day10_paxos_practise

为了整理思路，文章采用模拟2人对话方式，如有误，欢迎留言。

阅读本文 你讲获得如下收益。

Q1 Base Paxos解决什么问题？对某个决议达成共识

Q2 社区提问 2阶段提交优化

- 两阶段提交1次日志2rpc疑问
  1. 协调者无状态，不再持久化日志，
  2. prepare阶段完成之间返回客户端 提交成
  3. 这是2句话就解释清楚了。这样优化，具体什么场景不满足，不清楚。

Q3 活锁问题？两个提议者并发执行Basic Paxo出现2个执行顺序。
Q4 日志的连续性问题

Q1 题目1 Base Paxos解决什么问题

问题描述

对于一个由三个节点（S1、S2、S3）组成的Basic Paxos系统，假设其中存在两个提议者（S1和S2）和三个接受者（S1、S2、S3）。

分析以下情况是否可能发生：

1. **步骤（1）**：S1发送Prepare（1.1）消息给S1、S2和S3，并收到成功的响应。
2. **步骤（2）**：S1发送Accept（1.1，X）给S1和S3，且都收到成功的响应，满足多数派条件，S1批准了提案值X，然后S1宕机。
3. **步骤（3）**：S2发送Prepare（2.2）消息给S2和S3，并收到成功的响应。
4. **步骤（4）**：S2发送Accept（2.2，Y）消息给S2和S3，并收到成功的响应，因此S2批准了Y。

分析

核心逻辑：

Base Paxos协议用于在多个副本之间在有限时间内对某个决议达成共识。

一次决议出来的是什么，不是提案编号达成共识，而是提案内容达成共识
是一个值 在多个副本之间达成共识。（可能不同提案编号，但是提案值情况 如下图）

绝不会出现 2个 提议者 对同一个内容 有2个不同决议

> 注意：
>
> 1. 这个和2个客户端写入2个不同内容不要混淆。
> 2. 对整个系统而言 可能存因为延迟，故障等原因出现 2个不同的值瞬间时刻。
>   请问 决议出一个值是核心逻辑 ,根据什么条件呢？举例说明

场景描述：拍卖规则先到先得。

假设有一个在线拍卖系统，三个服务器节点（S1、S2、S3）需要就某个物品的最终成交价格达成一致。其中，S1和S2是提议者，分别代表两个不同的竞拍者提出的出价；S1、S2、S3都是接受者，负责对出价进行投票和确认。

拍卖物品举例说明

假设竞拍者A通过S1提出出价1000元，竞拍者B通过S2提出出价1200元。系统需要通过Basic Paxos协议就最终成交价格达成一致。

1. **S1发起Prepare请求**：
   - S1发送 `Prepare(1)` 给S1、S2、S3。
   - S1、S2、S3都响应了 `Promise` ，并附带已**接受的最高编号的提案**（此时都为空）。
   - S1收到多数派响应后，进入接受阶段。
2. **S1发起Accept请求**：

- S1发送 `Accept(1, 1000)` 给S1和S3。
- S1和S3都接受1000元，并返回 `Accepted` 响应。
- S1收到多数派响应后，认为1000元达成共识。

3. **S2发起Prepare请求**：
   - S2发送 `Prepare(2)` 给S2和S3。
   - S2和S3响应 `Promise` ，**并附带已接受的最高编号的提案（即(1, 1000)）**。
4. **S2发起Accept请求**：
   - S2根据收到的响应，发现最高编号的提案是(1, 1000)，因此发送 `Accept(2, 1000)` 给S2和S3。
   - S2和S3接受1000元，并返回 `Accepted` 响应。
   - S2收到多数派响应后，认为1000元达成共识。

最后决议一致的说明

尽管竞拍者B通过S2提出了更高的出价1200元，但由于在S2发起的Prepare请求中，受者S3已经响应了之前S1发起的Prepare请求，并附带了已接受的1000元出价。

根据Basic Paxos协议，在S2的Accept阶段，必须使用响应中最高编号的提案值，即1000元。

因此，最终所有受者达成一致，认为物品的成交价格是1000元（和女朋友一起看电影，看什么电影不重要，重要是达成一致）

小王提问：这么说 每次拍卖，都是第一个1000喊出的获胜了？

老王回答：不是，分为下面三个情况

- 提案已经被批准（写入大多数节点）
- 提交被接受，提议者可见
- 提交被接受，提议者不可见

由于提案编号 4 大于 3，并且提案已经被批准了，接受者 S3 会回复包含提案值 X 的响应。

（4）S5 会根据 S3 的响应，将提案值 Y 替换成 X，继续向 S3、S4 和 S5 发送 Accept 请求，之后提案再次被批准，但提案值依旧是 X。

2. 情况 2：提案被接受，提议者可见

情况 2 如图 4-7 所示，和情况 1 类似，区别在于此时提案 3.1 还未被批准，只是被 S3 接受，但 S3 仍然会回复包含已经接受的提案值 X 的 PROMISE()响应，所以 S5 仍然会将提案值替换为 X，最终所有接受者对 X 达成共识，虽然提案编号有所不同。

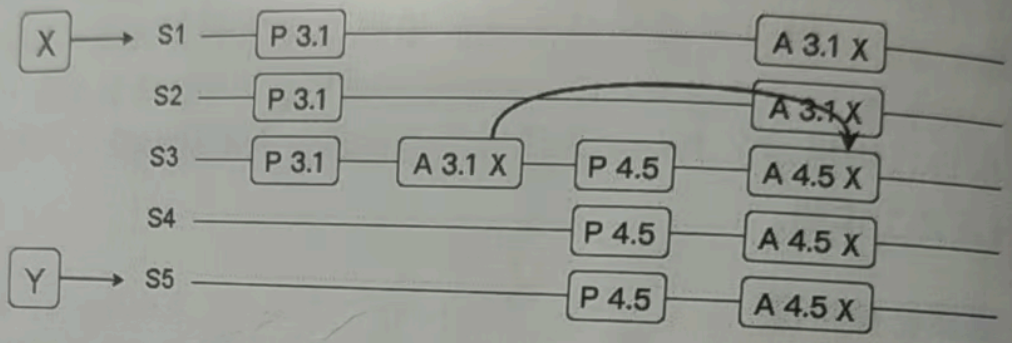


图 4-7

情况 2 还说明了，只要有一个接受者在 Promise()响应中返回了提案值，就要用它来替换提案值。

3. 情况 3：提案被接受，提议者不可见

情况 3 如图 4-8 所示，和情况 2 稍有不同，此处变成 S1 接受了提案，但是 S3 还未接受提案，因此在 S3、S4 和 S5 的 Promise()响应中没有任何提案信息。所以 S5 没有收到任何上一阶段的提案值，可以自行决定提案值为 Y，并在第二阶段发送 Accept(4.5，Y)请求。

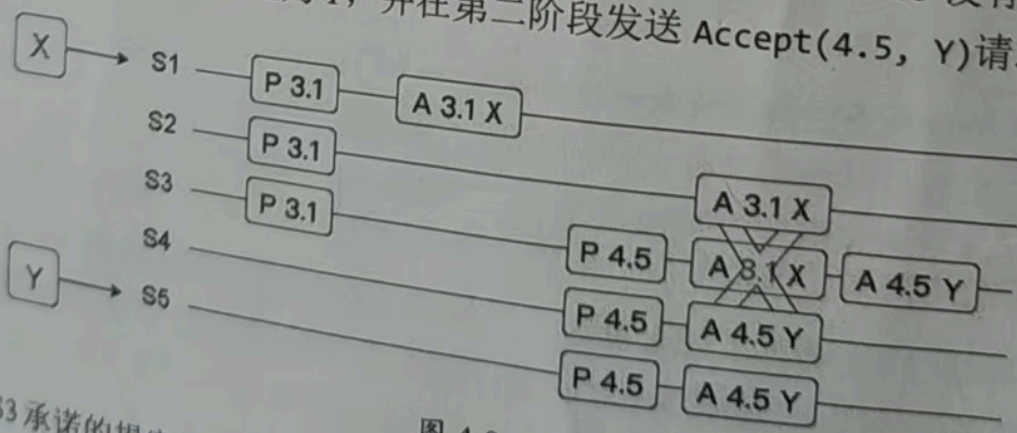
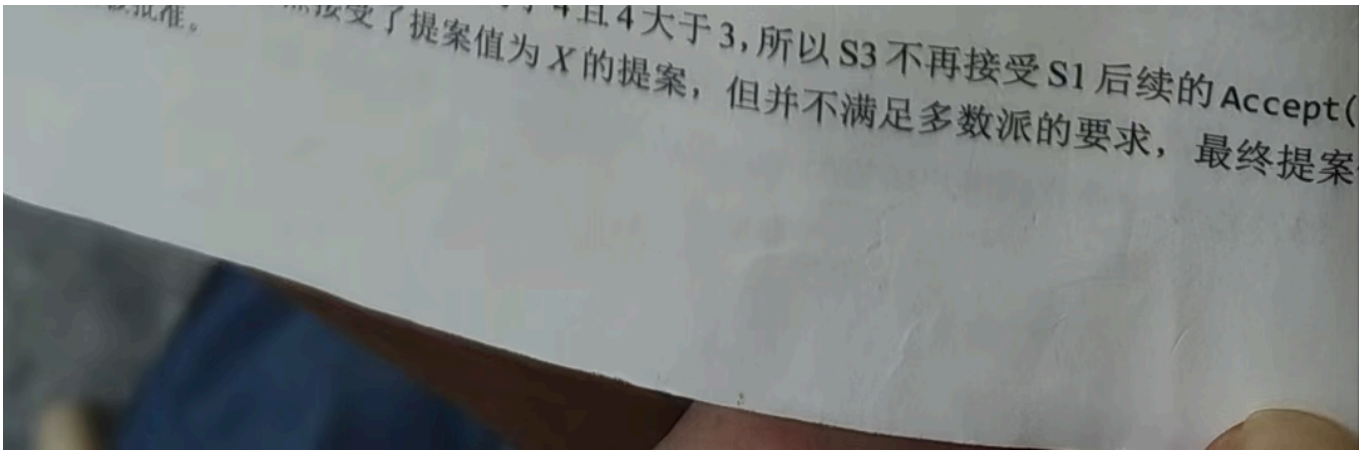


