分布式考试

填空 20

三驾马车：MapReduce，GFS，Bigtable

密集型计算可以分为三种:

数据密集型:这种计算主要操作大量的数据,CPU 利用率较低,瓶颈在于内存带宽和内存访问速度。这种情况下,使用数据并行来解决,如 MapReduce 等。

计算密集型:这种计算主要消耗 CPU 资源,涉及大量计算,CPU 利用率很高。这种情况下,可以使用任务并行来解决,将计算分解为多个任务同时执行。

IO 密集型:这种计算主要等待 IO 操作,如读写文件等,CPU 利用率较低,大部分时间等待 IO。这种情况下,可以使用 IO 并行来解决,如使用多线程或异步 IO 提高 IO 效率。

所以,总结如下:

数据密集型:使用数据并行,如 MapReduce 等。

计算密集型:使用任务并行,将计算分解为多个任务同时执行。

IO 密集型:使用 IO 并行,如多线程或异步 IO 提高 IO 效率。

12. 什么是幂等操作？说说幂等的含义以及其真正的价值。判断集合并、文件记录的插入、 缓冲区的读取、 **HTTP GET** 方法、**POST** 和 **PUT** 是否是幂等的。

幂等操作指重复执行多次与执行一次的效果是相同的操作。在分布式系统设计中，幂等性是一个十分重要的概念。采用幂等设计可以避免由于网络等原因导致操作被重复执行而产生的错误结果。幂等性的真正价值在于增加了系统的可靠性和稳定性。

例如，判断集合并、文件记录的插入、缓冲区的读取、HTTP GET 方法均是幂等的。而 HTTP POST 方法不具备幂等性，因为多次执行可能会导致资源的重复创建。HTTP PUT 方法具有幂等性，因为多次执行会得到相同的结果，即更新资源的状态。

为了保证 API 的幂等性，可以采用一些技巧，比如为每个操作关联一个唯一的标识符，确保每个标识符只能被处理一次。这样即使操作被多次执行，也不会产生错误结果。

(1)允许多个客户端操作共享资源

这种情况下，**对共享资源的操作一定是幂等性操作**， 无论你操作多少次都不会出现不同结果。在这里使用 锁，无外乎就是为了避免重复操作共享资源从而提高效率。

(2)只允许一个客户端操作共享资源

这种情况下， **对共享资源的操作一般是非幂等性操作**。在这种情况下， 如果出现多个客户端操作共享资源， 就可能意味着数据不一致，数据丢失。

幂等操作(Idempotent Operation)是指可以重复执行多次,但结果仍然相同的操作。在分布式系统中,由于网络等原因,同一个操作可能会被重复执行多次,所以设计幂等的操作很重要。

常见的分布式系统的幂等操作有:

- GET 请求:GET 请求获取资源,无论执行多少次,结果都相同。所以 GET 请求是幂等的。

- PUT 请求:PUT 请求是更新/替换资源,多次执行会使用同一输入,所以 PUT 请求也是幂等的。
- DELETE 请求:DELETE 请求是删除资源,无论执行多少次,结果都是资源被删除,所以 DELETE 请求也是幂等的。
- 统计计数器:统计计数器可以通过 CAS(Compare-And-Swap)操作实现,每次对值进行自增时,检查值是否修改过,如果没有修改就更新,如果修改过就重新获取值自增。这种实现方式使统计计数器是幂等的。
- 添加集合成员:向集合添加成员可以采用唯一标识符和 CAS 操作实现,先检查集合中是否已存在该成员,如果不存在才添加,这使得添加集合成员操作是幂等的。
- 并集计算:通过唯一标识符可以实现对并集的幂等求和运算。每次只将不存在的元素加入并集,重复元素不作处理,这使得并集计算成为幂等操作。
- 购物车结算:通过商品唯一标识和数量来实现购物车结算的幂等性,如果商品已存在就累加数量,否则添加新的商品记录。这使得结算操作成为幂等的。

通过 CAS,唯一标识符等机制,可以实现分布式系统中常见资源的幂等操作。

57. 拜占庭问题(Byzantine Problem)讨论的是允许存在少数节点作恶(消息可能被伪造) 场景下的一致性达成问题。拜占庭容错(Byzantine Fault Tolerant，BFT) 算法讨论的是 在拜占庭情况下对系统如何达成共识。

分布式死锁预防策略：wait-die，wound-wait 4-52

分布透明性=分片透明性+位置透明性+局部数据模型透明性（Chapter7-44）

86. 解释分片透明性、复制透明性和位置透明性等三级透明性的区别。

分片透明性是用户不必关心数据的逻辑分片和物理位置分配的细节；复制透明性是用户不必关心数据在各个站点的一致性如何实现的细节；位置透明性是用户不必知道数据存在于哪一个位置上。这三种透明性使得分布式数据库系统具有分布透明性，让用户感觉这是一个集中式数据库，提高了系统的可用性、可靠性和性能（目的）。分布式数据库的数据可以存储在靠近它正常使用的地方，减少响应时间与通讯费用，系统结构灵活，增加新的站比用一个更大的系统代替已有的集中式系统更容易。

注意：反向出题-为了实现分布式数据库操作起来像是一个集中式数据库，需要各种透明性实现。

63. CAP,BASE 以及 ACID 的关系

CAP 定理:CAP 定理指出,分布式系统中最多只能同时满足一致性(Consistency)、可用性(Availability)和分区容错(Partition tolerance)三个需求中的两个。由此可以分为:

- CA 系统:满足一致性和可用性,但不容忍网络分区。
- CP 系统:满足一致性和分区容错,但不保证可用性。
- AP 系统:满足可用性和分区容错,但不保证强一致性。

BASE 理论:BASE 理论是对 CAP 定理的补充,它阐述的分布式系统应具有的基本属性。即:

- 基本可用(Basically Available):系统保证可用性,但不一定实时保证。

- 软状态(Soft state):系统中对象的状态可能不是最新的,需要应用自行决定是否需要刷新。
- 最终一致(Eventually consistent):系统最终会达到一致,但在分区出现期间不会一致。

BASE 理论牺牲强一致性来获得可用性和分区容忍性（可伸缩性）,认为通过软状态和最终一致可以实现足够高的可用性。

ACID 模型:ACID 模型定义了数据库事务应具有的四个特性,即:

- 原子性(Atomicity):事务中的所有操作要么全部成功,要么全部失败回滚。
- 一致性(Consistency):事务完成后,数据库从一个一致状态转换到另一个一致状态。
- 隔离性(Isolation):并发事务之间不会互相影响,各自有隔离的工作空间。
- 持久性(Durability):一旦事务提交,其结果会永久保存。

ACID 模型强调数据的一致性和完整性。但在分布式数据库,难以做到隔离性和 ACID 要求的强一致性。

CAP 定理指出，在一个分布式系统中，一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三个特性不能同时满足。因此，在设计分布式系统时，需要在这三个特性之间做出权衡。当选择其中两个特性时，就必须放弃第三个特性。

BASE 是对于 CAP 定理的一种权衡选择。它放弃了强一致性，而强调可用性和分区容错性。BASE 代表的是 Basically Available（基本可用）、Soft-state（软状态）和 Eventually Consistent（最终一致性）。BASE 适用于那些对数据一致性要求不高的场景，比如 NoSQL 系统、微服务架构下的分布式系统等。

ACID 是传统关系型数据库的事务特性，其中 A 代表原子性（Atomicity）、C 代表一致性（Consistency）、I 代表隔离性（Isolation）和 D 代表持久性（Durability）。ACID 强调强一致性和可用性，但不考虑分区容错性。因此，**ACID 适用于单机数据的事务处理**。

关系：CAP 是描述分布式系统中数据一致性、可用性和分区容错性之间制约关系的三要素，选择其中两个要素就必须对剩下的一个做出牺牲。BASE 和 ACID 是对 CAP 三要素进行取舍后的特殊情况。BASE 强调可用性和分区容错性，放弃强一致性，适用于大部分分布式系统，如 NoSQL 系统和微服务架构下的分布式系统。ACID 是单机数据的事务特性，因为不涉及分区容错，所以在选择可用性和强一致性后可以实现。

简答 5＊8 = 40

【1. raft】

1.强 Leader：Raft 算法使用比其他共识算法更强的 Leader 机制。例如，日志条目只从 Leader 节点流向其他节点，这简化了复制日志的管理，使 Raft 算法更易于理解。

2.Leader 选举：**Raft 算法使用随机定时器的超时**来选举 Leader，这在保持心跳机制的前提下，增加了一些机制，同时可以简单而快速地解决冲突

3.成员变更：Raft 算法使用一种新的联合一致性方法来改变群集中的服务器集合，该方法使得两个不同配置的多数派在转换期间重叠，这使得群集可以在进行配置更改时继续正常运行

4.Raft 算法将共识算法分成了四个独立的模块：**Leader Election（Leader 选举）、Log Replication（日志复制）、Safety（安全性）和 Membership Changes（成员变更）**

