《分布式系统》复习

一、填空题

1. 分布式系统必须能够让用户方便地访问资源；必须隐藏资源在一个网络上分布这样一个事实； 必须是开放的；必须是可扩展的。

2. 分布式系统**类型**：分布式计算系统、分布式信息系统、分布式普适系统。

3. 图领奖获得者、著名数据库专家 Jim Gray 博士认为，人类自古以来在**科学研究**上先后 经历了实验科学、理论科学、计算科学、数据密集型科学。

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。

6. 一个开放的分布式系统就是根据一系列准则来提供服务，这些准则描述了所提供服务的语法和语义。

7. 集群计算系统一个突出的特征是它的同构性；它提供了最大限度的分布式透明性。可用于单个程序在多台计算机上并行地运行。

8. 分布式系统是一组自治的计算机集合， 通过通信网络相互连接， 实现资源共享和协作，而呈现给用户的是单个完整的计算机系统。

9. **客户/服务器结构**的应用程序通常划分为三层，它们是：用户接口层、处理层和数据层。

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。

11. 某游戏公司欲开发一个大型多人即时战略游戏，游戏设计的目标之一是能够支持玩家自行创建战役地图，定义游戏对象的行为和之间的关系。针对该目标， 公司应该采用解释 器架构风格最为合适。解释器架构风格是一种常见的软件架构风格，它的主要特点是将系统中的每个功能模块都封装成解释器，用来解释和执行用户定义的语言或规则。在这种架构风格下，系统中的所有操作都由解释器来完成，而不是直接操作数据或对象。这种架构风格的优点是能够提高系统的可扩展性和灵活性，因为用户可以通过定义自己的语言或规则来扩展系统的功能。在游戏开发中，解释器架构风格非常适合支持玩家自行创建地图和定义游戏对象行为的需求。玩家可以通过定义自己的语言或规则来实现这些功能，而这些语言或规则可以通过解释器来执行。这种架构风格使得游戏系统更加灵活和可扩展，可以满足不同玩家的需求和喜好，提高了游戏的可玩性和趣味性。因此，在这个情境下，采用解释器架构风格是最为合适的选择。

12. 在**消息队列系统**中， 队列由队列管理器来管理，它与发送或接收消息的应用程序直接交 互。

13. 所谓幂等， 简单地说，就是对接口的多次调用所产生的结果和调用一次是一致的。

14. 业务系统实现幂等的通用方式一般是排重表校验。

15. 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。

16. 在云计算技术中， 中间件 位于服务和服务器集群之间，提供管理和服务即云计算体系 结构中的管理系统。

17. **分布式系统体系结构**样式很多，其最重要的有：分层体系结构； 基于对象的体系结构、以数据为中心的体系结构以及基于事件的体系结构等四类。

18. **分布式软件体系结构**主要分集中式、非集中式和各种混合形式三大类。其**非集中式体系结构**又分为结构化的点对点、非结构化的点对点、超级对等体三种。

19. **TCP/IP 体系结构**的传输层上定义的两个传输协议为传输控制协议(TCP)和用户数据报协议(UDP)。

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。

21. 对于域名： test.com ，DNS 服务器查找顺序是(先查找.com 域， 再查找 test 主机)。

22. 实现**软件自适应**的基本技术分为要点分离、计算映像和基于组件的设计三种类型。

23. 分布式的**自主系统**指的是自我管理、自我恢复、自我配置和自我优化等各种自适应性

24. 一个线程独立地执行它自己的程序代码。线程系统一般只维护用来让多个线程共享 CPU 所必需的最少量信息。

25. 任何一个分布式系统都无法同时满足一致性(Consistency)、可用性(Availability)和 分区容错性(Partition tolerance)，最多只能同时满足两项。

26. 在**服务器的组织结构**中，迭代服务器是自己处理请求， 将响应返回给客户；而并发服务器将请求传递给某个独立线程或其他进程来处理。

27. **服务器集群在逻辑上由三层**组成，第一层是逻辑交换机； 第二层是应用/计算服务； 第三层是文件/数据库系统。

28. 有两种实现线程包的基本方法： 一是可以构造一个完全在用户模式下执行的线程； 二是由内核来掌管线程并进行调度。

29. 在**代码迁移的框架结构**中，**进程包含三个段**， 它们是代码段、资源段和执行段三个段。

30. **进程对资源的绑定**有三种类型：一是按标识符绑定；二是按值绑定；三是按类型绑定。而三种类型的资源对机器的绑定是未连接资源、附着连接资源和紧固连接资源。

在操作系统中，进程需要使用各种资源（如内存、文件、网络等）来完成任务。为了管理和控制这些资源，操作系统通常会将资源与进程之间建立一定的绑定关系。这些绑定关系可以按照不同的方式进行分类和描述，其中包括三种类型的绑定方式：按标识符绑定、按值绑定和按类型绑定。

按标识符绑定是指系统为每个资源分配一个唯一的标识符，进程通过该标识符来访问该资源。比如，在Unix系统中，文件通常用文件描述符来标识和访问。这种方式的优点是资源的访问和管理都比较简单和高效，但缺点是可能会出现标识符冲突和管理复杂的问题。

按值绑定是指将资源的值直接绑定到进程的地址空间中，进程可以直接访问和修改该资源。比如，在C语言中，可以通过指针来访问和修改内存中的变量。这种方式的优点是操作简单和高效，但缺点是可能会导致资源共享和安全性问题。按类型绑定是指将资源按类型进行分类和管理，进程可以通过类型来访问和管理该资源。比如，在Java语言中，可以通过类来访问和管理各种资源。这种方式的优点是可以更好地管理和控制资源，但缺点是可能会导致类型转换和效率降低等问题。

另外，资源的绑定还可以按照连接的方式进行分类，主要包括未连接资源、附着连接资源和紧固连接资源三种类型。未连接资源是指进程在使用该资源之前需要进行显式的连接操作，比如打开文件、申请内存等。这种方式需要进程手动管理资源的生命周期和连接状态，比较灵活但也容易出错。附着连接资源是指资源在进程启动时自动连接到进程中，并伴随进程的生命周期而存在，比如共享库和动态链接库等。这种方式可以提高资源的共享和复用性，但也可能导致依赖性和版本问题。紧固连接资源是指资源在进程编译时就被静态链接到进程中，成为进程的一部分。这种方式可以提高程序的运行效率和可靠性，但也会增加程序的大小和复杂度。

31. 在 RPC 操作中， **客户存根**的功能是将得到的参数打包成消息， 然后将消息发送给服务 器存根。

客户端存根（Client Stub）是指在RPC（Remote Procedure Call，远程过程调用）中，客户端使用的一个代理程序，用于向服务端发送请求并接收响应。客户端通过客户端存根来调用远程服务端提供的函数（即远程过程），客户端存根负责将函数参数打包成一个消息，通过网络发送给服务端，等待服务端的响应，然后将响应解包成函数返回值返回给客户端。

客户端存根通常由RPC框架自动生成，开发人员只需要调用客户端存根提供的接口函数即可。客户端存根的生成过程通常分为两个步骤：接口定义和存根生成。

接口定义是指开发人员定义远程过程的接口和参数类型，通常使用IDL（Interface Definition Language，接口定义语言）来描述。IDL是一种面向对象的描述语言，用于描述远程过程的接口和参数。

存根生成是指RPC框架根据接口定义自动生成客户端和服务端的存根代码。客户端存根通常会包含一个接口函数，该函数接受函数参数，然后将参数打包成一个消息，通过网络发送给服务端存根。客户端存根还需要处理网络异常、重试机制和安全性等问题。服务端存根则需要接收客户端发送的消息，解析出函数参数，并调用相应的函数。

总的来说，客户端存根是一种RPC框架提供的工具，用于简化远程过程调用的过程，开发人员只需要关注接口定义和使用存根提供的接口函数即可。客户端存根的主要功能是将函数参数打包成一个消息，通过网络发送给服务端，然后等待服务端的响应，将响应解包成函数返回值返回给客户端。

32. NoSQL 数据库具有弱一致性，关系型数据库具有强一致性

关系型数据库通常被设计用于处理事务性数据，要求任何时刻对数据的查询都应该返回最新的结果，不允许出现数据不一致的情况。为了保证强一致性，关系型数据库通常采用ACID事务模型，即原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）等严格的事务处理机制。在关系型数据库中，当一个事务提交时，数据库系统会保证这个事务对数据的修改是原子性的，要么全部成功，要么全部失败，从而保证数据的一致性。

NoSQL数据库通常被设计用于处理大规模、分布式、非结构化或半结构化数据，要求具有更好的可扩展性、高并发性和可用性等特性。为了满足这些需求，NoSQL数据库通常采用分布式架构和最终一致性（eventual consistency）模型。最终一致性指的是在一定时间内允许出现数据不一致的情况，但最终会达到一致的状态。在NoSQL数据库中，当一个数据写入操作完成后，不会立即对所有节点进行同步，而是通过异步复制机制将数据复制到其他节点，可能存在一段时间内某些节点读取到旧数据的情况。但随着时间的推移，所有节点最终会达到一致的状态，从而保证数据的最终一致性。

NoSQL（Not Only SQL）是一种非关系型数据库（Non-Relational Database），它的出现是为了解决关系型数据库在处理大规模数据、高并发访问和分布式存储等方面的局限性。相比于传统的关系型数据库，NoSQL数据库通常具有更加灵活的数据模型和更好的可扩展性，能够处理大量的非结构化和半结构化数据。

NoSQL数据库通常采用分布式架构、键值对存储、列族存储、文档存储或图形存储等不同的存储模型，能够支持高并发访问、水平扩展和自动故障转移等特性。NoSQL数据库通常不支持SQL语言，而是使用一些特定的查询语言或API来访问数据。

33. NoSQL 数据库的三大理论基石：CAP ； 最终一致性； BASE

CAP理论指出，在一个分布式系统中，Consistency（一致性）、Availability（可用性）和Partition tolerance（分区容错性）这三个指标无法同时满足，只能同时满足其中的两个。NoSQL数据库通常会选择满足Availability和Partition tolerance，而通过牺牲一定的一致性来

达到这两个指标。例如，Cassandra数据库采用了分布式无中心架构，支持分区容错和高可用性，但在一定时间内可能出现数据不一致的情况。

CAP理论指出，在分布式系统中，Consistency（一致性）、Availability（可用性）和Partition tolerance（分区容错性）这三个指标无法同时满足，只能同时满足其中的两个，因为：

Consistency（一致性）

一致性（Consistency）：指分布式系统中的所有节点在同一时刻看到的数据是一致的。为了保证一致性，分布式系统需要对数据进行同步和协调，这就导致了系统的性能问题和可用性问题。

分布式事务（Distributed Transaction）：通过协调多个节点，保证分布式系统中的数据操作具有ACID特性，从而实现强一致性模型。

Paxos算法：通过一系列的消息交换和投票机制，保证分布式系统中的数据一致性，从而实现强一致性模型。

2PC（Two-Phase Commit）协议：通过两个阶段的协商和确认，保证分布式系统中的数据操作具有ACID特性，从而实现强一致性模型。

Raft算法：通过一系列的心跳机制和领导者选举，保证分布式系统中的数据一致性，从而实现强一致性模型。

Availability（可用性）

可用性（Availability）：指分布式系统在任何时候都能够响应用户的请求，并返回正确的结果。为了保证可用性，分布式系统需要将请求分发到多个节点上进行处理，这就导致了数据的不一致性和同步问题。

负载均衡（Load Balancing）：通过将请求分发到多个节点上进行处理，实现请求的均衡分配和负载的均衡，从而提高系统的可用性和性能。

故障转移（Failover）：通过将故障节点的任务转移到其他节点上进行处理，实现故障的快速恢复和系统的连续性和稳定性。

缓存（Caching）：通过将频繁访问的数据缓存到内存中，提高数据的访问速度和系统的响应速度，从而提高系统的可用性和性能。

异步复制（Asynchronous Replication）：通过将数据异步复制到多个节点上，提高数据的冗余性和可靠性，从而提高系统的可用性和容错性。

Partition tolerance（分区容错性）

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。

分布式一致性算法（Distributed Consensus Algorithm）：例如Paxos算法、Raft算法等，通过一系列的消息交换和投票机制，保证分布式系统中的节点能够达成一致的状态，从而保证分区容错性。

最终一致性模型（Eventually Consistent）：通过允许一定时间内的数据不一致，最终达到一致的状态，来保证分区容错性。例如，通过异步复制和版本控制等技术，实现数据的同步和一致性。

分布式哈希表（Distributed Hash Table）：通过将数据根据哈希函数分散到多个节点上进行存储，实现数据的分布式存储和访问，从而保证分区容错性。

最终一致性指出，在一个分布式系统中，允许在一定时间内出现数据不一致的情况，但最终会达到一致的状态。这种模型相对于强一致性模型更为灵活，可以提高系统的可用性和

性能。例如，MongoDB数据库采用了最终一致性模型，支持多主复制和分片架构，可以处理大量的非结构化数据。

BASE理论是指基本可用（Basically Available）、软状态（Soft state）和最终一致性（Eventually consistent）。这种理论强调在处理大规模分布式系统时，需要权衡系统的可用性、性能和一致性等方面的因素，采用柔性的设计策略和最终一致性模型来优化系统的效率和可靠性。基本可用指系统在出现故障或异常情况时，仍然能够保证基本的可用性和服务质量；软状态指系统的状态可以在一定时间内不同步或不一致，但最终会达到一致的状态。

基本可用性（Basically Available）：系统能够在面临分区容错性的情况下，仍然保证基本的可用性和响应能力。

软状态（Soft state）：系统中的数据状态是可变的，因为数据可能会存在一定的不一致性和延迟，但是最终会达到一致的状态。

最终一致性（Eventually Consistent）：系统能够保证在一定时间内，所有节点的数据最终会达到一致的状态。

具体来说，NoSQL数据库和BASE的关系如下：

基本可用性（Basically Available）：NoSQL数据库通常采用分布式架构和数据复制技术，将数据存储在多个节点上，从而实现基本的可用性和响应能力。当某个节点出现故障或网络分区时，系统可以从其他节点获取数据，保证数据的可用性和一致性。

软状态（Soft state）：NoSQL数据库通常采用最终一致性模型，允许数据在一定时间内存在不一致性和延迟，但最终会达到一致的状态。为了实现最终一致性，NoSQL数据库通常采用基于向量时钟或时间戳的版本控制机制，保证数据同步和版本管理的正确性和可靠性。

最终一致性（Eventually Consistent）：NoSQL数据库通常采用基于最终一致性的数据同步和管理机制，放宽对一致性的要求，从而实现高可用性和性能。具体来说，NoSQL数据库通常采用以下技术和策略：

副本复制：将数据副本存储在多个节点上，通过异步或同步复制的方式将数据同步到其他节点上，从而实现数据的冗余和高可用性。

分片技术：将数据分散存储在多个节点上，通过分片技术将数据分散到不同的节点上进行存储和访问，从而实现数据的可扩展性和性能。

缓存技术：利用缓存技术将数据缓存在内存中，提高数据的访问速度和响应能力。

柔性事务：采用柔性事务机制，允许数据在一定时间内存在不一致性和延迟，但最终会达到一致的状态。

分布式一致性算法：采用分布式一致性算法，如Paxos算法、Raft算法等，保证数据在多个节点之间的一致性和正确性。

34. NoSQL 数据库的类型包括文档数据库、键值数据库、图数据库、列族数据库等等。

35. 将基础设施作为服务的云计算服务类型是：IaaS

IaaS（Infrastructure as a Service，基础设施即服务）是一种云计算服务模式，它将基础设施作为服务提供给用户。在IaaS模式下，云服务提供商向用户提供计算、存储、网络等基础设施资源，用户可以根据自己的需求选择需要的资源进行配置和部署，从而构建自己的应用程序和服务。用户只需要支付使用的资源费用，而不需要关注基础设施的维护和管理。一个常见的例子是，假设一个公司需要部署一个Web应用程序，他们可以选择使用IaaS服务来构建和部署该应用程序。该公司可以选择一个IaaS服务提供商，如AWS、Azure或GCP，并选择所需的基础设施资源，例如虚拟机、存储、数据库和网络等。然后，他们可以根据自己的需求进行配置和部署，例如选择虚拟机的大小和数量、选择存储类型和容量、配置数据库和网

络等。最后，他们可以将自己的Web应用程序部署到所选的基础设施资源上，并通过互联网向用户提供服务

36. 虚拟化作用于 IaaS 技术架构中那一层？ 动态资源层

动态资源层（Dynamic Resource Layer）是指IaaS（Infrastructure as a Service）技术架构中的虚拟化层，它包括了计算、存储和网络等基础设施资源的虚拟化技术，能够将物理资源抽象为虚拟资源，并为用户提供动态的资源分配和管理功能。动态资源层是IaaS架构中的核心层，它通过虚拟化技术将物理资源抽象为虚拟资源，并为用户提供按需分配和管理资源的功能

在IaaS中，虚拟化技术通常包括以下几种类型：

服务器虚拟化：将一台物理服务器划分为多个虚拟服务器，每个虚拟服务器可以运行不同的操作系统和应用程序，并独立使用计算、存储和网络资源。

存储虚拟化：将多个物理存储设备组合成一个逻辑存储池，用户可以从中分配存储资源，并按需进行扩容或缩减。

网络虚拟化：将物理网络设备虚拟化为多个虚拟网络设备，如虚拟交换机、路由器等，使用户可以进行动态网络配置和管理。

37. 虚拟化可采用两种方法， 一是构建一个运行时系统，提供一套抽象指令集来执行程序。二是提供虚拟机监视器。*虚拟化技术可以采用两种方法实现：一种是基于虚拟机监视器（Virtual Machine Monitor，VMM）的全虚拟化方式，另一种是基于操作系统层面的容器虚拟化方式。*

38. 中间件是一种应用程序， 它在逻辑上位于应用层中，但在其中包含有多种通用协议，这些协议代表各自所在的层，独立于其他更加特别的应用。

中间件可以提供多种服务和功能，如消息传递、数据存储、安全性、事务处理、负载均衡、缓存和集成等。中间件还可以为应用程序提供通用的API和接口，使应用程序可以更加方便地进行通信和交互。

消息队列中间件：用于实现异步消息传递和事件驱动架构，如ActiveMQ、RabbitMQ和Kafka等

数据库中间件：用于实现数据库的分布式访问和负载均衡，如MySQL Proxy和pgpool-II等。

Web服务器中间件：用于实现Web应用程序的访问和负载均衡，如Apache、Nginx和IIS等。

应用服务器中间件：用于实现应用程序的部署和管理，如Tomcat、JBoss和WebLogic等。

缓存中间件：用于实现缓存管理和加速访问，如Redis、Memcached和Ehcache等。

39. 在远程对象调用中， 远程接口使每个远程方法都具有方法签名。

远程对象调用（Remote Method Invocation，简称RMI）是一种分布式计算技术，它允许一个Java程序通过网络调用另一个远程Java对象上的方法。

在RMI中，客户端程序可以通过网络调用远程对象上的方法，就像调用本地对象的方法一样。远程对象可以是在本地计算机上运行的Java对象，也可以是在远程计算机上运行的Java对象。当客户端程序调用远程对象上的方法时，RMI会将方法调用请求发送到远程对象所在的计算机上，在远程计算机上执行该方法，并将结果返回给客户端程序。

方法签名是指一个方法的名称、参数类型和返回类型的组合，用于唯一标识一个方法。在远程对象调用中，需要方法签名来确保客户端程序和远程对象之间的通信正确无误。当客户端程序调用远程对象上的方法时，它需要知道要调用的方法的名称、参数类型和返回类型，以便将方法调用请求正确地发送给远程对象。远程对象也需要知道要执行的方法的名称、参数类型和返回类型，以便正确地执行方法并将结果返回给客户端程序。

40. 所有 DCE 的底层编程模型都是客户-服务器模型。而 DCE 本身的一部分是由分布式文件服务、目录服务、安全服务以及分布式时间服务等构成的。

DCE（Distributed Computing Environment）是一种分布式计算环境，它提供了一组标准化的API和服务，用于实现分布式计算和应用程序的开发、部署和管理。DCE的底层编程模型确实是客户-服务器模型，其中客户端程序通过网络连接到远程服务器上，调用远程服务器上的服务或函数，并接收返回结果。（也是一种中间件）

DCE本身包含了多个服务，包括分布式文件服务、目录服务、安全服务和分布式时间服务等，这些服务都是为了提高分布式计算的可靠性、安全性和可扩展性而设计的。其中，分布式文件服务提供了分布式文件系统的支持，允许用户在分布式环境中访问和管理文件；目录服务提供了分布式命名和目录服务的支持，允许用户在分布式环境中查找和管理对象；安全服务提供了分布式安全机制的支持，允许用户在分布式环境中进行身份认证和访问控制；分布式时间服务提供了分布式时间同步的支持，允许用户在分布式环境中对时间进行同步和管理

41. 在**面向消息的通信**中，通常分为面向消息的瞬时通信和持久通信两种机制。

面向消息的通信是一种通信模式，其中通信的基本单位是消息。在这种通信模式中，发送方将消息发送到消息中间件或消息队列系统中，接收方从消息中间件或消息队列系统中接收消息。

消息通常包括一些元数据和有效载荷。元数据通常包括消息的唯一标识符、发送方和接收方的地址、消息的优先级和时间戳等信息。有效载荷则包括实际要传输的数据，如文本、二进制数据或结构化数据。

瞬时通信是指消息在发送之后，只要接收方已经接收到消息并确认，那么该消息就可以被丢弃。在这种机制下，发送方不需要保留消息的副本，因此瞬时通信通常具有较低的延迟和更高的吞吐量。

相比之下，持久通信则要求消息在被发送之后始终保持存储，直到接收方明确地表示接收到该消息并已经成功处理。在这种机制下，发送方需要保留消息的副本，以便在需要重发消息时使用。持久通信通常用于需要可靠传输和消息持久化的场景，如消息队列、发布/订阅系统和分布式事务等。

42. 在面向消息的瞬时通信中，通常采用套接字接口和消息传递接口。

套接字接口是一种基于网络协议的编程接口，它允许应用程序通过网络进行通信。在套接字接口中，发送方和接收方之间建立一个网络连接，使用TCP或UDP协议进行数据传输。发送方通过套接字接口将消息发送到网络上的对应端口，接收方通过套接字接口监听对应端口并接收消息。