图 4-8

由于此时 S3 承诺的提案编号 n 变为了 4，只
X)请求。虽然有两个节点接受
Y 的提案被



参考答案

不会发生这种情况。
在第（3）步中，S2收到S3的响应时，会得知S3已经接受了提案（1.1，X）。
因此，在第（4）步中，S2会使用X来发起Accept消息，而不是Y。

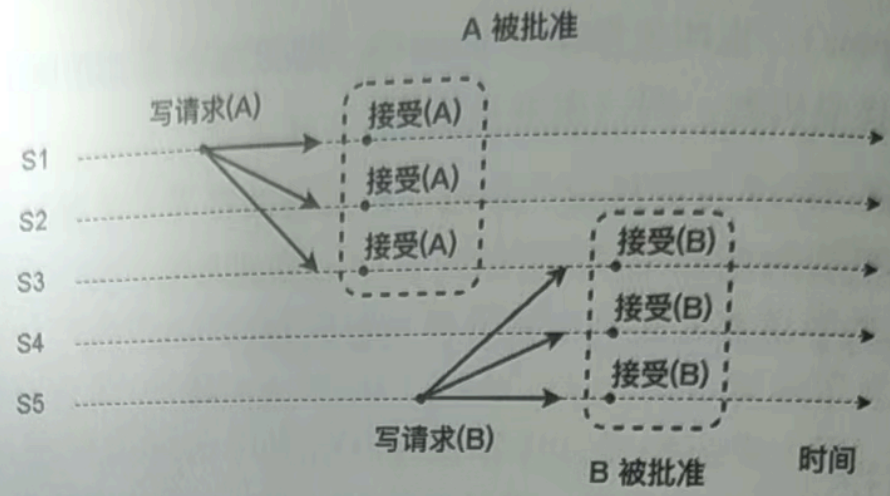


图 4-4

基础的 Paxos 算法强调：一旦一个值被批准了，未来的提案就必须提议相同的提

也就是说，我们讨论的基础 Paxos 只会批准一个提案值。基于此，就需要设计（2-phase）协议，将已经批准的值告知后续的请求，让后续的提案也使用相同的值

如图 4-5 所示这种情况，S3 直接拒绝写请求 B 的值，因为 S3 已经批准了写请求就不会再接受别的值。这样的两阶段协议就可以保证集群只批准一个值，即达成共

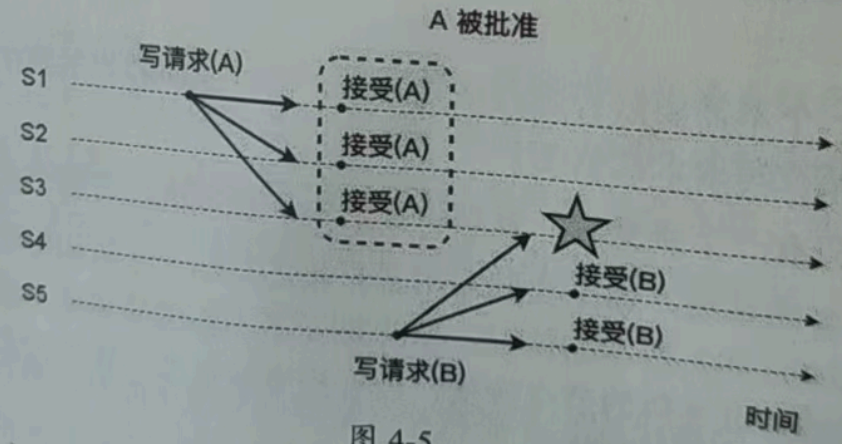
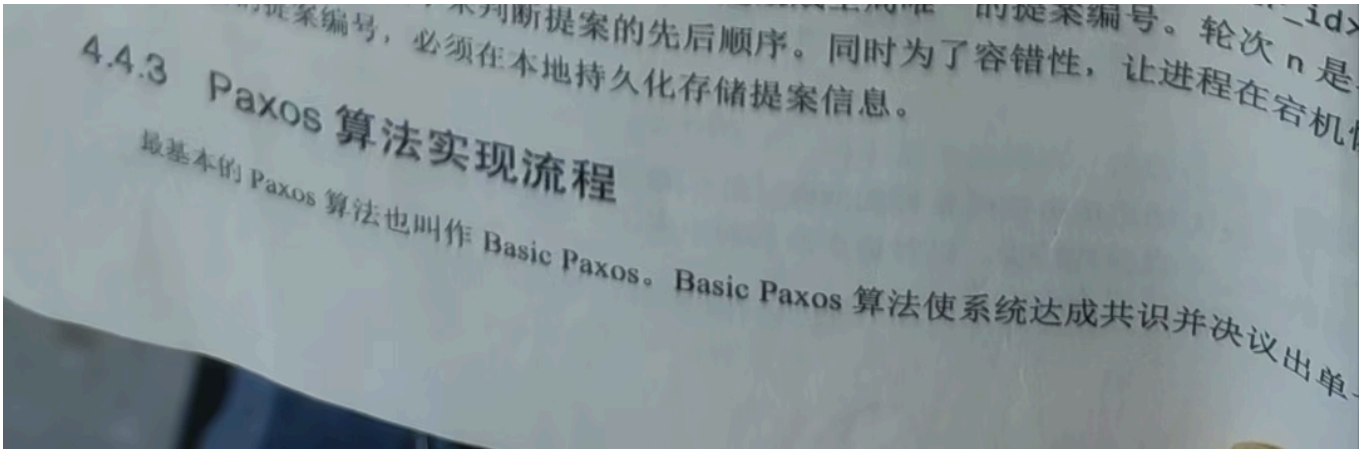


图 4-5

不过在使用这种方式时，我们需要知道提案的先后顺序，在单机系统中通常通过时比较提案的先后顺序，但在分布式系统中直接使用时间戳之类的物理时间可能并不准确（6 章），会存在各种各样的问题。因此，我们不能使用物理时间来判断提案的先后顺序。其次，Paxos 算法通过给每个提案附加一个唯一的编号，即提案编号，如<n，server n 被称为轮次（Round Number），和服务器 id 一起组成全局唯一的提案 增的，这样就能通过 n 的大小来判 仍然知道当前的提



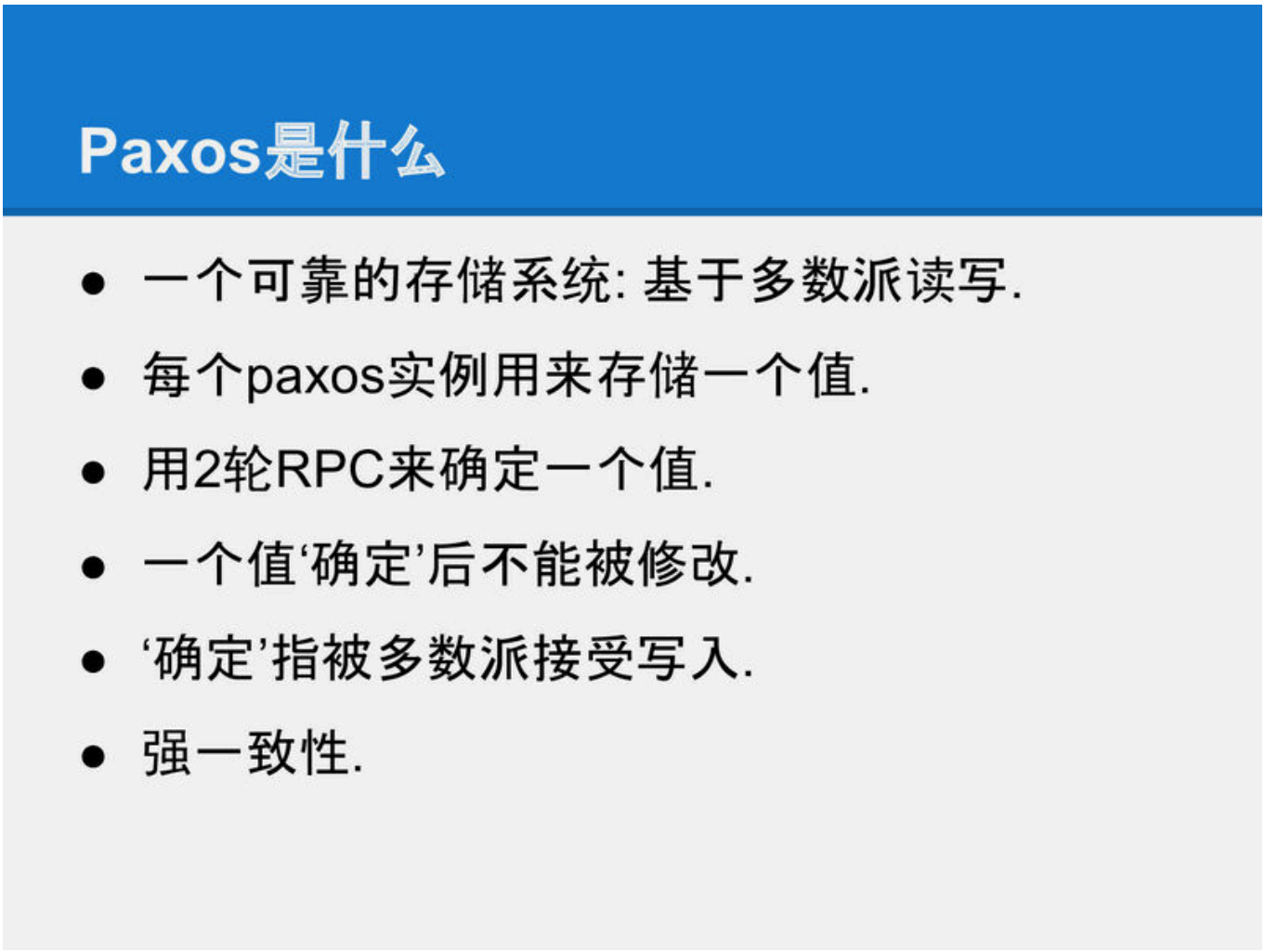
> 提示 上面的 提议者 并发情况2个，在无节点故障情况下
> 情况1：S1 完成 Prepare，Accept阶段 ，S2在执行 Prepare，Accept阶段 S1 提案成功
> 情况2： S1 完成 Prepare，Accept之间，S2开始执行 Prepare，Accept阶段 S2 提案成功。
> 因为延迟原因，S1 提案成功 或者S2提案成功。更加证明 决议出一个值。这个值是什么并“不重要”。