5.两个 RPC：
candidate 发起的 RequestVote RPC
leader 发起的 AppendEntries RPC
6.**三个角色：**
follower
candidate
leader

7.Raft 的 Leader 举机制：**基于随机化时钟和心跳机制的超时选举机制**

8.Raft 算法的目的就是通过同步多个机器上的状态进而实现一致性。具体的实现方法就是 **Log Replication** 策略。

9. 成员变化处理：快照（snapshot）；增量压缩

53. 在容错性中，人们定义了一些不同类型的故障，主要的有崩溃性故障、遗漏性故障、定 时性故障、响应性故障及随意性故障等五大类。

分区容错性（Partition tolerance）：指分布式系统在遇到网络分区（即节点之间无法互相通信）时，仍然能够继续运行，并保持数据的正确性和可用性。为了保证分区容错性，分布式系统需要将数据分散存储在多个节点中，这就导致了数据的不一致性和同步问题。

分布式系统的容错性是指系统在面对节点故障、网络故障或其他异常情况时，仍能够保持部分或全部的功能，并且不会对系统的正常运行和其他用户造成影响。

崩溃性故障：指系统出现严重错误或异常，导致系统崩溃或无法正常工作。这种故障往往需要系统管理员或运维人员介入进行修复。

遗漏性故障：指系统在执行任务时，由于某些原因漏掉了某些操作或数据，导致系统出现错误或异常。这种故障可能需要通过检查日志或回溯系统状态来进行修复。

定时性故障：指系统在特定的时间或周期内出现错误或异常，例如定时任务无法按时执行或在某个时间点崩溃。这种故障需要系统管理员或运维人员进行监控和维护，以确保系统在预定时间点正常工作。

响应性故障：指系统在接收用户请求或输入时出现错误或异常，例如无法响应用户的请求或输入。这种故障需要系统管理员或运维人员快速响应，并尽快修复故障，以保证用户的体验和系统的稳定性。

随意性故障：指系统出现意料之外的错误或异常，可能是由于未知的原因或者系统中的隐藏错误导致的。这种故障需要系统管理员或运维人员进行详细的排查和调试，以找出错误的根本原因并进行修复。

崩溃性故障、定时性故障、响应性故障：leader 通过**心跳机制**更新 follower 节点的状态，leader 节点发生崩溃或者没有响应时，**随机化时钟超时**并**重新选举**一个新的 leader 来继续处理日志复制过程。但是如果有半数 machine 都死亡，此时无法选举出 leader，系统失效，只能手动恢复。

遗漏性故障： leader 节点发送的 log 更新消息时消息丢失，造成 log 的不一致；当 follower 再次收到 Leader 的日志更新消息时，它会将该消息与自己的日志进行比较，并将缺失的日志条目复制到自己的日志中；Leader 在提交日志之前崩溃，则新的 Leader 会检查未提交的日志

并将其提交；当 follower 的 log 与 leader 不一致时，leader 会强制覆盖。

随意性故障： Raft 采用了多数派投票机制来处理随意性故障，所以即便有一些机器偶然间发生错误投票，系统任然能够发生作用。

【2.对云计算是什么东西/优点，不足，挑战，涉及技术，3 层次，3 部署模型了解】

9-23/9-25

云计算的三服务层次指的是基础设施即服务（IaaS）、平台即服务（PaaS）、软件即服务（SaaS）。

- 基础设施即服务（IaaS）：提供基础的计算、存储、网络等基础设施，用户可以在此基础设施上构建自己的应用、平台和服务。IaaS 的代表产品有 Amazon Web Services（AWS）的 EC2、S3 等、Microsoft Azure 的虚拟机、Google Cloud Platform 的 Compute Engine 等。
- 平台即服务（PaaS）：在 IaaS 的基础上，提供更高级别的平台服务，如数据库、应用服务器、消息队列等，用户可以在此平台上构建自己的应用。PaaS 的代表产品有 AWS 的 Elastic Beanstalk、Azure 的 App Service、Google Cloud Platform 的 App Engine 等。
- 软件即服务（SaaS）：在 PaaS 的基础上，提供完整的应用程序服务，用户只需使用提供的应用程序，无需关心底层基础设施和平台。SaaS 的代表产品有 Salesforce、Office 365 等。

云计算的三部署模型指的是公有云、私有云和混合云。

- 公有云：云服务提供商通过公共网络向公众提供计算资源和服务，用户可以根据自己的需求灵活使用，按需付费。公有云的代表产品有 AWS、Azure、Google Cloud Platform 等。
- 私有云：云服务提供机构或用户自建云平台，提供专属的计算资源和服务，通常部署在企业内部或数据中心内，提供更高的安全性和可控性，但需要更高的成本和维护。常用的私有云平台有 OpenStack、VMware vSphere 等。
- 混合云：将公有云和私有云结合使用，根据实际需求灵活选择使用公有云或私有云，可以在私有云和公有云之间灵活迁移应用和数据。混合云可以更好地平衡成本和安全性等因素，是企业云战略的重要选择。
- （社区云）

76. Google 的云计算应用均依赖于四个基础组件：1) 分布式文件存储，GFS 2) 并行数 据处理模型 MapReduce 3) 分布式锁 Chubby 4) 结构化数据表 BigTable

Google File System (GFS)：分布式文件存储系统。GFS 是 Google 开发的分布式文件系统，用于存储大规模数据集。

MapReduce：并行数据处理模型。MapReduce 是 Google 开发的一种分布式计算模型，用于处理大规模数据集。它将数据分解成小块，然后在分布式环境中并行处理，最终将结果合并起来。

Chubby：分布式锁服务。Chubby 是 Google 开发的分布式锁服务，用于管理分布式系统中的共享资源。

BigTable：结构化数据表。BigTable 是 Google 开发的分布式数据存储系统，用于存储结构化数据。

77. 对提供者而言， **云计算可以三种部署模式**，即 共有云、私有云 和混合云 。

78. 分布式 是公有云计算基础架构的基石。

79. 当前，几乎所有的知名 IT 提供商、互联网提供商，甚至电信运营商都在向云计算进军， 都在提供相关的云服务。但归纳起来，当前**云提供者可以分为三大类**，即 SaaS 提供商、 和 PaaS、IaaS 提供商。

73. **云体系结构**的开发有如下三层：基础设施层（IaaS）、平台层（PaaS）和应用程序层（SaaS）。这三个开发层使用 云中分配的经虚拟化和标准化的硬件与软件资源实现。

74. 部署基础设施层来支持 IaaS 服务。基础设施层是为支持 PaaS 服务构建云平台层的基础。平台层是为 SaaS 应用而实现应用层的基础。

75. SaaS 是一种基于互联网提供软件服务的应用模式

94. 云计算技术体系结构可以分为哪几层？

云计算技术体系结构分为四层：物理资源层、资源池层、管理中间件层和 SOA 构建层。

物理资源层包括计算机、存储器、网络设施、数据库和软件等基础设施资源；资源池层将大量相同类型的资源构成同构或接近同构的资源池，构建资源池更多是物理资源的集成和管理工作；管理中间件层负责对云计算的资源进行管理，并对众多应用任务进行调度，使资源能够有效、安全地为应用提供服务；SOA 构建层将云计算能力封装成标准的 Web Services 服务，并纳入到 SOA 体系进行管理和使用，包括服务注册、查找、访问和构建服务工作流等。其中，管理中间件和资源池层是云计算技术的最关键部分，SOA 构建层的功能更多依靠外部设施提供。

95. 简述云计算服务的三个层次

云计算服务的三个层次是：**基础设施级服务(IaaS)、平台级服务(PaaS)和软件级服务(SaaS)**。

IaaS(Infrastructure-as-a-Service)：基础设施级服务，消费者通过互联网可以从完善的计算机基础设施获得服务。IaaS 是把数据中心、基础设施等硬件资源通过 Web 分配给用户的商业模式，提供的服务包括计算、存储、网络、安全等。示例产品有 Flexiscale 和 Amazon EC2。

PaaS(Platform-as-a-Service)：平台级服务，提供软件开放运行平台层的服务，使得软件开发人员可以基于平台构建、测试及部署定制应用程序，降低了管理系统的成本。典型服务包括 Storage、Database、Scalability 等。示例产品有 Google App Engine、AWS:S3、Microsoft Azure。

SaaS(Software-as-a-Service)：软件级服务，提供基于 Web 的软件应用服务，用户无需购买软件，而是向提供商租用基于 Web 的软件，来管理企业经营活动。SaaS 模式大大降低了软件尤其是大型软件的使用成本，并且由于软件是托管在服务商的服务器上，减少了客户的管理维护成本，可靠性也更高。示例产品有 Google Docs、CRM、Financial Planning、Human Resources、Word Processing 等。

96. 简述云计算与分布式计算的关系

云计算是分布式计算的一种具体形式，它利用互联网技术将计算资源和服务按需提供给用户。云计算提供了高度灵活的计算、存储和网络资源，用户可以根据需要快速获取和释放这些资源，而无需购买和维护自己的硬件和软件基础设施。云计算基于分布式计算的理念，利用多台计算机共同完成计算任务，同时提供高可用性、高性能和高可扩展性的服务。因此，云计算可以看作是分布式计算的一种高级应用，是分布式计算技术在实际应用中的一种体现。

