

Q1：如何理解[分布式系统](#)的透明性？

【第一章】

在用户面前呈现为单个计算机系统，隐藏分布式系统底层的实现细节

Q2：持久化通信和非持久化通信的区别，电子邮件和 QQ 留言分别属于哪种通信？

【第四章】

持久化通信：只要传递消息，消息就会存储在通信服务器中

非持久化通信：消息发出后，如果接收方没有成功接收则消息被丢弃

电子邮件和QQ留言都属于持久化通信

Q3：RPC 中客户端崩溃的处理方法有哪些？

【第四章】

1. 终止。客户端在发送RPC之前将操作记录在log entry中，客户端恢复后读取log entry，将孤儿终止
2. 轮回。给时间划分不同的时期，当客户端重启的时候，开启新的时期，并将消息广播出去，服务器收到这个消息的时候，终止任务的计算
3. 过期。给RPC执行一个标准的时间段，未完成任务需要显式申请附加配额

Q4：简述同步通信和[异步通信](#)含义，一个 TCP 连接中数据传输是同步还是异步？为什么？

【第四章】

同步通信：消息发出后需要得到接收方收到的回复，发送方才能继续下一个动作，否则是[阻塞状态](#)

异步通信：消息发出后继续执行下一个操作，不管接收方是否收到

TCP传输时同步通信，在TCP连接中，数据传输的过程是需要等待确认的，即发送方需要等待接收方的确认信号，才能继续发送下一段数据。这个过程中，发送方会阻塞，直到接收到确认信号。这就是所谓的[同步操作](#)。

Q5：结构化 P2P 和非结构化 P2P 的区别是什么，分别有哪些优劣？

【第二章】

结构化P2P：节点按照一定的组织结构排列；组织结构化覆盖网络中的节点，如逻辑环或者[超立方体](#)，并使特定的节点仅根据其ID负责服务

优点：

- 确定的路由
- 高效的搜索
- 负载均衡

缺点：

- 节点引入引出开销大
- 不适合动态环境

非结构化P2P：节点随机选择邻居

优点：

- 简单灵活
- 适应动态环境

缺点：

- 不确定的路由
- 负载不均衡

Q6：流媒体传输的 QOS 指标有哪些？

【第四章】

1. 传输数据需要的[比特率](#)
2. 端到端的最大延时
3. 建立会话前最大延时
4. 往返的最大延时
5. 最大延迟方差或者抖动

Q7：为什么分布式系统使用[逻辑时钟](#)而不是物理时钟？

【第六章】

分布式系统仅考虑节点之间的交互事件的发生顺序达成一致，而不考虑时钟是否按照接近实际时间

Q8：设计时间同步协议 使得同步后时间为所有节点的平均值 写出同步和时间调整的具体流程

【第六章】

1. 选择一个节点作为master，master发送请求到其它的slave节点，询问时钟值
2. 其它节点返回结果，master计算均值
3. 将均值填回其它slave节点（master节点也需要改写成平均时间）

Q9：考虑 3 个进程 ABC，一开始三个进程的逻辑时钟均为 0，然后依次发生以下事件

- a. A 向 B 发送 M1,
- b. A 向 C 发送 M2,
- c. A 向 C 发送 M3,
- d. C 先收到 M3，然后收到 M2，然后向 B 发送 M4,
- e. B 收到 M4，然后向 A 发送 M5,
- f. A 收到 M5，然后向 C 发送 M6,
- g. C 收到 M6。

【第六章】

(1) 写出每个消息的 lamport [时间戳](#);

M1=1;M2=2;M3=3;

M4=6;M5=8;M6=10;

(2) 写出最后三个进程的本地时间;

B=8;A=10;C=11

(3) lamport 时钟是全序关系还是偏序关系?

偏序关系

(4) 写出 M1-M4 的向量时间戳。

M1=[1,0,0]

M2=[2,0,0]

M3=[3,0,0]

M4=[3,0,3]

Q10: 三个进程 P1(x=1, print(y,z)) P2(y=1,print(x,z)) P3(z=1,print(x, y))

(1) 如果满足顺序一致性, 给出两种执行顺序和结果;

(2) 如果满足 FIFO 一致性, 给出一个不满足顺序一致性的执行和结果, 说明原因。

【第六章】

没听课光看PPT的话还蛮抽象的, 这里参考这个[博客](#), 尝试写了一下答案 (不保真)

顺序执行 (同进程的执行顺序不能被打乱; 读出的都是开始写入的程序) :