小王提问：通过上面练习题，我感觉不需要2次提交，只要满足一次写入大多数节点也可以完成呢?

老王回答：

请重新看

1. 可靠分布式系统-paxos的直观解释
2. paxos是通过2次 [多数派读写]. 来完成强一致的读写。
   从多数派读写到paxos的推导

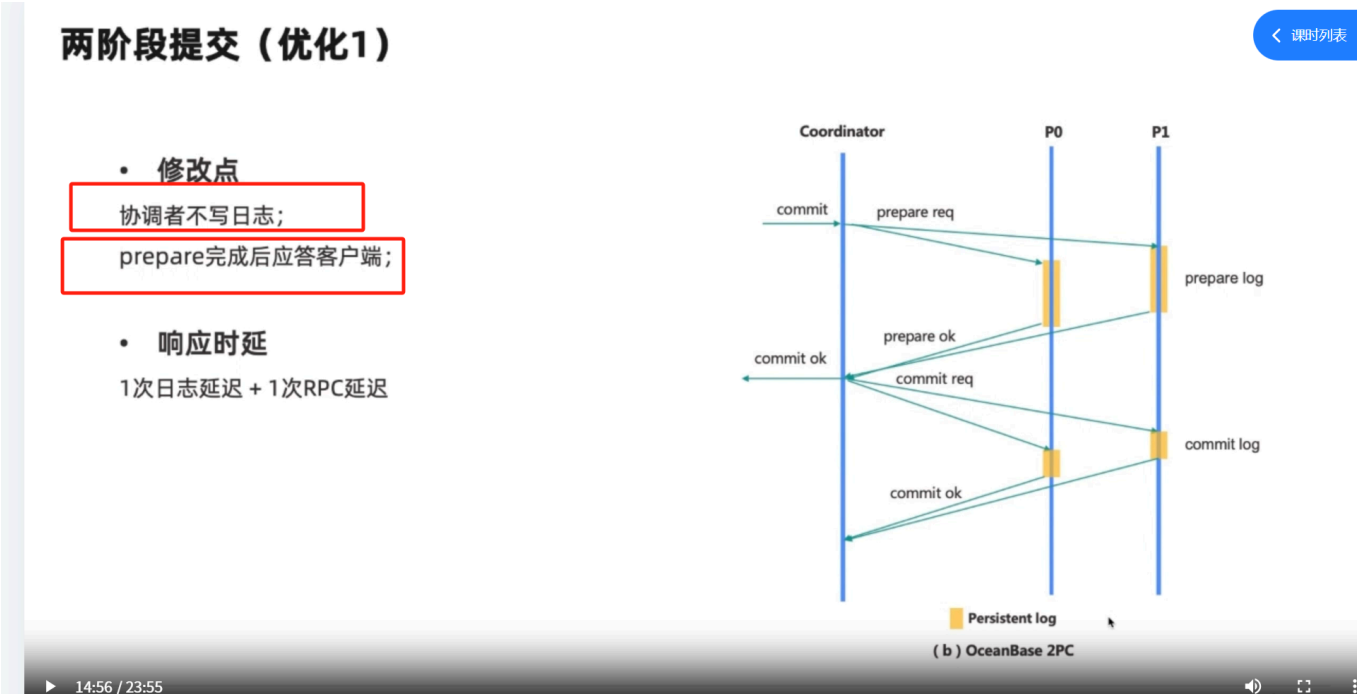


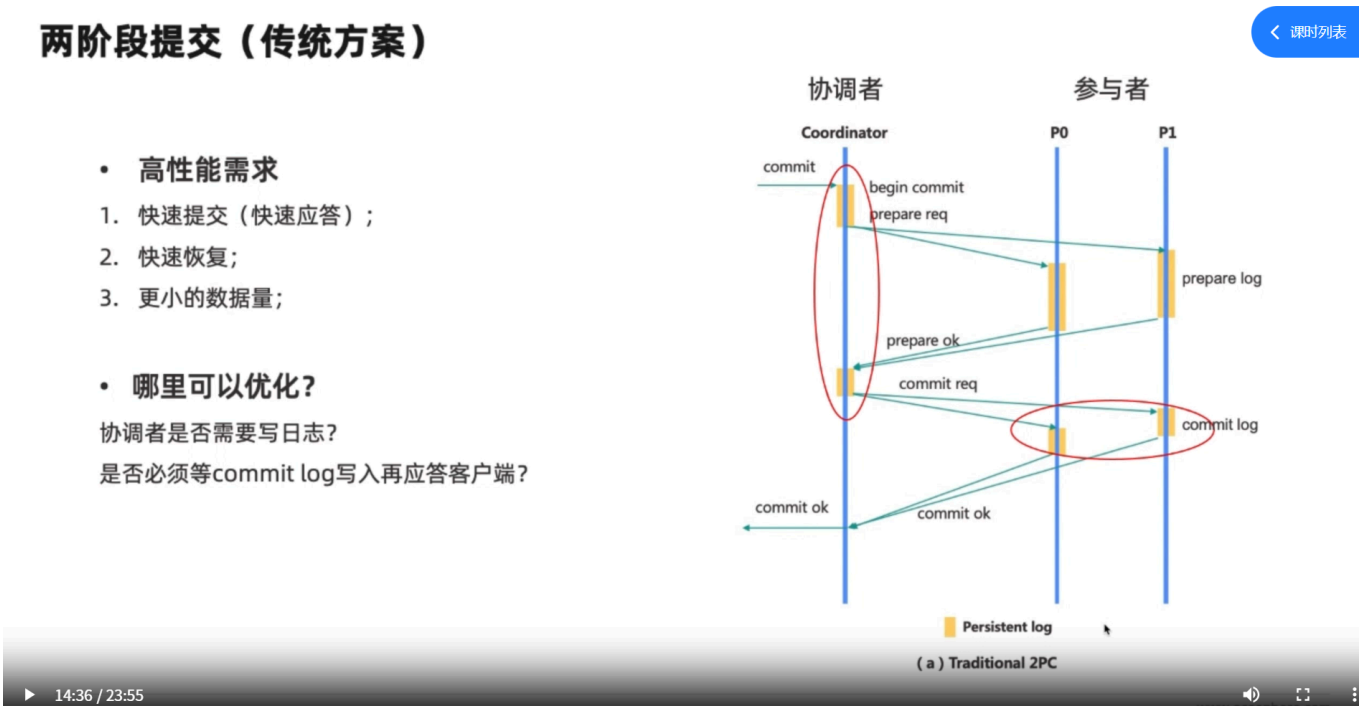
Q2 社区提问 2阶段提交优化 1次日志延迟+1次PRC延迟理解

> 2阶段提交 ob如何响应延迟从4次延迟+2次RPC
> 从 降低到1次日志延迟+1次PRC延迟，这样还能保证数据一致吗? 有什么科学依据

- https://ask.oceanbase.com/t/topic/35612599/20
- - https://www.oceanbase.com/docs/common-oceanbase-database-cn-1000000000821584

参考回答：






```
Coordinator                                                Participant

                              QUERY TO COMMIT
                    ------------------------------->

                              VOTE YES/NO
prepare*/abort*
                    <-------------------------------
```

```
commit*/abort*                COMMIT/ROLLBACK
                  ------------------------------->
                              ACKNOWLEDGEMENT
commit*/abort*
                  <-------------------------------
end
```

为了优化2PC性能，减少关键路径的持久化和RPC次数是关键，一种对经典2PC的优化思路如下：

- 协调者无状态，不再持久化日志，但是为了方便宕机重启后恢复事务状态，需要向每个参与者发送事务的参与者名单并持久化。这样即使协调者宕机，参与者也可以方便地询问其他参与者事务状态了。