97. 简述私有云、公用云和混合云的基本概念

私有云是由单个客户所拥有、按需提供的基础设施，该客户控制哪些应用程序在哪里运行，拥有自己的服务器、网络和磁盘，并且可以决定允许哪些用户使用基础设施。

公用云是由第三方运行的云，将来自许多不同客户的作业混合一起部署在云内的服务器、存储系统和其他基础设施上。因此，用户不知道运行其作业的同一台服务器、网络或磁盘上还有哪些用户。

混合云结合了公用云和私有云的优点，客户可以通过一种可控的方式对云资源实现部分拥有，部分与他人共享。混合云通常由私有云和公用云组成，私有部分用于敏感数据和应用程序的保护，公用部分用于处理非敏感数据和应用程序。混合云的目的是为了实现更好的资源利用、更灵活的部署和更高的安全性。

98. 举例描述*aaS 的概念。

1) IaaS，全称 Infrastructure as a Service，基础设施即服务。将多台服务器组成的“云端” 计算资源和存储，作为计量服务提供给用户。它将内存、 I/O、存储和计算能力整合成一个 虚拟的资源池向业界用户提供存储资源和虚拟化服务器等服务。如 Amazon EC2/S3。

2) PaaS，全称 Platform as a Service，平台即服务，把服务器平台或者开发环境作为一种服 务提供的商业模式， 以 SaaS 的模式提交给用户。用户在服务提供商的基础架构上开发程序 并通过网络传送给其他用户(最终用户)。如 Force.com，GoogleAppEngine，Microsoft Windows Azure。

3) SaaS，全称 Software as a Service，软件即服务， 是基于互联网提供软件服务的软件应用 模式。将应用软件统一部署于服务器(集群)，通过网络向用户提供软件。用户根据实际需 求定制或者租用应用软件。消除了企业或者机构购买、构建和维护基础设施和应用程序的投 入。如 Saleforce online CRM。

4) DaaS，全称 Data as a Service，数据即服务， 是继 SaaS，PaaS 之后又一个新的服务概念。

5) MaaS，全称 M2M as a Service ，M2M 即服务，M2M 是将数据从一台终端传送到另一台 终端，也就是就是机器与机器(Machine to Machine) 的对话，是物联网四大支撑技术之一。

6) TaaS，全称 everyTHING As A Service，虚拟化云计算技术,SOA 等技术的结合实现物联 网的泛在即服务。

【3.互斥算法-coral-令牌-非令牌】

10. 两个旅行社甲和乙为旅客到某航空公司订飞机票， 形成互斥的资源是 飞机票。

52. 互斥集中式算法的优点是易于实现、很公平、保证了顺序一致性。而缺点是协作者是单个故障点， 如果它崩溃了，整个系统可能瘫痪。

互斥集中式算法通常由一个中心节点来控制共享资源的访问，该中心节点维护一个互斥集，用于记录当前访问共享资源的进程集合。当一个进程需要访问共享资源时，它必须向中心节点发送请求，中心节点会将该进程加入互斥集，然后向该进程发送访问共享资源的许可。当该进程完成访问后，它会将自己从互斥集中删除，然后通知中心节点将访问共享资源的许可转交给下一个进程。

56. 分布式互斥算法的优点是不会发生死锁与饿死现象，也不存在单个故障点。

64. 分布式互斥算法主要分为基于令牌的算法、非基于令牌的算法和基于 Quorum 的方案

37. 什么是 **Quorum** 机制。

Quorum 机制是一种在分布式系统中应用的权衡机制，其意思是仲裁人数，也就是最小合法人数。在一个 N 个副本的系统中，如果最少要写 W 个副本，最多要读 R 个副本才能读到更新的数据，那么 W+R>N，一般 W+R=N+1。Quorum 机制通常用于选择主副本和中心节点，通过读取 R 个副本中版本号最高的副本来选择新的主副本，但新的主副本不能马上服务，至少要与 W 个副本同步完成后才能进行服务。由于 Quorum 机制无法保证强一致性，因此是一种弱一致性算法。

38. 分布式令牌环算法存在令牌丢失的问题，如果令牌丢失，会导致算法失败， 请将该算 法改进一下，使该算法既能检测到令牌丢失， 也能进行补救。

分布式令牌环算法中可能存在令牌丢失的问题，可以通过指定一个站点作为主动令牌管理站

来进行改进。主动令牌管理站通过超时机制来检测令牌丢失，如果在超时时间内没有检测到令牌，则认为令牌已经丢失，并清除环路上的数据碎片并发出一个新的令牌。此外，为了检测到持续循环的数据帧，管理站在经过的任何一个数据帧上置其监控位为 1。如果管理站检测到一个数据帧的监控位已经置为 1，则说明有某个站未能清除自己发出的数据帧，管理站将清除环路上的残余数据并发出一个新的令牌。通过这种方式，分布式令牌环算法可以检测到令牌丢失并进行补救。

152. 分布式互斥算法

分布式互斥算法通过进程之间的消息交换实现进程对共享资源的互斥访问。当一个进程想要进入临界区时，会向其他进程发送带有时间戳的 Request 消息。其他进程根据自己的状态发送 Reply 消息，进程收到所有 Reply 消息后才能进入临界区。这种方式保证了多个进程对共享资源的互斥访问。

– 如该进程自己不想进入临界区，则**立即发送** Reply 消息

– 如该进程想进入临界区， 则把自己的 Request 消息时间戳与收到的 Request 消息时间戳相比较，

- 如自己的晚，则立即发送 Reply 消息
- 否则，就推迟发送 Reply 消息

59. 为什么使用分布式锁？

在分布式系统中，由于存在多个节点并发地访问共享资源的情况，因此需要一种机制来防止资源的竞争和冲突，以确保数据的一致性和正确性。（这个地方逆向出填空也可以）

使用分布式锁的目的，无外乎就是保证同一时间只有一个客户端可以对共享资源进行操作。 但是 Martin 指出，根据锁的用途还可以细分为以下两类

(1)允许多个客户端操作共享资源

这种情况下，**对共享资源的操作一定是幂等性操作**，无论你操作多少次都不会出现不同结果。在这里使用 锁，无外乎就是为了避免重复操作共享资源从而提高效率。

(2)只允许一个客户端操作共享资源

这种情况下， 对共享资源的操作一般是非幂等性操作。在这种情况下， 如果出现多个客户端操作共享资源， 就可能意味着数据不一致，数据丢失。

60. 什么是分布式锁？如何实现之？

分布式锁的目的是并发控制。

分布式锁的特性是：

互斥性：在任意时刻，只有一个客户端能持有锁；

不会发生死锁：即使有一个客户端在持有锁期间崩溃而没有主动解锁，也能保证其它客户端能加锁； 加锁和解锁必须是同一个客户端；

具有容错性，只要大多数 redis 节点正常运行，客户端就能获取和释放锁。

主要实现有三种途径：

采用 Redis 的 setnx；

采用数据库乐观锁(唯一性索引)；

采用 Zookeeper 分布式锁；

锁重入： 指任意线程在获取到锁之后， 再次获取该锁而不会被该锁所阻塞。关联一个线程持有者+计数 器，重入意味着锁操作的颗粒度为“线程”。

20. 在逻辑时钟算法中， Lamport 定义了一个称作“先发生”的关系，表达式 a→b 表示 a 在 b 之前发生。先发生关系是一个传递关系。

50. 一次将所有的消息以相同的顺序传送给每个接收的多播操作称为全序多播。Lamport 时间戳可以用于以完全分布式的方式实现。

69. 在原子多播里，消息排序通常有 4 种不同的排序方法，它们分别是：不排序的多播、FIFO 顺序的多播、按因果关系排序多播和全序多播。

20. 要使用 **Lamport** 时间戳实现全序多播，是不是每个消息都必须要被严格地确认？

答：不需要，任何类型的消息， 只要它的时间戳大于所接收到的消息的时间戳， 就可以被加 入**消息队列**，使用 Lamport 时间戳实现全序多播。

21. **IBM MQSeries** 以及许多其他消息队列系统中的路由表是人工配置的。描述一种自动完 成配置工作的简单方法。

答： 最简单的是现实使用一个集中的组件， 该组件维护消息队列系统的拓扑结构。它使用一种已知的 路由算法来计算各个队列管理器之间的最佳路由， 然后为每一个队列管理器生成路由表， 这些表可以 由各个管理器分别下载。这种方法适合于队列管理器相对较少但是特别分散的消息队列系统。

88. 实现分布式系统同步的复杂性表现在哪几个方面？说明先发生关系， 并说明在 **LAMPORT** 算法中怎样给事件分配时间。

实现分布式系统同步的复杂性表现在以下几个方面：1. 相关信息分散在多台机器上；2. 进程决策仅依赖于本地信息； 3. 系统中单点故障应避免； 4. 没有公用时钟和其他精确的全局时间资源存在。这些特点表明，在分布式系统中要实现时间同步并不是一件容易的事情。

