

**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： M20xxxxxxx**

**姓 名： 张三**

**指导教师： 金海、石宣化**

**报告日期： 20xx.xx.xx**

**计算机科学与技术学院**

**目录**

[1. 实验一 1](#_Toc126609289)

[1.1 实验目的 1](#_Toc126609290)

[1.2 实验内容 1](#_Toc126609291)

[1.3 实验方法 2](#_Toc126609292)

[1.3.1 Proposer 3](#_Toc126609293)

[1.3.2 Acceptor 5](#_Toc126609294)

[1.4 实验结果 6](#_Toc126609295)

[2. 实验二 8](#_Toc126609296)

[2.1 实验目的 8](#_Toc126609297)

[2.2 实验内容 8](#_Toc126609298)

[2.3 实验方法 10](#_Toc126609299)

[2.3.1 Read 10](#_Toc126609300)

[2.3.2 Write 11](#_Toc126609301)

[2.4 实验结果 11](#_Toc126609302)

[3. 总结与收获 13](#_Toc126609303)

# 实验一

## 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。

## 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。

1. 相关知识
2. 角色
3. 提议者(proposer)：进行提议的角色；
4. 批准者(acceptor)：通过提议的角色；
5. 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。

1. 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。
4. 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。

1. 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。**

## 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。

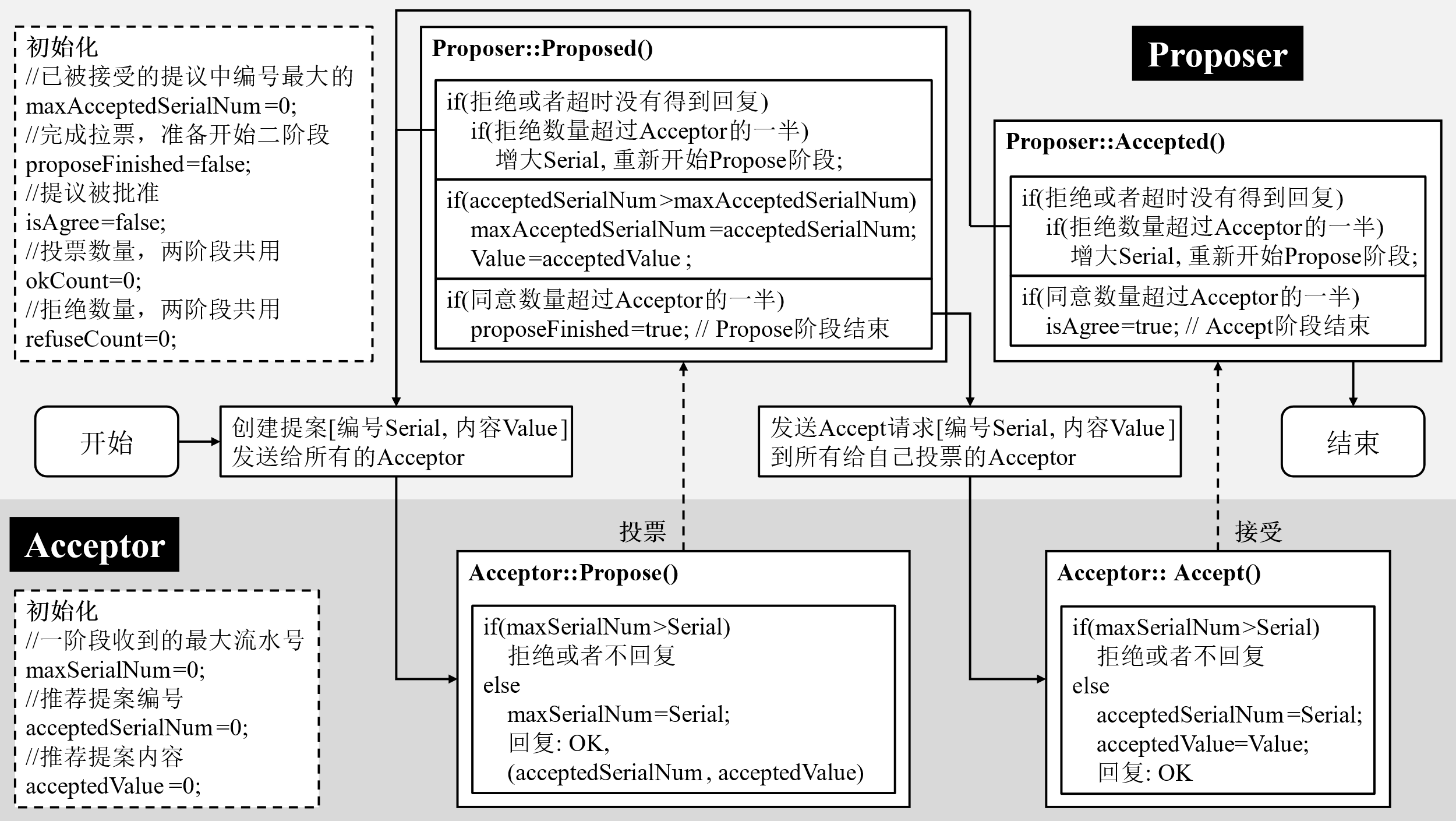


图1‑1 Paxos算法达成一致流程示意图

### 1.3.1 Proposer

1. Proposer在Propose阶段的行为如下：
2. 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；
3. 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；
4. 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;