[该思路相当于参与者在协调者宕机时，自己担当起协调者询问事务状态的任务]
这句话有问题，在故障后在吗？例如Paxos 和Raft 拒绝这样查询？❌❌❌

- 只要所有参与者prepare成功，事务一定会成功提交。

因此为了减少提交延时，协调者可以在收到所有参与者prepare成功后就返回客户端成功，但如此，读请求可能会因为提交未完成而等待，从而增大读请求的延时

反过来，如果协调者确认所有参与者都提交成功才返回客户端成功，提交延时比较长，但会减少读请求延时

PREPARE 阶段 【保持不变】

协调者：协调者向所有的参与者发起 prepare request

参与者：参与者收到 prepare request 之后，决定是否可以提交，如果可以则持久化 prepare log 并且向协调者返回 prepare 成功，否则返回 prepare 失败。

COMMIT阶段 【响应客户端顺序】

协调者：协调者收齐所有参与者的 prepare ack 之后，

- **进入 COMMIT 状态，向用户返回事务 commit 成功**，

- 然后向所有参与者发送事务 commit request。

参与者：参与者收到 commit request 之后释放资源解行锁，然后提交 commit log，日志持久化完成之后给协调者回复 `commit ok` 消息，最后释放事务上下文并退出。

参考

1. **可靠分布式系统-paxos的直观解释**
   https://zhuanlan.zhihu.com/p/145044486
2. **成为OB贡献者（4）：从单点到多节点i++并发方案**
   https://open.oceanbase.com/blog/14180550530
3. **二阶段提交**
   https://en.wikipedia.org/wiki/Two-phase_commit_protocol

Q3 活锁问题

Paxos系统中的接受者是否可能接受不同的值?

参考答案：有可能。

分析原因：

考虑一个Basic Paxos系统，包含两个提议者（S1和S2）和三个接受者（S1、S2和S3）。以下是其运行过程：
S1 `Accept(1.1, Y)` ，S2 `Accept(2.2, X)`

1. **S1发起提案**
   - S1发送 `Prepare(1.1)` 消息给S1和S2，并收到成功的响应。
   - 在响应中，S1发现没有已接受的提案。
2. **S2发起提案**
   - S2发送 `Prepare(2.2)` 消息给S2和S3，并收到成功的响应。

- 由于S2的提案编号（2.2）更大，S2和S3接受了S2的提案`Accept(2.2, X)`。

3. **S1尝试提交提案**
   - 此时，S1发送`Accept(1.1, Y)`消息给S1和S2。
   - S2拒绝接受该提案，因为S2已经接受了编号更大的提案（2.2）。
   - 然而，S1仍然接受了自己的提案`Accept(1.1, Y)`。

结果

尽管S1接受了`Accept(1.1, Y)`，但S2和S3已经接受了编号更大的提案`Accept(2.2, X)`。

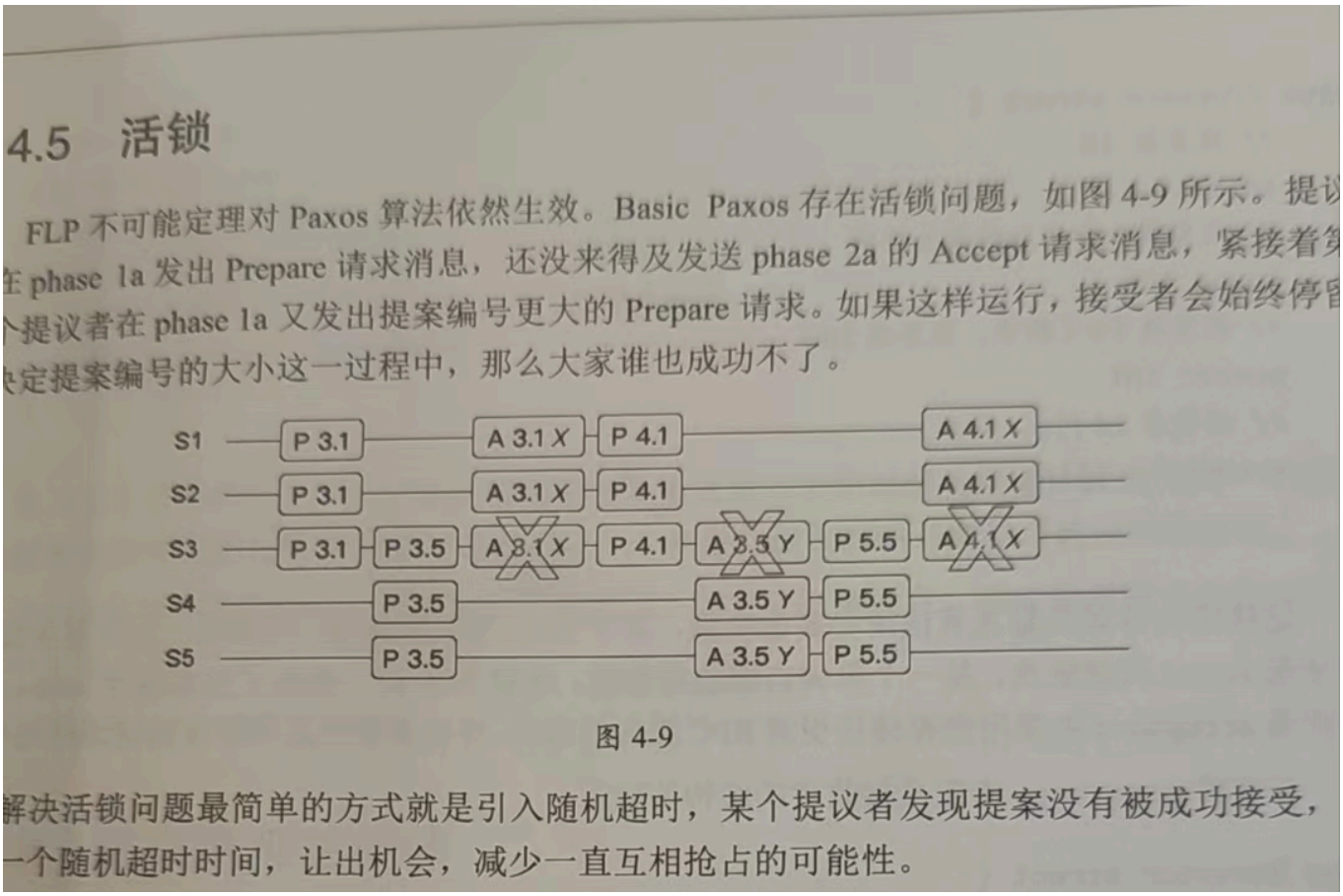
因此，整个系统最终批准的提案值仍然是`X`，成功达成了共识

小王提问：上面例子 和之前例子 说明 服务正常情况下 Accept阶段执行会失败情况

老王：最坏情况 出现活锁问题，这个概念不考虑什么含义

意思是 Prepare Accept 这个是2个独立请RPC请求，中间被其他请求干扰

活锁指的是任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试，失败，尝试，失败



Q4 日志的连续性问题

(10 分) 假设一个 Proposer 以初始值 v1 运行 Basic Paxos，但是它在协议执行过程中或执行后的某个（未知）时间点宕机了。

假设该 Proposer 重新启动并从头开始运行协议，使用之前使用的相同的提案编号，但初始值为 v2，这样安全吗？请解释你的答案。

答案：

不安全。不同的提案必须具有不同的提案编号。

三节点集群为举例说明

1. **S1 发送 Prepare(n=1.1) 消息至 S1 和 S2**
   - S1 向 S1 和 S2 发送 `Prepare(n=1.1)` 消息。

2. **S1 发送 Accept(n=1.1, v=v1) 消息至 S1 并宕机**
   - S1 向 S1 发送 `Accept(n=1.1, v=v1)` 消息。
   - 在此过程中，S1 宕机。（s1接受 ）
3. **S1 重启**
   - S1 重新启动。
4. **S1 发送 Prepare(n=1.1) 消息至 S2 和 S3**
   - S1 向 S2 和 S3 发送 `Prepare(n=1.1)` 消息。
   - S1 收到响应，发现没有任何节点返回被接受的提案。
5. **S1 发送 Accept(n=1.1, v=v2) 消息至 S2 和 S3**
   - S1 向 S2 和 S3 发送 `Accept(n=1.1, v=v2)` 消息。
6. **S1 将 v2 被批准的消息返回给客户端**
   - S1 认为值 `v2` 被批准，并将此消息返回给客户端。
7. **S2 收到新的客户端请求，发送 Prepare(n=2.2) 消息至 S1 和 S2**
   - S2 收到新的客户端请求。
   - S2 向 S1 和 S2 发送 `Prepare(n=2.2)` 消息。
   - S2 收到来自 S1 的响应：`acceptedProposal=1.1, acceptedValue=v1`。
   - S2 收到来自 S2 的响应：`acceptedProposal=1.1, acceptedValue=v2`。
8. **S2 直接选择 v1 作为提案值**
   - S2 根据收到的响应，选择 `v1` 作为提案值。
9. **S2 发送 Accept(n=2.2, v=v1) 消息至 S1、S2 和 S3**
   - S2 向 S1、S2 和 S3 发送 `Accept(n=2.2, v=v1)` 消息。
10. **S2 将 v1 被批准的消息返回给客户端**
    - S2 认为值 `v1` 被批准，并将此消息返回给客户端。