消息传递接口是一种基于消息传递的编程接口，它允许应用程序通过发送和接收消息来进行通信。在消息传递接口中，发送方将消息发送到消息队列或消息中间件中，接收方从消息队列或消息中间件中接收消息。消息传递接口通常具有更高的可靠性和扩展性，因为消息可以被持久化和缓存，从而保证消息传输的可靠性和顺序性。

需要注意的是，套接字接口和消息传递接口并不是互斥的，而是可以在同一个系统中并存。例如，在分布式系统中，可以使用套接字接口进行节点之间的实时通信，同时也可以使用消息传递接口将消息发送到消息队列中进行异步处理。

43. 在面向持久的通信中，消息队列系统为持久异步通信提供多种支持。它提供消息的中介存储能力。

44. 在**消息队列系统**中， 队列由队列管理器来管理，它与发送或接收消息的应用程序直接交互。

队列管理器是一个消息队列系统的核心组件，它负责管理消息队列的创建、删除、配置和监控等任务，同时也负责处理消息的路由、转发、持久化、安全性和事务管理等方面的工作。

具体来说，队列管理器通常具有以下功能：

队列的创建和删除：队列管理器负责创建和删除消息队列，并为队列分配唯一的标识符。

队列的配置：队列管理器可以配置队列的深度、优先级、过期时间、持久化策略、安全性等参数，以满足不同的业务需求。

消息的路由和转发：队列管理器可以根据消息的目的地址、消息的类型、消息的优先级等信息将消息路由到相应的消息队列中，并且可以将消息转发到其他队列或应用程序中。

消息的持久化：队列管理器可以将消息持久化到磁盘中，以保证即使在系统崩溃或断电的情况下，消息也不会丢失。

消息的安全性：队列管理器可以提供消息的加密、认证、授权等安全性保护功能，以确保消息传输的安全性和完整性。

事务管理：队列管理器可以支持事务操作，以保证消息的原子性、一致性、隔离性和持久性

45. **应用层多播**的基本思想是结点组织成一个覆盖网络，然后用它来传播信息给其成员。一个重要的因素是网络路由器不在组成员中。

应用层多播的基本思想是结点组织成一个覆盖网络，因为这种设计可以提供更灵活、更高效和更可靠的多播服务。覆盖网络可以提供更灵活的拓扑结构，可以根据具体的应用需求进行定制和优化。与传统的基于IP协议的多播不同，应用层多播可以采用更复杂的拓扑结构，如树形、网格、蜂窝等结构，以适应不同的应用场景。

46. 在覆盖网络构建时， 主要有两种方法，一种是结点本身直接组织成树； 另一种是结点组织成一个网状网络。

树形结构是指将节点组织成一棵树状结构，其中根节点作为多播源，树的分支代表了不同的多播路径，叶子节点代表了多播组的成员。树形结构具有层次结构和单向传输的特点，可以减少重复的数据传输和降低网络延迟，但是容易产生瓶颈和单点故障。

网状结构是指将节点组织成一个网状网络，其中每个节点都可以直接与其他节点通信，没有中心节点。网状结构具有去中心化、容错性强的特点，可以提高网络的可靠性和灵活性，但是容易产生数据冗余和增加网络负载。

在实际的应用中，树形结构和网状结构可以结合使用，以提高网络的性能和可靠性。例如，在点对点视频流传输中，可以采用网状结构进行多播，以提高网络的可靠性和灵活性；同时，也可以采用树形结构进行单向传输，以减少数据传输的冗余和降低网络延迟。

47. 分布式系统中，有**三种不同的命名系统**，它分别是无层次命名；结构化命名和基于属性的命名。

无层次命名：在无层次命名中，可以为每个文件和目录分配一个唯一的名字，例如"file1"、"dir1"等。这种方式简单、灵活，但是命名冲突的可能性较大，不适用于大规模系统。

结构化命名：在结构化命名中，可以采用层次结构进行命名，例如将文件和目录组织成树形结构，每个节点都被分配到一个唯一的路径中，例如"/root/dir1/file1"等。这种方式层次明确、简单清晰，可以有效地避免命名冲突，适用于大规模系统。

基于属性的命名：在基于属性的命名中，可以为每个文件和目录赋予一组属性，例如文件类型、文件大小、创建时间等，可以通过查询属性值来获取文件和目录，例如查询所有类型为文本文件、大小超过1MB、创建时间在2020年之后的文件等。这种方式灵活、高度自适应，适用于动态变化和高度异构的系统。

48. 在无层次命名中，通常有广播和多播、转发指针、基于宿主位置、分布式散列表、分层结构等方法实现实体定位。

在广播方式中，请求会被发送到整个系统中的所有节点，每个节点都会接收到该请求并进行响应。在多播方式中，请求只会被发送到指定的多个节点中，只有这些节点会接收到该请求并进行响应。广播和多播方式的优点是简单、易于实现，但是在大规模系统中会产生较大的网络负载和延迟。

转发指针中，每个实体都保存了一个或多个指向其他实体的指针，通过这些指针可以实现实体的定位。当一个实体接收到请求时，它可以根据指针将请求转发到其他实体中，直到找到目标实体为止。转发指针的优点是具有较高的灵活性和可扩展性，但是需要较多的通信开销和存储开销。

在基于宿主位置中，每个实体都记录了自己所在的宿主位置，并将宿主位置信息共享给其他实体。当一个实体接收到请求时，它可以根据宿主位置信息将请求转发到目标宿主上，并在该宿主上查找目标实体。基于宿主位置的方法具有较高的效率和可靠性，但是需要较多的宿主位置信息共享和管理。

在分布式散列表中，实体被分布在多个节点上，每个节点负责管理一部分实体，并记录每个实体的位置信息。当一个实体接收到请求时，它可以根据散列表的分布规则将请求转发到对应的节点上，并在该节点上查找目标实体。分布式散列表的优点是具有较高的效率和可扩展性，但是需要较多的散列表管理和维护。散列表的value对于本机资源就是资源地址，对于其他机器的资源就是其他机器的地址。

在分层结构中，实体组织成多个层次结构，每个层次结构负责管理一部分实体，并记录每个实体的位置信息。当一个实体接收到请求时，它可以根据层次结构的分布规则将请求转发到对应的层次上，并在该层次上查找目标实体。分层结构的优点是具有较高的效率和可扩展性，但是需要较多的层次结构管理和维护

49. **基于属性的命名系统**实现的方式有两种。一种是分层实现，使得目录项集合形成了分层的目录信息树。而另一种是非集中式实现，它是采用映射到分布式散列表的方式。

可以使用数据库实现key-value的目录

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。

全序多播是一种将所有消息按照相同的顺序传递给每个接收者的多播操作，它可以用于实现共享状态或复制状态的一致性。Lamport 时间戳是一种广泛使用的算法，可以在分布式系统中实现全序多播。（注意：多播的对象是一个指定的组，而不是全部机器）

Lamport 时间戳的基本原理是为每个事件分配一个唯一的时间戳，这个时间戳由进程标识符和递增的计数器组成。当一个进程发送一条消息时，它将自己的时间戳附加到消息上，然后将消息发送到所有接收者。当一个接收者收到一条消息时，它将自己的时间戳与消息中的时间戳进行比较，然后根据时间戳的大小关系来决定消息的顺序。

具体来说，当一个接收者收到一条消息时，它将自己的时间戳与消息中的时间戳进行比较。如果自己的时间戳比消息中的时间戳小，则说明自己的事件发生在消息的事件之前，此时接收者将暂时缓存这条消息，并等待更早的消息到达。如果自己的时间戳比消息中的时间戳大，则说明自己的事件发生在消息的事件之后，此时接收者可以立即处理这条消息。如果自己的时间戳与消息中的时间戳相等，则需要进一步比较进程标识符来决定消息的顺序。

通过使用 Lamport 时间戳，分布式系统可以在完全分布式的方式下实现全序多播，从而确保所有接收者按照相同的顺序处理消息。但需要注意的是，使用 Lamport 时间戳并不能解决所有的一致性问题，例如处理并发写入的数据一致性问题。因此，在实际应用中，还需要综合考虑其他一致性算法和技术。

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。

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。

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。

崩溃性故障：leader节点发生崩溃性故障时，其他节点通过**心跳机制**检测到该节点的状态，并重新选举一个新的leader来继续处理决策过程。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。

定时性故障：通过时钟超时检测故障

响应性故障：

随意性故障：随机性故障（或者称为非恶意故障）是指节点由于硬件故障、网络问题或其他原因而出现的错误行为。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发生故障任然能够发生作用。

54. 在分布式锁中，**惊群效应**指的是，在有多个请求等待获取锁的时候，一旦占有锁的线程释放之后，如果所有等待的方都同时被唤醒， 尝试抢占锁。但是这样的情况会造成比较 大的开销， 那么在实现分布式锁的时候， 应该尽量避免惊群效应的产生。

惊群效应（Thundering Herd）是指在高并发系统中，当某个资源（如锁）的独占权被释放时，所有等待该资源的线程都会被唤醒，但只有一个线程能够获得该资源，其他线程则会再次进入等待状态。这种情况下，由于大量线程同时竞争同一资源，导致系统性能急剧下降，甚至出现系统崩溃的情况。

惊群效应通常发生在分布式系统中，由于网络延迟等原因，多个节点或进程同时请求同一个资源。当该资源的独占权被释放时，所有节点或进程都会收到通知，这时就会发生惊群效应。

为避免惊群效应的出现，通常采用一些技术手段。例如在分布式锁中可以采用随机等待时间、队列调度、分布式锁降级、精细化锁设计等方法来减少锁竞争的激烈程度，从而避免惊群效应的发生。采用随机等待时间：等待获取锁的线程可以在一定的时间范围内随机等待，这样可以避免所有线程同时抢占锁，减少锁竞争的激烈程度。利用队列进行调度：等待获取锁的线程可以加入一个队列中，当锁被释放时，只有队列中的第一个线程可以获取锁，这样可以避免所有线程同时抢占锁。

55. 分布式系统中，传统的**选举算法**有两种，一是欺负选举算法；二是环选举算法。

欺负选举算法（Bully Algorithm）：欺负选举算法是一种基于节点编号的选举算法，它的基本思想是，节点编号最大的节点可以成为主节点。当一个节点发现主节点不可用时，它会向比自己编号小的节点发送选举请求，要求它们放弃主节点的权利，然后自己成为主节点。如果没有比它编号小的节点响应，则该节点成为主节点。欺负选举算法的缺点是，当节点数量较多时，选举时间会很长，且选举期间会出现网络拥塞的情况。

环选举算法（Ring Algorithm）：环选举算法是一种基于环形结构的选举算法，它的基本思想是，将所有节点排成一个环形，每个节点只与相邻的节点通信。当一个节点发现主节点不可用时，它向它的下一个节点发送选举请求，请求它成为主节点。下一个节点如果发现主节点不可用，则向它的下一个节点发送选举请求，以此类推，直到找到一个可用节点为止。环选举算法的缺点是，如果节点故障率较高，选举时间会很长。

Raft选举算法的基本思想是，当集群中没有主节点时，节点会随机等待一段时间，然后发起选举。选举过程分为两个阶段：首先，节点成为候选人，向其他节点发送投票请求，要求它们投票给自己；其次，如果候选人获得了多数节点的投票，则成为新的主节点。

Raft选举算法的设计考虑了多种情况，例如节点网络延迟、故障、重启等情况，从而保证了选举的正确性和可靠性。与欺负选举算法和环选举算法相比，Raft选举算法具有以下特点：

随机等待时间：Raft选举算法在节点没有主节点时，会随机等待一段时间再发起选举，从而避免了所有节点同时发起选举的情况，减少了网络拥塞和竞争激烈程度。

非对称投票：Raft选举算法采用非对称投票的方式，即每个节点只能投一次票。这样可以避免欺负选举算法中节点编号较大的节点一直占据主节点的情况，提高了选举的公正性。

选举超时：Raft选举算法引入了选举超时机制，即当节点等待时间过长时，会重新发起选举。这样可以避免节点网络延迟或故障的情况下选举时间过长。

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。

避免死锁：在分布式互斥算法中，为避免死锁的发生，通常采用超时机制、时间戳机制等。例如，在使用分布式锁的场景中，如果一个节点在一定时间内没有完成对共享资源的访问，那么其它节点可以发起请求，抢占锁的使用权，从而避免了死锁的发生。

避免饥饿：为避免某个节点一直被其他节点排挤的情况，分布式互斥算法通常采用公平性策略，如时间戳机制，使得所有节点都有机会获得共享资源的访问权，避免了饥饿的发生。

在Raft算法中，所有节点通过日志复制机制来达成共识，即所有节点都接受相同的日志序列。为了保证日志复制的正确性和一致性，Raft算法引入了Leader节点，即主节点，负责日志的复制和提交。当某个节点发现Leader节点不可用时，会耗尽时钟并再次发起选举，选出新的Leader节点。

在Raft算法中，选举过程采用了随机等待和超时机制，避免了死锁的发生。同时，Raft算法中的Leader节点采用了心跳机制来保持其活跃性，避免了饥饿的发生。

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。

拜占庭问题通常采用拜占庭将军问题作为形象化的描述。在一个拜占庭帝国的将军团队中，一些将军可能会叛变或者失去联系，而其他将军需要通过沟通来达成一致决策，保证最终的行动是正确的。这个问题的关键在于如何在存在叛徒或者失联将军的情况下，保证所有忠诚的将军能够达成一致，并且能够排除叛徒的影响。

在分布式系统中，拜占庭问题通常指的是在存在故障或者恶意节点的情况下，如何保证分布式系统的正确性和一致性。在这个问题中，分布式系统中的节点可以是计算机、服务器、传感器等设备。由于分布式系统中的节点之间通过网络进行通信，因此存在通信延迟、消息丢失、节点故障等问题，可能会影响系统的正确性和一致性。拜占庭问题的解决方案是设计一种算法或协议，在存在故障或者恶意节点的情况下，保证所有正确节点能够达成共识，并且排除错误节点的影响。

拜占庭容错算法通常是为了处理分布式系统中节点失效或者恶意行为等问题，因此其主要目标是保证系统的可靠性和安全性。不同的拜占庭容错算法可能会有不同的一致性级别，例如强一致性、最终一致性或者弱一致性等。其中，强一致性是最高级别的一致性级别，可以保证所有操作都具有ACID特性。

PBFT（Practical Byzantine Fault Tolerance）算法是比较著名的拜占庭容错算法之一，它可以实现强一致性。PBFT算法采用了多数派决策机制和消息签名等技术来保证节点之间的通信安全和正确性，通过多轮消息交换来达成共识并实现状态机复制。在PBFT算法中，只有当超过2/3的节点达成一致，系统才认为达成了共识，因此PBFT算法可以实现强一致性。

Raft算法处理节点失效时：

如果领导者失效，剩余节点会发起新的领导者选举。Raft 算法通过投票机制来选举新的领导者，只有得到超过半数的节点投票才能成为新的领导者。新的领导者会复制原来领导者未提交的日志并提交它们。

如果跟随者失效，领导者会尝试向该节点发送心跳消息。如果该节点在一定时间内没有响应，则认为该节点失效。领导者会将该节点标记为失效状态，并继续向其他节点发送心跳消息。如果该节点恢复，则会发起同步请求，领导者会向该节点发送缺失日志条目并将其同步到最新状态。但需要注意的是，这种同步方式不是强一致性的，因为领导者可能在恢复的跟随者还没有完全同步之前就提交了新的日志条目。

如果有超过半数的节点失效，则 Raft 算法可能无法达成共识，因为 Raft 算法要求超过半数的节点都复制了相同的日志条目才能认为该条目已经被提交并生效。如果节点失效的数量超过了系统所能容忍的最大失效数，那么系统可能会陷入无法恢复的状态，无法选出一个 leader，此时需要手动干预来恢复系统的可用性。

58. 容错系统一般包括固态存储、故障-停止处理器以及原子操作。

坚固存储器（Solid-state storage）通常是指使用闪存等非易失性存储介质的存储设备，具有高速读写、低耗能、抗震动等特点，同时也具有较高的可靠性和耐久性。在容错系统中，使用坚固存储器可以保证数据的持久性和可靠性，即使系统发生故障也能够保证数据的完整性和可用性。

故障-停止处理器（Fail-stop processor）通常是指在出现故障时可以停止工作的处理器，这种处理器通常具有高可靠性和可用性。在容错系统中，使用故障-停止处理器可以避免节点出现故障后继续参与系统运行，从而避免数据的错误传播和系统的崩溃。

原子操作（Atomic operation）是指不可分割的操作，要么全部执行成功，要么全部不执行。在容错系统中，如果某个操作不是原子操作，当系统发生故障或错误时，可能会导致操作被中断，而操作只完成了一部分，这就会导致数据不一致，从而影响系统的正确性和可靠性。

59. 服务器集群在逻辑上由三层组成，第一层是逻辑交换机； 第二层是应用/计算服务； 第三层是文件/数据库系统。

服务器集群在逻辑上由三层组成，第一层是逻辑交换机； 第二层是应用/计算服务； 第 三层是文件/数据库系统。

这样的服务器集群可以被称为三层架构（Three-tier Architecture），也称为分层架构（Layered Architecture），是一种常见的系统架构设计。

在这种架构下，第一层逻辑交换机通常是用来实现负载均衡、网络路由等功能，可以将请求分发到不同的服务器上，从而提高系统的性能和可扩展性。

第二层应用/计算服务层通常是实现业务逻辑的核心部分，包括应用程序、计算服务、消息队列等，主要负责处理业务数据和逻辑，提供功能服务。

第三层文件/数据库系统层通常是用来存储和管理数据的，包括文件系统、数据库系统等，主要负责数据的存储、查询和管理。

60. MapReduce 借鉴了函数式程序设计语言的设计思想，其软件实现是指定一个Map 函数，把键值对(key/value) 映射成新的键值对(key/value) ，形成一系列中间结果形式 的key/value 对， 然后把它们传给 Reduce(规约)函数， 把具有相同中间形式 key 的 value 合并在一起。

61. 有两种实现**线程包**的基本方法： 一是可以构造一个完全在用户模式下执行的线程； 二是由内核来掌管线程并进行调度。

线程包是指一组用于创建和管理线程的类和函数库。线程包提供了一系列的接口，使得程序员可以方便地创建、启动、挂起、恢复和销毁线程，以及进行线程同步和互斥操作。

62. IDL 编译器的输出包括三个文件，它们是头文件、客户存根和服务器存根。

头文件：IDL编译器会根据IDL文件生成对应的头文件，头文件中包含了IDL接口的定义以及数据类型的定义。头文件通常用于客户端和服务器端的开发，客户端需要包含头文件来调用服务器端的接口，服务器端需要包含头文件来实现接口。（编译IDL就能得到一个）

客户存根：IDL编译器会根据IDL文件生成客户存根，客户存根是客户端程序中用于调用远程服务的代理程序。客户存根封装了远程服务的调用细节，客户端程序只需要调用客户存根提供的接口即可，无需关心远程服务的具体实现。

服务器存根：IDL编译器会根据IDL文件生成服务器存根，服务器存根是服务器端程序中用于实现远程服务的代理程序。服务器存根接收客户端的请求，然后调用实际的服务实现来处理请求，并将处理结果返回给客户端。

感觉和protoBuf很像，都需要一个类似profile的东西来指定格式，然后编译生成一个头文件参与具体的代码构建。

63. 在面向消息的通信中，通常分为面向消息的瞬时通信和持久通信两种机制。

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。

基于令牌的算法：这种算法通过传递令牌来实现互斥访问共享资源。当一个节点拥有令牌时，它才能够访问共享资源。在访问完成后，节点将令牌传递给下一个节点。常见的基于令牌的算法包括Token Ring算法（不断向后传递，拿到令牌的可以访问资源）、Bully算法（选择编号最大的作为协调者，由其调控资源分给谁）等。

非基于令牌的算法：这种算法不需要传递令牌，而是通过节点之间的相互协调来实现互斥访问共享资源。常见的非基于令牌的算法包括Lamport算法、Ricart-Agrawala算法、Suzuki-Kasami算法等。

基于Quorum的方案：这种算法利用Quorum集合来实现互斥访问共享资源。Quorum集合是指一个集合，其中任意两个节点都有至少一个共同的节点。当一个节点想要访问共享资源时，它必须先获取一个Quorum集合的锁，并且只有持有该锁的节点才能够访问共享资源。常见的基于Quorum的方案包括Quorum-based Mutual Exclusion算法等。

65. **网络协议有三要素**组成， 时序是对事件实现顺序的详细说明；语义是指需要发出何种控制信息以及要完成的动作与作出的响应； 语法是指用户数据与控制信息的结构与格式。

66. 在**名称解析**的实现中，通常采用两种方法，一是迭代名称解析；二是递归名称解析。

67. 用户第一次请求时， 负载均衡器将用户的请求转发到了 A 服务器上， 如果负载均衡器 设置了粘性 Session 的话，那么用户以后的每次请求都会转发到 A 服务器上。

粘性 Session（Sticky Session），也称为会话保持或会话粘滞，是一种负载均衡技术，用于确保用户在一段时间内始终与同一台后端服务器进行通信。在粘性 Session 中，负载均衡器在第一次请求时将用户请求发送到一个特定的后端服务器上，并在负载均衡器上维护与该用户相关的会话信息。随后，负载均衡器会将该用户的所有后续请求路由到同一台服务器上，以确保用户的会话状态在整个应用程序中保持一致。

粘性 Session 可以通过多种方式实现，最常见的方式是在用户的浏览器中设置一个 cookie，将该 cookie 的值与特定的后端服务器绑定。当用户发起后续请求时，负载均衡器会检查该 cookie，并将请求路由到与该 cookie 绑定的后端服务器上。

68. 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。

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。

原子多播（Atomic Multicast）是一种分布式系统中的通信模式，它可以将一条消息同时发送给多个接收方，并且确保所有接收方都接收到相同的消息，而且只接收一次(普通多播就无法保证)。在原子多播中，消息的排序非常重要，因为不同的排序方法可能会导致不同的消息传递语义。

常见的原子多播消息排序方法包括：

不排序的多播：消息没有任何排序限制，接收方可以按照任何顺序接收消息。

FIFO 顺序的多播：消息按照发送的顺序进行排序，每个接收方都按照相同的顺序接收消息

按因果关系排序多播：消息按照因果关系进行排序，即如果消息 A 发送后消息 B 发送，那么消息 B 的排序必须在消息 A 之后。这种排序方法可以确保消息的因果关系得到维护。