}

1. Proposer在Accept阶段的行为如下：
2. 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；
3. 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：
2. 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。
3. 提案通过(m\_maxSerialNum<=serialNum)，会执行两个动作：
4. 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；
5. 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;

}

1. Acceptor在Accept阶段的行为如下：
2. 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；
3. 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;

}

## 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。



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。

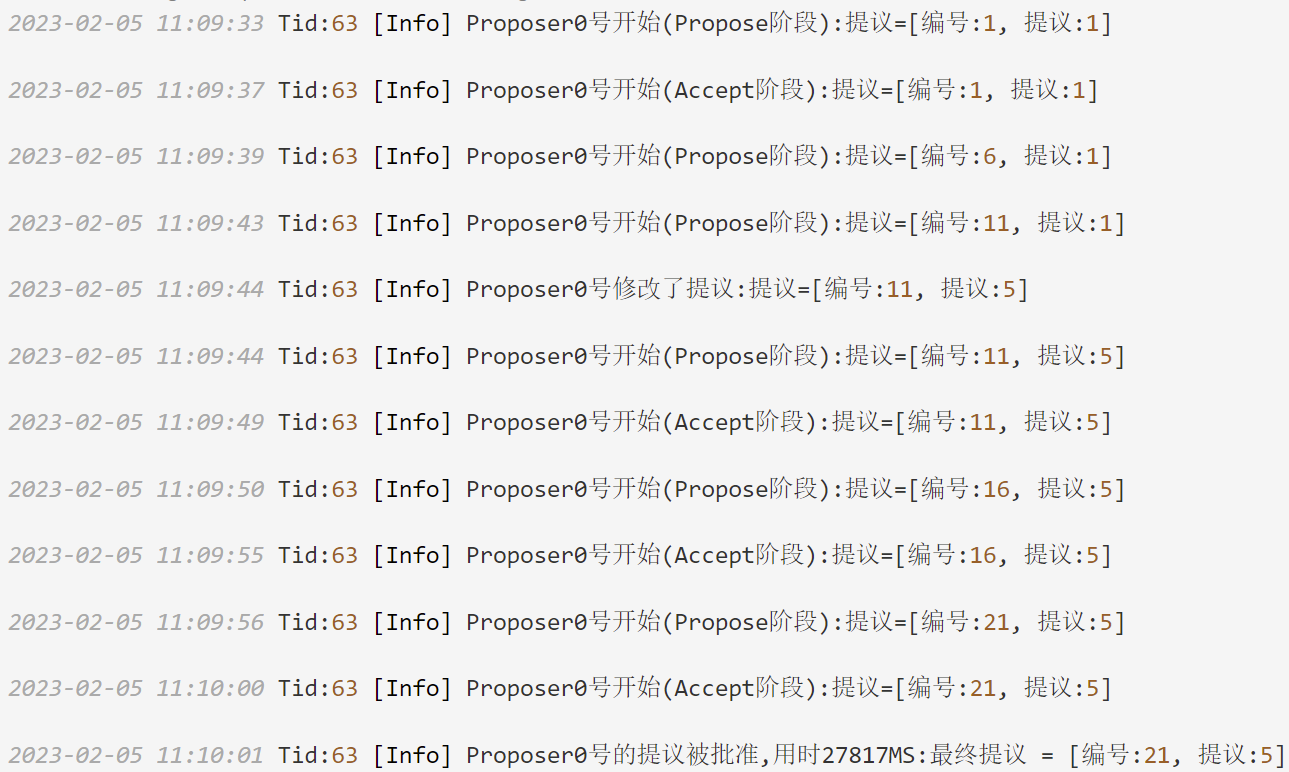


图1‑4 Proposer0的提议-接受过程

类似地，结合图1‑2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。

# 实验二

## 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。

## 实验内容

1. 任务描述

补充程序实现写穿算法。

1. 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
2. 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。
3. 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。
4. 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。

1. 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。

1. 编程提示

类与函数解释

1. sim\_file模拟文件条目类
2. sim\_file.write(data,verion)向文件条目写入data和version。
3. sim\_file.get\_data()获取文件条目数据，返回file\_name,data,version。
4. sim\_file.get\_version()获取文件version。
5. 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
6. 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。
7. 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。
8. server.get\_version(target\_file)尝试向服务器获取目标文件version。
9. cache模拟高速缓存类(在实现的函数中时候self调用下列函数)
10. cache.\_get\_new\_version()返回一个时间戳作为version。
11. 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。
12. 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。
13. cache.\_search\_cache(target\_file)在高速缓存中检索文件是否已经缓存。
14. 注意：
15. 测试数据确保不会出现读取服务器不存在的文件；
16. 测试程序不存在也不要求高速缓存换出的情况；
17. 请勿直接操作cache类和server类私有变量(除了调用\_target\_server外)。
18. 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)。