为了整理思路，文章采用模拟2人对话方式，如有误，欢迎留言。

阅读本文 你讲获得如下收益。

Q1 Base Paxos解决什么问题？对某个决议达成共识

Q2 社区提问 2阶段提交优化

- 两阶段提交1次日志2rpc疑问

1. 协调者无状态，不再持久化日志，
2. prepare阶段完成之间返回客户端 提交成
3. 这是2句话就解释清楚了。这样优化，具体什么场景不满足，不清楚。

Q3 活锁问题？两个提议者并发执行Basic Paxo出现2个执行顺序。
Q4 日志的连续性问题

Q1 题目1 Base Paxos解决什么问题

问题描述

对于一个由三个节点（S1、S2、S3）组成的Basic Paxos系统，假设其中存在两个提议者（S1和S2）和三个接受者（S1、S2、S3）。

分析以下情况是否可能发生：

1. **步骤（1）**：S1发送Prepare（1.1）消息给S1、S2和S3，并收到成功的响应。
2. **步骤（2）**：S1发送Accept（1.1，X）给S1和S3，且都收到成功的响应，满足多数派条件，S1批准了提案值X，然后S1宕机。
3. **步骤（3）**：S2发送Prepare（2.2）消息给S2和S3，并收到成功的响应。
4. **步骤（4）**：S2发送Accept（2.2，Y）消息给S2和S3，并收到成功的响应，因此S2批准了Y。

分析

核心逻辑：

Base Paxos协议用于在多个副本之间在有限时间内对某个决议达成共识。

一次决议出来的是什么，不是提案编号达成共识，而是提案内容达成共识
是一个值 在多个副本之间达成共识。（可能不同提案编号，但是提案值情况 如下图）

绝不会出现 2个 提议者 对同一个内容 有2个不同决议

> 注意：

> 1. 这个和2个客户端写入2个不同内容不要混淆。
> 2. 对整个系统而言 可能存因为延迟，故障等原因出现 2个不同的值瞬间时刻。
>
>   请问 决议出一个值是核心逻辑 ,根据什么条件呢？举例说明

场景描述：拍卖规则先到先得。

假设有一个在线拍卖系统，三个服务器节点（S1、S2、S3）需要就某个物品的最终成交价格达成一致。其中，S1和S2是提议者，分别代表两个不同的竞拍者提出的出价；S1、S2、S3都是接受者，负责对出价进行投票和确认。

拍卖物品举例说明

假设竞拍者A通过S1提出出价1000元，竞拍者B通过S2提出出价1200元。系统需要通过Basic Paxos协议就最终成交价格达成一致。

1. **S1发起Prepare请求**：
   - S1发送 `Prepare(1)` 给S1、S2、S3。
   - S1、S2、S3都响应了 `Promise`，并附带已**接受的最高编号的提案**（此时都为空）。
   - S1收到多数派响应后，进入接受阶段。
2. **S1发起Accept请求**：
   - S1发送 `Accept(1, 1000)` 给S1和S3。
   - S1和S3都接受1000元，并返回 `Accepted` 响应。
   - S1收到多数派响应后，认为1000元达成共识。
3. **S2发起Prepare请求**：
   - S2发送 `Prepare(2)` 给S2和S3。
   - S2和S3响应 `Promise`，**并附带已接受的最高编号的提案（即(1, 1000)）**。
4. **S2发起Accept请求**：
   - S2根据收到的响应，发现最高编号的提案是(1, 1000)，因此发送 `Accept(2, 1000)` 给S2和S3。
   - S2和S3接受1000元，并返回 `Accepted` 响应。
   - S2收到多数派响应后，认为1000元达成共识。

最后决议一致的说明

尽管竞拍者B通过S2提出了更高的出价1200元，但由于在S2发起的Prepare请求中，受者S3已经响应了之前S1发起的Prepare请求，并附带了已接受的1000元出价。

根据Basic Paxos协议，在S2的Accept阶段，必须使用响应中最高编号的提案值，即1000元。

因此，最终所有受者达成一致，认为物品的成交价格是1000元（和女朋友一起看电影，看什么电影不重要，重要是达成一致）

小王提问：这么说 每次拍卖，都是第一个1000喊出的获胜了？

老王回答：不是，分为下面三个情况

- 提案已经被批准（写入大多数节点）
- 提交被接受，提议者可见
- 提交被接受，提议者不可见

由于提案编号 4 大于 3，并且提案已经被批准了，接受者 S3 会回复包含提案值 X 的响应。

（4）S5 会根据 S3 的响应，将提案值 Y 替换成 X，继续向 S3、S4 和 S5 发送 Accept 请求，之后提案再次被批准，但提案值依旧是 X。

2. 情况 2：提案被接受，提议者可见

情况 2 如图 4-7 所示，和情况 1 类似，区别在于此时提案 3.1 还未被批准，只是被 S3 接受，但 S3 仍然会回复包含已经接受的提案值 X 的 PROMISE()响应，所以 S5 仍然会将提案值替换为 X，最终所有接受者对 X 达成共识，虽然提案编号有所不同。

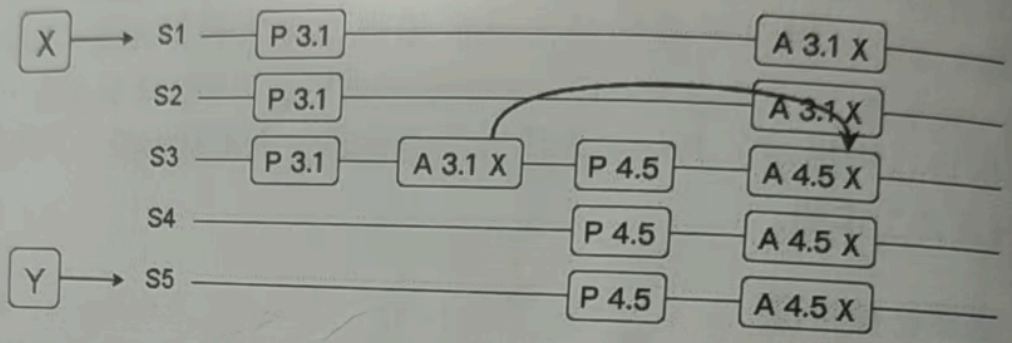


图 4-7

情况 2 还说明了，只要有一个接受者在 Promise()响应中返回了提案值，就要用它来替换提案值。

3. 情况 3：提案被接受，提议者不可见

情况 3 如图 4-8 所示，和情况 2 稍有不同，此处变成 S1 接受了提案，但是 S3 还未接受提案，因此在 S3、S4 和 S5 的 Promise()响应中没有任何提案信息。所以 S5 没有收到任何上一阶段的提案值，可以自行决定提案值为 Y，并在第二阶段发送 Accept(4.5，Y)请求。

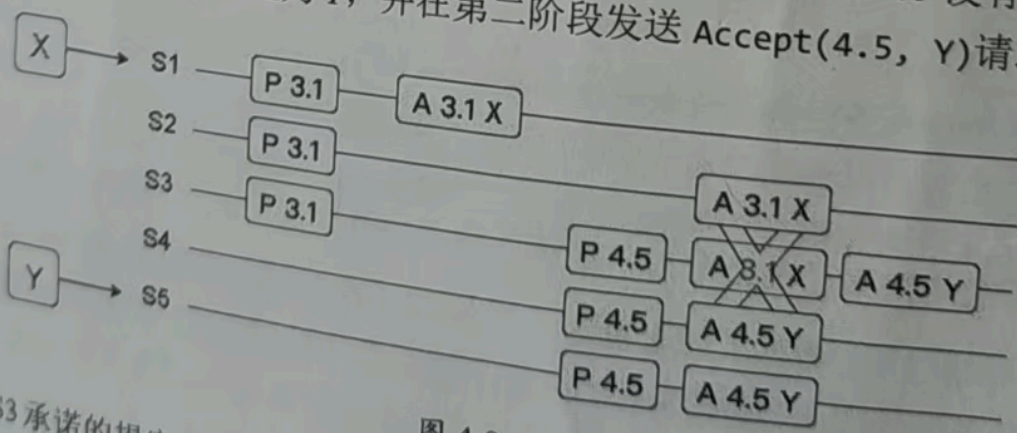
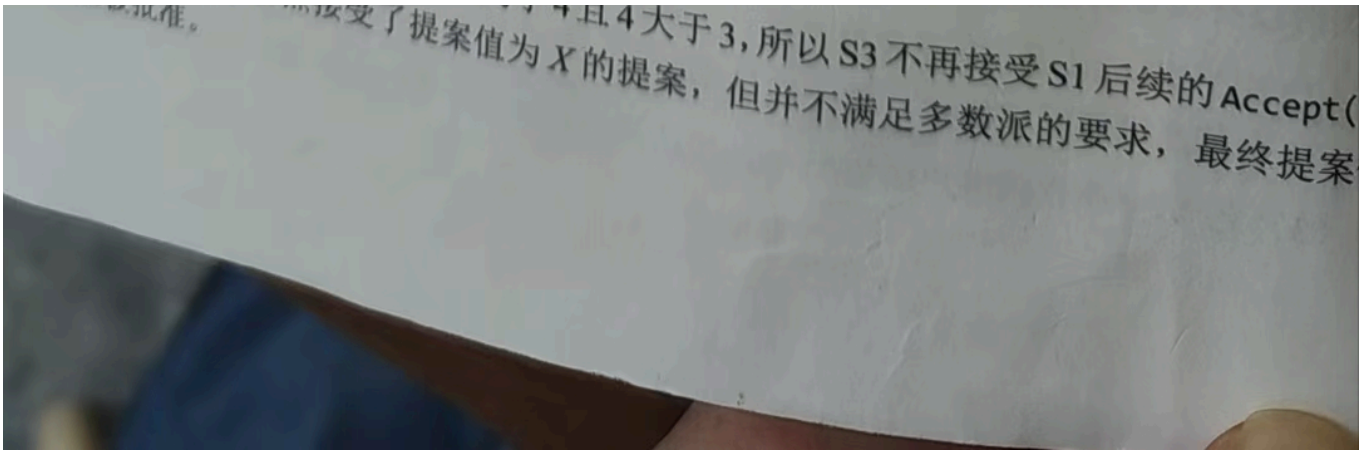