全序多播：消息按照全局的顺序进行排序，即所有接收方都按照相同的顺序接收消息。这种排序方法可以确保消息的全局排序。

70. **虚拟化的典型类型**: 基础设施虚拟化、系统虚拟化、软件虚拟化。

71. **虚拟化的目的**：对象脱离原有环境、在计算机上被表示、通过计算机控制按需获取。

具体来说，虚拟化的目的包括：

资源隔离：通过虚拟化，可以将物理资源划分为多个虚拟资源，实现资源的隔离，避免资源之间的干扰和冲突。

资源共享：通过虚拟化，可以将物理资源共享给多个虚拟资源，提高资源的利用率，减少资源浪费。

资源动态分配：通过虚拟化，可以根据实际需求动态分配资源，提高系统的灵活性和响应能力。

应用程序隔离：通过虚拟化，可以将应用程序隔离在不同的虚拟环境中，避免应用程序之间的干扰和冲突，提高系统的安全性和稳定性。

环境隔离：通过虚拟化，可以为不同的应用程序创建不同的虚拟环境，避免应用程序受到底层操作系统和硬件环境的影响，提高应用程序的可移植性和兼容性。

综上所述，虚拟化的目的是为了提高资源的利用率和灵活性，以及提高系统的安全性和稳定性，实现资源的隔离、共享和动态分配，以及应用程序的隔离和环境的隔离。

72. **分布式加密系统**通常有三种类型，一是对称加密系统(DES)；二是公钥加密系统(RSA)、三是散列函数(MDS) 系统。

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。

基础设施层（Infrastructure as a Service，IaaS）：这是云计算架构中的底层基础设施，包括服务器、存储设备、网络设备等。部署基础设施层是为了支持IaaS服务，将计算、存储和网络资源通过虚拟化技术进行抽象和隔离，为上层的云平台层提供支撑。

云平台层（Platform as a Service，PaaS）：这是在基础设施层之上构建的中间层，为开发人员提供应用程序开发和部署的平台。云平台层将应用程序运行环境、中间件和开发工具等进行虚拟化和隔离，提供了一种更高层次的抽象，使得开发人员可以专注于应用程序的开发和部署，而不需要关注底层的基础设施。

应用层（Software as a Service，SaaS）：这是云计算架构中的最上层，为最终用户提供各种应用程序和服务，例如电子邮件、在线办公、社交媒体等。在应用层之上的服务都是通过云平台层和基础设施层来实现的。

75. SaaS 是一种基于互联网提供软件服务的应用模式

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。

GFS和BigTable是Google的两个分布式存储系统，它们在存储模型和应用场景上有所不同。下面是它们在存储区别方面的简单比较：

存储模型：GFS是一个分布式文件系统，主要用于存储大型文件和数据集。它以文件为单位进行数据存储和管理，并支持随机访问和追加写入。而BigTable是一个分布式数据库，主要用于存储结构化数据。它以行列式存储模型进行数据存储和管理，支持高性能的随机读写和查询。

存储方式：GFS使用分布式的文件系统来存储数据，采用多个服务器构成的存储集群，数据被分成多个块分布在不同的服务器上。而BigTable使用分布式的数据表来存储数据，采用分区、列族、列和行的层次结构，数据被分散存储在多个服务器上。

应用场景：由于存储模型和存储方式的不同，GFS和BigTable在应用场景上也有所不同。GFS适用于存储和处理大型文件和数据集，例如Google的日志存储、Web索引和地图数据等。而BigTable适用于存储和处理结构化数据，例如Google广告系统的用户数据存储和检索等

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。

80. 一个分布式系统就是一组 独立的 计算机的集合，但是这组计算机在用户看来是一个 整体的 系统。

81. **分布式系统的可扩展性**包括 大小规模的 可扩展性、 地理上的 可扩展性和 管理上的 可扩展性三个层面。

82. **分布式对象**是指 实现对象的接口 与 对象本身 位于不同的机器上的对象。 (通信 1)

83. **时钟同步算法**中，Cristian 算法的时间服务器的工作方式是 被动的 ，而 Berkeley 算法的时间服务器是 主动的 。

Cristian算法是一种基于客户-服务器架构的时钟同步算法，它使用一个时间服务器来为客户端节点提供准确的时间。当一个客户端需要同步其本地时钟时，它会向时间服务器发送一个请求，服务器会回复一个时间戳，客户端通过计算服务器返回的时间戳与请求时间的差值，来调整本地时钟。Cristian算法的主要缺点是它需要一个可靠的时间服务器，并且它无法处理来自不可信客户端的请求。（Cristian服务器被动接受）

Berkeley算法是一种基于对等网络架构的时钟同步算法，它使用一个主节点来协调所有其他节点的时钟。它更常见的是基于对等网络（P2P）架构实现。在Berkeley算法中，每个节点都可以向其他节点发送请求，并且任何一个节点都可以充当主节点来协调其他节点的时钟，

因此它不需要一个特定的中心节点或服务器，而是具有更加分散的特点。当节点需要同步其本地时钟时，它会向主节点发送一个请求，主节点会计算出所有节点的时间偏差，并将平均值返回给节点，节点通过调整本地时钟来同步时间。Berkeley算法的优点是它可以处理来自不可信节点的请求，并且它的时间同步精度相对较高。但是，它需要一个可靠的主节点，并且可能会受到网络延迟和时钟漂移等因素的影响。

84. Web 服务器中，servlet 是在 服务器地址空间中由服务器 执行，而 CGI 程序 则是以 单独的进程 方式执行。(Servlet 程序与CGI 程序最大区别在于 CGI 程序 以单独的进程执行，而 servlet 则是在服务器地址空间中由服务器执行。)

85. **分布式系统进行复制的副本**类型有 永久副本 、 服务器启动的副本 和 客户启动的副本 三种。

86. **递归名称解析比迭代名称解析**的通信开销 低 ，对名称服务器的性能要求 高 。

87. RPC 通信中， 服务器的崩溃又可以进一步分为 执行之后崩溃 和 执行之前崩溃 两种情况。

88. 加密系统可以分为 对称加密 与 非对称加密 两种。

89. NFS (网络文件系统) 系统的基本底层模型叫 远程访问模型(remote access model) 。

90. 区块链是在没有中央控制点的分布式对等网络中，使用分布式集体运作的方法，维护一套不易篡改、可靠数据库的技术方案，其特点为去中心化存储、信息高度透明、不易篡改、加密安全性高等。

91. 机器对资源的绑定有 标识符绑定 ， 按值绑定 ， 按类型绑定 三种 方式。

进程对资源的三种绑定方式：

最强的绑定方式是**标志符绑定**，进程使用资源的标志符来引用资源，例如 URL。

较弱的一种方式是只使用资源的值， 称为**按值绑定**。例如 C 或者 Java 程序使用的库或者类包文件。 最弱的一种绑定方式是**按类型绑定**。进程只制定资源的类型。

标识符绑定（Symbolic binding）：进程使用资源的标识符来引用资源。例如，一个进程使用一个文件的路径名来访问该文件，或者使用一个 URL 来访问一个网络资源。这种绑定方式是最强的，因为它们提供了对资源的直接和精确的访问。

值绑定（Value binding）：进程使用资源的值来引用资源。这种方式通常用于访问库或类包文件，例如在 C 或 Java 程序中使用库函数或类包。这种绑定方式比标识符绑定弱，因为它们并不提供对资源的直接访问。

类型绑定（Type binding）：进程只指定资源的类型，而不是具体的标识符或值。这种绑定方式通常用于访问操作系统资源，例如通过调用操作系统提供的 API 函数来访问资源。这种绑定方式是最弱的，因为它们提供的访问是间接的，需要通过操作系统来实现。

二、非填空题

1. 中间件在分布式系统中扮演着什么角色？

答：中间件主要是为了增强分布式系统的透明性(这正是网络操作系统所缺乏的)，换言之， 中间件的目标是分布式系统的单系统视图。

2. 什么是开放的分布式系统？开放性带来哪些好处？

答：开放的分布式系统根据明确定义的规则来提供服务。开放系统能够很容易地与其它系统协作， 同时也允许应用移植到同一个系统的不同实现中。

开放的分布式系统是指遵循一组明确定义的规则和接口，可以提供服务并与其他系统进行协作的分布式系统。开放性带来了许多好处，包括以下几点：

1. 可移植性：开放的分布式系统采用标准接口和协议，使得应用程序可以很容易地移植到不同的系统中运行
2. 互操作性：开放的分布式系统可以与其他系统进行互操作，使得不同系统之间可以共享数据和服务，从而促进了系统之间的整合和协作。
3. 可扩展性：开放的分布式系统可以通过添加新的组件和服务来扩展其功能和性能，而不需要对整个系统进行大规模的修改。
4. 灵活性：开放的分布式系统可以灵活地适应不同的应用场景和需求，因为它们通常具有可配置的参数和选项。
5. 开放性：开放的分布式系统基于标准接口和协议，使得系统的开发和部署更加容易和灵活，同时也促进了技术和标准的进步。

3. 分布式系统的类型。

(1)分布式计算系统(分为群集计算系统和网格计算系统)

(2)分布式信息系统(分为事务处理系统和企业应用集成)

(3)分布式普适系统(如家庭系统、电子健保系统、传感器网络)

4. 计算机系统的硬件异构性、软件异构性主要表现在哪几方面？

计算机系统的硬件异构性主要表现在三个方面：指令系统、数据表示方法和机器配置。这些差异会导致程序模块不能在不兼容的机器上执行，也会影响系统的兼容性和可移植性。软件异构性包括操作系统和编程语言的异构性。

计算机系统的硬件异构性主要有三个方面的表现， 即：

①计算机的指令系统不同。这意味着一种机器上的程序模块不能在另一种不兼容的机器 上执行，很显然，一种机器上的可执行代码程序不能在另一种不兼容的机器上执行。

②数据表示方法不同。例如不同类型的计算机虽然都是按字节编址的，但是高字节和低 字节的规定可能恰好相反。浮点数的表示方法也常常不一样。

③机器的配置不同。尽管机器的类型可能相同，其硬件配置也可以互不兼容。

计算机系统的软件异构性包括操作系统异构性和程序设计语言异构性。

操作系统异构性的三个主要表现方面为：

①操作系统所提供的功能可能大不相同。例如，不同的操作系统至少提供了不同的命 令集。

②操作系统所提供的系统调用在语法、语义和功能方面也不相同。

③文件系统不同。

程序设计语言的异构性表现在不同的程序设计语言用不同方法在文件中存储数据。

5. 对服务器进程中的线程数目进行限制有意义吗？

答：有。原因有两个：内存资源开销和线程无序运行的不稳定性

线程需要内存来设置他们的私有堆栈。因此， 线程太多可能导致消耗过多的存储器。

更严重的情况是，对于一个操作系统，独立的线程是以无序的方式在运行。在虚拟存储器系统中，构建一个相对稳定的工作环境可能比较困难，从而导致许多的页错误和过多的 I/O 操作，结果可能导致系统性能的下降。

6. 请描述在客户和服务器进程间使用套接字时如何进行无连接通信？

答： 同时在客户端和服务器上创建一个套接字，但只有服务器套接字绑定到本地终结点。然后， 服务器可以随后做一个阻塞读取调用用以等待接收从任何客户端传入的数据。同样， 在 创建套接字后，客户端仅仅做一个阻塞调用以向服务器写入数据.这是没有必要关闭连接的。

6. 移动计算和普适计算的区别

普适计算包括移动计算，但强调环境驱动性，需要具有高度的环境感知性和自适应能力，包括中间件、人机交互、嵌入式技术、传感器、网络技术等领域。普适计算需要解决扩展性、异构性、集成、上下文感知和不可见性等问题，其中不可见性要求系统具有自动和动态的配置机制，无需用户或最少干预。

普适计算可包括移动计算，但普适计算不是移动计算， 前者更强调环境驱动性。从技术上来说， 这就要求普适计算对环境信息具有高度的可感知性，人机交互更自然化，设备和网络的自动配置和自适应能力更强， 所以普适计算的研究涵盖中间件、移动计算、人机交互、嵌入式技术、传感器、网络技术等领域。普适计算要解决的问题包括： 扩展性、异构性、不同构

件的集成、上下文感知和不可见性(Invisibility) 。其中不可见性对普适计算来说是至关重要的， 因为它要求系统无需用户干预或只需要最少干预，也就是要求系统具有自动和动态的配置机制。

7. 简述分布式系统中异构性的解决方案。

在分布式系统中，可以采用以下解决方案来应对异构性问题：中间件、移动代码和虚拟机。中间件提供编程抽象，屏蔽了底层网络、硬件、操作系统和编程语言的异构性，如软总线、CORBA和RMI；移动代码可以在不同的操作系统和硬件平台上运行，例如Java applet；虚拟机提供了一种代码可以在任何计算机上运行的方法，编译器生成虚拟机代码，虚拟机通过解释的方式来执行它。

● 中间件---是指一个软件层，它提供了一个编程抽象，同时屏蔽了底层网络、硬件、 操作系统和编程语言的异构性。 softbus

- 公共对象请求代理(Common Object Request Broker, CORBA)
- JAVA 远程调用方法(Remote Method Invocation， RMI)
- 大多数中间件基于互联网协议实现，这些协议屏蔽了底层网络的差异，但

是中间件要解决操作系统和硬件的不同。

● 移动代码---是指从能从一台计算机发送到另一台计算机，并在目的计算机上运行的 代码(例如：Java applet)

● 虚拟机方法提供了一种代码可以在任何计算机上运行的方法：某种语言的编译器生 成一台虚拟机代码而不是某种硬件代码， 虚拟机通过解释的方式来执行它。

9. 简要描绘全局唯一标识符的一个有效实现

答： 这些标识符可以在以下方式中可以局部产生:将产生标识符的机器所在的网络地址，附 加上当地时间,沿用一个伪随机数.虽然，在理论上,另一台机器也很有可能产生相同的数字, 这种机会微乎其微。

2、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？

参考答案：

①资源的多重性带来的问题。由于处理资源的多重性，分布计算系统可能产生的差错类型和次数都比集中式单机系统多。最明显的一个例子是部分失效问题：系统中某一个处理资源出现故障而其他计算机尚不知道，但单机系统任何一部分出现故障时将停止整个计算。另一个例子是多副本信息一致性问题。可见，资源多重性使得差错处理和恢复问题变得很复杂。资源多重性还给系统资源管理带来新的困难。

②资源的分散性带来的问题。在分布计算系统中，系统资源在地理上是分散的。由于进程之间的通信采用的是报文传递的方式进行的，通信将产生不可预测的、有时是巨大的延迟，特别是在远程网络所组成的分布计算系统中更是这样。例如使用卫星通信会产生 270 毫秒的延迟。在分布计算系统中，系统的状态信息分布在各个分散的节点上。分布式的状态信息和不可预知的报文延迟使得系统的控制和同步问题变得很复杂，要想及时地、完整地搜集到系统各方面的信息是很困难的，从而使处理机进行最佳调度相当困难。

③系统的异构性带来的问题。在异构性分布计算系统中，由于各种不同资源（特别是计算机和网络）的数据表示和编码、控制方式等均不相同，这样一来就产生了翻译、命名、保护和共享等新问题。

由于上述原因，分布计算系统的研制，特别是软件的验证、调试、测量和维护问题变得很复杂。这些正是分布计算系统研制者要解决的主要问题。

10.

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。

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。

某个操作可以重复多次而无害出。也就是重复执行多次与执行一次的效果是相当的。 幂等性是分布式系统设计中十分重要的概念， 假设有一个从账户取钱的 API：

boolean withdraw(account_id, amount)

它实现的功能是从 account_id 对应的账户中扣除 amount 数额的钱； 如果扣除成功则返 回 true，账户余额减少 amount；如果扣除失败则返回 false ，账户余额不变。

因为生产环境的网络状况，用户操作的情况非常复杂，比如 API 的请求已经被服务器 端正确处理，但返回结果由于网络原因导致客户端无法得知处理结果，可能会使用户认 为上一次操作失败了，然后刷新页面，

这就导致了 withdraw 被调用两次，账户也被多扣了一次钱。

这个问题的解决方案一是采用分布式事务，但是分布式事务一方面架构太重量级， 容易 被绑在特定的中间件上，不利于异构系统的集成；另一方面分布式事务虽然能保证事务 的ACID 性质， 而但却无法提供性能和

可用性的保证。

另一种更轻量级的解决方案是幂等设计。我们可以通过一些技巧把 withdraw API 变成 幂等的，比如：

int create_ticket()

boolean idempotent_withdraw(ticket_id, account_id, amount)

create_ticket 实现的功能是获取一个服务器端生成的唯一的 ticket_id，它用于标识后续 的操作。

idempotent_withdraw 和 withdraw 的区别在于关联了一个 ticket_id，一个 ticket_id 表示的 操作至多只会被处理一次，每次调用都将返回第一次调用时的处理结果。这样，idempotent_withdraw 就符合幂等性了，客户端就可以放心地多次调用。

HTTP GET 方法用于获取资源， 不应有副作用，所以是幂等的。POST 方法不具备幂等性。PUT 方法具有幂等性。

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

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，可以重复发送请求，保证至少执行一次。但需要注意，由于系统异步的特性，响应可能不总是最新的。

A. Submitting a blog post that has already been composed in a local file and includes a timestamp and the users name at the beginning of the blog text.

Solution: Either one of the following is correct, as long as the explanation is reasonable.

_ At-least-once. It is not harmful, in this case, to keep sending call requests to the server machine until we get an ACK. In other words, it wouldn’t hurt to post two identical blogs in a row.

_ At-most-once. User can simply re-post if the previous attempt is not successful. This semantic helps avoid indefinite blocking, or any other reasonable concerns.

B. Creating a subdirectory in a distributed file system.

Solution: Either one of the answers can be accepted, with appropriate justification:

_ At-least-once: The same directory can only be created once, after which the operation becomes idempotent. Also, in this case, issuing the same calls multiple times (when no ACK is received from the DFS server) is harmless to the user.

_ At-most-once: Server could crash (or unreachable), and if it happens, we don't want to wait forever. Instead, we can use at-most-once semantic to declare a timeout.

C. Buying a mobile phone from an e-commerce website

Exactly-once: We need to make sure that the transaction happens, and happens once only.

D. Checking the stock price of a company

At-least-once: Since the operation is idempotent, meaning the global state is preserved even if multiple calls may have been issued, we can keep sending requests; and this semantic guarantees that the checkPrice() call executes at least once.

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。

a) 基于渠道

发布者发布事件到命名的渠道， 订阅者订阅其中一个已命名的渠道，并接收所有发 送到那个渠道的事件

b) 基于主题

假设每个通知中包含多个域，其中一个域表示主题。订阅是根据感兴趣的主题来定 义的。

c) 基于内容

基于内容的方法是基于主题方法的一般化，它允许订阅表达式具有一个事件通知上 的多个域。基于内容的过滤器是事件属性值的约束组合定义的查询。

d) 基于类型

订阅根据事件类型来定义，匹配根据给定的过滤器的类型或者子类型来定义。

e) 基于概念的订阅模型

过滤器可以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。

泛洪（Flooding）：向网络中的所有节点发送事件通知，在订阅者端执行适当的匹配，或者向所有可能的发布者发送订阅，然后在发布端执行匹配，并将匹配成功的事件通过点对点通信直接发送到相关的订阅者。

过滤（Filtering）：代理网络中采用过滤的方法，通过一个有路径到达有效订阅者的网络转发通知，代理在每个节点上存储相关状态，包括邻居列表、订阅列表和路由表。

广告（Advertisement）：向订阅者传播广告，减少订阅传播时的网络流量负担。

汇聚（Rendezvous）：将可能的事件集合看做一个事件空间，并将事件空间的责任划分到网络中的代理集合上，汇聚结点是负责一个给定的事件空间的子集的代理节点，控制订阅传播的方法。

17. 用组通信方式时，举例说明消息顺序的重要性，并说明解决方法说明。

在组通信中，消息顺序的重要性体现在接收方收到的消息顺序可能与发送方发送的顺序不一致，从而引起不一致的情况。为解决这个问题，可以采用全局时间顺序或一致时间顺序的方法，前者立即发送所有消息并保持发送顺序，后者通过先取其中一条消息作为第一个发送给所有组内成员，再取下一条消息发送，保证组内成员按照同一顺序收到消息。

答：要使组通信易于理解和使用，有两种性质是不可缺少的，首先是原子广播原语，它确保了一条消息 要么被所有组内成员收到， 要么没有一个成员能收到。其次是消息的顺序。例如： 有四台机器每台机器 有一个进程，进程 1 、2 、3 、4 属于同一个进程组，进程 0 与进程 4 同时想给该组发送一条消息，当两 个进程竞相访问 LAN 时，在网络中消息传送的顺序是无法确定的，可能是 0->1, 4->0,4->1,4->3,0->3,0->4。 这样进程 1 先收到 0 再收到 4，进程 3 先收到进程 4 在收到 0，则 1 与 3 之间可能会出现不一致。

解决方法：1) 全局时间顺序，保证立即发送所有消息并让他们保持发送顺序， 该方法能将消息精确 的按照发送顺序传递到目的地。 2)一致时间顺序，若有两条消息 A 和 B ，以很少的时间间隔发送， 系统先取其中一个作为第一个发送给所有组内成员， 然后再取下一个发送给组内成员， 这种方法保证 组内成员按照统一的顺序收到了消息，但是这个顺序可能并不是发送消息的顺序。

18. 在深度为 k 的分层定位服务中，当移动实体改变它的位置时，最多需要更新多少条位置记录？

答：改变位置可以看作是插入和删除操作的组合。插入操作要求至少 k+1 条记录变动， 同 样地，删除操作也要求改变 k+1 个记录，根的记录被这两个操作分享，导致 2k+1 条记录被 更新。

19. 分层定位服务中的根结点可能是一个潜在的瓶颈。如何能有效地避免这个问题？

答： 一项重要观察发现我们只使用随机位的字符串作为标识符，这样，我们很容易就划分标识符空间并且为每一个部分分配一个独立根结点。划分的根结点以及通路将遍布网络。

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。

22. 许多分布式算法需要使用协调进程。简单讨论一下，这样的算法实际上可以在什么程度上被看作为分布式的？

答： 在集中式的算法中，常常是固定的进程充当协调者。分布来源于其他进程在不同的机器

上运行的事实。在分布式算法中，没有固定的协调者，协调者从组成部分算法的进程中选出。事实是协调者能使算法更具分布性。

协调进程是一种多任务处理方式，它要求不同的进程或线程之间主动协作完成任务，而不是通过操作系统强制调度来实现并发执行。在协调进程中，每个进程或线程都可以控制自己的执行流程，当它需要等待其他进程或线程完成某些操作时，会主动放弃CPU控制权，让其他进程或线程运行。这种方式可以减少因为进程切换带来的开销，提高系统的性能。