Lamport 时间戳算法的解决方案是直接使用先发生关系，每条消息都携带发送者的时钟以指出其发送的时间。当消息到达时，接收者的时钟比消息发送者时钟小，接收者就立即将自己的时钟调到比发送者的时间大 1 或更多的值。在给事件分配时间时，需要遵循以下规则：1. 在同一进程中，事件 a 发生在事件 b 之前，则 $C(a) < C(b)$；2. 若事件 a 和事件 b 分别代表发送消息和接收消息，则 $C(a) < C(b)$；3. 对于所有的事件 a 和 b，$C(a) \neq C(b)$。这些规则可以保证每个事件都能够被分配到唯一的时间戳。

✧ 153. 给出 Lamport 时间戳互斥算法的主要思想

Lamport 时间戳互斥算法的主要思想是通过给每个事件分配一个唯一的时间戳来实现进程之间的同步和互斥。当一个进程想要进入临界区时，会向其他进程发送带有时间戳的申请资源报文，并把自己的申请报文放到自己的申请队列中。其他进程收到申请报文后，会向申请进程发送带有时间戳的承认报文，并把申请报文放到自己的申请队列中。当申请进程收到所有其他进程的承认报文，并且它的申请报文排在所有其他进程的申请报文前面时，它就可以进入临界区了。当进程释放资源时，会向其他进程发送带有时间戳的释放资源报文，并从自己的申请队列中删除对应的申请报文。使用 Lamport 时间戳互斥算法，进入一次临界区共需要 3(n-1) 个报文，n 为参加互斥的进程数。

38. 分布式令牌环算法存在令牌丢失的问题，如果令牌丢失，会导致算法失败，请将该算法改进一下，使该算法既能检测到令牌丢失，也能进行补救。

分布式令牌环算法中可能存在令牌丢失的问题，可以通过指定一个站点作为主动令牌管理站来进行改进。主动令牌管理站通过超时机制来检测令牌丢失，如果在超时时间内没有检测到令牌，则认为令牌已经丢失，并清除环路上的数据碎片并发出一个新的令牌。此外，为了检测到持续循环的数据帧，管理站在经过的任何一个数据帧上置其监控位为 1。如果管理站检测到一个数据帧的监控位已经置为 1，则说明有某个站未能清除自己发出的数据帧，管理站将清除环路上的残余数据并发出一个新的令牌。通过这种方式，分布式令牌环算法可以检测到令牌丢失并进行补救。

40. 由于分布计算系统包含多个(可能是不同种类的）分散的、自治的处理资源，要想把它们组织成一个整体，最有效地完成一个共同的任务，做到这一点比起传统的集中式的单机系统要困难得多， 需要解决很多新问题。这些问题主要表现在哪些方面？

资源的多重性：处理资源的多重性可能导致差错类型和次数增多，例如部分失效问题和多副本信息一致性问题，同时也使得资源管理变得更加困难。

资源的分散性：资源在地理上分散，进程之间通信采用报文传递方式导致不可预测的、有时是巨大的延迟，状态信息分布在各个节点上，使得系统的控制和同步问题变得复杂，难以及时搜集到完整的信息，从而使处理机进行最佳调度相当困难。

系统的异构性：异构性分布计算系统中不同资源的数据表示和编码、控制方式等不相同，导致翻译、命名、保护和共享等新问题。

RPC 调用问题：在分布式系统中，不同的节点之间需要进行远程过程调用（RPC），但是网络通信可能存在延迟、丢包等问题，导致调用失败或者出现超时等情况。

80. 选举算法中环算法的思想

环算法是一种分布式选举算法，用于在一个环形网络中选举一个进程作为协调者或领导者。每个进程按照进程号的顺序形成一个环，当需要选举一个协调者时，进程发送 ELECTION 消息给其后继进程，并将自己的进程号加入该消息中，直到消息回到进程 P，选出其中最大的进程号作为新的协调者。为了避免进程故障导致选举失败，通常会在环上设置一个监视器进程。

81. 请求驱动式令牌传递方法中，若 pi 发出request 消息后久未获得Token，该怎么处理？ 若引入时钟戳，该算法应做何修改？

在请求驱动式令牌传递方法中，若 pi 发出的 request 消息后久未获得 Token，可以引入时钟戳来增加算法的强健性。具体做法是，若 pi 发出的 request 消息后久未获得 Token，则向其它进程发询问消息；若其它进程无反对消息到达，则重新生成令牌，否则继续等待。如果接收到询问消息的进程是令牌的持有者，或已发出一样 request 消息，且自己的 request 消息的时钟戳先于询问进程的时戳，则立即发回一条反对消息。当令牌持有者传递令牌时，如果发现接收者故障，需要调用逻辑环重构算法进行环重构，再重新选择接收者。

【4.典型分布式存储类型-选择最佳拍档-nds/sna-nosql 的类型例子-cap/base/acid 理论-反范式设计=增加冗余列】

存储类型：DAS；SAN；NAS

最佳拍档：

MongoDB 和 Elasticsearch：MongoDB 用于存储结构化或半结构化数据，而 Elasticsearch 用于全文搜索和分析。这种组合通常用于 Web 应用程序和日志记录。

Cassandra 和 Apache Spark：Cassandra 用于存储大规模的结构化数据，而 Spark 用于数据处理和分析。这种组合通常用于大数据分析和实时数据处理。

Redis 和 MySQL：Redis 用于缓存和会话管理，MySQL 用于持久化存储。这种组合通常用于 Web 应用程序和电子商务网站

Neo4j 和 Elasticsearch：Neo4j 用于存储图数据，而 Elasticsearch 用于全文搜索和分析。这种组合通常用于社交网络分析和知识图谱。

Nosql 类型与例子：列存储 Hbase；文档存储；key-value 存储；图存储；对象存储；xml 数据库；6-17

65. 请简述数据库反规范设计方法

反范式化设计是一种在数据库设计中为了提高查询性能而有意违反范式化规则的方法。常见的反范式化技术包括增加冗余列、增加派生列、重新组表、分割表和垂直分割。这些技术可以减少连接操作、减少计算量、降低数据和索引的页数、提高查询效率。但反范式化也会增加数据的冗余，增加维护难度和风险。在进行反范式化设计时需要权衡查询性能和数据冗余之间的关系，考虑数据的访问模式、数据量和系统的可扩展性等因素。

66. 反规范设计的数据完整性维护方法。

反范式化设计会增加数据的冗余，为保证数据的完整性需要额外的工作来维护。常见的维护方法包括应用逻辑、批处理维护和触发器。应用逻辑在应用程序的事务中对所有涉及的表进行操作，但难于管理和容易遗漏。批处理维护通过定期运行批处理程序或存储过程来处理所有非规范化关系涉及的数据，适用于对实时性要求不高的环境。触发器在数据库端建立，对原数据的修改会立即触发对冗余列的修改，实时性好，但会降低插入和更新速度。维护方法的选择需要根据具体情况进行权衡。

63. CAP,BASE 以及 ACID 的关系（重点!!）

CAP理论指在分布式系统中，一致性（Consistency）、可用性（Availability）和分区容错性（Partition Tolerance）三个目标无法同时满足，需要在其中进行权衡；BASE理论是对CAP理论的补充，指在分布式系统中，基本可用（Basically Available）、软状态（Soft state）和最终一致性（Eventually consistent）三个目标是可以同时满足的；ACID理论是传统关系型数据库中的事务处理理念，指原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）四个属性。

65. 请简述数据库反规范设计方法

数据库规范化可以减少数据冗余，节约存储空间，加快增删改的速度，但对于完全规范的数据库查询，通常需要更多的连接操作，影响查询速度。因此，有时为了提高某些查询或应用的性能而破坏规范规则，即反规范化。常见的反规范化技术包括增加冗余列、增加派生列、重新组表、水平分割和垂直分割。这些方法可以在查询时避免连接操作、减少计算量、降低需要读的数据和索引的页数，提高查询效率。反规范化设计需要根据数据库的性能需求进行评估，确保数据的一致性和可维护性。

66. 反规范设计的数据完整性维护方法。

反规范设计可能导致数据的不一致性，因此需要额外的工作来维护数据的完整性。常见的维护方法包括应用逻辑、批处理维护和触发器。应用逻辑是在应用程序的事务中对所有涉及的表进行增、删、改操作进行维护，但难以管理且不易于维护。批处理维护是由批处理程序批量处理所有的非规范化关系涉及的数据，适用于对实时性要求不高的环境。触发器可以在数据库端建立，对原数据的修改会立即触发对冗余列的修改，是实时的且易于维护，但会降低数据的插入和更新速度。在选择维护方法时，需要根据业务需求和性能要求进行权衡。