图 4-8

由于此时 S3 承诺的提案编号 n 变为了 4，只
X)请求。虽然有两个节点接受
Y 的提案被



参考答案

不会发生这种情况。
在第（3）步中，S2收到S3的响应时，会得知S3已经接受了提案（1.1，X）。
因此，在第（4）步中，S2会使用X来发起Accept消息，而不是Y。

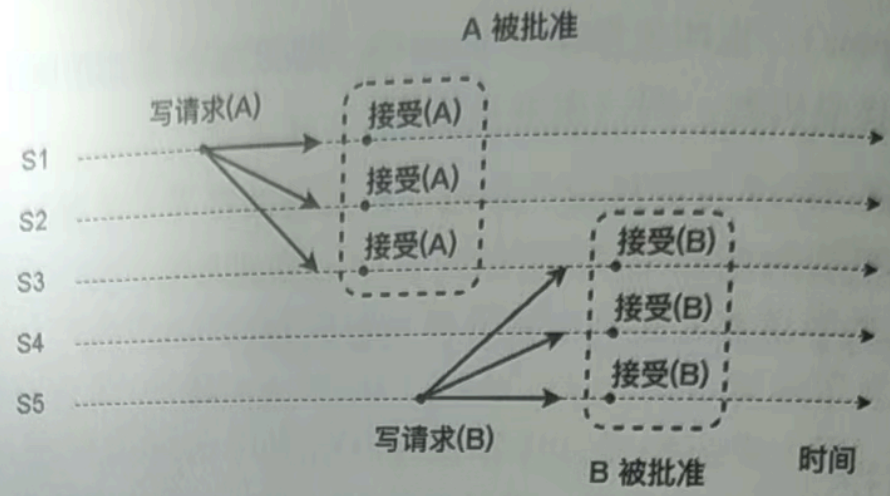


图 4-4

基础的 Paxos 算法强调：一旦一个值被批准了，未来的提案就必须提议相同的提

也就是说，我们讨论的基础 Paxos 只会批准一个提案值。基于此，就需要设计（2-phase）协议，将已经批准的值告知后续的请求，让后续的提案也使用相同的值

如图 4-5 所示这种情况，S3 直接拒绝写请求 B 的值，因为 S3 已经批准了写请求就不会再接受别的值。这样的两阶段协议就可以保证集群只批准一个值，即达成共

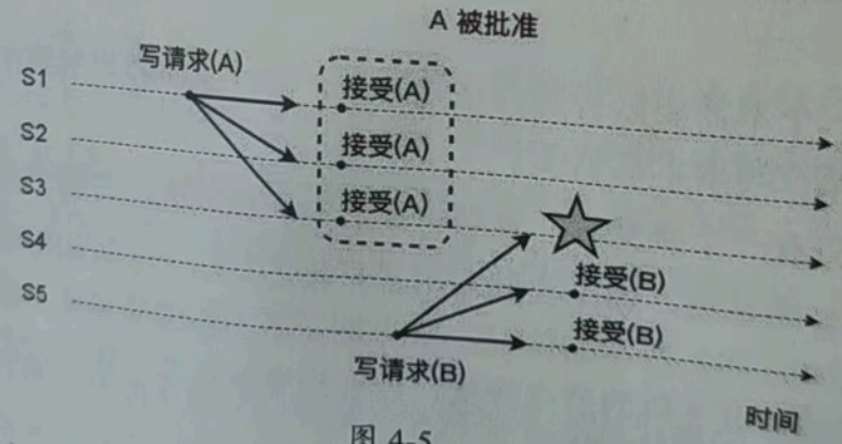
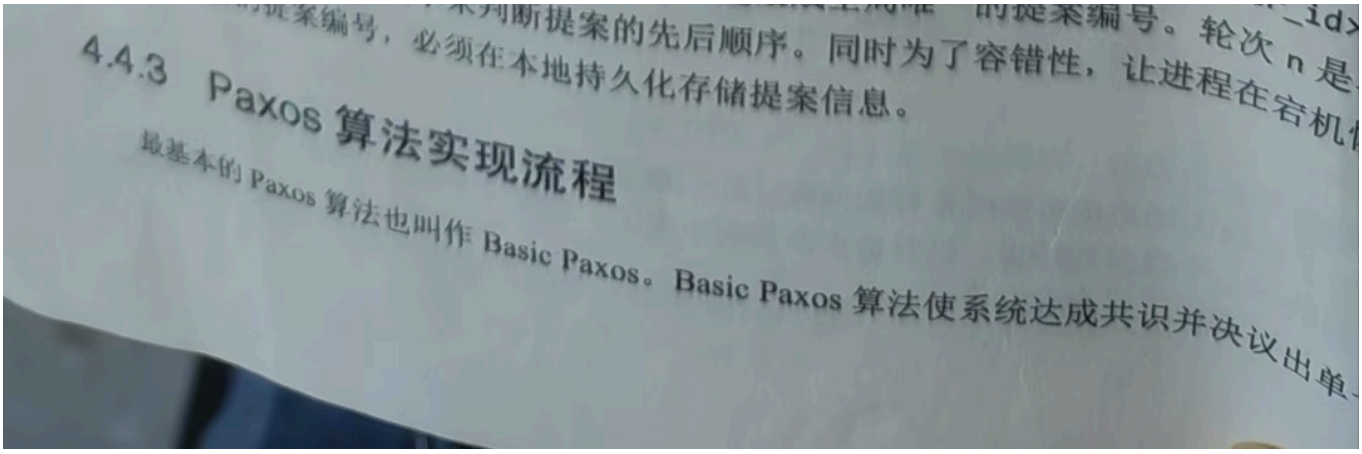


图 4-5

不过在使用这种方式时，我们需要知道提案的先后顺序，在单机系统中通常通过时比较提案的先后顺序，但在分布式系统中直接使用时间戳之类的物理时间可能并不准确（6 章），会存在各种各样的问题。因此，我们不能使用物理时间来判断提案的先后顺序。其次，Paxos 算法通过给每个提案附加一个唯一的编号，即提案编号，如<n，server n 被称为轮次（Round Number），和服务器 id 一起组成全局唯一的提案 增的，这样就能通过 n 的大小来判 仍然知道当前的提



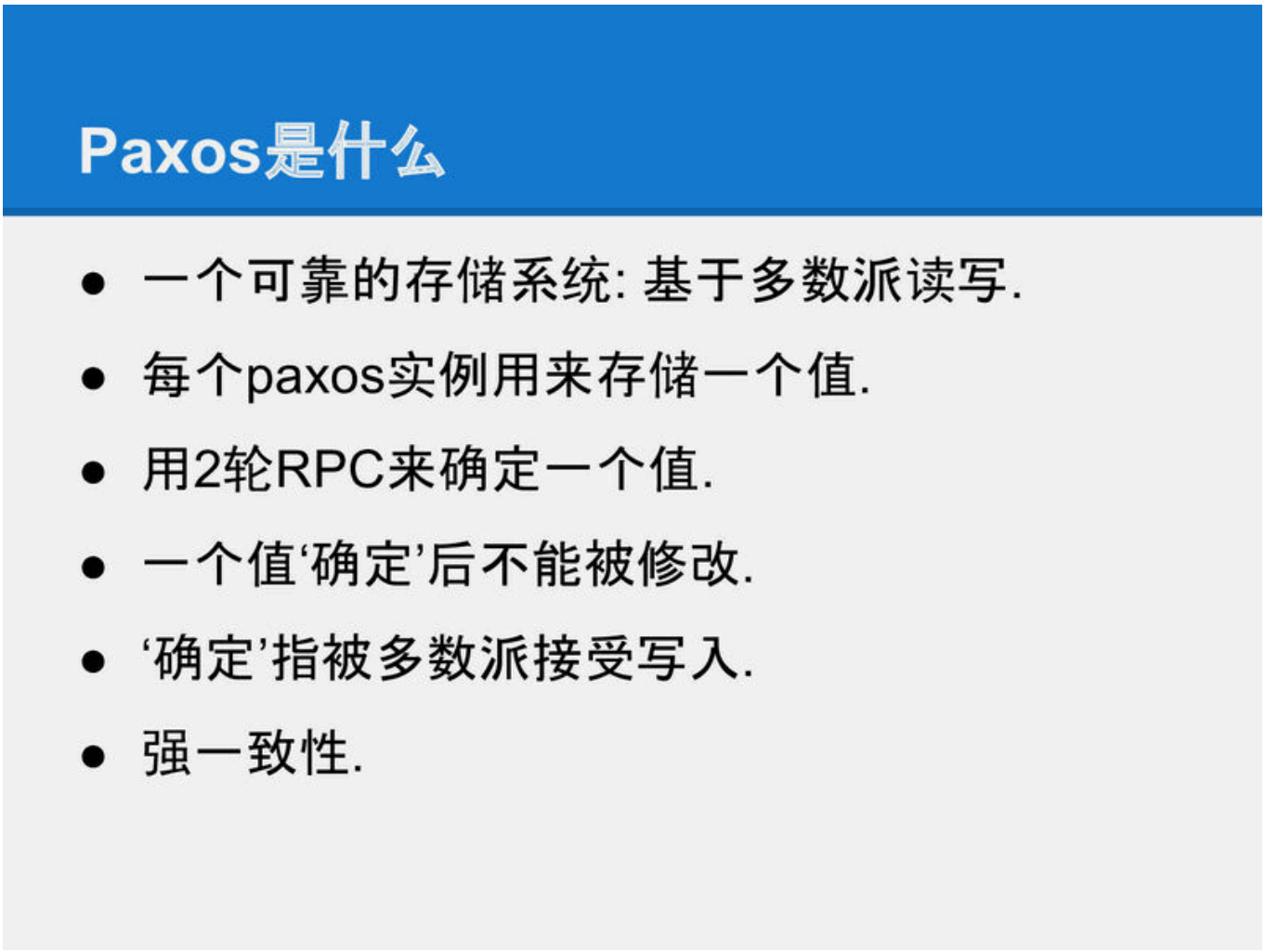
> 提示 上面的 提议者 并发情况2个，在无节点故障情况下
> 情况1：S1 完成 Prepare，Accept阶段 ，S2在执行 Prepare，Accept阶段 S1 提案成功
> 情况2： S1 完成 Prepare，Accept之间，S2开始执行 Prepare，Accept阶段 S2 提案成功。
> 因为延迟原因，S1 提案成功 或者S2提案成功。更加证明 决议出一个值。这个值是什么并“不重要”。