协调进程需要进行显式的协作，因此需要编写更复杂的代码来处理进程之间的通信和同步。但是，协调进程可以更好地利用系统资源，避免了进程切换的开销，同时也可以更好地控制进程之间的通信和同步，避免了竞态条件等问题。因此，在某些场景下，协调进程可以比传统的并发模型更加高效和可靠。

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)多个进程动态地共享系统的资源可能会产生死锁现象。死锁的产生，必须同时满足四个条件，第一个是互斥条件， 即一个资源每次只能由一个进程占用； 第二个为等待条件，即一个进程请求资源不能满足时，它必须等待，但它仍继续保持已得到的所有其它资源；第三个是非出让条件，任何一个进程不能抢占另一个进程已经获得且未释放的资源； 第四个为循环等待条件， 系统中存在若干个循环等待的进程， 即其中每一个进程分别等待它前一个进程所持有的资源。防止死锁的机构只须确保上述四个条件之一不出现，则系统就不会发生死锁。

只要资源分配策略能保证进程不出现循环等待，则系统就不会发生死锁。

(2) 银行家算法分配资源的原则是：系统掌握每个进程对资源的最大需求量，当进程要求申请资源时， 系统就测试该进程尚需资源的最大量，如果系统中现存的资源数大于或等于该进程尚需的最大量时，则就满足进程的当前申请。这样可以保证至少有一个进程可能得到全部资源而执行到结束，然后归还它所占用的全部资源供其它进程使用。银行家算法破坏了产生死锁的第四个条件，即不可能产生循环等待，从而可以避免死锁的发生。

防止进程发生循环等待的另一种资源分配策略是按序分配算法，其基本思想如下：把系统中所有的资源排一个顺序，例如系统共有 m 个资源，用ri 表示第 i 个资源， 那么这 m 个资源是：

r1, r2, r3 ……, rm

规定任何进程不得在占用资源 ri (1<i≤m)后再申请 rj (j<i≤m)，或者说，如果里程需要资源 rj，那么它必须在申请 ri 之前申请(j<i)。可以证明，按这种策略分配资源时破坏了循环等待 条件，故能防止发生死锁

按序分配算法可以避免循环等待，因为它规定了进程请求资源的顺序，保证了每个进程在请求资源时必须按照一定的顺序进行，从而避免了循环等待的情况。

具体来说，在按序分配算法中，系统中所有的资源都被排列成一个序列，例如 r1, r2, r3, …, rm。规定进程只能按照这个序列的顺序请求资源，即一个进程只有在占用资源 ri 后才能请求资源 rj (j>i)。这样，如果进程 A 请求了资源 ri，那么只有当进程 B 已经释放了资源 ri 才能请求资源 rj (j>i)，从而避免了进程 A 和进程 B 形成循环等待的情况。(所有进程都按照这个顺序来申请资源，不会导致越位申请)

因此，按序分配算法可以保证每个进程在请求资源时必须按照一定的顺序进行，避免了循环等待的情况，从而避免了死锁的发生。

24. 在分布式系统中， 为什么需要代码迁移？ 代码迁移可以分为 **Sender-initiated** 和 **Receiver-initiated**，解释其中的含义，并举例说明。

在分布式系统中，代码迁移指将应用或服务的执行代码从一个节点移动到另一个节点的过程，可以帮助优化系统的资源利用率、提高系统性能和可扩展性。代码迁移分为 Sender-initiated 和 Receiver-initiated 两种模式，分别用于实现负载均衡、能源管理和故障恢复等场景。（负载均衡、故障恢复）

如果把进程由负载较重的机器上转移到负载较轻的机器上去，就可以提升系统的整体性能。 Sender-initiated 是在发送者启动的迁移，代码当前驻留在哪台机器上或者正在哪台机器上执行， 就由该机器来启动迁移。通过因特网向 Web 数据库服务器发送搜索程序以在该服务器上进行查询就是发送者启动迁移的一个例子。 Receiver-initiated 是在接收者启动的迁移，代码迁移的主动权掌握在目标机器手中。 Java 小程序就是接收者启动的一个例子。

25. 代码迁移的动机有哪些？

代码迁移将应用或服务的执行代码从一个节点移动到另一个节点，其动机主要有以下几点：**实现负载均衡、改善通信性能、提高可用性、使用特殊功能和提高灵活性**。通过将进程从负载较重的节点迁移到负载较轻的节点，可以实现负载均衡；通过将交互密集的进程迁移到同一节点，可以减少通信开销；通过将进程迁移到数据所在的节点，可以提高数据处理效率；通过将进程从故障节点迁移到正常节点，可以提高系统的可用性；通过在特定节点上执行进程，可以充分利用特定节点上独有的硬件或软件功能；通过将软件作为服务提供给客户，可以提高系统的灵活性。

代码迁移指的是将程序(或执行中的程序) 传递到其它计算机。 (基本思想： 把进程由负载较重的机器 上转移到负载较轻的机器上去，就可以提升系统的整体性能)

迁移动机：

(1) 实现负载均衡：将进程从负载重的系统迁移到负载轻的系统，从而改善整体性能。 (2)改善通 信性能：交互密集的进程可迁移到同一个节点执行以减少通信开销， 当进程要处理的数据量较大时， 最 好将进程迁移到数据所在的节点。

(3) 可用性： 需长期运行的进程可能因为当前运行机器要关闭而需要迁移。 (4) 使用特殊功能： 可 以充分利用特定节点上独有的硬件或软件功能。 (5) 灵活性： 客户首先获取必需的软件， 然后调用服 务器。

26. 简述三模冗余的基本思想，并举例说明三模冗余能否处理 **Byzabtine** 故障。

三模冗余是一种使用主动复制方法提供容错的技术，基本思想是利用物理冗余，将每个设备复制三次，每个副本由一个表决器控制，当两个或三个输入相同时，输出等于输入，否则输出不确定值。因此，当一个副本出现故障时，仍然有两个副本可以继续工作，保证了系统的容错能力。

然而，当处理机是 Byzantine 类型的，出错的处理机仍然可以工作并发出错误的随机应答，因此至少需要 2k+1 个处理机才能达到 k 级容错。在最坏情况下，如果 k 个失效的处理机恰好产生相同的应答，而剩下的 k+1 个未出错的处理机也产生相同的应答，那么客户机将无法得到正确结果。

因此，三模冗余可以处理一个部件出现 Byzantine 故障的情况，但如果一组中有两个或三个部件同时出现 Byzantine 故障，则无法处理。

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。

答： 主要思想是允许程序去调用位于其他机器上的过程。当位于机器 A 的一个进程调用机器 B 上的某个过程时， 机器 A 上的过程被挂起，被调用的过程在机器 B 上执行。调用者讲消息放在参数表中传送给被调用者，结果作为过程的返回值返回给调用者。消息的传送与 I/O 操作对于编程人员是不可见的。

主要步骤如下：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)服务器存根取出消息中的参数后

调用服务器过程； 6)服务器完成工作后将结果返回至服务器存根； 7)服务器存根将它们打包并激活内核陷阱； 8)远程内核将消息发送会客户内核； 9) 客户内核将消息提交给客户存根； 10)客户存根从消息中 取出结果返回给客户。

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。

服务器崩溃：

- 至少一次语义：等待服务器重新启动，然后重发请求，直到客户端收到应答消息，保证RPC至少执行一次。
- 至多一次语义：立即放弃并报告失效，确保RPC至多执行一次，但也可能根本没有执行。
- 不作保证。
- 精确一次语义。

客户端崩溃：

- 根除：在客户端发送RPC消息前先做日志，系统重新启动后，检查日志，发现孤儿存在并将其杀死。
- 再生：把时间分成有序的纪元，当客户端重启时，向所有机器广播一个消息通知一个新纪元的到来，并结束所有的远程计算。
- 温和再生：服务器接收到新纪元广播时，检查自己是否有远程计算，只有那些找不到所有者的远程计算终止。
- 过期：每个RPC都分配一个标准时间T来完成任务，如果超时没有完成则显示分配一个数额。

Raft算法的崩溃处理：
如果是leader节点崩溃，follower节点会在一段时间后超时并发起新的选举，选出新的leader。如果是follower节点崩溃，leader节点会不断重复向其发送消息，直到其恢复。在Raft中，只有leader节点可以向follower节点发送附加日志RPC请求，follower节点不能主动从集群中移除自己。如果follower节点恢复正常，它会接收到leader节点的最新日志条目并更新自己的日志。

如果RPC调用失败，Raft采用重试机制。如果是请求投票RPC或附加日志RPC，会根据具体情况重发RPC请求，直到收到响应；如果是心跳RPC，会立即重发RPC请求，以确保leader节点的心跳能够及时到达follower节点。

答： 如果是服务器崩溃了， 用户无法区分服务器是在执行前还是执行后崩溃， 解决方案如下： 1) 至少一次 语义， 指等待服务器重新启动， 然后重发请求。这种方法要求不断重试直至客户收到应答消息。它保证 RPC 至少执行一次。 2) 至多一次语义，指立即放弃并报告失效。它确保 RPC 至多执行一次，但也可能根本没 有执行； 3)不作保证； 4)精确一次语义；

如果是客户机崩溃了， 存在孤儿问题(客户已发送请求，在应答到来之前崩溃了， 此时已经激活服务器中的过程并获得结果，但是没有客户在等待结果)解决方案如下： 1)根除，在客户存根发送 RPC 消息前先做日志(用来恢复崩溃)，系统重新启动后， 检查日志，发现孤儿存在并将其杀死； 2) 再生，把时间分成 有序的纪元， 当客户端重启时， 向所有机器广播一个消息通知一个新纪元的到来， 并结束所有的远程计算； 3) 温和再生， 服务器接收到新纪元广播时， 检查自己是否有远程计算， 只有那些找不到所有者的远程计算 终止。 4)过期，每个 RPC 都分配一个标准时间T 来完成任务，如果超时没有完成则显示分配一个数额。

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。

解答：发生现象： 客户机在发送请求后在服务器应答消息到来之前崩溃，其已经激活了服务器的相应计算， 而客户没有等待它的结果，将遗留“计算孤儿”。

清除“孤儿”方法：

a) 根绝(extermination)法：客户存根发送 RPC 前在日志文件中记录将要执行的RPC， 若客户重启则依据日志作准确清除远程计算。

b) 再生(reincarnation) 法： 划分时间为序号纪元(时间戳)， 客户重起则广播新 纪元开始，所有远程计算被终止。

c) 温和再生(gentle reincarnation) 法：改进“再生”法，由服务器检查远程计算 有无调用者，若无则远程计算被终止。

d) 过期(expinration) 法：每个 rpc 执行前给定时间段 T，rpc 到期未完成的必须 再申请新的 T 。服务器将清除没有再申请新的 T 的 rpc。

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。

RPC 是建立在 Socket 之上的， 相对于 Socket ，RPC 实现了以下功能，

首先， 通讯方面， 主要是通过在客户端和服务器之间建立 TCP 连接， 远程过程调用的所有交换的数据都在这个连接里传输。连接可以是按需连接，调用结束后就断掉，也可以是长连接，多个远程过程调用共享同一个连接。

第二，寻址方面，比如 A 服务器上的应用怎么告诉底层的 RPC 框架，如何连接到 B 服务器(如主机或 IP 地址)以及特定的端口，方法的名称名称是什么，这样才能完成调用。比如基于 Web 服务协议栈的 RPC， 就要提供一个 endpoint URI，或者是从 UDDI 服务上查找。如果是 RMI 调用的话， 还需要一个 RMI Registry 来注册服务的地址。

第三，当 A 服务器上的应用发起远程过程调用时，方法的参数需要通过底层的网络协议如 TCP 传递到 B 服务器， 由于网络协议是基于二进制的， 内存中的参数的值要序列化成二进制的形式，也就是序列化(Serialize)或编组(marshal)，通过寻址和传输将序列化的二进制发送给 B 服务器。

第四， B 服务器收到请求后， 需要对参数进行反序列化(序列化的逆操作)，恢复为内存中的表达方式， 然 后找到对应的方法(寻址的一部分)进行本地调用，然后得到返回值。

第五，返回值还要发送回服务器 A 上的应用， 也要经过序列化的方式发送， 服务器 A 接到后，再反序列化， 恢复为内存中的表达方式，交给 A 服务器上的应用去处理

RPC 方法的基本原则是-- 以模块调用的简单性忽略了通讯的具体细节, 以便程序员不用关心 C/S 之间的通讯协议, 集中精力实现业务过程. 这就决定了 RPC 生成的通讯不可能对每种应用都有恰当的处理方法. 与 Socket 方法相比,传输相同的有效数据,RPC 占用更多的网络带宽和系统资源。

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

客户端与服务器端通过套接字通信。

在服务器端，创建远程服务对象，接收请求并执行并返回结果。其中Skeleton对象负责解码参数、调用实际远程对象实现的方法，并将结果返回给调用程序。

在客户端，Stub对象负责初始化连接、编码并发送参数、等待方法调用结果、解码返回值或返回的异常，并将值返回给调用程序。

34. 试举例说明没有统一时钟的分布式系统会发生什么问题？

解答： 当每台机器有它自己的时钟时， 一个发生于另一事件之后的事件可能会 被标记为一个比另一个事件更早的时间。例：
假设节点A在下午3点发送了一个请求到节点B，请求中包含了时间戳(3:00:00)，而节点B的本地时钟只显示为下午2点59分，那么节点B就可能会认为这个请求是在它自己的请求(时间戳为2:59:30)之后才到达的。这样就会导致节点B错误地将请求记录在日志文件中的时间戳为2:59:30，而不是实际的3:00:00，从而给排错带来很大的麻烦。

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；

36. 分布式系统中的体系结构样式有那几种？并简述之。

分布式系统中常见的**体系结构样式有四种：分层体系结构、基于对象的体系结构、以数据为中心的体系结构和基于事件的体系结构**。分层体系结构是控制系统从上到下，请求从上往下，结

果从下往上；基于对象的体系结构每个对象对应一个组件，易于扩展和修改；以数据为中心的体系结构通过共享数据结构进行进程通信，易于维护一致性；基于事件的体系结构进程之间通过事件进行通信，易于实现异步通信，进程之间耦合度低。

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。

Quorum 意思是仲裁人数，也就是最小合法人数。比如你是一个选民，选择下一任总统， 要求投票的人数必须大于 500，这个 500 就是 quorum。quorum 机制是一种 C 和 A 之间的权 衡机制。

考虑 N 个副本，我们设最少要写 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，即读写副本有重叠情况， 一般 W+R=N+1。例如 N=5，W=3，R=3，我们任意更新三 个副本，再最多读 3 个副本就一定能读到更新的数据。

Quorum 机制无法保证强一致性，所以是一种弱一致性算法。通常基于 Quorum 来选择 primary (主副本， 用于读写)，中心节点(服务器) 读取 R 个副本， 选择版本号最高的副本 作为新的 primary，但新选出 primary 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。

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。

令牌环的故障处理功能主要体现在对令牌和数据帧的维护上。令牌本身就是比特串， 绕环传 递过程中也可能受干扰而出错， 以至造成环路上无令牌循环的差错； 另外， 当某站点发送数据帧 后，由于故障而无法将所发的数据帧从网上撤消时，又会造成网上数据帧持续循环的差错。令牌 丢失和数据帧无法撤消，是环网上最严重的两种差错， 可以通过在环路上指定一个站点作为主动 令牌管理站，以此来解决这些问题。

主动令牌管理站通过一种超过机制来检测令牌丢失的情况， 该超时值比最长的帧为完全遍历 环路所需的时间还要长一些。如果在该时段内没有检测到令牌， 便认为令牌已经丢失， 管理站将 清除环路上的数据碎片，并发出一个令牌。

为了检测到一个持续循环的数据帧， 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经过的数据帧的监控拉的已经置为 1，便知道有某个站未能清除自己发出的 数据帧， 管理站将清除环路的残余数据，并发出一个令牌。

39. 简述分布式系统设计所面临的问题及遇到的挑战。

分布式系统设计面临的问题和挑战包括难以合理设计分配策略、部分失效问题、性能和可靠性过分依赖于网络、缺乏统一控制、以及安全保密性问题。这些问题和挑战需要在设计分布式系统时考虑和解决，以确保系统的性能、可靠性和安全性。

1)：难以合理设计分配策略,在集中式系统中，所有的资源都由系统系统管理和分配，但在分布式系统中，资源属于局部工作站或个人计算机，所以调度的灵活性不如集中式系统，资源的物理分布可能与服务器的分布不匹配，某些资源可能空闲，而另外一些资源可能超载。

2)：部分失效问题：由于分布式系统通常是由若干部分组成的，各个部分由于各种各样的原因可能发生故障，如硬件故障。如果一个分布式系统不对这些故障对这些问题进行有效的处理，系统某个组成部分的故障可能导致整个系统的瘫痪。 3)性能和可靠性过分依赖于网络：由于分布式系统是建立在网络之上的，而网络本身是不 可靠的，可能经常发生故障， 网络故障可能导致整个系统的终止； 另外，网络超负荷会导致 性能下降， 增加系统的响应时间。

4)缺乏统一控制： 一个分布式系统的控制通常是一个典型的分散式控制，没有统一的中心控制。因此，分布式系统通常需要相应的同步机制来协调系统中各个部分的工作； 5)安全保密性问题：为了获得可扩展性，分布式系统中的许多软件接口都提供给用户，这 样的开放结构对于开发人员非常有价值， 担同时也为破坏者打开了方便之门

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。

①资源的多重性带来的问题。由于处理资源的多重性，分布计算系统可能产生的差错类 型和次数都比集中式单机系统多。最明显的一个例子是部分失效问题：系统中某一个处理资 源出现故障而其他计算机尚不知道，但单机系统任何一部分出现故障时将停止整个计算。另 一个例子是多副本信息一致性问题。可见，资源多重性使得差错处理和恢复问题变得很复杂。 资源多重性还给系统资源管理带来新的困难。

②资源的分散性带来的问题。在分布计算系统中，系统资源在地理上是分散的。由于进 程之间的通信采用的是报文传递的方式进行的，通信将产生不可预测的、有时是巨大的延迟， 特别是在远程网络所组成的分布计算系统中更是这样。例如使用卫星通信会产生 270 毫秒的 延迟。在分布计算系统中，系统的状态信息分布在各个分散的节点上。分布式的状态信息和 不可预知的报文延迟使得系统的控制和同步问题变得很复杂，要想及时地、完整地搜集到系 统各方面的信息是很困难的，从而使处理机进行最佳调度相当困难。

③系统的异构性带来的问题。在异构性分布计算系统中，由于各种不同资源(特别是计 算机和网络) 的数据表示和编码、控制方式等均不相同， 这样一来就产生了翻译、命名、保 护和共享等新问题。

由于上述原因， 分布计算系统的研制，特别是软件的验证、调试、测量和维护问题变得很复杂。这些正是分布计算系统研制者要解决的主要问题。

41. 挑战：设计与实现一个对用户来说是透明的且具有容错能力的分布式系统。

设计和实现一个对用户透明的、具有容错能力的分布式系统需要考虑以下几个方面：异构性、开放性、安全性、可扩展性、错误处理、并发一致性和透明度。为了解决这些挑战，需要采用相应的机制和技术，如中间件、虚拟机、安全机制、分散算法、错误处理机制和并发控制机制等。

挑战：1. Heterogeneity 异构性，包括网络、硬件、操作系统、编程语言、Implementation 组件， <中间件的作用就是隐藏这些异构，并提供一致的计算模式(模块) >，<虚拟机，编 译器成虚拟机使用的 code>；2. Openness 开放性：提供 Services, Syntax, Semantics， 合适的接口定义需要兼顾完整性和中立性，同时互操作性和便携性也是重要的，分布式系统 需要 flexible，即易于配置且把策略和结构分开来获得 flexible；3. Security 安全性： 安全性包括有效性， 机密性， 完整性三个要素，拒绝服务攻击和移动代码安全是两个安全方 面的挑战。4. Scalability 可扩展性(可测性)，即在资源和用户增加的时候保持效率， size、geographically、administratively 的 scaleble。要求分散(分布式) 算法， 没有 机器有系统状态的完整信息，机器做决定仅仅是取决于本地信息，一个机器的错误不会毁掉 整个算法， 没有统一时序。5. Failure Handling 错误处理。6. Concurrency 协力，
一致(并发) 7. Transparency 透明度

42. 说明客户/服务器模式的主要思想，并说明在采用了阻塞的、有缓存的、可靠的发送和接收原语的情况下， 系统是如何工作的。

客户/服务器模式的主要思想是构建由一组协同进程组成的操作系统，其中服务器进程提供服务，客户进程请求服务，客户和服务器运行在相同的微内核中。在采用了阻塞的、有缓存的、可靠的发送和接收原语的情况下，进程调用send原语指定目的地和发送缓冲区数据，发送进程被阻塞直到消息传送完毕，接收进程建立邮箱并指定地址，收到具有该地址的消息放入邮箱中，调用receive时从邮箱中取出一条消息，当邮箱为空时阻塞。可靠的原语有三种方式：重新定义非可靠原语、接收机器的内核给发送机器的内核发送确认消息、服务器的内核不发送确认消息，而是将应答作为确认消息。这些技术和机制使得客户/服务器模式能够提供高效、可靠、安全的分布式服务。

主要思想： 构造一种操作系统， 它由一组协同进程组成， 这组进程称作服务器。为用户提供 服务的进程称作客户，客户和服务器都运行在相同的微内核中。 进程调用 send 原语， 它指 定了目的地以及发送到该目的地的缓冲区数据。消息被发送时，发送的进程被阻塞。直到消 息传送完毕，其后的指令才能继续执行。之后对接收信息感兴趣的进程可以让内核为之建立 一个邮箱，并指定一个地址以便于寻找网络信包。所有具有该地址的输入消息被放入邮箱中， 调用 receive 时只要从邮箱中取出一条消息，邮箱为空时阻塞。可靠的原语有三种： ①重新 定义非可靠原语。②要求接收机器的内核给发送机器的内核发送一个确认消息。只有当收到 这个确认消息后，发送内核释放用户进程。③客户机在发送消息阻塞后，服务器的内核不发 送确认消息，而是将应答作为确认消息。

43. 对于接收消息 **Receive** 原语，为什么需要缓存**,** 缓存的作用是什么**?**

答：如果不适用缓存， 服务器接收来的消息会被丢弃或者存在诸如服务器需要存储和管理早到来的消息 这样的问题。缓存的作用就是用来统一管理消息的： 它定义了一种叫邮箱的数据结构， 接收客户端请求 的进程通知内核创建邮箱存储消息， 并且指定了访问地址。当 Receive 原语调用是， 系统内核就会提取 消息并知道如何处理它。