85. 分布式数据库系统的模式结构（重点！分布透明性）

分布式数据库系统的模式结构包括集中式数据库的模式结构和分布式数据库系统增加的模式级别。集中式数据库采用外模式、概念模式和内模式实现数据的逻辑和物理独立性。分布式数据库增加了全局外模式、全局概念模式、分片模式和分布模式，用于表示全局应用所涉及的数据部分、定义分布式数据库的全局逻辑结构、说明如何放置数据库的特殊部分和定义片段存放节点。分布式数据库系统中增加的模式和映像使得用户和应用程序可以像使用单一数据库一样使用分布式数据库系统，从而实现分布透明性，隐藏数据的分布和复杂性。

分布透明性=分片透明性+位置透明性+局部数据模型透明性（Chapter7-44）

86. 解释分片透明性、复制透明性和位置透明性等三级透明性的区别。

分片透明性是用户不必关心数据的逻辑分片和物理位置分配的细节；复制透明性是用户不必关心数据在各个站点的一致性如何实现的细节；位置透明性是用户不必知道数据存在于哪一个位置上。这三种透明性使得分布式数据库系统具有分布透明性，让用户感觉这是一个集中式数据库，提高了系统的可用性、可靠性和性能（目的）。分布式数据库的数据可以存储在靠近它正常使用的地方，减少响应时间与通讯费用，系统结构灵活，增加新的站比用一个更大的系统代替已有的集中式系统更容易。

注意：反向出题-为了实现分布式数据库操作起来像是一个集中式数据库，需要各种透明性实现。

133. 解释透明性的含义， 并举例说明不同类型的透明性。

透明性指分布式系统隐藏多个计算机的处理过程和资源的物理分布，使其呈现为单个计算机系统。不同类型的透明性包括：访问透明性、位置透明性、迁移透明性、重定位透明性、复制透明性、并发透明性、故障透明性、持久性透明性。

答：对于分布式系统而言， 透明性是指它呈现给用户或应用程序时，就好像是一个单独是计算机系统。 具体说来， 就是隐藏了多个计算机的处理过程， 资源的物理分布。透明性： 如果一个分布式系统能够在 用户和应用程序面前呈现为单个计算机系统，这样的分布式系统就称为是透明的。

分类:1、访问透明性:隐藏数据表示形式以及访问方式的不同 2、位置透明性:隐藏数据所在位置 3、迁移透 明性:隐藏资源是否已移动到另一个位置 4、重定位透明性:隐藏资源是否在使用中已移动到另一个位置 5、 复制透明性:隐藏资源是否已被复制 6、并发透明性:隐藏资源是否由若干相互竞争的用户共享 7、故障透明 性:隐藏资源的故障和恢复 8、持久性透明性:隐藏资源(软件)位于内存里或在磁盘上。

4. **访问透明性**是指对不同数据表示形式以及资源访问方式的隐藏。而**位置透明**是用户无法 判别资源在系统中的物理位置。

5. **迁移透明性**是指分布式系统中的资源移动不会影响该资源的访问方式。而**复制透明**是指 对同一个资源存在多个副本的隐藏。

11. 分布式系统具有透明性时，系统有什么优点。

系统具有透明性时有以下一些优点：

①使软件的研制变得容易，因为访问资源的方法只有一种，软件的功能与其位置无关。

②系统的某些资源变动时不影响或较少影响应用软件。

③系统的资源冗余（硬件冗余和软件冗余）使操作更可靠，可用性更好。透明性使得在实现这种冗余的时候，各种冗余资源的互相替换变得容易。

④在资源操作方面，当把一个操作从一个地方移到若干地方时没有什么影响。

116. SAN 和 NAS 在存储虚拟化方面有哪些不同？

存储类型：DAS；SAN；NAS

1.DAS

直联式存储，存储磁盘与服务器之间的连接通常采用 SCSI 连接（可以理解为直接集成在计算机的总线系统中），SCSI 通道存在 I/O 瓶颈，服务器主机的 SCSI ID 资源有限，能够建立的 SCSI 连接有限

2.NAS

网络附加存储，全面改进了以前低效的 DAS 存储方式，采用独立的服务器，单独为网络数据存储而开发的一种文件服务器来连接所存储设备，这样的数据存储不再是服务器的附属，而是作为网络节点存在于网络之中，由所有的网络用户共享

3.SAN

存储区域网络，SAN 是独立于数据网络的单独的存储网络，有 IP SAN 和 FC SAN 两种部署模式。SAN 可以将存储和服务器隔离，简化了存储管理，能够统一，集中的管理各种资源。SAN 可以屏蔽系统的硬件，可以同时采用不同厂商的存储设备

四.三种存储方式的比较

NAS	SAN	DAS
基于IP网络传输	基于光纤通道或IP网络	基于IP网络
传输文件	传输块	传输文件
可利用带宽低	可利用带宽高	可利用带宽低
系统应用与存储功能分开，两者互不影响	系统应用与存储功能分开，两者互不影响	系统应用与存储功能由同一台服务器负责，两者相互影响
NAS存储自带共享功能，可在多个应用服务器间自动实现共享访问	必须安装共享软件才可在多台应用服务器之间实现存储设备的共享访问	依靠DAS存储服务器的网络共享功能实现多台应用服务器间的共享访问
适用于各种规模的系统，应用服务器数量越大，其简单方便性相比越高	适用于各种规模的系统，应用服务器数量越大，网络设备的成本所占比例越高	一般只适用单台或两台服务器的系统中
价格中等	价格较高	价格较低
依赖于解决方案，网关NAS扩展性比较好，统一NAS扩展性较差	依赖于解决方案	扩展性差
管理性效率低	管理性效率低	有较高的管理效率
适合文件存储	适合文件存储	适合文件存储
不适合数据库存储	适合数据库存储	适合数据库存储
安装简单	安装复杂	安装简单
一定程度的容错性	容错性很好	一定程度的容错性
没有灾难恢复的能力	有较强的灾难恢复的能力	没有灾难恢复的能力

SAN（Storage Area Network）和 NAS（Network Attached Storage）都是存储虚拟化技术的实现方式。 SAN 连接多个独立存储设备形成一个共享存储池，通过块存储访问方式提供存储服务；而 NAS 连接在现有以太网上，通过文件存储访问方式提供存储服务。SAN 通常使用专门的存储控制器管理存储设备，而 NAS 则通常使用通用的服务器管理存储设备。SAN 的数据访问速度通常比 NAS 更快，但成本更高。

SAN 在存储虚拟化方面通常使用**存储区域网络虚拟化技术**（Storage Area Network Virtualization，SAN Virtualization），**将多个物理存储设备（如磁盘阵列）虚拟化为一个逻辑存储池，以提高存储利用率和灵活性**。SAN 虚拟化通常由存储虚拟化器（Storage Virtualization Appliance）来实现，它可以将存储池中的物理存储资源分配给需要的虚拟服务器，将存储资源与服务器物理拓扑分离。这种方式使得存储资源可以更好地管理和利用，同时也提高了存储的可靠性和可扩展性。

相比之下，NAS 在存储虚拟化方面通常**使用文件层面的虚拟化技术**（File-Level Virtualization），将不同的文件系统虚拟化为一个逻辑文件系统，以提高文件共享和管理的灵活性。NAS 虚拟化通常由 NAS 控制器（NAS Controller）来实现，它可以**将多个物理存储设备聚合为一个逻辑文件系统**，并通过网络共享给多个客户端。这种方式使得文件可以更方便地管理和共享，同时也提高了文件的可靠性和可扩展性。

大块连续 I/O 密集的环境：SAN

高并发随机小块 I/O 或共享文件的环境：NAS

CPU 密集的环境：NAS

【5.rpc，rmi，间接通信-消息队列/出版订购】

13. For each of the following use cases, identify the weakest RPC semantics that can be used. In each case explain your answer【最弱的RPC语义】

在分布式系统中，常见的远程过程调用（RPC）语义包括以下三种：

- At-most-once语义：保证请求被执行一次或者不执行，但不保证是否执行成功。
- At-least-once语义：保证请求被执行至少一次，但可能会导致请求被执行多次。
- Exactly-once语义：保证请求被执行恰好一次，不会出现数据不一致的情况。

A. 对于提交已经完成的博客文章的 use case，可以使用 At-least-once 或 At-most-once 语义。At-least-once 可以重复发送请求直到收到确认，At-most-once 则可以让用户重新提交文章，避免无限制地阻塞。

B. 对于在分布式文件系统中创建子目录的 use case，可以使用 At-least-once 或 At-most-once 语义。At-least-once 可以确保同一目录只能被创建一次，之后操作变为幂等。At-most-once 可以在服务器崩溃或无法访问时声明超时。

C. 对于从电子商务网站购买手机的 use case，应使用 Exactly-once 语义，确保交易只发生一次，避免错误的收费或其他问题。

D. 对于查询公司股票价格的 use case，可以使用 At-least-once 语义，因为该操作是幂等的，可以重复发送请

14. 什么是间接通信？简述间接通信的基本策略。

间接通信是通过第三个实体实现发送者和接收者之间解耦合的通信方式，包括组通信、发布-订阅系统、消息队列、元组空间和分布式共享内存等基本策略。其中，组通信支持一对多通信，发布-订阅系统由大量生产者向大量消费者发布信息，通过中间服务实现路由，消息队列提供点对点服务，元组空间支持在持久空间中读取或删除元组，分布式共享内存支持在不同内存进程之间共享数据。