## 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。

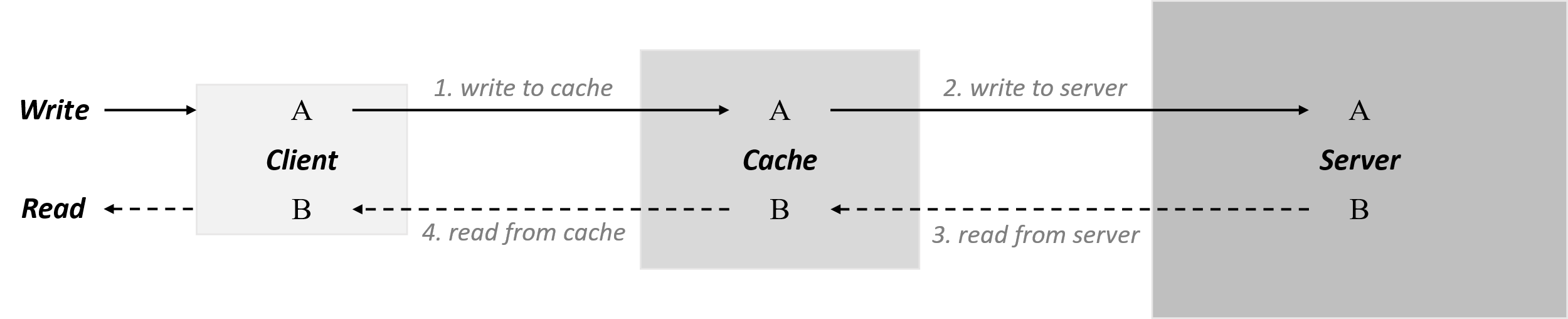


图2‑1 写穿算法流程示意图

### 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。

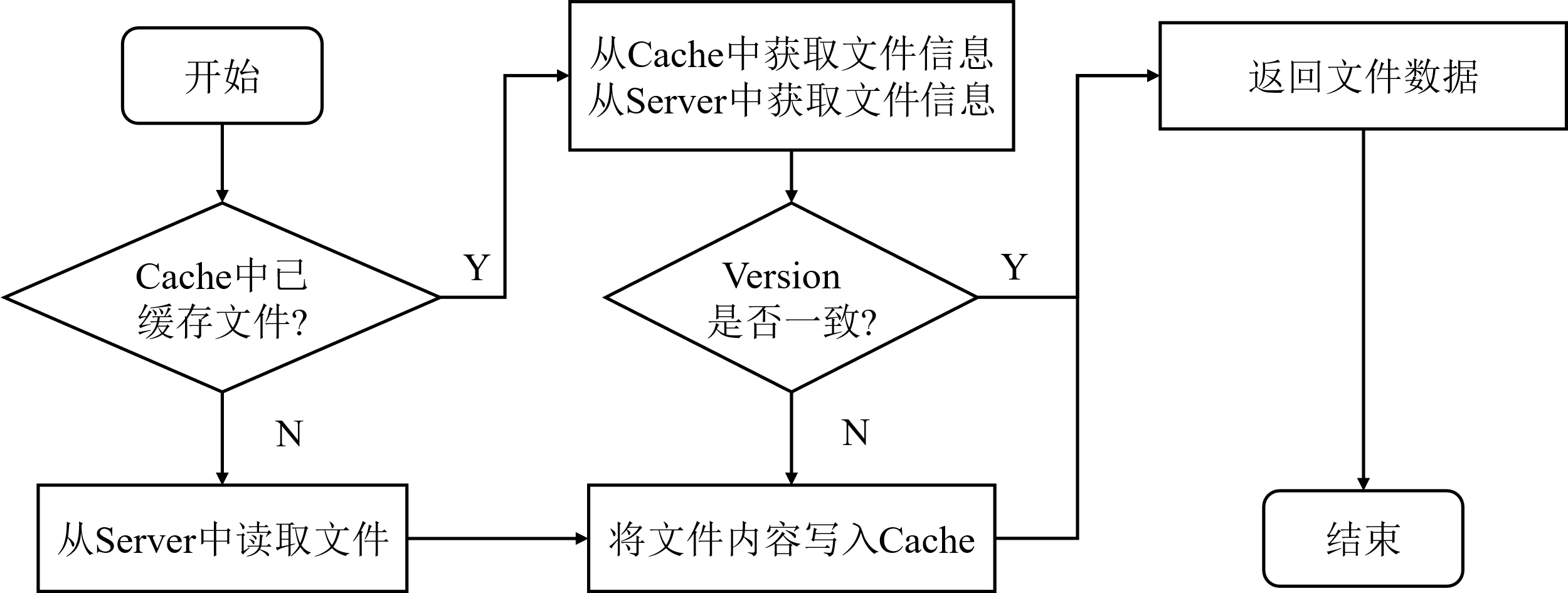


图2‑2 读取数据流程图

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

## 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

# 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。