小王提问：通过上面练习题，我感觉不需要2次提交，只要满足一次写入大多数节点也可以完成呢?

老王回答：

请重新看

1. 可靠分布式系统-paxos的直观解释
2. paxos是通过2次 [多数派读写]. 来完成强一致的读写。
   从多数派读写到paxos的推导

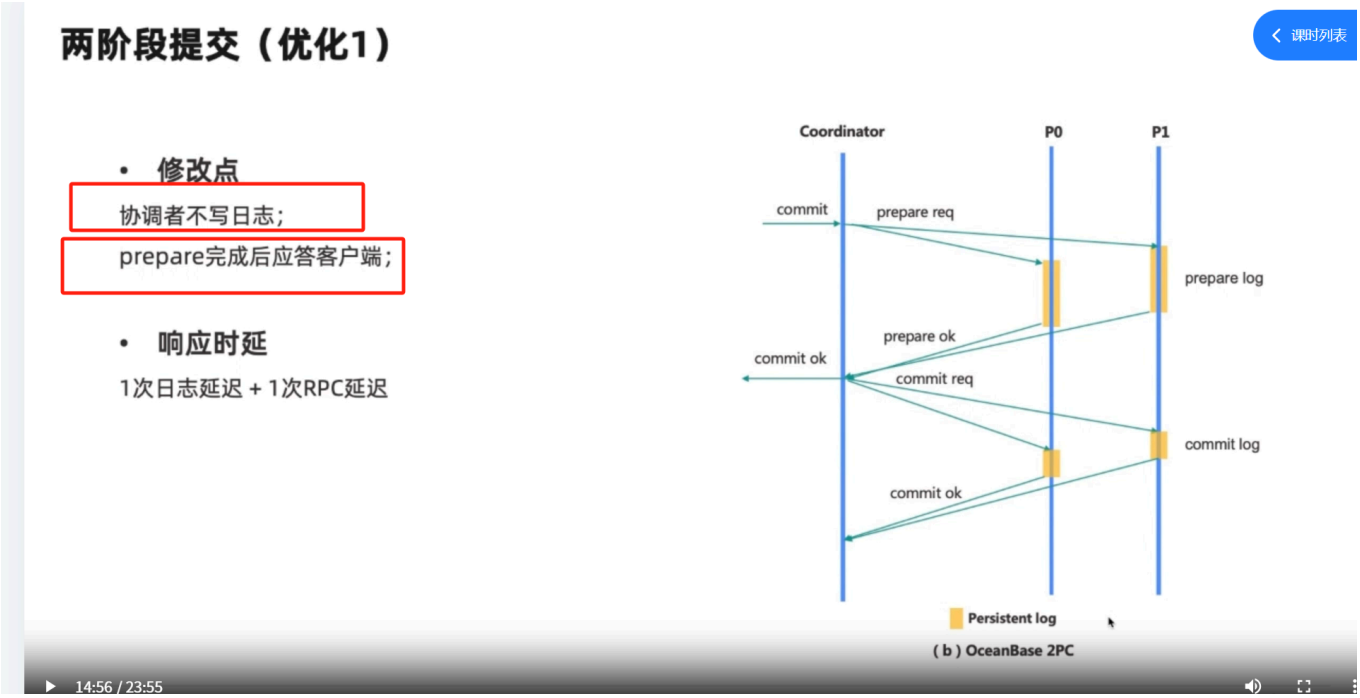


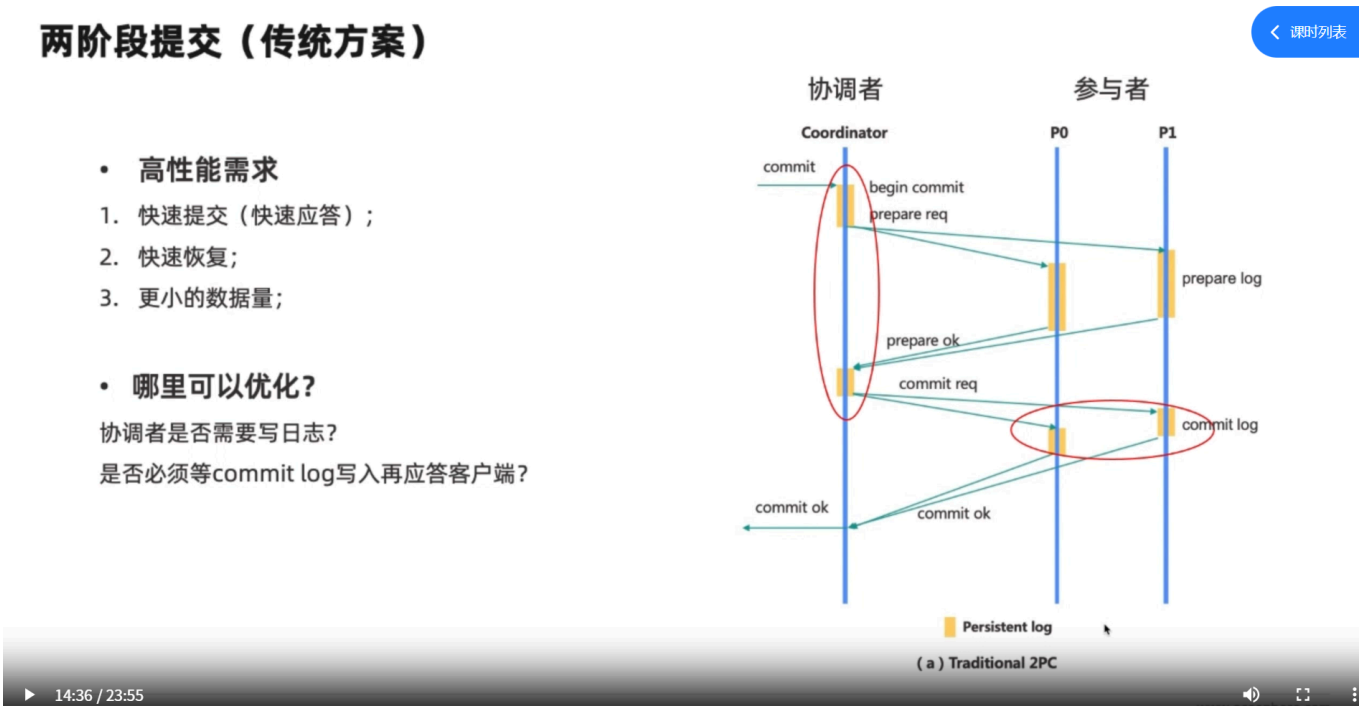
Q2 社区提问 2阶段提交优化 1次日志延迟+1次PRC延迟理解

> 2阶段提交 ob如何响应延迟从4次延迟+2次RPC
> 从 降低到1次日志延迟+1次PRC延迟，这样还能保证数据一致吗? 有什么科学依据

- https://ask.oceanbase.com/t/topic/35612599/20
- - https://www.oceanbase.com/docs/common-oceanbase-database-cn-1000000000821584

参考回答：






```
Coordinator                                                Participant

                              QUERY TO COMMIT
                    ------------------------------->

                              VOTE YES/NO
prepare*/abort*
                    <-------------------------------
```

```
commit*/abort*                COMMIT/ROLLBACK
                  ------------------------------->
                              ACKNOWLEDGEMENT
commit*/abort*
                  <-------------------------------
end
```

为了优化2PC性能，减少关键路径的持久化和RPC次数是关键，一种对经典2PC的优化思路如下：

- 协调者无状态，不再持久化日志，但是为了方便宕机重启后恢复事务状态，需要向每个参与者发送事务的参与者名单并持久化。这样即使协调者宕机，参与者也可以方便地询问其他参与者事务状态了。