通过第三个实体，允许在发送者和接收者之间的深度解耦合，即发送者不需要知道正在发送给谁(空间解耦合)，发送者和接收者不需要同时存在(时间解耦合)

a)组通信：组通信涉及消息传递给若干接收者，是支持一对多通信的多方通信泛型

b)发布—订阅系统：大量生产者(或发布者) 为大量的消费者(订阅者)发布他们感兴趣的信息项　关键特征：中间服务---用于确保由生产者生成的信息被路由到需要这个信息的消费者。

c)消息队列：提供点对点服务。生产者发送消息到指定队列，消费者能从队列中接收消息，或者被通知队列有消息到达。

d)元组空间：进程可以把任意的结构化数据项(元组) 放到一个持久元组空间，其他进程可以指 定感兴趣的模式，从而可以在元组空间读取或者删除元组。

e)分布式共享内存(Distributed Shared Memory ，DSM)：系统提供一种抽象，用于支持在不同共享物理内存的进程之间共享数据。

15. 发布—订阅系统的几种已定义的常见模式：

发布-订阅系统常见的模式包括**基于渠道、主题、内容、类型和概念，用于定义订阅策略**。其中，基于渠道是通过命名的渠道发布和接收事件，基于主题是根据感兴趣的主题定义订阅，基于内容允许使用事件属性值约束组合进行查询，基于类型是根据事件类型定义订阅，基于概念的订阅模型则根据事件的语义和语法进行表述。

16. 列出出版订购系统中事件路由的常见技术？

出版订购系统中事件路由的常见技术包括泛洪、过滤、广告和汇聚。其中，泛洪是向所有节点广播事件通知，匹配在订阅者端执行；过滤是代理网络中使用过滤方法，通过有路径到达有效订阅者的网络转发通知；广告是向订阅者传播广告，减少订阅传播时的网络流量负担；汇聚是将事件空间的责任划分到网络中的代理集合上，控制订阅传播的方法。

27. 说明 RPC 的主要思想及 RPC 调用的主要步骤。

RPC 的主要思想是允许程序调用其他机器上的过程，调用者发送消息给被调用者，被调用者执行过程并返回结果给调用者，消息传送和 I/O 操作对编程人员是透明的。RPC 调用的主要步骤包括客户端调用客户端存根、构造消息并发送到远程内核、远程内核将消息发送到服务器端存根、服务器端存根从消息中取出参数并调用服务器端过程、服务器端将结果返回给服务器端存根、服务器端存根将结果打包并发送回客户端内核、客户端内核将消息提交给客户端存根、从消息中取出结果并返回给客户端。这些步骤使得远程过程调用看起来像是本地过程调用，从而简化了分布式系统的开发和维护。

RPC 是远程过程调用的缩写，它的目的是让远程的过程调用就像在本地一样，调用者不必意识到此调用的过程是在其他机器上执行的。RPC 的执行步骤包括：客户过程调用客户存根；客户存根建立消息，激活内核陷阱；内核将消息发送到远程内核；远程内核将消息发送到服务器存根；服务器存根解包消息并调用服务器过程；服务器完成工作或返回结果给服务器存根；服务器存根再次打包消息并激活内核陷阱；远程内核将消息发送至客户内核；客户内核将消息交给客户存根；客户存根解包消息，取出结果并返回给客户。

28. 在RPC调用时，如果服务器或客户机崩溃了，各有哪些解决方法。

在 RPC 调用中，如果服务器或客户端崩溃了，可以采取不同的解决方法。对于服务器崩溃，At-least-once 语义可以等待服务器重新启动，重发请求，保证 RPC 至少执行一次；At-most-once 语义可以立即放弃并报告失效，确保 RPC 至多执行一次；不作保证需要手动处理；Exactly-once 语义可以通过事务管理机制保证 RPC 恰好执行一次。对于客户端崩溃，可以采用根除、再生、温和再生或过期等方式进行处理。需要根据具体场景进行权衡和选择。

29. 在RPC中，如果客户机在发送请求后在服务器应答消息到来之前崩溃了，将会发生什么问题？如何解决

在 RPC 中，如果客户端在发送请求后在服务器应答消息到来之前崩溃，会产生“计算孤儿”的问题。为了解决这个问题，可以采用以下四种方法：根绝方法、再生法、温和再生法和过期法。这些方法可以帮助清除孤儿计算，确保 RPC 的可靠性和正确性。

30. 一个影响 RPC 执行时间的问题是消息的拷贝问题，试说明在那些环节需要拷贝，并说明减少拷贝次 数的方法。

RPC 中的消息拷贝问题会影响执行时间。需要拷贝的环节包括在发送端、消息从客户存根拷贝到客户内核缓冲区，再拷贝到客户接口芯片缓冲区，最后到达服务器存根(共 5 次)。为了减少拷贝次数，可以采用分散-集中方法，具有分散-集中能力的网络芯片可以通过拼接多个内存缓冲区来组装报文，减少拷贝次数。

在发送端，由客户内核缓冲区生成报文消息头，由客户存根生成消息体，由网络芯片组装报文。在接收端，网络芯片将接收到的报文分解成消息头和消息体，并放入相应的缓冲区。

31. RPC通信与通常的自定义协议的 Socket 通信有哪些区别和联系

区别：

- RPC是建立在Socket之上的，相对于Socket，RPC实现了更多的功能。
- RPC的寻址需要提供特定的方法名称和连接到目标服务器的信息，而自定义协议的Socket通信则需要提供IP地址和端口号。
- RPC需要将参数序列化为二进制并进行网络传输，而Socket通信则直接传输数据。
- RPC的通信过程需要进行反序列化和本地调用，而Socket通信则直接将数据传递给应用程序处理。

联系：

- RPC通信和Socket通信都是基于TCP连接的。
- 两者都可以使用长连接或按需连接的方式进行通信。
- 两者都需要进行数据传输和数据处理。

33.简述远程方法调用(Remote Method Invocation，RMI)的基本通信原理。

客户端与服务器端内在通过套接字通信

服务器端

1、创建远程服务对象

2、接收请求、执行并返回结果(Skeleton)

1)解码(读取)远程方法的参数； 2)调用实际远程对象实现的方法；

3)将结果(返回值或异常)返回给调用程序。

客户端

1、建立与服务器的连接

2、发送请求、接收返回结果(Stub)

1)初始化连接； 2)编码并发送参数； 3)等待方法调用结果；

4)解码(读取)返回值或返回的异常； 5)将值返回给调用程序。

35. 什么是 RPC？试简述 RPC 的执行步骤。

RPC 是远程过程调用的缩写，它的目的是让远程的过程调用就像在本地一样，调用者不必意识到此调用的过程是在其他机器上执行的。RPC 的执行步骤包括：客户过程调用客户存根；客户存根建立消息，激活内核陷阱；内核将消息发送到远程内核；远程内核将消息发送到服务器存根；服务器存根解包消息并调用服务器过程；服务器完成工作或返回结果给服务器存根；服务器存根再次打包消息并激活内核陷阱；远程内核将消息发送至客户内核；客户内核将消息交给客户存根；客户存根解包消息，取出结果并返回给客户。

答： RPC 是 remote procedure call (远程过程调用)的简称。 RPC 思想是使远程的过程调 用就像在本地的过程一样，调用者不应该意识到此调用的过程是在其他机器上实行的。 RPC 的执行步骤：

(1) 客户过程以普通方式调用相应的客户存根；

(2) 客户存根建立消息， 打包并激活内核陷阱；

(3) 内核将消息发送到远程内核；

(4) 远程内核将消息发送到服务器存根；

(5) 服务器存根将消息解包， 取出其中参数后调用服务器过程；

(6) 服务器完成工作或将结果返回服务器存根；

(7) 服务器存根将它打包并激活内核陷阱；

(8) 远程内核将消息发送至客户内核；

(9) 客户内核将消息交给客户存根；

(10) 客户存根将消息解包，从中取出结果返回给客户；

计算综合 4＊10=40

（一）逻辑时钟计算(标量/向量/全局时钟的计算)，判断两个事件是否有并发(互相在先)和在先关系

20. 在逻辑时钟算法中， Lamport 定义了一个称作“先发生”的关系，表达式 a→b 表示 a 在 b 之前发生。先发生关系是一个传递关系。

Lamport 逻辑时钟：由 Leslie Lamport 在 1978 年提出，**基于事件的先后顺序来推导时间**。

Lamport 逻辑时钟的基本思想是，每个进程在本地维护一个逻辑时钟，该时钟以事件为单位递增，并且在事件发生时将其时间戳附加到事件中。Lamport 逻辑时钟的时间戳由两部分组成：进程本地时钟的值和该进程最后一次发送或接收事件的时间戳的最大值。