44. 一个最完备的分布式体系由以下模块组成。请说明各模块的功能？

分布式处理系统：能够在短时间内动态地组合成面向不同服务对象的系统，对用户是透明的。

分布式查询模块：可以访问来自多种异类数据源的数据，这些数据可存储在相同或不同的计算机上。

分布式数据库模块：由分布于多个计算机结点上的若干个数据库系统组成，提供有效的存取手段来操纵这些结点上的子数据库。

分布式缓存模块：支持**重复、分配和分层**三种基本配置，用于提高缓存数据的可用性和实现高可伸缩性。

分布式文件系统模块：具有执行远程文件存取的能力，并以透明方式对分布在网络上的文件进行管理和存取。

分布式网络通信模块：能够连接跨越了多台计算机的应用程序各节点。

分布式监控管理模块：可以有效避免最上层服务器因顾及不暇而出现管理疏漏的现象。

分布式程序设计模块：用于编写运行于分布式计算机系统上的分布式程序。

分布式算法模块：起到分布性和并发性的作用，与集中式算法不同，能够应对分布式系统的非稳定性因素。

①分布式处理系统必须有能力在短时间内动态地组合成面向不同服务对象的系统。对用户来说系统是透明的， 用户只需指定系统干什么而不必指出哪个部件可以提供这一服务。系统各组成部分是自主的，但不是无政府状态， 而是遵循某个主计划由高级操作系统进行协调工作。在一个计算机网中有多台主机不一定都是分布式处理。如果这样的系统不具备动态组合及任务再指派的能力，那么它们仍然是集中式处理。

②分布式查询可以访问来自多种异类数据源的数据，而这些数据可存储在相同或不同的计算机上。

③分布式数据库系统由分布于多个计算机结点上的若干个数据库系统组成,它提供有效的存取手段来操纵这些结点上的子数据库。分布式数据库在使用上可视为一个完整的数据库,而实际上它是分布在地理分散的各个结点上。当然,分布在各个结点上的子数据库在逻辑上是相关的。

④分布式缓存支持一些基本配置： 重复(replicated)、分配(partitioned) 和分层(tiered)。 重复(Replication)用于提高缓存数据的可用性。在这种情况下，数据将重复缓存在分布式系 统的多

台成员机器上， 这样只要有一个成员发生故障， 其他成员便可以继续处理该数据的提供。 另一方面，分配(Partitioning) 是一种用于实现高可伸缩性的技巧。通过将数据分配存放在许 多机器上，内存缓存的大小加随着机器的增加而呈线性增长。结合分配和重复这两种机制创建出 的缓存可同时具备大容量和高可伸缩的特性。

⑤分布式文件系统具有执行远程文件存取的能力,并以透明方式对分布在网络上的文件进行管理和存取。

⑥ 分布式网络通信能够连接跨越了多台计算机的应用程序各节点。

⑦分布式监控管理可以有效避免最上层服务器因顾及不暇而出现管理疏漏的现象。

⑧用于编写运行于分布式计算机系统上的分布式程序。一个分布式程序由若干个可以独立执行的程序模块组成,它们分布于一个分布式处理系统的多台计算机上被同时执行。它与集中式的程序设计语言相比有三个特点： 分布性、通信性和稳健性。

⑨分布式系统的执行存在着许多非稳定性的因素。分布式算法起到分布性和并发性的作用，这一点不同于集中式算法。

45. 集群“脑裂”怎么产生， 如何解决。

集群“脑裂”通常是因为在网络通信出现故障时，选举机制可能**在不同的网络分区中选出两个Leader，导致数据不一致的现象**。解决脑裂的方式包括：使用仲裁(quorum)机制、增加心跳线、启用磁盘锁/IO Fence等方法，但不能完全保证避免脑裂的发生，需要人工干预

- 仲裁(quorum)机制：通过在集群中设置一个仲裁节点，来保证只有一个Leader被选举出来，从而避免了多个Leader同时对系统进行操作的情况。
- 增加心跳线：通过增加网络线路或使用多根以太网线来进行通信，可以提高集群的可用性，减少网络通信故障的发生。
- 启用磁盘锁/IO Fence：通过在共享磁盘上使用锁定机制，来防止多个Leader同时对同一份数据进行操作，从而避免了数据不一致的情况。
- 使用分布式锁：通过使用分布式锁来保证在任何时候只有一个Leader可以访问共享资源，从而避免多个Leader同时对系统进行操作的情况。

46. 为什么分布式系统需要用到 **ID** 生成系统

在复杂分布式系统中， 往往需要对大量的数据和消息进行唯一标识。如在美团点评的金融、支付、餐饮、 酒店、猫眼电影等产品的系统中，数据日渐增长，对数据库的分库分表后需要有一个唯一 ID 来标识一 条数据或消息，数据库的自增 ID 显然不能满足需求；特别一点的如订单、骑手、优惠券也都需要有唯 一 ID 做标识。此时一个能够生成全局唯一 ID 的系统是非常必要的。

概括下来，业务系统对 ID 号的要求有哪些呢？

ID 生成系统的需求

1.全局唯一性：不能出现重复的 ID ，最基本的要求。

2.趋势递增： MySQL InnoDB 引擎使用的是聚集索引， 由于多数 RDBMS 使用 B-tree 的数据结构来存储 索引数据，在主键的选择上面我们应尽量使用有序的主键保证写入性能。

3.单调递增：保证下一个 ID 一定大于上一个 ID。

4.信息安全：如果 ID 是连续递增的，恶意用户就可以很容易的窥见订单号的规则，从而猜出下一个订 单号，如果是竞争对手，就可以直接知道我们一天的订单量。所以在某些场景下，需要 ID 无规则。 第 3 、4 两个需求是互斥的，无法同时满足。

同时，在大型分布式网站架构中，除了需要满足 ID 生成自身的需求外，还需要 ID 生成系统可用性极 高。

47. 应用哪些技术可以使得一个分布式系统具有可伸缩性？

可伸缩性是指系统在面对不断增长的用户和数据量时，能够保持良好的性能和可用性，而不需要进行过多的修改或重构。一个具有良好可伸缩性的系统能够在不增加额外的资源或成本的情况下，适应不断增长的负载和数据。这包括水平扩展和垂直扩展两个方面：

水平扩展：通过增加更多的服务器节点、应用程序实例等方式来水平扩展系统的能力，从而实现更高的并发性和吞吐量。

垂直扩展：通过增加服务器的硬件资源，如CPU、内存等来垂直扩展系统的能力，从而提高单个服务器的性能和处理能力。

实现分布式系统的可伸缩性主要有三种技术：

- 减少通信延迟，使用异步通信方式，处理其他本地任务。
- 分层，将组件分解为几个小层，均衡了系统负载。
- 复制冗余，使得资源更容易就近获取，并且均衡了负载。

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？

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；

2) P1 申请 P2 占用的资源， 使用 wound-wait 方法；

49. 请介绍一下分布式两阶段提交协议？

分布式两阶段提交协议是一种用于保证分布式事务的原子性、一致性和持久性的协议，它分为两个阶段。在阶段一，所有事务参与者向协调者发送预备提交请求，并等待协调者的回复；在阶段二，如果协调者发送了提交请求，则所有参与者执行正式提交操作，并释放占用的资源；如果协调者发送了回滚请求，则所有参与者放弃执行操作，并释放占用的资源。在分布式两阶段提交协议中，主事务管理器充当协调者的角色，负责整个事务并与网络中的其他事务管理器协同工作。为了实现分布式事务，必须使用一种协议在分布式事务的各个参与者之间传递事务上下文信息。所有资源的更新操作都会被写入日志，并且所有参与者都需要遵循协议的规定，以保证事务的正确性和一致性。

确保事务的ACID属性（原子性、一致性、隔离性和持久性）不受损害

阶段一：开始向事务涉及到的全部资源发送提交前信息。此时，事务涉及到的资源还有最后一 次机会来异常结束事务。如果任意一个资源决定异常结束事务， 则整个事务取消， 不会进行资源 的更新。否则，事务将正常执行， 除非发生灾难性的失败。为了防止会发生灾难性的失败，所有 资源的更新都会写入到日志中。这些日志是永久性的，因此， 这些日志会幸免遇难并且在失败之 后可以重新对所有资源进行更新。 * 阶段二： 只在阶段一没有异常结束的时候才会发生。此时， 所有能被定位和单独控制的资源管理器都将开始执行真正的数据更新。 在分布式事务两阶段提 交协议中，有一个主事务管理器负责充当分布式事务协调器的角色。事务协调器负责整个事务并 使之与网络中的其他事务管理器协同工作。 为了实现分布式事务，必须使用一种协议在分布式 事务的各个参与者之间传递事务上下文信息，IIOP 便是这种协议。这就要求不同开发商开发的 事务参与者必须支持一种标准协议， 才能实现分布式的事务。

50. 描述可能发生在因特网上的几类主要的安全威胁**(**对进程的威胁、对信道的威胁、拒绝 服务**)**可能的发生情况。

因特网上可能发生的主要安全威胁包括对进程的攻击（如欺骗服务器）、对通信信道的攻击（如IP欺骗和中间人攻击）以及拒绝服务攻击（如洪水攻击）。攻击者可能会利用这些漏洞来窃取敏感信息、篡改数据或使目标系统不可用。

Threats to processes: without authentication of principals and servers, many threats exist. An enemy could

access other user's files or mailboxes, or set up 'spoof' servers. E.g. a server could be set up to 'spoof' a bank's service and receive details of user's financial transactions.

Threats to communication channels: IP spoofing - sending requests to servers with a false source address, man-in-the-middle attacks.

Denial of service: flooding a publicly-available service with irrelevant messages.

51. Describe a strong cache-consistency mechanism for a distributed file system with a stateless server. Describe a weak cache-consistency mechanism for the same system.

一个无状态服务器的分布式文件系统需要依靠客户端来维护缓存一致性。强一致性机制会在缓存命中时与服务器通信，验证缓存副本是否为最新的数据。弱一致性机制则是定期或基于缓存对象的最后修改时间，检查服务器上的数据是否有更新。强一致性机制能够保证数据始终为最新，但性能较差；而弱一致性机制则能提高性能，但有可能获取到旧数据。具体应该选择哪种机制取决于系统的具体需求。

强一致性是指在分布式系统中，无论在哪个节点上访问数据，都能够获得相同的结果。在强一致性下，所有节点的副本都是同步的，当一个节点更新数据时，其他节点会立即同步更新。这样可以保证数据的一致性，但有时会牺牲一些性能。

弱一致性是指在分布式系统中，不同节点间访问同一数据的结果可能会有所不同，但是这种不同是可以接受的。在弱一致性下，数据的更新可能会存在一定的延迟，因此在不同节点上访问数据时，可能会得到不同的结果。但是由于数据的更新是异步的，所以弱一致性可以提高性能。

BASE理论中的"B"指的是基本可用性（Basically Available）BASE理论是基于弱一致性的分布式系统设计理论。其中的"A"指异步通信，"S"指数据复制的可用性，"E"指事件ual consistency（最终一致性）。

Answer: Since the server is stateless, the client must handle cache consistency. A strong mechanism has the cache, upon a cache hit, communicate with the server to verify the cache copy is up to date, before using the cached data. A weak mechanism has the cache, upon a cache hit, only sometimes communicate with the server to verify the cache copy is up to date. This can happen periodically (e.g., every X seconds) or heuristically based on the last modified times of the cached object.

53. 怎么理解分布式一致性？

分布式一致性是指在分布式系统中，多个节点之间协同工作，保证数据在节点之间的一致性。数据一致性是指在系统中的多个副本中，数据的内容是否一致。在分布式系统中，一致性通常分为强一致性、弱一致性和最终一致性三种。强一致性是指任何时候访问到的数据都是最新的数据；弱一致性是指在一定时间内达成一致；最终一致性是指在经过一段时间后，最终会达到一致状态。最终一致性相对于强一致性和弱一致性来说，实现难度较小，同时对系统的性能影响较小，因此在实际的分布式系统中被广泛采用。

Raft属于强一致性(Strong Consistency)类型的分布式一致性算法。Raft算法在保证系统数据强一致性方面，采用了 Leader-Follower 结构，将集群中的节点分为三种角色：Leader、Follower 和 Candidate。Leader 负责接收客户端请求并进行日志复制，Follower 负责接收 Leader 的复制请求并进行数据同步，Candidate 则用于选举出新的 Leader。通过这种方式，Raft算法保证了分布式系统中数据的强一致性，同时具备选举速度快、容错性强等优点。

分布式一致性是指在分布式系统中，多个节点之间协同工作，保证数据在节点之间的一致性。由于分布式系统中存在网络延迟、节点故障等问题，因此保证分布式一致性是非常困难的。

数据一致性是指在系统中的多个副本中，数据的内容是否一致。在分布式系统中，数据的复制会导致数据一致性问题，因此需要采取一些措施来保证分布式一致性。在分布式系统中，一致性通常分为强一致性、弱一致性和最终一致性三种。

强一致性是指在分布式系统中，不论在任何时候、任何情况下，不同节点之间的数据都是强一致的，即任何时候访问到的数据都是最新的数据。实现强一致性需要付出很高的代价，因为需要保证分布式系统中所有节点之间的通信都是同步的。

弱一致性是指在分布式系统中，不同节点之间的数据在某些时候可能是不一致的，但是系统需要在一定时间内达成一致。弱一致性相对于强一致性来说，实现难度较小，但是需要付出的代价也较高。

最终一致性是指在分布式系统中，数据在经过一段时间后，最终会达到一致状态。最终一致性通常是通过异步复制的方式来实现的，即在不同节点之间的数据复制是异步的，但是在一定时间后，系统会自动调节数据达到一致状态。最终一致性相对于弱一致性来说，实现难度较小，同时对系统的性能影响较小，因此在实际的分布式系统中被广泛采用。

54. 讨论在 **TCP/IP** 连接之上实现请求一应答协议时可以获得的调用语义， 该调用语义要确 保数据按发送顺序到达， 既不丢失也不重复。考虑导致连接中断的所有条件。

55. 一个页面从输入 **URL** 到页面加载显示完成， 这个过程中都发生了什么？

56. 为什么要用 **ajax**

57. 简述 **ajax** 的过程

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。木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。

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；

分类	方案	实现原理	优点	缺点

基于数据库	基 于 mysql 表唯一索引	1.表增加唯一索引 2.加锁：执行 insert 语句，若报 错，则表明加锁失败 3.解锁：执行 delete 语句	完全利用 DB 现有 能力，实现简单	1.锁无超时自动失效机制， 有死锁风险 2.不支持锁重入，不支持阻 塞等待 3.操作数据库开销大，性能 不高
基于分布式 协调系统	基 于 ZooKeeper	1.加锁： 在/lock 目录下创建 临时有序节点， 判断创建的节点 序号是否最小。若是， 则表示获 取到锁； 否，则 watch /lock 目录 下序号比自身小的前一个节点 2.解锁：删除节点	1.由 zk 保障系统高 可用 2.Curator 框架已原 生支持系列分布式 锁命令，使用简单	需单独维护一套 zk 集群，维 保成本高
基 于 redis 命令 基于缓存		1. 加锁：执行 setnx，若成功再 执行 expire 添加过期时间 2. 解锁：执行 delete 命令	实现简单，相比数 据库和分布式系统 的实现，该方案最 轻，性能最好	1.setnx 和 expire 分 2 步执 行，非原子操作；若 setnx 执行成功， 但 expire 执行失 败，就可能出现死锁 2.delete 命令存在误删除非 当前线程持有的锁的可能 3.不支持阻塞等待、不可重入
基 于 redis Lua 脚本能 力		1. 加锁：执行 SET lock_name random_value EX seconds NX 命令 2. 解锁：执行 Lua 脚本	同上；实现逻辑上 也更严谨，除了单 点问题，生产环境 采用用这种方案， 问题也不大。	不支持锁重入， 不支持阻塞等待

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”。

61. 在 **DHT-based** 系统中可以采用 **finger table** 的方法来相应的提高查询效率， 试着说明其工作原理以及在实际中的应用案例

62. 举例简述负载均衡策略

负载均衡策略有基于 DNS 的负载均衡和反向代理负载均衡。基于 DNS 的负载均衡是将同一个主机名配置多个 IP 地址，DNS 服务器按顺序返回不同的解析结果，将客户端的访问引导到不同的节点上去。反向代理负载均衡将来自 Internet 上的连接请求以反向代理的方式动态地转发给内部网络上的多个节点处理，结合高速缓存技术提高系统性能，具备额外的安全性。反向代理负载均衡的缺点在于应用范围受限和代理服务器成为服务瓶颈。

基于 DNS 的负载均衡。基于 DNS 的负载均衡是在 DNS 服务器中为同一个主机名配置多个 IP 地址， 在应答 DNS 查询时， DNS 服务器对每个查询将以 DNS 文件中主机记录的 IP 地址按顺序返回不同的解析 结果， 将客户端的访问引导到不同的节点上去， 使得不同的客户端访问不同的节点， 从而达到负载均衡的 目的。

DNS 负载均衡的优点是经济、简单易行，并且节点可以位于 Internet 上任意的位置。但它也存在不少 缺点，例如， 为了保证 DNS 数据及时更新， 一般都要将 DNS 的刷新时间设置得较小， 但太小就会造成太 大的额外网络流量， 并且更改了 DNS 数据之后也不能立即生效； DNS 负载均衡采用的是简单的轮转算法， 不能 区分节点之间的差异， 不能反映节点的当前运行状态， 不能做到为性能较好的节点多分配请求， 甚至 会出现客户请求集中在某一个节点上的情况。另外， 要给每个节 点分配一个 Internet 上的IP 地址， 这势必 会占用过多的 IP 地址。

反向代理负载均衡。反向代理负载均衡是将来自 Internet 上的连接请求以反向代理的方式动态地转发给 内部网络上的多个节点进行处理， 从而达到负载均衡 的目的。反向代理负载均衡既能以软件方式实现， 也 能在高速缓存器和负载均衡器等硬件设备上实现。反向代理负载均衡可以将优化的负载均衡策略和代理服 务器的 高速缓存技术结合在一起， 提升静态网页的访问速度， 提高系统性能。另外， 由于网络外部用户不 能直接访问真实的节点计算机，反向代理负载均衡还具备额外的安 全性(同理，基于 NAT 的负载均衡也 有此优点)。

反向代理负载均衡的缺点主要表现在两个方面。首先，反向代理处于 OSI 参考模型应用层，因此，必 须为每种应用服务专门开发一个反向代理服务器， 这样， 就限 制了反向代理负载均衡技术的应用范围， 现 在一般都用于对 Web 服务器的负载均衡； 其次， 针对每一次代理， 代理服务器都必须打开两个连接， 一个 对外，一个对 内。在并发连接请求数量非常大的时候， 代理服务器的负载也就非常大， 代理服务器本身会 成为服务的瓶颈。

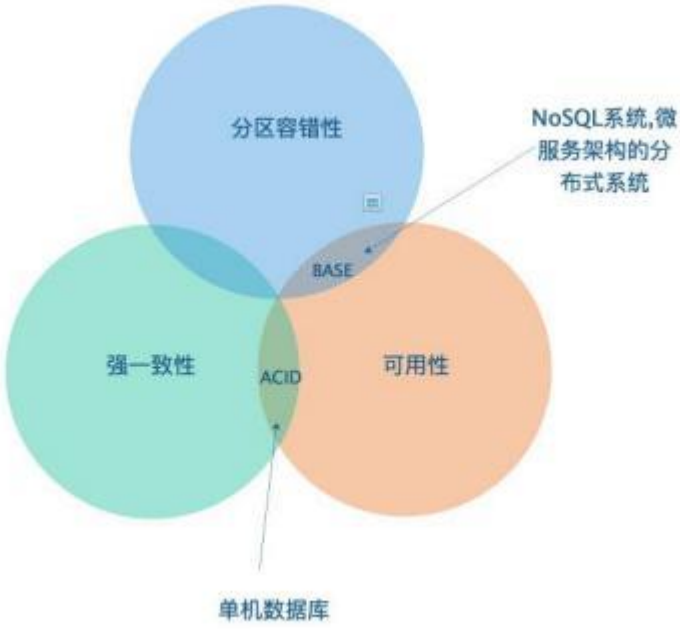
63. CAP,BASE 以及 ACID 的关系（重点！！）

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。



64. Redis 中缓存穿透、缓存击穿、缓存雪崩解决方案？

缓存穿透:指查询一个一定不存在的数据， 如果从存储层查不到数据则不写 入缓存， 这将导致这个不 存在的数据每次请求都要到 DB 去查询，可能导 致 DB 挂掉。

解决方案：

1. 查询返回的数据为空，仍把这个空结果进行缓存，但过期时间会比较短;

2. 布隆过滤器:将所有可能存在的数据哈希到一个足够大的 bitmap 中，一 个一定不存在的数据会被这 个 bitmap 拦截掉，从而避免了对 DB 的查 询。

缓存击穿:对于设置了过期时间的 key,缓存在某个时间点过期的时候，恰 好这时间点对这个 Key 有大 量的并发请求过来， 这些请求发现缓存过期一 般都会从后端 DB 加载数据并回设到缓存， 这个时候大并发 的请求可能会 瞬间把 DB 压垮。

解决方案：

1. 使用互斥锁:当缓存失效时， 不立即去 I oad db,先使用如 Redis 的 setnx 去 设置一个互斥锁，当操作 成功返回时再进行 Load db 的操作并回设缓存，否 则重试 get 缓存的方法。

2. 永远不过期:物理不过期，但逻辑过期(后台异步线程去刷新)。

缓存雪崩： 设置缓存时采用了相同的过期时间， 导致缓存在某一时刻同时失效， 请求 全部转发到 DB, DB 瞬时压力过重雪崩。与缓存击穿的区别:雪崩是很多 key,击穿是某一个 key 缓存。

解决方案:将缓存失效时间分散开， 比如可以在原有的失效时间基础上增加 一个随机值， 比如 1-5 分钟 随机，这样每一个缓存的过期时间的重复率就 会降低，就很难引发集体失效的事件。

65. 请简述数据库反规范设计方法

数据库中的**数据规范化的优点**是减少了数据冗余， 节约了存储空间， 相应逻辑和物理的 I/O 次数减少，

同时加快了增、删、改的速度，但是对**完全规范的数据库查询，通常需要更多的连接操作，从而影响查询速度**。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(非规范化处理)。