第一种: x=1;y=1;z=1;print(x,y);print(y,z);print(x,z); 结果111111

第二种: y=1;z=1;x=1;print(y,z);print(x,y);print(x,z) 结果111111

FIFO执行 (每次读到的都是最近写入的数据)

x=1;print(y,z);y=1;z=1;print(x,y);print(x,z) 结果001111

理由: print(y,z)读取y和z的时候没有 读取最新写入的版本

Q11: 下面图中的两个模型是否满足因果一致性和顺序一致性? 如果不满足, 说明理由。

P1:	W(x)a		
P2:	R(x)a	W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

P1:	W(x)a		W(x)c
P2:	R(x)a	W(x)b	
P3:			R(x)c
P4:			R(x)a

(b) 知乎 @Peanuts

(a) 不满足顺序一致性, 因为P3读取b的时候不是最新写入的b; 不是因果一致性, 违反了因果一致的存储

(b) 这条存疑, gpt回复是顺序一致性可以并发读写同一变量, 按照这个思路的话, 就是满足顺序执行条件; 同理, 也是同样的回复。

【第六章】但是不确定, 非常不确定 $\pi \rightsquigarrow \pi \dots$

Q12：门户网站应该采用哪种一致性模型？具体设计一个一致性模型，它具有什么效果和具体要求？

【第七章】

以客户为中心的一致性模型。

参考课件的回答：

效果：

以客户为中心的一致性模型只为单个用户提供一致性保证，并不为不同用户的并发访问提供一致性保证；如果在相当长的一段时间内没有更新操作，所有副本逐渐成为一致

要求：

单调读：进程在t时刻看到x保证在后续操作不会看到x更老的版本

单调写：写操作必须完全串行执行不能交叉

写后读：一个写操作总是在同一个进程后续读操作之前完成，不管这个后续读在什么位置

读后写：对数据项x任何后续写操作，保证发生在x读取值相同或者比之更新的值上

gpt的回答：

具体要求：

1. 弱一致性：由于门户网站通常面向大量用户，并且用户之间的操作相对独立，因此可以容忍一定程度的数据不一致。即时而非强一致性可以提高系统的[可扩展性](#)和性能。
2. 用户体验：确保用户在操作门户网站时，系统的行为对用户来说是可预测和一致的。例如，在提交表单或更新个人资料时，用户期望立即看到更新后的结果。
3. 实时性：门户网站需要尽量减少延迟，以确保用户能够及时获取最新的信息。这可以通过使用缓存、异步处理和[负载均衡](#)等技术来实现。
4. 容错性：系统应该具备一定的容错能力，以应对网络故障、服务不可用等情况，确保用户的操作不会因为局部故障而受影响

设计：

1. 缓存：使用缓存来提高读取操作的性能，减少对后端数据存储系统的访问。可以使用[分布式缓存](#)来实现高可用性和扩展性。
2. 异步处理：将一些非关键的操作异步处理，以减少用户操作的响应时间。例如，将用户提交的数据先保存到[消息队列](#)中，然后异步处理这些消息。
3. 冲突解决：当多个用户同时对同一资源进行操作时，需要设计冲突解决策略，例如使用乐观并发控制（Optimistic Concurrency Control）或[冲突检测](#)与合并等技术。
4. [容错机制](#)：使用复制和故障转移技术来提高系统的容错性。例如，使用主从复制来保证数据的可用性和一致性。

Q13：在没有协调者的非集中式的分布式系统中，一个正确副本一定会给正确的结果，错误的副本可能沉默，也可能给错误的随机值。K容错的情况下想获得正确结果至少需要多少个副本？原因是什么？给出一个获得正确结果的协议

【第八章】

2k+1最极端的情况下，k个故障节点发出同一个消息，则需要k+1个正确节点的信息才能使得最终按照多数投票的结果保持正确

协议：

1. 初始化：假设有 N 个节点参与计算，并且每个节点都有相同的输入数据。
2. 执行计算：每个节点基于相同的输入数据进行计算，并独立地生成一个结果。
3. 结果比较：将所有节点生成的结果进行比较，统计每个结果的出现次数。
4. 多数投票：选择出现次数最多的结果作为最终的结果。
5. 验证：可选步骤，可以通过将最终结果发送给所有节点进行验证，确保每个节点都得到相同的结果。