在 Lamport 逻辑时钟中，如果事件 A 发生在事件 B 之前，那么事件 A 的时间戳一定小于事件 B 的时间戳。如果事件 A 和事件 B 在不同的进程上发生，那么它们的时间戳可能相等，因为两个进程之间的消息传递可能需要一定的时间。

51. 向量时钟能捕获因果关系。创建向量时钟是让每个进程 Pi 维护一个向量 VCi 来完成。

向量时钟可以捕获因果关系，是因为它通过向量的方式记录了分布式系统中各个进程之间的事件顺序关系。具体来说，向量时钟中的每个元素记录了对应进程发生的事件数量，每个进程的向量时钟包含了所有进程的事件数量信息。因此，通过比较不同进程的向量时钟，可以判断不同进程之间的事件顺序关系，进而判断事件之间的因果关系。

在向量时钟中，每个元素的含义是该进程在该维度上的事件数量。当一个进程 Pi 发生一个事件时，它会将**自己的向量时钟 VCi** 的第 i 个元素加 1。当一个进程 Pi 接收到另一个进程 Pj 发送的事件时，它会将自己的向量时钟 VCi 的第 j 个元素更新为 max(VCi[j], VCj[j])+1，表示进程 Pi 发生的事件以及进程 Pj 发生的事件之间的因果关系。因此，通过比较不同进程的向量时钟，可以判断事件之间的因果关系，即哪些事件是因为其他事件的结果而发生的。

51. 向量时钟能捕获因果关系。创建向量时钟是让每个进程 Pi 维护一个向量 VCi 来完成。

向量时钟可以捕获因果关系，是因为它通过向量的方式记录了分布式系统中各个进程之间的事件顺序关系。具体来说，向量时钟中的每个元素记录了对应进程发生的事件数量，每个进

程的向量时钟包含了所有进程的事件数量信息。因此，通过比较不同进程的向量时钟，可以判断不同进程之间的事件顺序关系，进而判断事件之间的因果关系。

在向量时钟中，每个元素的含义是该进程在该维度上的事件数量。当一个进程 Pi 发生一个事件时，它会将**自己的向量时钟 VCi** 的第 i 个元素加 1。当一个进程 Pi 接收到另一个进程 Pj 发送的事件时，它会将自己的向量时钟 VCi 的第 j 个元素更新为 max(VCi[j], VCj[j])+1，表示进程 Pi 发生的事件以及进程 Pj 发生的事件之间的因果关系。因此，通过比较不同进程的向量时钟，可以判断事件之间的因果关系，即哪些事件是因为其他事件的结果而发生的。

164. 使用逻辑时钟的“先于”关系在以下表格中填入校正的时间(如需要)，并标记出所有 的先于关系以及所有的并发事件

进程 1		
时间	校正时间	事件
2		A
4		B （向进程 2 发送消息）
6	7	C （从进程 2 接收消息 F）
8		D （向进程 3 发送消息）

进程 2		
时间	校正时间	事件
3	5	E（从进程 1 发送消息 B）
6		F （向进程 1 发送消息）
9		G（从进程 3 接收消息 J）
12		H （向进程 3 发送消息）

进程 3		
时间	校正时间	事件
4		I
8		J （向进程 2 发送消息）
12	13	K （从进程 2 接收消息 H）
16		L （从进程 1 接收消息 D）

先于关系： A—>B—>E>F—>C—>D>G—>H---------->K—>L

并发事件：B,I ；　　D,J

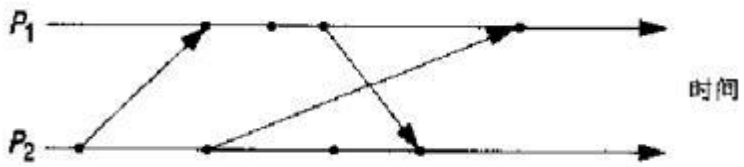
89. 有三个进程分别运行在不同的机器上，每个机器都有自己的时钟并以不同且不变的速率工作(进程1的时钟嘀嗒了6下时，进程 2 的时钟嘀嗒了8下，而进程3的时钟嘀嗒了10下)。举例说明进程之间消息传递中违反先发生关系的情况，并说明如何用 Lamport 方法解决。

Lamport解决方案直接使用先发生关系，每条消息携带发送者的时钟以指出其发送的时刻，当消息到达时，接收者时钟若比消息发送时钟小，就立即将自己的时钟调到比发送时间大1或更多的值。

如上图，其中消息C从进程2到进程1是在60时刻离开，56时刻到达。同理，消息D从进程1到进程0是在64时序离开，54时刻到达，这是绝对不可能出现的。

Lamport的解决方案是直接使用先发生关系，因为C在60时刻离开，只能在61时刻或更晚时刻到达，所以每条消息都携带发送者的时钟以指出其发送的时刻，当消息到达时，接收者时钟惹比消息发送时钟小，就立即将自己的时钟调到比发送时间大1或更多的值。在b中，我们看到C现在到达的时间是61，同样D到达的时间是70。

113. 下图给出在两个进程 P1 和 P2 中发生的事件。进程之间的箭头表示消息传递。从初始 状态〔0,0)开始，画出并标注一致状态(p1 的状态， p2 的状态)的网格。



146. 全局状态-多个事件轴的分割-判断一致性割和非一致性割

全局状态定义了每个进程的本地状态和正在传输中的消息

一致的全局状态：如果已经记录了一个进程 P 收到了来自进程 Q 的一条消息，那么也应该记录 Q 确实已经发送了那条消息。

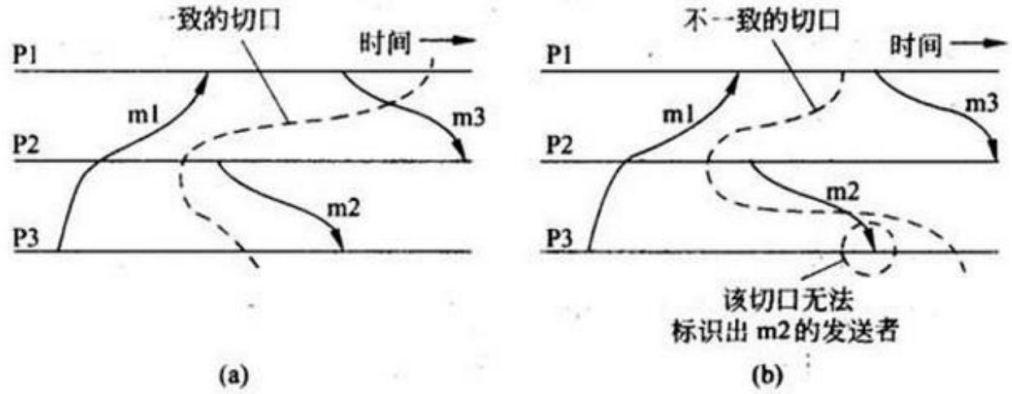


图 5.9 一致的切口和不一致的切口

(a) 一致的切口；(b) 不一致的切口

145. 当发现一个时钟 4s 时，它的读数是 10:27:54.0(小时:分钟:秒)。解释为什么在这时不愿 将时钟设成正确的时间， 并(用数字表示)给出它应该如何调整以便在 8s 后变成正确的 时间。

52. 互斥集中式算法的优点是易于实现、很公平、保证了顺序一致性。而缺点是协作者是单个故障点， 如果它崩溃了，整个系统可能瘫痪。

互斥集中式算法通常由一个中心节点来控制共享资源的访问，该中心节点维护一个互斥集，用于记录当前访问共享资源的进程集合。当一个进程需要访问共享资源时，它必须向中心节点发送请求，中心节点会将该进程加入互斥集，然后向该进程发送访问共享资源的许可。当该进程完成访问后，它会将自己从互斥集中删除，然后通知中心节点将访问共享资源的许可转交给下一个进程。

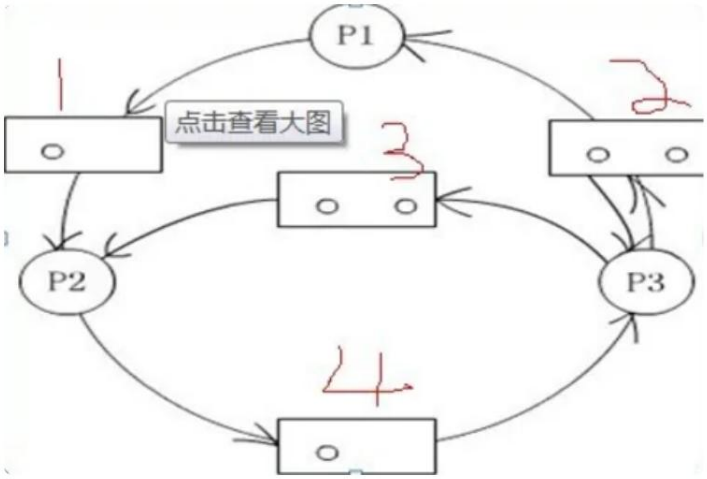
（二）死锁-图模型-等待图-资源图-判断是否存在死锁/有无假死锁/做资源调度

首先看P1，P1申请资源1，但资源1只有1个，且被P2占用，所以P1被阻塞，无法删除P1的边；

接着看P2，P2申请资源4，同理，资源4只有一个且被P3占用，所以P2的边也不能删除；

最后P3，P3申请资源3和2，资源3有2个，其中一个被P2占用，剩余一个空闲资源，可被P3申请，但资源2中，一个被P1占用，另一个被P3占用，无空闲资源，所以P3也被阻塞。无法删除P3的边。

三个结点经分析后都不能化简为孤立结点，所以形成死锁。



23. 有三个进程 P1,P2 和 P3 并发工作。进程 P1 需用资源 S3 和 S1;进程 P2 需用资源 S1 和 S2;进程 P3 需用资源 S2 和 S3 。回答:

(1)若对资源分配不加限制，会发生什么情况?为什么?