常见的反规范化技术包括：

(1)增加冗余列

增加冗余列是指在多个表中具有相同的列，它常用来在查询时避免连接操作。例如：以规范化设计的 理念，学生成绩表中不需要字段“姓名”，因为“姓名”字段可以通过学号查询到， 但在反规范化设计中， 会将“姓名”字段加入表中。这样查询一个学生的成绩时， 不需要与学生表进行连接操作，便可得到对应 的“姓名”。

(2)增加派生列

增加派生列指增加的列可以通过表中其他数据计算生成。它的作用是在查询时减少计算量，从而加快 查询速度。例如：订单表中， 有商品号、商品单价、采购数量，我们需要订单总价时，可以通过计算得到 总价，所以规范化设计的理念是无须在订单表中设计“订单总价”字段。但反规范化则不这样考虑，由于 订单总价在每次查询都需要计算，这样会占用系统大量资源，所以在此表中增加派生列“订单总价”以提 高查询效率。

(3)重新组表

重新组表指如果许多用户需要查看两个表连接出来的结果数据，则把这两个表重新组成一个表来减少 连接而提高性能。

(4)分割表

有时对表做分割可以提高性能。表分割有两种方式。

水平分割： 根据一列或多列数据的值把数据行放到两个独立的表中。水平分割通常在下面的情况下使 用。

情况 1：**表很大**， 分割后可以降低在查询时需要读的数据和索引的页数，同时也降低了索引的层数， 提高查询效率。

情况 2：表中的**数据本来就有独立性**，例如表中分别记录各个地区的数据或不同时期的数据，特别是 有些数据常用，而另外一些数据不常用。

情况 3：**需要把数据存放到多个介质**上。

(5) 垂直分割： 把主码和一些列放到一个表， 然后把主码和另外的列放到另一个表中。如果一**个表中某些列常用， 而另外一些列不常用，则可以采用垂直分割**， 另外垂直分割可以使得数据行变小，一个数据 页就能存放更多的数据， 在查询时就会减少 I/O 次数。其缺点是需要管理冗余列， 查询所有数据需要连接 操作。

66. 反规范设计的数据完整性维护方法。

无论使用何种反规范技术，需要额外的工作来维护数据的完整性，一般可以通过以下几 种方式进行：

应用逻辑

在应用程序的事务中对同一同一事务中对所有涉及的表进行增、删、改操作进行维护。 这种方式比较难于管理， 一个维护逻辑很容易出现在多个应用程序当中，容易遗漏，特别是 在需求变化时， 不易于维护。。

批处理维护

由批处理程序批量的处理所有的非规范化关系涉及的数据。一般定期运行一批处理作业 或存储过程，运行间隔根据业务来决定， 并且可以利用 Job 来自动运行批处理程序。可用 于对冗余数据的实时性要求不高或者有一定规则的环境。

触发器

在数据库端建立触发器，对原数据的修改会立即触发对冗余列的修改。触发器是实时的， 而且相应的处理逻辑只在一个地方出现，易于维护。可用于对数据实时性要求较高的环境，

但同时会降低数据的插入和更新速度。

67. 简述虚拟化的实现层次。

虚拟化技术通过在同一个硬件主机上多路复用虚拟机的方式来共享昂贵的硬件资源，虚拟化的基本思 想是分离软硬件以产生更好的系统性能

引入虚拟化后，不同用户应用程序由自身的操作系统(即客户操作系统)管理， 并且那些客户操作系 统可以独立于主机操作系统同时运行在同一个硬件上，这通常是通过新添加一个称为虚拟化层的软件来完 成，该虚拟化层称为 hypervisor 或虚拟机监视器(Virtual Machine Monitor ， VMM)

- 指令集体系结构级：代码解释和动态二进制翻译
- 硬件抽象级：虚拟化一个计算机硬件资源
- 操作系统级：在单一物理服务器上创建隔离的容器和操作系统实例
- 库支持级：库接口的虚拟化
- 应用程序级：进程级虚拟化、高级语言(High Level Language ， HLL)虚拟机

68. IaaS提供了哪些虚拟化？

IaaS提供虚拟化的硬件资源，如服务器、存储和网络。用户需要自己管理在这些硬件资源之上的软件，比如操作系统和开发环境等。Amazon EC2是一个提供IaaS服务的例子，它提供一个虚拟的PC供用户安装和维护自己的操作系统实例。还有其他可能的例子。

69. 虚拟机的两种架构的本质差别表现在那些方面？

虚拟机有两种架构： 寄居架构(Hosted Architecture) 和裸金属架构 (“Bare Metal” Architecture)。所谓寄 居架构就是在已安装的操作系统之上安装和运行虚拟化程序 VMM，然后再利用 VMM 创建和管理虚拟 机，它依赖于主机操作系统对设备的支持和物理资源的管理； 而裸金属架构就是直接在硬件上面安装虚 拟化软件，即将 VMM 安装在物理服务器上， 再在其上安装操作系统和应用， 依赖虚拟层内核和服务器 控制台进行管理。

普遍认为， 裸金属架构比寄居架构高， 因为裸金属架构是直接运行在硬件上的。一般寄生型适用于桌面 系统，裸金属适用于服务器系统。

70. 简述什么是 **SOAP**，以及 **SOAP** 与 **XML** 之间的关系。

SOAP (Simple Object Access Protocol)即简单对象访问协议，是一个简单的用在 Web 上交换结构信息 的 XML 协议， 没有定义应用语义和传输语义， 它以一种基于 XML 且与平台无关的 Web 编程方式改进 了 Internet 的互操作性。 SOAP 消息传递协议使用HTTP 承载消息，而使用XML 格式化消息。

SOAP 包括四个部分：

1，信封； 2，数据的编码规则；3 ，RPC 调用规范； 4 ，SOAP 绑定。 SOAP 与XML 之间的关系是：

1 ，SOAP 是在 XML 基础上定义的，所有的 SOAP 消息都使用 XML 消息格式来编码， XML 是 SOAP 的底层技术规范；

2，对于 SOAP 中的简单类型， SOAP 采用了在 XML Schema 规范的数据类型部分定义的内嵌数据类型 中所有类型，包括这些类型的值和词汇空间的定义；；

3 ，SOAP 完全继承了 XML 的开放性和描述可扩展性。

4 ，SOAP 还具有 XML 的其他一些优点，如可广泛应用于 web 的任何地方，使得编程更加简单，便于 阅读等等。

71. 选举算法中 **Bully** 算法的思想

选举算法： 选择一个进程作为协调者、发起者或其他特殊角色， 一般选择进程号最大的进程(假设每个 进程都知道其他进程的进程号， 但不知道是否还在运行) 它的目的是保证在选举之行后， 所有进程都认

可被选举的进程。

Bully 算法：

当进程 P 注意到需要选举一个进程作协调者时：

(1) 向所有进程号比它高的进程发 ELECTION 消息

(2) 如果得不到任何进程的响应，进程 P 获胜，成为协调者

(3)如果有进程号比它高的进程响应，该进程接管选举过程，进程 P 任务完成

(4)当其他进程都放弃，只剩一个进程时，该进程成为协调者

(5)一个以前被中止的进程恢复后也有选举权

72. 在分布式系统中，许多算法都需要一个进程充当协调者，因此需要协调者选举算法。试说明霸道算法的主要思想，并说明在 **8** 个进程的情况下号码为 **3** 的进程发现协调者 崩溃后的选举过程。

霸道算法： 当一个进程发现协调者不再响应请求时，它发起选举。进程 P 选举过程如下：

① P 向所有号码比它大的进程发送选举消息

② 若无人响应，P 获胜成为协调者。

③ 若有号码比它大的进程响应， 响应者接管， P 的工作完成。 假设 8 个进程为进程 0 到进程 7，原协调者为进程 7

进程 3 发现协调者崩溃，进程 3 主持选举， 进程 4 进程 5 和进程 6 应答， 通知进程 3 停止， 进程 4 进程 5 进程 6 分别主持选举，进程 6 通知进程 5 和进程 4 停止，进程 6 获胜并通知所 有进程。

（说白了，Bully算法中选择时间戳最大的进程，不行就选第二大的）

73. 无线环境下的选举算法

74. 描述一下客户和服务器之间使用套接字的无连接通信是如何进行的？

答：首先服务器和客户端都要创建一个套接字，并遵循 UDP 协议，服务器将其所在的 IP 地址以及一个端口号绑定到套接字，完成绑定后，服务器就能接收来自客户端的 UDP 数据包了。同样， 客户端在创建套接字后， 能够向服务器发送 UDP 包进行通信，通信过程中， 服务器和客户端之间是不用建立连接的。

75. 试描述客户和服务器之间使用套接字的面向连接的通信是如何进行的？

答： 为了实现服务器与客户机的通信，服务器和客户机都必须建立套接字。服务器与客户机的工作原理可以用下面的过程来描述。 (1)服务器先用 socket 函数来建立一个套接字，用

这个套接字完成通信的监听。 (2)用bind 函数来绑定一个端口号和 IP 地址。因为本地计

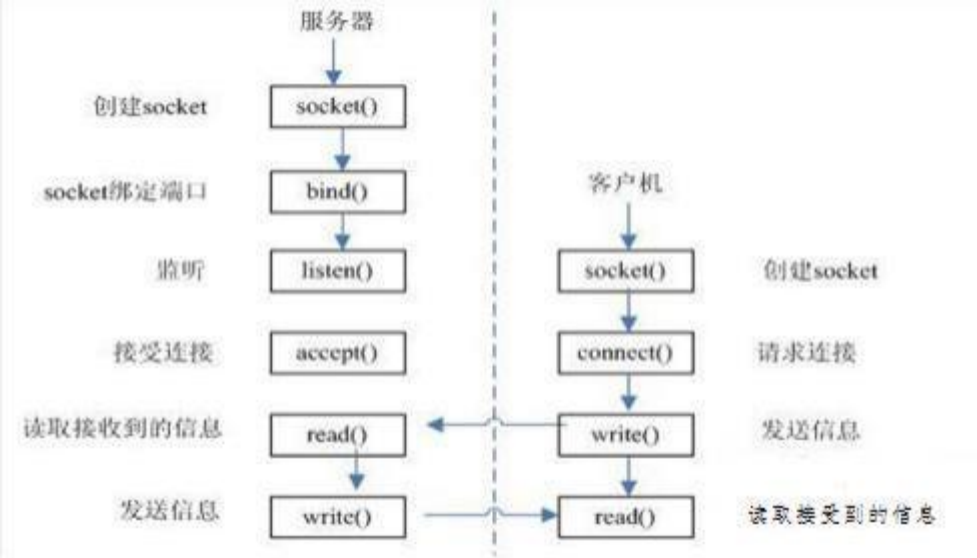
算机可能有多个网址和 IP，每一个 IP 和端口有多个端口。需要指定一个 IP 和端口进行监

听。 (3)服务器调用 listen 函数， 使服务器的这个端口和 IP 处于监听状态，等待客户机

的连接。 (4)客户机用 socket 函数建立一个套接字，设定远程 IP 和端口。 (5)客户机调用 connect 函数连接远程计算机指定的端口。 (6)服务器用accept 函数来接受远程计算机的 连接， 建立起与客户机之间的通信。 (7)建立连接以后， 客户机用 write 函数向 socket

中写入数据。也可以用 read 函数读取服务器发送来的数据。 (8)服务器用read 函数读取

客户机发送来的数据，也可以用 write 函数来发送数据。 (9)完成通信以后，用 close 函数 关闭 socket 连接。 客户机与服务器建立面向连接的套接字进行通信， 请求与响应过程可 用图来表示。



76. 简述 **TCP** 和 **UDP** 协议在通信中的区别

TCP 是面向连接的可靠的协议，适用于传输大批量的文件，检查是否正常传输。而 UDP 是面向非连接的不可靠的协议，适用于传输一次性小批量的文件， 不对传输数据报进行检查。TCP 需要先建立连接才能通话； 而 UDP 不需要，实时性要高点。

TCP 可以形象比喻为打电话的过程； UDP 可以比喻为发短信的过程。

TCP 不能发送广播和组播，只能单播；UDP 可以广播和组播。

77. 分布式系统的类型。

(1)分布式计算系统(分为群集计算系统和网格计算系统) (2)分布式信息系统(分为事务处理系统和企业应用集成) (3)分布式普适系统(如家庭系统、电子健保系统、传感器网络)

78. 移动边缘计算

答：MEC 是 ETSI 提出并主推的概念，经历了从移动边缘计算(Mobile Edge Computing) 到多接入边缘计算(Multi-access Edge Computing) 的演变。移动边缘计算是指在移动网络 边缘提供 IT 服务环境和云计算能力，将网络业务下沉到更接近移动用户的无线接入网侧， 旨在降低延时，实现高效网络管控和业务分发，改善用户体验；多接入边缘计算是指在网络 边缘为应用研发商和内容提供商提供 IT 服务环境和云计算能力，该环境为应用提供超低延 时、高带宽、实时接入等特性能力。

MEC 主要特性包括：就近接入、超低时延、位置可见、数据分析等。

移动边缘计算(Mobile Edge Computing, MEC) 可利用无线接入网络就近提供电信用户 IT 所需服务和云端计算功能，而创造出一个具备高性能、低延迟与高带宽的电信级服务环境， 加速网络中各项内容、服务及应用的快速下载，让消费者享有不间断的高质量网络体验。

79. 简述面向服务的体系结构(**SOA**)。

面向服务的体系结构，是一个组件模型， 它将应用程序的不同功能单元(称为服务) 通过这 些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的， 它应该独 立于实现服务的硬件平台、操作系统和编程语言。

3 个主要角色是：

1、服务请求者：服务请求者是一个应用程序、一个软件模块或需要一个服务的另一个服务。它发起对注册中心中的服务的查询，通过传输绑定服务，并且执行服务功能。服务请求者根据接口契约来执行服务。

2、服务提供者：服务提供者是一个可通过网络寻址的实体，它接受和执行来自请求者的请求。它将自己的服务和接口契约发布到服务注册中心， 以便服务请求者可以发现和访问该服 务。

3、服务注册中心： 服务注册中心是服务发现的支持者。它包含一个可用服务的存储库，并允许感兴趣的服务请求者查找服务提供者接口。

3 个主要的服务是：

发布： 为了使服务可访问．需要发布服务描述以使服务请求者可以发现和调用它。 查询： 服务请求者定位服务．方法是查询服务注册中心来找到满足其标准的服务。

绑定和调用：在检索完服务描述之后，服务请求者继续根据服务描述中的信息来调用服务。

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。其基本思想是，每个进程都按照一定的规则将选举消息发送给其后继进程，直到最终选举出一个进程作为协调者。

在环算法中，每个进程都按照进程号的顺序来排序，从而形成一个环。当需要选举一个协调者时，进程 P 发送一条包含自己进程号的 ELECTION 消息给其后继进程，后继进程再将自己的进程号加入该消息中，并将消息继续发送给自己的后继进程，依次类推，直到消息绕整个环一周后回到进程 P。

在这个过程中，每个进程都比较自己的进程号与收到的 ELECTION 消息中的进程号，如果自己的进程号比消息中的进程号大，则忽略该消息，否则将自己的进程号加入消息中，并将消息继续发送。最终，当消息回到进程 P 时，它就知道了环上所有在线进程的进程号，并选出其中最大的进程号作为新的协调者。

最后，进程 P 发送一条 COORDINATOR 消息到环上，包含新选出的协调者进程号和所有在线进程，这条消息在循环一周后被删除，随后每个进程都恢复原来的工作。这样，整个选举过程就完成了。

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？

答： 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，应该决定 是站点故障还是 Token 丢失，需要有对应逻辑环重构方法和 Token 生成方法。 可以引入时 时钟戳增加算法的强健性，具体如下: (1)当 request 消息后久未获得令牌，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。 (2)若接收到询 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 Request 消息的时钟戳 先于询问进程 Request 消息的时戳，则立即发回一条反对消息。(3)令牌持有者传递令牌时， 若发现接收者故障， 需要调用逻辑环重构算法进行环重构，再重新选择接收者。

82. 设计一个能支持分布式无用单元回收和在本地与远程对象引用之间转化的远程对象表。给出一个例子，包括在不同地址上的几个远程对象和代理，以阐述该表的使用。说明 当一个调用导致创建新代理时这个表的变化。说明当一个代理不能用时这个表的变

83. Jones 正在运行一组进程 **p1,p2,...，PN**。每个进程 **pi** 包含一个变量 **vi**。她希望判定所有变量 **v1,v2，...，vn** 在执行中是否相等。

(1) Jones 的进程在同步系统中运行。她使用一个监控器进程判定变量是否相等。应用进程何时应该与监控器进程通信，它们的消息应该包含什么?

(2)解释语句:可能的(v1=v2=...vn) .Jones 如何能判定该语句在她的执行中成立。

(i) communicate new value when local variable v_i changes;

with this value send: current time of day *C(e)* and vector timestamp *V(e)* of the event of the change, *e*.

(ii) *possibly* (...): there is a consistent, potentially simultaneous global state in which the given predicate is true.

Monitor process takes potentially simultaneous events which correspond to a consistent state, and checks predicate $v_1 = v_2 = \ldots = v_N$.

Simultaneous: estimate simultaneity using bound on clock synchronization and upper limit on message propagation time, comparing values of *C* (see p. 415).

Consistent state: check vector timestamps of all pairs of potentially simultaneous events e_i, e_j: check $V(e_i)[i] \geq V(e_j)[i]$.

84. 修改用于互斥的中央服务器算法，使之能够处理任何客户〔在任何状态**)**的崩溃故障，假设服务器是正确的，并且有一个可靠的故障检测器。讨论这个系统是否能够容错。 如果拥有令牌的客户被错误地怀疑为出了故障，会发生什么样的情况**?**

为了使系统容错，可以使用主备复制技术，即有一个主服务器和一个备份服务器。主服务器处理客户端请求，备份服务器监测主服务器是否出现故障，一旦检测到故障，备份服务器接管主服务器的角色。如果一个客户端被错误地怀疑为出了故障，系统可以通过等待一段时间来确定客户端是否真的故障，并在超时后采取适当的措施。通过这些措施，系统可以实现容错，能够处理任何客户端的崩溃故障。

The server uses the reliable failure detector to determine whether any client has crashed. If the client has been granted the token then the server acts as if the client had returned the token. In case it subsequently receives the token from the client (which may have sent it before crashing), it ignores it.

The resultant system is not fault-tolerant. If a token-holding client crashed then the application-specific data protected by the critical section (whose consistency is at stake) may be in an unknown state at the point when another client starts to access it.

If a client that possesses the token is wrongly suspected to have failed then there is a danger that two processes will

be allowed to execute in the critical section concurrently.

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两部分：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。

集中式数据库的模式结构采用三级模式体系结构，包括外模式、概念模式和内模式，用于实现数据的逻辑独立性和物理独立性。

分布式数据库系统增加了更多级别的模式，包括全局外模式、全局概念模式、分片模式和分布模式。全局外模式是全局概念模式的子集，用于表示全局应用所涉及的数据部分；全局概念模式定义了分布式数据库的全局逻辑结构；分片模式用于说明如何放置数据库的特殊部分，如何划分逻辑片和如何定义不同片段之间的关系；分布模式定义了片段存放节点，使得全局查询可以分为若干子查询，每个子查询所访问的数据属于同一场地的局部数据库。

分布式数据库系统中所增加的模式和相应的映像，使得分布式数据库系统具有分布透明性，可以隐藏数据的分布和复杂性，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。

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。

分片透明性是指用户不必关心数据的逻辑分片，不必关心数据物理位置分配的细节，也 不必关心各个场地上数据库的数据模型。

复制透明性(数据的一致性) 指用户不必关心数据的一致性问题， 不必关心数据在各个 站点的一致性如何实现的细节， 由分布数据库自动完成。

分布式数据库具有分布透明性，或称作位置透明性(Location Transparency) 即程序的 正确性不受数据从一个节点到另一个节点移动的影响，使用者自己并不需要知道数据是存在 哪一个位置上。并且分布式数据库中个别位置或位置间通讯链发生故障时， 也能继续运行， 提供了比中央集权系统更高的可靠性。

分布式数据库的数据可以存储在靠近它正常使用的地方，可以减少响应时间与通讯费用。 分布式数据库是将一些功能较少的数据库综合成一个功能更强的数据库系统，显然， 增加一 个新的站，远比用一个更大的系统代替已有的集中式系统要容易。使整个系统结构十分灵活，部分增减对系统的其它部分影响较小

87. 请你给出单点登录的实现方案？

单点登录的实现方案一般包括基于 Cookie 和基于分布式 Session 两种方式。

基于 Cookie 的单点登录实现方式简单，使用 Cookie 存储用户凭证，但存在安全和跨域问题。基于分布式 Session 的方式是目前大型网站采用的方式，通过将会话信息存储在分布式缓存系统中实现多个应用共享会话状态，从而实现单点登录。该方式可以解决基于 Cookie 的方式存在的安全和跨域问题。分布式 Session 的实现一般采用 Cache 中间件，建议使用 Redis，实现分布式 Session 的宕机恢复和持久化存储。

87-2. 补充：分布式哈希表DHT的应用：

在负载均衡场景中，分布式哈希表通常被用于实现一种称为一致性哈希的算法。一致性哈希将数据映射到一个哈希环上，并将节点也映射到该哈希环上。当需要存储或访问某个数

据项时，先将该数据项的键值通过哈希函数计算出在哈希环上的位置，然后顺时针找到最近的节点进行处理。这样可以实现动态添加或删除节点时对数据的均衡分布，从而实现负载均衡。

在分布式锁场景中，分布式哈希表通常被用于实现一种称为基于 Redis 的分布式锁。该锁的实现需要使用 Redis 的分布式缓存系统，将锁的状态存储在分布式哈希表中。当多个客户端需要获取锁时，先通过哈希函数计算出锁所在的节点，然后在该节点上执行获取锁的操作，从而实现分布式锁的互斥性。

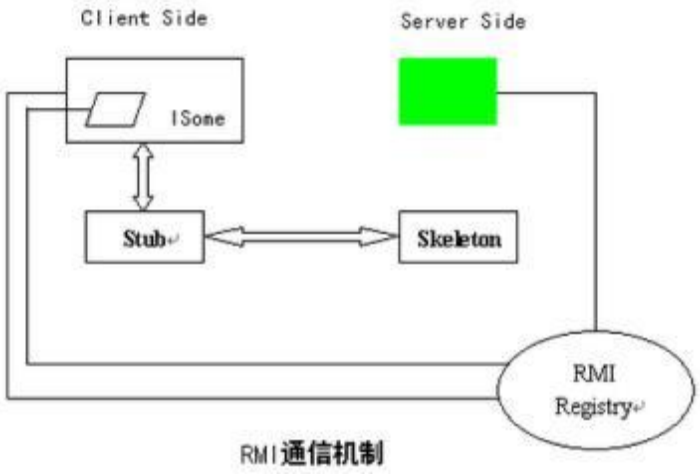
88. 实现分布式系统同步的复杂性表现在哪几个方面？说明先发生关系， 并说明在 **LAMPORT** 算法中怎样给事件分配时间。

