

2) cache._read_cache(target_file)尝试从高速缓存中读取目标文件，如果高速缓存中存在该文件,则会返回 file_name,data,version；如果不存在，则会返回 None。

3) cache._write_cache(target_file,data,version)尝试向高速缓存中写入目标文件, 如果目标文件已经存在，则会覆盖写入。

4) cache._search_cache(target_file)在高速缓存中检索文件是否已经缓存。

(4). 注意：

1) 测试数据确保不会出现读取服务器不存在的文件；

2) 测试程序不存在也不要求高速缓存换出的情况；

3) 请勿直接操作 cache 类和 server 类私有变量(除了调用_target_server 外)。

5. 测试说明

平台会对你编写的代码进行测试。

示例 2-1 测试输入样例

```
5 2
write 0 0 10
read 0 0
read 1 0
write 1 0 5
read 0 0
```

第一行两个数字分别代表读写的操作次数 m 和用户数量 n。接下来的 m 行中每一行分为读和写两种操作类型，当操作类型是“write”时，后面三个参数分别是用户编号、文件编号、写入的文件数据(int 类型)；当操作类型是“read”时，后面两个参数分别是用户编号、文件编号。在程序实际运行中，你并不需要对输入进行处理，测试程序会处理输入并按照输入调用 cache 类的 read 和 write 函数。

示例 2-2 预期输出

```
10
10
5
```

每一行的数字是每个“read”操作后返回的文件数据(int 类型)。

2.3 实验方法

本实验模拟了写穿算法的读写过程，设计了一个 Cache 对用户的读写行为和对服务端的访存行为进行模拟。

写穿算法的流程如图 2-1 所示，基于此对代码逻辑进行补全。

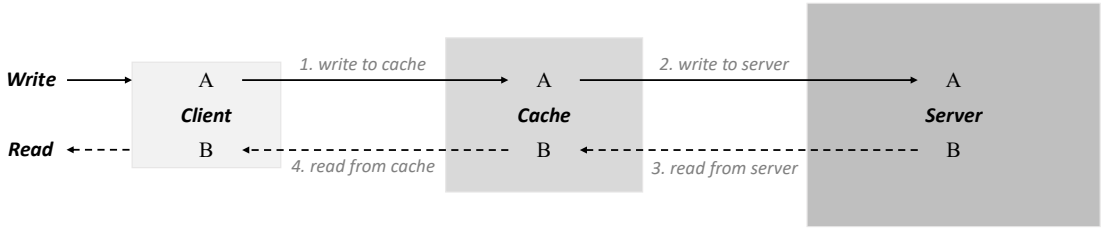

图 2-1 写穿算法流程示意图

2.3.1 Read

读取数据时的流程如图 2-2 所示。具体而言：

1. 检查 Cache 中是否已缓存文件。若已缓存则执行 2，否则执行 3；

2. 读取 Cache 和 Server 中的 version，检验二者是否一致。若一致则返回 cache data，否则将数据更新至 Cache 中后返回 server data；