Q14: Paxos 协议中，（1）Basic Paxos 的假设是什么？（2）什么情况下会活锁？怎么解决？

gpt:(1)可靠的消息传递：Basic Paxos 假设消息在网络中能够可靠地传递，即消息不会丢失、乱序或重复。强一致性：Basic Paxos 假设所有参与者都是诚实的，并且它们遵守协议的规则。这意味着所有参与者都会按照协议的要求执行操作。[进程崩溃](#)：Basic Paxos 假设进程可能会发生崩溃故障，即进程停止发送消息或停止响应消息。当进程崩溃后，它可能在恢复后重新参与协议。消息传递延迟：Basic Paxos 假设消息传递可能会有延迟，即消息可能需要一定的时间才能到达目标参与者。

(2)活锁在 Paxos 中可能发生的一个情况是称为[拜占庭故障](#)的特殊情况。拜占庭故障是指存在一些恶意的或者失效的进程，它们可能会发送错误的或者矛盾的消息，从而导致系统无法达成一致。为了解决活锁问题，Paxos 引入了一些机制和改进：提案编号（Proposal Number）：Paxos 使用提案编号来为每个提案进行排序。通过引入单调递增的提案编号，可以避免活锁问题。投票阶段限制（Phase Restriction）：Paxos 引入了投票阶段（Prepare 和 Accept），并对这些阶段进行了限制。特别是，每个阶段只能进行一次，并且进程必须按照特定的规则进行投票和接受。多数决定（Majority Decision）：Paxos 要求提案必须获得多数接受者的支持才能被接受。这样可以确保只有得到足够多的接受者支持的提案才能进入决定状态，从而避免了活锁。超时机制

（Timeouts）：Paxos 的参与者和提议者可以使用超时机制来检测消息传递延迟或其他故障。通过合理设置超时时间，可以避免进程无限等待或持续进行无效的交互。

Q15: Raft 一致性算法中，（1）简述 leader 的选举过程；（2）简述客户端与 leader 交互，数据最终进入到[状态机](#)同步的过程。

（1）Leader 的选举过程

1. 任期（Term）：每个节点都有一个递增的任期号，用于标识选举的轮次。初始情况下，每个节点的任期号为 0。
2. 选举超时（Election Timeout）：每个节点随机设置一个选举超时时间，在超过该时间后，节点会启动新一轮的选举过程。选举超时时间通常是在几百毫秒到一秒之间。
3. 选举触发：当节点的选举超时时间到达时，节点会进入候选者（Candidate）状态，并开始新的选举。
4. 成为候选者：节点将自己的任期号增加 1，并请求其他节点投票给自己。同时，候选者会重新初始化选举超时时间并开始计时。
5. 请求投票：候选者向其他节点发送请求投票的消息，包含候选者的任期号、候选者的 ID 等信息。
6. 投票响应：收到投票请求的节点在满足一定条件后会对候选者进行投票。节点会判断候选者的任期号是否大于自己的任期号，并且自己没有投票给其他候选者。若满足条件，则将投票授予候选者，并更新自己的任期号。
7. 选举胜利：如果候选者收到了大多数节点（超过半数）的选票，那么该候选者将成为新的 Leader。候选者会向其他节点发送包含自己的任期号的心跳消息，以告知其他节点自己成为了 Leader。
8. 失败处理：如果候选者在选举过程中没有收到足够多的选票或者在选举过程中收到了其他节点成为 Leader 的心跳消息，候选者会重新初始化选举超时时间并进入下一轮选举。

（2）客户端和Leader的交互是什么样的？

1. 客户端发送请求：客户端向当前的 Leader 发送请求，例如写入操作或查询操作。
2. Leader 接收请求：Leader 接收到客户端的请求后，将请求转化为一条日志（Log Entry）并附加到自己的日志中。
3. 日志复制：Leader 将新的日志条目发送给其他节点（Followers）。
4. Follower 接收日志：Followers 接收到 Leader 发送的日志后，将该日志附加到自己的日志中。
5. 日志复制确认：一旦 Follower 成功地将日志条目附加到自己的日志中，它会向 Leader 发送一条确认消息。
6. Leader 日志提交：Leader 在收到大多数（超过半数）Followers 的确认消息后，将该日志条目标记为已提交（committed）。
7. 状态机应用：一旦日志条目被标记为已提交，Leader 将该日志条目应用到自己的状态机中，执行相应的操作。
8. 响应客户端：Leader 将执行结果返回给客户端。
9. 数据同步：Followers 在接收到已提交的日志条目后，也会将该日志条目应用到自己的状态机中，并执行相应的操作，以保持状态机的一致性。

第十六题略（23课件里面没有~）