答：分布式算法有如下性质：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。前三点说明在一处收集所有 信息并对他们进程处理是不可接受的，左后一点说明在分布式系统获得时间上的一致并不是容易的。 LAMPORT 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， 接受者的时钟比消息发送者时钟小， 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，我们给出一种测量时间的方法， 使得对每一事件 a，在所有进程中都认可给它一个时间值 C(a)， 在给事件分配时间时要遵循一下规则： 1)在同一进程中 a 发生在 b 之前则 C(a)<C(b)；2)若 a 和 b 分 别代表发送消息和接收消息，则 C(a)<C(b)；3)对所有事件 a 和b ，C(a)≠C(b)

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。

(**Remote Method Invocation，RMI**)的基本通信原理。

答:：远程方法调用(RMI)的基本通信原理:



RMI通信机制

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。

92. 服务器虚拟化的关键技术？

服务器虚拟化的关键技术包括计算虚拟化、存储虚拟化、设备与 I/O 虚拟化以及实时迁移技术。

- 计算虚拟化：实现CPU虚拟化、计算负载的动态分配和能耗管理。
- 存储虚拟化：实现内存虚拟化和磁盘存储动态分配。
- 设备与 I/O 虚拟化：通过软件方式实现，提供统一、标准化的接口，以及操作指令转译。
- 实时迁移技术：将整个虚拟机的运行状态完整、快速地从原宿主机的硬件平台转移到新的宿主机硬件平台。

93. 举例说明 **Lamport** 等人提出的算法是如何解决 **Byzantine** 将军问题的。

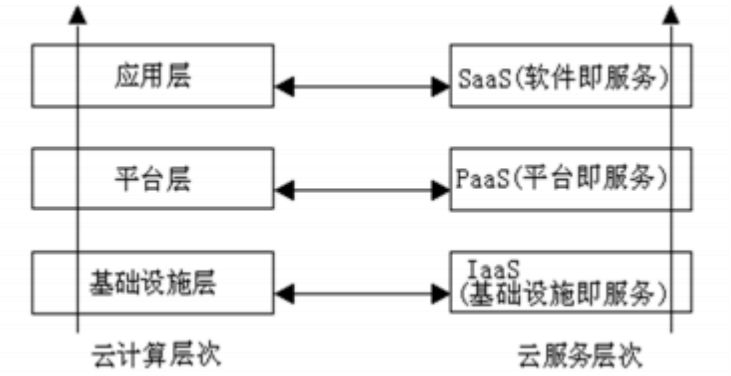
94. 云计算技术体系结构可以分为哪几层？

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

答：分布式计算研究如何把一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。分布式计算依赖于分布式系统， 分布式系统由通过网络连接的多台计算机组成。网络把大量分布在不同地理位置的计算机连接在一起，每台计算机都拥有独立的处理器及内存。这些计算机互相协作， 共同完成一个目标或者计算任务。

分布式计算是一个很大的范畴，在当今的网络时代，不是分布式计算的应用已经很少了。因此，云计算也是分布式计算的一种。

97. 简述私有云、公用云和混合云的基本概念

答：私有云也叫做专用云，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 (公有云) 是由第三方运行的云，第三方可以把来自许多不同客户的作业混合一起部署在云内的服 务器、存储系统和其它基础设施上。 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。

混合云把公用云模式与私有云模式结合在一起，客户可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。

98. 举例描述***aaS** 的概念。

云计算按照其提供的“产品”或者是用户获得资源的类型，大致可以分为一些几种类别：

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， Google AppEngine， 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。

99. 服务器虚拟化的特性？

多实例：一个物理服务器上可以运行多个虚拟服务器，支持多个客户操作系统，物理系统资源以可控的方式分配给虚拟机。

隔离性：虚拟机之间完全隔离，一个虚拟机的崩溃不会对其他虚拟机造成影响，虚拟机之间的数据相对独立，不会泄露。虚拟机之间如果需要互相访问，方式等同于独立物理服务器之间的互相访问。

封装性：硬件无关，对外表现为单一的逻辑实体，一个虚拟机可以方便的在不同硬件之间复制、移动。将不同访问方式的硬件封装成统一标准化的虚拟硬件设备，保证了虚拟机的兼容性。

高性能：可通过扩展获得“无限”的性能，虚拟化抽象层需要一定管理开销。

100. 服务器虚拟化的关键技术？

○1计算虚拟化

–CPU 虚拟化

–计算负载的动态分配

–能耗管理

○2存储虚拟化

–内存虚拟化

–磁盘存储动态分配

○3设备与 I/O 虚拟化

–软件方式实现

–统一、标准化的接口

–操作指令转译

○4实时迁移技术

–将整个虚拟机的运行状态完整、快速地从原宿主机的硬件平台转移到新的宿主机硬件平 台。

101. 简述管理虚拟集群的四种方式。

第一种方式是基于客户的管理器，其中集群管理器处于客户系统中。在这种管理方式中，多个虚拟机形成一个虚拟集群。

第二种方式是基于主机的集群管理器。监督客户系统且能在另一个物理机器上重启客户系统。第三种方式是在主机系统和客户系统中使用相互独立的集群管理器来管理虚拟集群。然而，这会使基础设施管理变得更为复杂。

第四种方式是在主机系统和客户系统中使用集成的集群。这表示管理器能区分虚拟资源和物理资源。

102. Multitenant technique

Multitenant technique refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple tenants. With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance. Therefore, it is one of the essential attributes of Cloud Computing.

According to this understanding, basically Amazon Web Services (AWS), Force.com, Google App Engine, and Microsoft Azure are all examples of multitenant technique, as they all have multi-tenancy at some level in the system architecture stack.

多租户 (就是能让一个单独的应用实例可以为多个组织服务， 而且保持良好的隔 离性和安全性，并且通过这种技术，能有效地降低应用的购置和维护成本) :i. 是一种软件架构技术， 它是在探讨与实现如何于多用户的环境下共用相同的系统 或程序组件， 并且仍可确保各用户间数据的隔离性。ii.实现重点， 在于不同租户 间应用程序竟的隔离以及数据的隔离, 以维持不同租户间应用程序不会相互干扰， 同时数据的保密性也够强。 1)程序部分： 通过进程或是支持多应用程序同时运行 的装载环境来做进程间的隔离， 或是在同一个服务程序进程内以运行就绪的方式 隔离。2)数据部分：通过不同的机制将不同租户的数据隔离， Force 是采用中介

数据的技术来切割，微软 MSDN 的技术文件则是展示了使用结构描述的方式隔离。

103. 简述什么是 **SOAP**，以及 **SOAP** 与 **XML** 之间的关系。

SOAP (Simple Object Access Protocol)即简单对象访问协议，是一个简单的用在 Web 上交换结构信息 的 XML 协议， 没有定义应用语义和传输语义， 它以一种基于 XML 且与平台无关的 Web 编程方式改进 了 Internet 的互操作性。 SOAP 消息传递协议使用HTTP 承载消息，而使用XML 格式化消息。

SOAP 包括四个部分：

1，信封； 2，数据的编码规则；3，RPC 调用规范； 4，SOAP 绑定。 SOAP 与XML 之间的关系是：

1，SOAP 是在 XML 基础上定义的，所有的 SOAP 消息都使用 XML 消息格式来编码， XML 是 SOAP 的底层技术规范；

2，对于 SOAP 中的简单类型， SOAP 采用了在 XML Schema 规范的数据类型部分定义的内嵌数据类型 中所有类型，包括这些类型的值和词汇空间的定义；；

3，SOAP 完全继承了 XML 的开放性和描述可扩展性。

4，SOAP 还具有 XML 的其他一些优点，如可广泛应用于 web 的任何地方，使得编程更加简单，便于 阅读等等。

104. 在面向消息的通信中，什么是持久通信和暂时通信？试举例说明。

答：持久通信：发送者发送消息后不需要再保持运行状态， 接收者在发送者发送消息时也不需要处于运 行状态。典型例子：电子邮件系统。传输的消息在提交之后由通信系统存储，直到将其交付给接收者。 工作方式类似于驿马快递制度。

暂时通信：通信系统只在发送和接收消息的应用程序运行期间存储消息。典型例子： 所有传输层通信服 务，存储转发式路由器。

105. 讨论下列操作是否是幂等的**:**

·按电梯的请求按钮。

.写数据到文件。

.将数据追加到文件。

操作不应该与任何状态相关是不是幂等的必要条件?

The operation to write data to a file can be defined (i) as in Unix where each write is applied at the read-write pointer, in which case the operation is not idempotent; or (ii) as in several file servers where the write operation is applied to a specified sequence of locations, in which case, the operation is idempotent because it can be repeated any number of times with the same effect. The operation to append data to a file is not idempotent, because the file is extended each time this operation is performed.
The question of the relationship between idempotence and server state requires some careful clarification.
It is a necessary condition of idempotence that the effect of an operation is independent of previous operations. Effects can be conveyed from one operation to the next by means of a server state such as a read-write pointer or a bank balance. Therefore it is a necessary condition of idempotence that the effects of an operation should not depend on server state. Note however, that the idempotent file write operation does change the state of a file.

106. 如何解决幂等性问题

全局唯一 ID：生成唯一 ID，传入接口，检索是否存在，避免重复执行。

去重表：在操作和写入去重表放入一个事务中，当出现重复操作时，数据库抛出唯一约束异常，操作回滚。

多版本并发控制：增加版本号来做幂等性控制，更新 SQL 语句增加版本号判断。

状态机控制：增加状态字段表示状态，更新时增加状态判断。

插入或更新：在 MySQL 中使用 ON DUPLICATE KEY UPDATE 子句，避免记录重复插入

更多细节查看源文档

107. 举例说明时间触发和事件触发的区别。

答：事件触发是指，当一个重要的外部事件触发时，它被传感器察觉到，并导致与传感器相连的 cpu 得到一个中断请求。

时间触发是指， 在每隔固定的时间 t 后产生一次时钟中断， 对选定的传感器进行采样， 并且驱动(特定 的)执行机构。

举例，考虑一个 100 层楼的电梯控制器设计。假定电梯正在 60 层安静的等待顾客，有人在一层按下按 钮。就在 100 毫秒后， 另一人在 100 层按下按钮。在事件触发系统中， 第一次按钮产生一个中断， 将使 电梯启动下行， 就在他做出下行决定后， 第二个按下按钮的事件到来， 因此第二个事件被记录下来以作 将来的参考，但电梯还是继续下行。

若考虑时间触发系统， 没 500 毫秒采样一次。若两次按下按钮都在一次采样周期中出现， 控制器就不得 不进行决定，例如按最近用户优先原则，此时电梯将上行。

由以上例子可以看出，事件触发的设计在低负载时会更快响应，但在高负载时可能崩溃。时间触发相反， 仅适用于相对静态的环境。

108. 使用上载/下载模式的文件服务器系统与使用远程访问模式的文件系统之间有什么区别?

答：文件服务分为两种类型：上载/下载模式和远程访问模式。

上载/下载模式。只提供两种主要的操作(读文件和写文件)，读操作将整个文件从文件服务器传输到请

求客户端， 写操作则刚好相反， 文件系统运行在客户端。优点是系统概念简单， 应用程序取得需要的文 件，并在本地使用它。当程序结束时， 将所有修改的文件或新创建的文件写回去， 不需要管理复杂的文 件服务接口， 文件传输效率高。缺点是客户端需要足够的存储空间， 当需要部分文件是需要传输整个文 件。

远程访问模式。提供了大量的操作，如打开、关闭文件，读写部分文件等等。文件系统在服务器端

运行。优点是客户不需要大量存储空间，当需要部分文件是不需要传输整个文件。

109. 说明保持客户高速缓存一致性的四种算法。

直接写：缓存和服务器同时更新，但其他进程读取可能会读取过期的值，需要从服务器中读取文件版本进行比较。

延迟写：操作不立即发送给服务器，而是延迟一段时间，但依赖于时间，读取的结果不确定，语义模糊。

关闭写：操作只有在文件关闭后才写回服务器，在 session 语义下使用。

集中控制：在文件服务器上保存了进程对文件的操作方式等信息，类似于锁机制的管理，避免写操作的文件被其他进程操作，操作结束时通知服务器，操作的结果立即送到服务器

答： 1)直接写，当缓存中的文件被更新后，新的值在缓存在保存，而且同时发送到服务器，而当另外 的进程访问文件时， 读到的是最新值， 但是存在一个问题， 其他进程在更新之前读到的文件内容可能是 过期的，那么在每次用到文件时需要从服务器中读取文件版本进行比较， 查看是否过期， 但是每次都要 在服务器和客户端之间通信，这样就体现不出缓存的作用了。

2) 延迟写，操作不立即发送给服务器，而是延迟一段时间，也就减少了网络消息，当进程读取文件时，依赖于时间。具体读到的是那次操作的结果不确定，语义模糊。 3)关闭写，操作只有在文件关闭后才写回服务器，配合 session 语义。 4)集中控制，就是在文件服务器上保存了进程对文件的操作方式等信息，类似于锁机制的管理，避免 写操作的文件被其他进程操作，但是当修改的操作结束时，会将操作结束消息通知服务器，操作的结果 也就立即会送到服务器

110. 试分别解释严格一致性、顺序一致性、因果一致性、PRAM 一致性等几种以数据为中心的一致性模型的含义。下图中的事件序列对上述哪几种一致性模型是有效的？

严格一致性要求所有进程看到的事件顺序都相同；顺序一致性要求所有进程最终看到的顺序相同，但可能会有短暂的不一致；因果一致性要求所有进程按照相同的顺序看到因果联系的事件；PRAM一致性要求进程按照预定顺序看到来自一个处理器的写操作，但对于来自其他处理器的写操作，顺序不做要求。图中的事件序列满足因果一致性和PRAM一致性，但不满足严格一致性和顺序一致性。

严格一致性模型：所有共享访问事件都有绝对时间顺序；

顺序一致性模型：所有进程都以相同的顺序检测到所有的共享访问事件；

因果一致性模型：所有进程都以相同的顺序检测到所有因果联系的事件；

PRAM 一致性模型：所有的进程按照预定的顺序检测到来自一个处理器的写操作，来自 其他处理器的写操作不必以相同的顺序出现；

图中的事件序列对因果一致性、PRAM 一致性是有效的。

111. 列出 **URL** 的 **3** 个主要成分，叙述如何表示它们的边界，并用例子说明每个成分。在什么程度上 **URL** 是位置透明的**?**

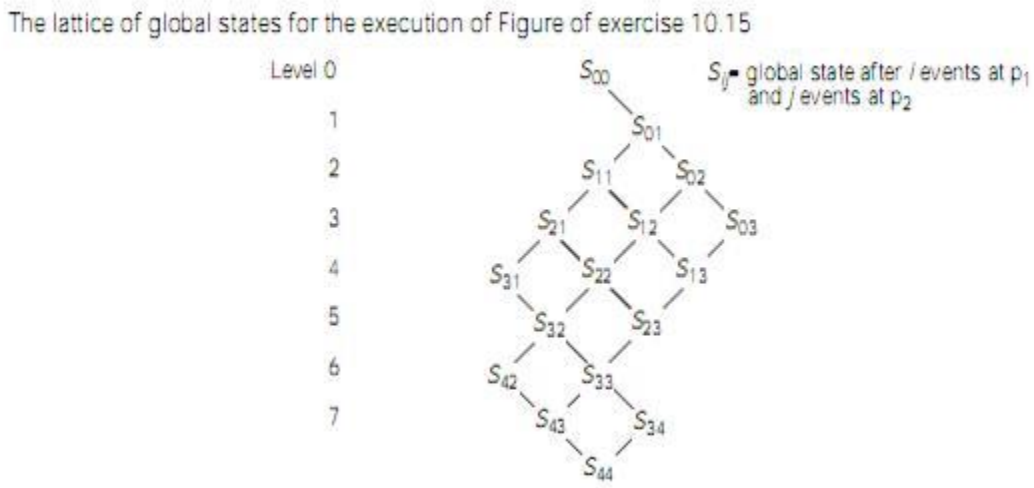
112. 简述基于中间件技术的分布式计算系统的组成结构。

基于中间件技术的分布式计算系统由分布式应用、中间件服务和网络操作系统服务三个层次组成。**中间件服务作为高层抽象层，连接应用程序和网络操作系统服务，提高了应用程序的透明度和易用性**。网络操作系统服务是底层的软件系统，提供基础服务。

有时间版本：

基于中间件技术的分布式计算系统由三个层次组成：分布式应用、中间件服务和网络操作系统服务。分布式应用是用户需要执行的应用程序，它们通过中间件服务来实现分布式计算。中间件服务是介于应用程序和网络操作系统之间的软件层次，提供了一个高层抽象层，使得应用程序能够更方便地访问网络操作系统的服务，同时隐藏底层的细节，提高了应用程序的透明度。网络操作系统服务是底层的软件系统，提供网络通信、文件系统等基础服务，中间件服务可以利用网络操作系统服务提供的接口来实现分布式计算，使得分布式计算系统更加易用、高效、可靠。

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。



114. 在多计算机系统中的 **256** 个 **CPU** 组成了一个 **16 X 16** 的网格方阵。在最坏的情况下， 消息的延迟时间有多长(以跳**(hop)**的形式给出，跳是结点之间的逻辑距离) ？

答： 假设路由是最优的，最长的路由是从网格方阵的一个角落到对角的角落。那么这个路由的长度是 30 跳。如果一行或一列中的处理器彼此相连，则路由长度为 15 跳。

115. 虚拟机的两种架构的本质差别表现在那些方面？

虚拟机有两种架构： 寄居架构(Hosted Architecture)和裸金属架构 (“Bare Metal” Architecture)。

116. SAN 和 **NAS** 在存储虚拟化方面有哪些不同？

117. GAE 平台的应用服务架构与传统的 **Web** 应用服务架构有什么异同？

118. 为了解决大量数据的存储和计算能力不足的问题， 我们有两种选择：

1.纵向扩展

即升级硬件设备。通过如升级存储量更高的硬盘、单位时间运算速度更高的芯片， 可以为计算机提升性 能和存储量， 来解决上述问题。但这种硬件的升级会受到计算机本身架构的局限， 同时进一步升级所需 要的成本会快速上升， 从而使得升级硬件的边际效益快速下降。此外， 升级所用的硬件依然具有其自身 的局限性，并不能彻底解决上述的问题。

2.横向扩展

使用多台普通计算机来模拟“超级计算机”。也就是使用多台机器各自并行地进行存储和计算这种方式进 行模拟。使用这种方式构建的计算机系统叫做分布式系统， 它具有以下三个特点： 一组计算机、通过网 络传递信息、协调计算机的行为。通过这种方式， 我们可以近似地获得无限的存储和计算能力， 解决单 机下存储和计算的局限。

119. 说明分布式系统相对于集中式系统的优点和缺点。 从长远的角度看， 推动分布式系统发展的主要动力是什么？

相对于集中式系统，分布式系统的优点包括经济性、计算能力、分布性、可靠性和前景。而分布式系统的缺点则包括软件问题、通信网络问题和安全问题。从长远的角度看，推动分布式系统发展的主要动力是大量个人计算机的存在和人们共同工作于信息共享的需要，这种信息共享必须是以一种方便的形式进行，不受地理或人员、数据以及机器的物理分布的影响。

答：相对于集中式系统，分布式系统的优点： 1)从经济上， 微处理机提供了比大型主机更好的性能价 格比；2) 从速度上， 分布式系统总的计算能力比单个大型主机更强； 3) 从分布上， 具有固定的分布性，

一些应用涉及到空间上分散的机器； 4)从可靠性上，具有极强的可靠性，如果一个极强崩溃，整个系 统还可以继续运行； 5)从前景上，分布式操作系统的计算能力可以逐渐有所增加。

分布式系统的缺点： 1) 软件问题， 目前分布式操作系统开发的软件太少； 2) 通信网络问题，一旦一个 系统依赖网络， 那么网络的信息丢失或饱和将会抵消我们通过建立分布式系统所获得的大部分优势； 3) 安全问题，数据的易于共享也容易造成对保密数据的访问。

推动分布式系统发展的主要动力： 尽管分布式系统存在一些潜在的不足， 但是从长远的角度看， 推动分 布式系统发展的主要动力是大量个人计算机的存在和人们共同工作于信息共享的需要， 这种信息共享必 须是以一种方便的形式进行。而不受地理或人员，数据以及机器的物理分布的影响

120. Browser/Server 体系结构与 **Client/Server** 体系结构相比不仅具有 **Client/Server** 体系结构的优点，而且又有 **Client/Server** 体系结构所不具备的独特优势。

开放的标准： Client/Server 所采用的通信协议往往是专用的。Browser/Server 所采用的标准， 如 HTTP、

HTML 等，都是开放的、非专用的，是经过标准化组织所确定的， 保证了其应用的通用性和跨平台性。 较低的开发和维护成本： Client/Server 的应用必须开发出专用的客户端软件， 无论是安装、配置还是升

级都需要在所有的客户机上实施，极大地浪费了人力和物力。 Browser/Server 的应用只需在客户端装有 通用的浏览器即可， 维护和升级工作都在服务器端进行， 不需对客户端进行任何改变， 故而大大降低了 开发和维护的成本。

使用简单，界面友好： Client/Server 用户的界面是由客户端软件所决定的，其使用的方法和界面各不相 同，每推广一个 Client/Server 系统都要求用户从头学起，难以使用。 Browser/Server 用户的界面都统一 在浏览器上， 浏览器易于使用、界面友好， 不须再学习使用其他的软件， 一劳永逸地解决了用户的使用 问题。

客户端简单： Client/Server 的客户端具有显示与处理数据的功能， 对客户端的要求很高， 是一个“胖” 客户机。Browser/Server 的客户端不再负责数据库的存取和复杂数据计算等任务，只需要根据结果数据 中指明的格式对其进行显示， 充分发挥了服务器的强大作用， 这样就大大地降低了对客户端的要求， 客