3. 读取 Server 中的数据，并将其更新至 Cache 中，最后返回 server data。

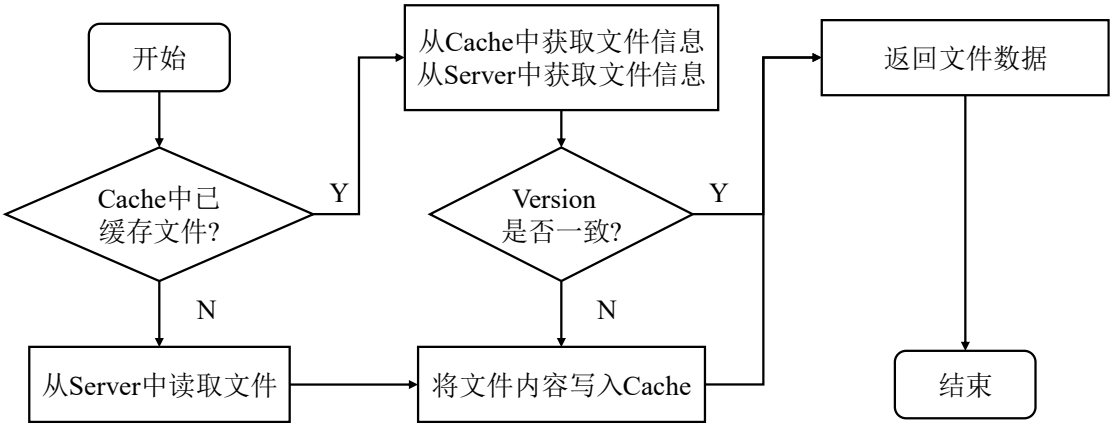

图 2-2 读取数据流程图

基于上述逻辑对代码进行补全。

代码 2-1 read()代码逻辑补全

```
def read(self,target_file):
    #read file from file system
    ####### Begin #######
    #检查缓存中是否已缓存文件
    #向 server 确认 version 是否一致
    #选择从 cache/server 中读取文件
```

```
    #缓存从 server 中读取的文件
    if self._search_cache(target_file):
        _, cache_data, cache_version = self._read_cache(target_file)
        _, server_data, server_version = self._target_server.read(target_file)
        if cache_version == server_version:
            return cache_data
        else:
            self._write_cache(target_file, server_data, server_version)
            return server_data
    elif self._target_server._search_server(target_file):
        _, server_data, server_version = self._target_server.read(target_file)
        self._write_cache(target_file, server_data, server_version)
        return server_data
    ######## End ########
    # there is no file in the system
    return
```

2.3.2 Write

写入数据时的行为如下：

1. 生成新 version；
2. 向 cache 中写入数据并更新 version；
3. 向 server 中写入数据并更新 version。

代码 2-2 write()代码逻辑补全

```
def write(self,target_file,data):
    ####### Begin #######
    #生成新 version
    #向 cache 中写入数据并更新 version
    #向 server 中写入数据并更新 version
    new_version = self._get_new_version()
    self._write_cache(target_file, data, new_version)
    self._target_server.write(target_file, data, new_version)
    ######## End ########
    return
```

2.4 实验结果

educoder 平台中实验结果如图 2-3 所示，顺利完成了该实验。可以获得的测试集给出的测试用例较为简单，读写过程中 Cache 的变化情况也较为简单，在此不展开过多的讨论。

图 2-3 educoder 平台实验通过结果截图

3. 总结与收获

通过本次高级分布式系统实验，了解了高级分布式系统中 Paxos 算法和写穿算法的设计与实现，深刻领会了 Paxos 算法的思想和写穿算法保证的分布式缓存一致性。具体而言，实验一对 Paxos 算法进行了学习和实现，实验二对写穿算法进行了学习和实现。

实验一过程中对 Paxos 算法中 Proposer 和 Acceptor 的行为逻辑和交互过程进行了学习，并根据提示对代码进行了补全，最后顺利完成了实验。通过实验，学习了 Paxos 算法的基本思想和深刻内涵，认识了 Paxos 算法在分布式系统中的基石地位。实验二过程中对写穿算法中的读取(Read)和写入(Write)行为进行了学习，并根据提示对代码进行了补全，最后顺利完成了实验。通过实验，学习了写穿算法的基本思想和深刻内涵，认识了写穿算法如何保证了分布式缓存一致性。

总而言之，通过本次课程实验，学习到了分布式系统中保证一致性的基础算法，简单但有效。感谢课程组老师设计出如此优秀的实验，让我学到了知识，开阔了眼界，对分布式系统中的一致性问题有了感性的认知，对我今后的学习和工作有很大的帮助和指导意义。心有所向，日复一日，必有精进。感谢老师、助教和同学在本次课程实验中对我的帮助，祝大家身体健康、生活愉快。