[该思路相当于参与者在协调者宕机时，自己担当起协调者询问事务状态的任务]
这句话有问题，在故障后在吗？例如Paxos 和Raft 拒绝这样查询？❌❌❌

- 只要所有参与者prepare成功，事务一定会成功提交。

因此为了减少提交延时，协调者可以在收到所有参与者prepare成功后就返回客户端成功，但如此，读请求可能会因为提交未完成而等待，从而增大读请求的延时

反过来，如果协调者确认所有参与者都提交成功才返回客户端成功，提交延时比较长，但会减少读请求延时

PREPARE 阶段 【保持不变】

协调者：协调者向所有的参与者发起 prepare request

参与者：参与者收到 prepare request 之后，决定是否可以提交，如果可以则持久化 prepare log 并且向协调者返回 prepare 成功，否则返回 prepare 失败。

COMMIT阶段 【响应客户端顺序】

协调者：协调者收齐所有参与者的 prepare ack 之后，

- **进入 COMMIT 状态，向用户返回事务 commit 成功**，

- 然后向所有参与者发送事务 commit request。

参与者：参与者收到 commit request 之后释放资源解行锁，然后提交 commit log，日志持久化完成之后给协调者回复 `commit ok` 消息，最后释放事务上下文并退出。

参考

1. **可靠分布式系统-paxos的直观解释**
   https://zhuanlan.zhihu.com/p/145044486
2. **成为OB贡献者（4）：从单点到多节点i++并发方案**
   https://open.oceanbase.com/blog/14180550530
3. **二阶段提交**
   https://en.wikipedia.org/wiki/Two-phase_commit_protocol

Q3 活锁问题

Paxos系统中的接受者是否可能接受不同的值?

参考答案：有可能。

分析原因：

考虑一个Basic Paxos系统，包含两个提议者（S1和S2）和三个接受者（S1、S2和S3）。以下是其运行过程：
S1 `Accept(1.1, Y)` ，S2 `Accept(2.2, X)`

1. **S1发起提案**
   - S1发送 `Prepare(1.1)` 消息给S1和S2，并收到成功的响应。
   - 在响应中，S1发现没有已接受的提案。
2. **S2发起提案**
   - S2发送 `Prepare(2.2)` 消息给S2和S3，并收到成功的响应。

- 由于S2的提案编号（2.2）更大，S2和S3接受了S2的提案`Accept(2.2, X)`。

3. **S1尝试提交提案**
   - 此时，S1发送`Accept(1.1, Y)`消息给S1和S2。
   - S2拒绝接受该提案，因为S2已经接受了编号更大的提案（2.2）。
   - 然而，S1仍然接受了自己的提案`Accept(1.1, Y)`。

结果

尽管S1接受了`Accept(1.1, Y)`，但S2和S3已经接受了编号更大的提案`Accept(2.2, X)`。

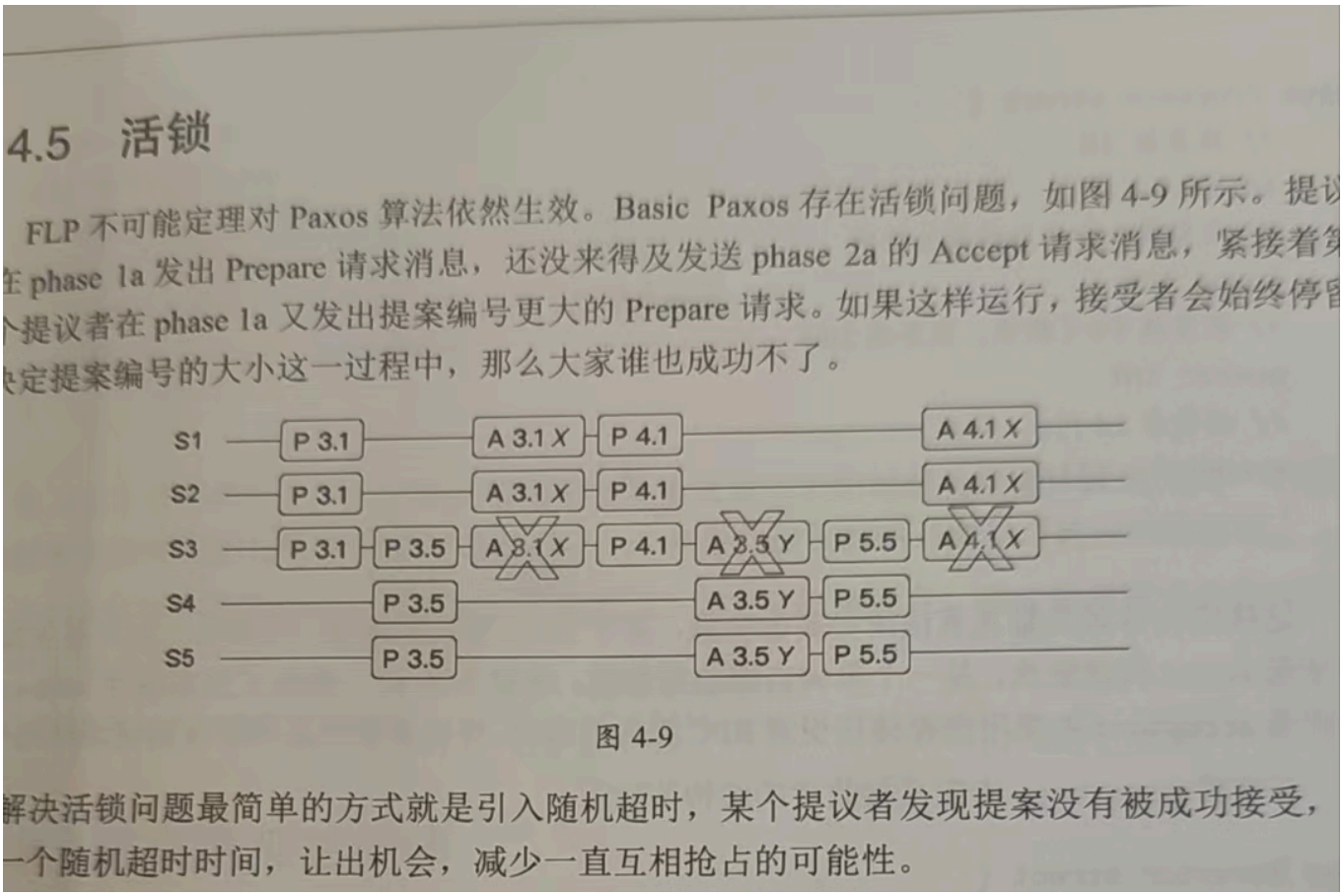
因此，整个系统最终批准的提案值仍然是`X`，成功达成了共识

小王提问：上面例子 和之前例子 说明 服务正常情况下 Accept阶段执行会失败情况

老王：最坏情况 出现活锁问题，这个概念不考虑什么含义

意思是 Prepare Accept 这个是2个独立请RPC请求，中间被其他请求干扰

活锁指的是任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试，失败，尝试，失败



Q4 日志的连续性问题

(10 分) 假设一个 Proposer 以初始值 v1 运行 Basic Paxos，但是它在协议执行过程中或执行后的某个（未知）时间点宕机了。

假设该 Proposer 重新启动并从头开始运行协议，使用之前使用的相同的提案编号，但初始值为 v2，这样安全吗？请解释你的答案。

答案：

不安全。不同的提案必须具有不同的提案编号。

三节点集群为举例说明

1. **S1 发送 Prepare(n=1.1) 消息至 S1 和 S2**
   - S1 向 S1 和 S2 发送 `Prepare(n=1.1)` 消息。

2. **S1 发送 Accept(n=1.1, v=v1) 消息至 S1 并宕机**
   - S1 向 S1 发送 `Accept(n=1.1, v=v1)` 消息。
   - 在此过程中，S1 宕机。（s1接受 ）
3. **S1 重启**
   - S1 重新启动。
4. **S1 发送 Prepare(n=1.1) 消息至 S2 和 S3**
   - S1 向 S2 和 S3 发送 `Prepare(n=1.1)` 消息。
   - S1 收到响应，发现没有任何节点返回被接受的提案。
5. **S1 发送 Accept(n=1.1, v=v2) 消息至 S2 和 S3**
   - S1 向 S2 和 S3 发送 `Accept(n=1.1, v=v2)` 消息。
6. **S1 将 v2 被批准的消息返回给客户端**
   - S1 认为值 `v2` 被批准，并将此消息返回给客户端。
7. **S2 收到新的客户端请求，发送 Prepare(n=2.2) 消息至 S1 和 S2**
   - S2 收到新的客户端请求。
   - S2 向 S1 和 S2 发送 `Prepare(n=2.2)` 消息。
   - S2 收到来自 S1 的响应：`acceptedProposal=1.1, acceptedValue=v1`。
   - S2 收到来自 S2 的响应：`acceptedProposal=1.1, acceptedValue=v2`。
8. **S2 直接选择 v1 作为提案值**
   - S2 根据收到的响应，选择 `v1` 作为提案值。
9. **S2 发送 Accept(n=2.2, v=v1) 消息至 S1、S2 和 S3**
   - S2 向 S1、S2 和 S3 发送 `Accept(n=2.2, v=v1)` 消息。
10. **S2 将 v1 被批准的消息返回给客户端**
    - S2 认为值 `v1` 被批准，并将此消息返回给客户端。