(2)为保证进程正确工作， 应采用怎样的资源分配策略?为什么?

（1）可能会发生死锁

例如：进程P1，P2和P3分别获得资源S3，S1和S2后再继续申请资源时都要等待，这是循环等待。

（或进程在等待新源时均不释放已占资源）1·

（2）可有几种答案：

A.采用静态分配 由于执行前已获得所需的全部资源，故不会出现占有资源又等待别的资源的现象（或不会出现循环等待资源现象）。

或B.采用按序分配，不会出现循环等待资源现象。

或C.采用银行家算法，因为在分配时，保证了系统处于安全状态。

48. 简述利用时间戳预防死锁的不同方法。如果进程 P1、P2、P3 分别有时间戳 5 、 10、 15 ， 在下列情况下， 应该怎样处理? PPT 4-52

等待-死亡方案是一种老的优先原则，即如果一个进程想要获取已经被另一个进程占用的资源，那么只有当它的时间戳比另一个进程的时间戳早的时候，它才会等待另一个进程释放资

源。如果它的时间戳比另一个进程晚，它就**会自己放弃请求**，并回滚到之前的状态。

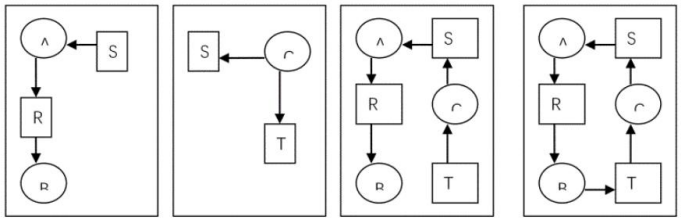
伤害-等待方案是一种年轻的优先原则，即如果一个进程想要获取已经被另一个进程占用的资源，那么只有当它的时间戳比另一个进程的时间戳晚的时候，它才会等待另一个进程释放资源。如果它的时间戳比另一个进程早，它就**会让另一个进程放弃占用资源**，并回滚到之前的状态。

1) P1 申请 P2 占用的资源， 使用 wait-die 方法；P1 等待。

2) P1 申请 P2 占用的资源， 使用 wound-wait 方法；P1 抢夺资源，P2 被杀死。

假死锁问题：4-57

答:集中式的死锁检测算法每台机器的资源图中只包含它自己的进程和资源,协调者节点保存整个系统〔所有资源图的集合〕的资源图。当机器资源图发生变化时相应的消息发送给协调者以提供更新,当协调者检测到环路时,它终止一个进程以解决死锁。



a 机器 0 b 机器 1 c 协调者 d 协调者

如上图圆表示进程,方框表示资源,开场时如同a,b,c所示,过来一段时间,B释放R并请求T,这是一个合法的操作,机器0向协调者发送一条消息申明它释放资源R,机器1向协调者发送一条消息声明进程B正在等待它的资源T,不幸的是机器1的消息先到达协调者,导致生成资源图如图d所示。协调者得出错误的结论——死锁存在,这种情况称为假死锁。

解决方法是:使用Lamport算法以提供全局统一的时间,对协调者收到的消息按照时间戳排序

（三）数据库分片方式-三级透明性

58. 客户向服务器发出远程过程调用。客户花 5ms 时间计算每个请求的参数，服务器花 10ms 时间处理每个请求。本地操作系统处理每次发送和接收操作的时间是 0.5ms,网络传递 每个请求或者应答消息的时间是 3ms.编码或者解码每个消息花 0.5ms 时间。

计算下列两种条件下客户创建和返回消息所花费的时间:

计算参数时间（calc. args）：每个请求的参数计算时间为 5ms。

参数编码时间（marshal args）：每个消息的编码或解码时间为 0.5ms，因此参数编码时间为 4 * 0.5ms = 2ms。

操作系统发送时间（OS send time）：每个发送和接收操作的时间是 0.5ms，因此操作系统发送时间为 2 * 0.5ms = 1ms。

消息传输时间（message transmission）：每个请求或者应答消息的网络传递时间是 3ms。

操作系统接收时间（OS receive time）：每个发送和接收操作的时间是 0.5ms，因此操作系统接收时间为 2 * 0.5ms = 1ms。

参数解码时间（unmarshal args）：每个消息的编码或解码时间为 0.5ms，因此参数解码时间为 4 * 0.5ms = 2ms。

执行服务器端过程时间（execute server procedure）：每个请求在服务器端处理的时间是 10ms。

结果编码时间（marshall results）：每个消息的编码或解码时间为 0.5ms，因此结果编码时间为 2 * 0.5ms = 1ms。

操作系统发送时间（OS send time）：每个发送和接收操作的时间是 0.5ms，因此操作系统发送时间为 2 *

0.5ms = 1ms。

消息传输时间（message transmission）：每个请求或者应答消息的网络传递时间是 3ms。

操作系统接收时间（OS receive time）：每个发送和接收操作的时间是 0.5ms，因此操作系统接收时间为 2 * 0.5ms = 1ms。

结果解码时间（unmarshal args）：每个消息的编码或解码时间为 0.5ms，因此结果解码时间为 2 * 0.5ms = 1ms。

每一章都会有：

第一讲概念

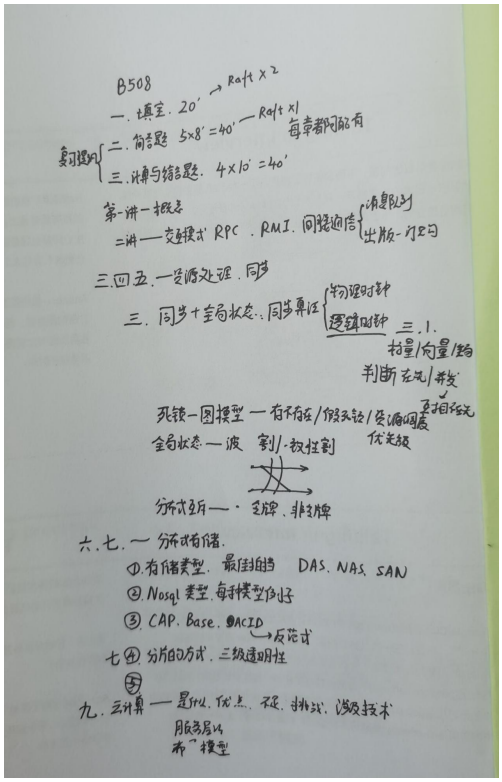
第二讲(rpc，rmi，间接通信-消息队列/出版订购)

第三四五讲(典型同步算法-物理时钟/逻辑时钟) (死锁-图模型-等待图-资源图-判断是否存在死锁/有无假死锁/做资源调度) (全局状态-多个事件轴的分割-判断一致性割和非一致性割) (互斥算法-coral-令牌-非令牌)

第六七章(典型分布式存储类型-选择最佳拍档-nds/sna-nosql 的类型例子-cap/base/acid 理论-反范式设计=增加冗余列) (数据库分片方式-三级透明性)

第九章 (对云计算是什么东西/优点，不足，挑战，涉及技术，3 层次，3 部署模型了解)

raft：简答 x1，填空 x2


B508
一. 填空. 20' → Raft X2
复习提纲 二. 简答题 5×8'=40' — Raft X1 每章都可能有
三. 计算与综合题. 4×10'=40'
第一讲一概念
二讲——远程模式 RPC . RMI. 同步通信 消息队列 出版-订阅
三.四.五.一致性处理. 同步
三. 同步+全局状态:. 同步算法 物理时钟 逻辑时钟 三.1. 标量/向量/矩阵
判断 先见/并发
死锁-图模型 一 有不存在/假死环/资源饥饿 互相纠缠 优先级
全局状态一波 割/-致性割
分布式互斥—. 主牌、非主牌
六.七.— 分布式存储.
①.存储类型. 最佳归档 DAS. NAS. SAN
②.Nosql 类型. 每种模型特好
③ CAP. Base. ACID → 反范式
七.④ 分片的方式. 三级透明性
⑤
九. 云计算 — 是什么. 优点. 不足. 挑战. 涉及技术
服务层次
两个模型