户端变得非常“瘦”。

121. 应用哪些技术可以使得一个分布式系统具有可伸缩性？
伸缩性：增加请求时依旧保持高性能

异步通信方式：通过异步通信方式，发送请求后不必阻塞等待答复，而是可以处理其他本地任务，从而减少通信延迟，提高系统性能。

分层架构：将一个组件分解为几个小层，实现分层架构，可以减少单个组件的复杂度，提高系统的可维护性和可扩展性，同时也能均衡系统负载。

复制冗余：通过复制冗余技术，可以将资源分布于整个系统，使得资源更容易就近获取，并且均衡系统负载，提高系统可靠性和可用性。

此外，还可以采用负载均衡、缓存技术、分布式数据库等技术，来提高分布式系统的可伸缩性和性能。

答：实现分布式可伸缩性，基本的三种技术为：

1、减少通信延迟，即使用异步通信方式，使得发送方发送请求后不必阻塞以等待答复， 而是处理其他 本地任务。

2、分层， 即将一个组件分解为几个小层。一个好的例子是 DNS 域名系统， 它将域名分为三层， 均衡了 系统负载。

3 、复制冗余，它能使得资源更容易就近获取，并且它能使资源分布于整个系统，均衡了负载。

122. 给出一个多线程客户端的例子，并给出一种构造多线程服务器的方法。

123. 在支持多线程的系统中， 可以采用三种模型来组织多线程，详细说明这三种模型。如果在不支持多线程的系统中实现文件服务，如何构造文件服务器呢**?**

在支持多线程的系统中，可以采用三种模型来组织多线程:

派遣者/工作者模型：一个线程作为派遣者，从系统邮箱读出请求，然后从空闲的工作者中选取一个工作者线程去处理请求。

团队模型：所有线程平等存在，每个线程接受来自客户端的请求并处理请求，可以维护一个队列缓存请求。

管道线模型：一组线程处理一个请求，前一个线程处理的结果输出会作为下一个线程的输入，数据持续从一个线程到另一个线程，每经过一个线程都会对数据进行处理。

如果在不支持多线程的系统中实现文件服务，可以通过创建一个无限的死循环来不断响应来自客户端的请求连接，每次请求处理完毕后就断开连接。但这种方法不能同时响应多个客户端的多个请求，也不支持缓存等高级功能。

124. 在代码迁移时，需要迁移代码片断、资源片断和执行片断，说明在迁移资源片断时需要考虑的主要问题。

将一个已经存在的分布式系统迁移到一个新的环境中，例如迁移到新的硬件平台、新的操作系统、新的云平台或者新的编程语言等

在代码迁移中，需要迁移代码片断、资源片断和执行片断。在迁移资源片段时，需要考虑相关引用的改变、机器之间的绑定关系以及资源片段的兼容性，以确保迁移后系统的性能、可用性和可维护性得到提高。

答：迁移资源片段时，有时需要考虑改变资源片段的相关引用，以适应迁移后的使用，但是又不能改变 该资源与其他进程之间的绑定关系。此外，有时还需考虑该资源片段与机器之间的绑定关系。例如有些 资源片段迁移后可用，但是要花很大代价，有些则迁移后在其他机器上不可执行。

125. 移动实体、宿主代理和宿主位置的理论

126. 什么是有状态服务器和无状态服务器，给出相应的例子，并说明有状态服务器存在的问题。

答：无状态服务器，在请求之间，服务器不保存具体客户的信息，以及与客户端交互活动的有关信息。 它要求每个请求必须是独立的， 必须包含全文件名和文件中的偏移量，因此消息长度较长。 有状态服 务器，在请求之间，服务器保存客户信息以及与客户交互活动的有关信息

127. 分布式系统中处理共享文件的四种方法(文件共享的四种语义)。

方法	说明
UNIX 语义	对一个文件的任何操作对所有进程都是及时可见的
会话语义	在文件关闭之前，对文件的修改对其他进程是不可见的
不可更改语义	不能进行更改，只是简单的共享和复制
事务	所有的更改要么都完成，要么都不能完成

128. 说明无状态服务器和有状态服务器，并说明二者的区别。

·无状态服务器：当客户发送一个请求到给服务器， 服务器完成请求，发送一个应答，然后从内部表中移 出关于该请求的所有信息。在请求之间，服务器不保存具体客户的信息。

·有状态服务器：服务器保存两个请求之间的客户的状态信息。

无状态服务器的优点	有状态服务器的优点
容错	请求消息比较短
不需要 OPEN/CLOSE 调用	更好的性能
没有服务器表空间的浪费	可以预读
没有打开文件数目的限制	易于幂等性

客户崩溃时不会造成服务器错误	可以对文件加锁

129. 客户为了发送消息给服务器，它必须知道服务器的地址。试给出服务器进程编址的几种方法，并说明如何定位进程。

答：方法一。机器号加进程号， 内核使用机器号将消息正确地发送到适当的机器上， 用进程号决定将消 息发送给哪一个进程。

方法二。进程选择随机地址， 通过广播方式定位进程， 进程在大范围的地址空间中随机指定自己的标识 号。在支持广播式的 LAN 中， 发送者广播一个特殊的定位包，其中包含目的进程地址，所有的内核查 看地址是不是他们的，如果是则返回消息给出网络地址，然后发送内核缓存地址。

方法三。客户机运行时，使用 ASCII 码访问服务。客户机运行时，向名字服务器发送请求信息，名字 服务器将 ASCII 服务器名映射成服务器地址，客户机收到给地址后，可以访问服务器。

130. 分布式系统的类型。

(1)分布式计算系统(分为群集计算系统和网格计算系统) (2)分布式信息系统(分为事务处理系统和企业 应用集成) (3)分布式普适系统(如家庭系统、电子健保系统、传感器网络)

131. 客户端-服务器模型。

132. 论大规模分布式系统缓存设计策略

大规模分布式系统通常需要利用缓存技术来减轻服务器负载、降低网络拥塞、增强系统可扩展性。常见的**缓存工作模式包括单实例模式、复制模式和分区模式**。每种工作模式都有其适应的场景和优缺点。单实例模式适用于缓存要求简单、系统吞吐量和数据量不大、性能要求不高的场景。复制模式适用于数据量不是特别大、需要高性能、数据改动频率不高的场景。分区模式适用于总体数据量较大、需要可伸缩性、客户端数量庞大且单个客户端对缓存数据的数据量要求不大的场景。在进行大规模分布式系统开发时，必须从一开始就全面考虑应用需求和场景对系统的缓存机制进行设计，以实现可伸缩的系统缓存架构。

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。

134. 进程和线程的比较。

进程定义为执行中的程序。未引入线程前是资源分配单位(存储器、文件) 和 CPU 调度(分配) 单位。 引入线程后，线程成为 CPU 调度单位，而进程只作为其他资源分配单位。

线程是 CPU 调度单位，拥有线程状态、寄存器上下文和栈这些资源，同线程一样也有就绪、阻塞和执 行三种基本状态。 (1)对于地址空间和其他资源(如打开文件)来说，进程间是相互独立的，同一进程的各线程间共享 该进程地址空间和其他资源(某进程内的线程在其他进程不可见)。

(2)在通信上，进程间通信通过 IPC，线程间可以直接读写进程数据段(如全局变量)来进行通信－

－需要进程同步和互斥手段的辅助，以保证数据的一致性。

(3) 在调度上， 线程上下文切换比进程上下文切换要快得多。线程是 CPU 调度单位， 而进程只作为其 他资源分配单位。线程的创建时间比进程短； 线程的终止时间比进程短； 同进程内的线程切换时间比进 程短。 因

此，多线程能提高性能；线程不像进程那样彼此隔离，并受到系统自动提供的保护，因此多 线程应用程序开发需要付出更多努力。

135. 多线程服务器的优点？

多线程技术不仅能够显著简化服务器代码， 还能够使得应用并行技术来开发高性能的服务器变得更加容 易，即使在单处理器系统上也是如此。多线程能够保留顺序处理的思路，使用阻塞性系统的系统调用， 仍然能到达并行处理的目的。使用阻塞系统调用使编程更容易，并行处理能提高系统的性能。

136. 什么软件代理？举例说明其作用。

软件代理是一些独立的单元， 能与其他的代理进行协作， 一同执行任务。定义为对环境的变化做出反应，并且启动这种变化的自治进程， 而且可以与用户代理或其他代理协同。与进程的区别在于能够对自己执行操作，在适当的时候采取主动。

代理分类：

(1)合作代理：通过协作达到某个共同的目标： 会议安排

(2)移动代理：能够在不同机器间迁移

(3)接口代理：协助最终用户使用应用程序，拥有学习能力：促成买卖

(5)信息代理：管理来自多个信息源的信息：排序、过滤和比较 a. 固定信息代理：电子邮件代理

b.移动信息代理：网络漫游，搜集所需信息

137. 代码迁移的动机有哪些？

代码迁移指的是将程序(或执行中的程序) 传递到其它计算机。 (基本思想： 把进程由负载较重的机器 上转移到负载较轻的机器上去，就可以提升系统的整体性能)

迁移动机：

(1) 实现负载均衡：将进程从负载重的系统迁移到负载轻的系统，从而改善整体性能。 (2)改善通

信性能： 交互密集的进程可迁移到同一个节点执行以减少通信开销， 当进程要处理的数据量较大时， 最 好将进程迁移到数据所在的节点。

(3) 可用性： 需长期运行的进程可能因为当前运行机器要关闭而需要迁移。 (4) 使用特殊功能：可 以充分利用特定节点上独有的硬件或软件功能。 (5) 灵活性： 客户首先获取必需的软件， 然后调用服 务器。

138. 代码迁移时进程对资源的绑定类型有哪些？

进程对资源的绑定类型有三类：分别是按**标志符(URL)、按值和按类型**。

139. 代码迁移时资源对机器的绑定类型有哪些？

资源对机器绑定类型分成：未连接(数据文件)、附着连接(数据库)和紧固连接(本地设备)三类。

资源对机器绑定

进程对资源绑定	未连接	附着连接	紧固连接
按标志符	MV (or GR)	GR (or MV)	GR
按值	CP (or MV, GR)	GR (or CP)	GR
按类型	RB (or MV, CP)	RB (or GR, CP)	RB (or GR)

- MV：移动资源
- GR：建立全局系统范围内引用
- CP：复制资源的值
- RB：将进程重新绑定到本地同类型资源

140. 什么是远程过程调用？远程过程调用的步骤。

远程过程调用（RPC）是本地程序调用远程进程的方法，通过消息传递参数和返回结果。其步骤包括生成消息、发送消息、接收消息和提取结果。在RPC中，客户过程调用客户存根，客户存根生成消息并调用本地操作系统，消息被发送到远程操作系统，远程操作系统将消息交给服务器存根，服务器存根提取参数并调用服务器，服务器执行操作并将结果返回给服务器存根，服务器存根将结果打包成消息并调用本地操作系统，消息被发送回客户端操作系统，客户端操作系统将消息交给客户存根，客户存根提取结果并返回给调用它的客户过程。

远程过程调用(Remote Procedure Call)RPC 是指本地程序调用位于其他机器上的进程，调用方通过消息 的形式把参数传递到被调用方的进程， 然后等待被调用方执行完后用消息的方式把结果传回调用方。 具 体步骤是：

(1)客户过程以正常的方式调用客户存根

(2) 客户存根生成一个消息，然后调用本地操作系统 (3)客户端操作系统将消息发送给远程操作系 统 (4)远程操作系统将消息交给服务器存根

(5)服务器存根将参数提取出来，然后调用服务器

(6) 服务器执行要求的操作，操作完成后将结果返回给服务器存根 (7) 服务器存根将结果打包成一 个消息，然后调用本地操作系统

(8) 服务器操作系统将含有结果的消息发送回客户端操作系统 (9)客户端操作系统将消息交给客户 存根

(10)客户存根将结果从消息中提取出来，返回给调用它的客户过程

141. 什么是远程对象调用？

远程对象调用指的是在本地调用位于其他机器上的对象。和远程过程调用主要的区别在于方法被调用的 方式。在远程对象调用中， 远程接口使每个远程方法都具有方法签名。如果一个方法在服务器上执行， 但是没有相匹配的签名被添加到这个远程接口上， 那么这个新方法就不能被远程对象调用的客户方所调 用。在远程过程调用中， 当一个请求到达远程过程调用的服务器时， 这个请求就包含了一个参数集和一 个文本值。

142. 消息同步通信与异步通信的区别？

异步通信特征在于发送者要把传输的消息提交之后立即执行其他的程序， 这意味着该消息存储在位于发 送端主机的本地缓冲区里中，或者存储在送达的第一个通信服务器上的缓冲区上中。 而对于同步通信 来说，发送者在提交信息之后会被阻塞直到消息已经到达并储存在接收主机的本地缓冲区中以后也就是 消息确实已经传到接收者之后，才会继续执行其他程序。

143. DNS 名称解析的方法有哪两种？各自优缺点

144. Lamport 时间戳算法的思想

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。

146. 全局状态

147. 删除无引用实体的方法

148. 设计一个分布式垃圾回收算法，简述其原理

引用计数(Reference Counting)

分布式标记清除算法

标记-清除(Mark-Sweep)

149. 分布式账本与区块链技术

下面的内容前面我都精简过了，看前面的版本就行了！

150. 选举算法中 Bully 算法的思想

选举算法：选择一个进程作为协调者、发起者或其他特殊角色，一般选择进程号最大的进程(假设每个 进程都知道其他进程的进程号，但不知道是否还在运行) 它的目的是保证在选举之行后，所有进程都认 可被选举的进程。 Bully 算法：

当进程 P 注意到需要选举一个进程作协调者时： (1)向所有进程号比它高的进程发 ELECTION 消息 (2) 如果得不到任何进程的响应，进程 P 获胜，成为协调者 (3)如果有进程号比它高的进程响应，该进程接管选举过程，进程 P 任务完成 (4)当其他进程都放弃，只 剩一个进程时，该进程成为协调者 (5)一个以前被中止的进程恢复后也有选举权

151. 选举算法中环算法的思想

不使用令牌，按进程号排序，每个进程都知道自己的后继者，当进程 P 注意到需要选举一个进程作协调 者时：

(1)创建一条包含该进程号的 ELECTION 消息，发给后继进程 (2)后继进程再将自己的进程号加入 ELECTION 消息，依次类推

(3)最后回到进程 P，它再发送一条 COORDINATOR 消息到环上，包含新选出的协调者进程(进程号最 大者)和所有在线进程，这消息在循环一周后被删除，随后每个进程都恢复原来的工作。

152. 分布式互斥算法

(1) 当进程想进入临界区时，它向所有其他进程发一条打了时间戳的消息 Request (2) 当收到所有其 他进程的 Reply 消息时，就可以进入临界区了 (3)当一个进程收到一条 Request 消息时，必须返回一 条 Reply 消息：

– 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

153. 给出Lamport 时间戳互斥算法的主要思想

参考答案：

Lamport 时间戳互斥算法由以下 5 条规则组成，为方便起见，认为每条规则定义的活动 形成单个事件：

①一个进程 P_i 如果为了申请资源，它向其它各个进程发送具有时间戳 T_m:P_i 的申请资源 的报文，并把此报文也放到自己的申请队列中；

②一个进程 P_j 如果收到具有时间戳 T_m:P_i 的申请资源的报文，它把此报文放到自己的申 请队列中，并将向 P_i 发送一个带有时间戳的承认报文。如果 P_j 正在临界区或正在发送自己 的申请报文，则此承认报文要等到 P_j 从临界区中退出之后或 P_j 发送完自己的申请报文之后 再发送，否则立即发送；

③一个进程 P_i 如果想释放资源，它先从自己的申请队列中删除对应的 T_m:P_i 申请报文，并向所有其他进程发送具有时间戳的 P_i 释放资源的报文；

④一个进程P_j 如果收到P_i 释放资源的报文，它从自己的申请队列中删除 T_m:P_i 申请报文；

⑤当满足下述两个条件时，申请资源的进程 P_i 获得资源：

(a) P_i 的申请队列中有 T_m :P_i 申请报文，并且根据时间戳它排在所有其它进程发来 的申请报文前面；

(b) P_i 收到所有其它进程的承认报文，其上面的时间戳值大于 T_m。

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。

154. 生产系统每天会产生一个日志文件 **F**，数据量在 **5000W** 行的级别。文件 **F** 保存了两列 数据，一列是来源渠道， 一列是来源渠道上的用户标识。文件 **F** 用来记录当日各渠道 上的所有访问用户， 每访问一次，记录一条。请问如何快速计算出各渠道上新增的用 户?

对于生产系统每天产生的日志文件 F，可使用以下方案快速计算出各渠道上新增的用户：

定义一个函数 f(来源渠道，用户标识)，将记录按照该函数的返回值分发到 N 个存储块中，尽量保证分发均匀。

将历史访问用户数据使用 f 函数分发至 N 个历史文件 H1、H2、…、HN 中。

将文件 F 中的内容使用 f 函数分发至文件 F1、F2、…、FN 中，可开多个并行处理，同时向同一文件写入数据的概率越小。

对文件 F1、F2、…、FN 内的访问记录进行去重，可开多个并行分别处理对应的 N 个文件。

将文件 Fn (1<=n<=N) 去重后的结果与对应的历史文件 Hn 比较得出新增用户结果 Rn，可开多个并行分别处理对应的 N 个文件。

合并第 5 步得到的结果 R1、R2、…、RN 即可得到当日新增用户，可并行处理。

为使历史数据文件 H1、H2、…、HN 中的数据最全，将结果 R1、R2、…、RN 分别写入对应的历史文件中，可并行处理。

该方案的优点是数据的分发、处理、合并都可并行处理，提高了处理效率，同时每个存储块上的新增数据只需要与它对应存储块上的历史数据比较即可，大大减少了比较次数，不需要考虑历史全量数据的保存及获取问题。缺点是处理方案比较复杂，需要多台服务器并行处理。难点在于需要一个稳定、快速、均匀的分发函数来将数据分发至不同的存储块。

155. 什么叫容错性？

容错意味着系统即使发生故障也能提供服务， 容错与可靠性相联系， 包含以下需求： 可用性(Availability)： 任何给定的时刻都能及时工作 可靠性(Reliability)： 系统可以无故障的持续运行

安全性(Safety)：系统偶然出现故障能正常操作而不会造成任何灾难 可维护性(Maintainability)：发生 故障的系统被恢复的难易程度

156. 掩盖故障的冗余有哪几种？

信息冗余：增加信息，如海明码校验。 时间冗余：多次执行一个动作，如事务。

物理冗余：增加额外的设备或进程。如三倍的模块冗余。

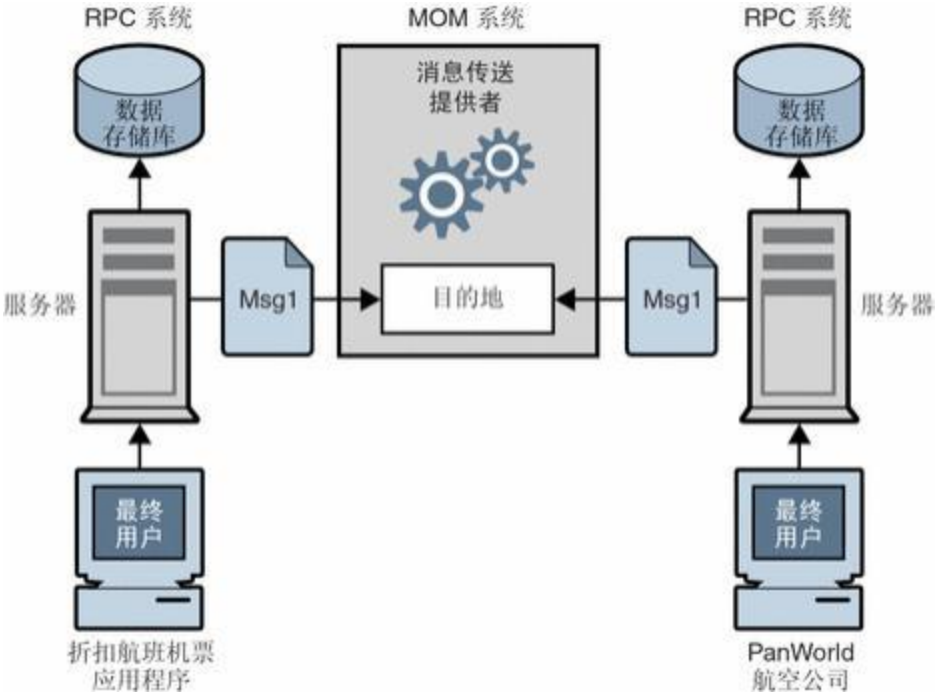
157. 拜占庭将军问题

158. NFS 的共享预约

159. 分布式提交—两阶段提交的思想？

160. 分布式提交—三阶段提交的思想？

161. 假如开发一个高吞吐的航空机票订单服务，需要手机端、网页端、以及其他第三方接口调用， 为了保证高吞吐，如何用 **RPC+**消息队列方式来实现， 说说你的设计思想和实 现思路



实现思路大致如上图，不同的客户端通过 rpc 提交订单请求， rpc 接到订单请求，封装消息发送给消息中间件。从而rpc 订单系统可以快速响应客户端，同时解耦订单系统与后台系统。后台系统以及第三方接口系统可以作为消息队列的消费者。

162. 现要为某网上商城实现一个商品价格查询服务，该服务具有以下功能：

用户可以主动查询某个商品的价格。

用户可以订购某个商品的价格，当商品价格低于用户指定的阈值时，该服务通知订购用户当前的价格。

多个用户可同时使用该服务。

现要使用面向对象的技术，如 CORBA 技术实现该服务：请描述该服务对象和客户端程序分别需要实现的接口。接口可以采用任何一种程序设计语言描述(甚至夹杂自然语言)，但要明确每个接口名、接口中的方法名、方法的返回值和参数名以及类型。

商品价格查询服务的接口：

方法一：价格查询。

Float getPrice (String goodID) throws someFailure；返回值为价格。

方法二：订购价格变化情况

Void subscribe(String goodID, float myInterestPrice, Ref myCallback) throws some someFilure
其中， myInterestPrice 为指定的价格阈值， myCallback 为实现 nicePrice () 方法的客户端回 调接口对象引用。

客户端实现的接口 方法一：

Void nicePrice(String goodID, float nicePrice) throws some someFilure; 其中，nicePrice 是低于阈值的新价格。

参数类型和名字等， 可在合理范围内变动。

163. 给定 **a**、**b** 两个文件，各存放 **50** 亿个 **url**，每个 **url** 各占**64** 字节，内存限制是**4G**，让 你找出 **a**、**b** 文件共同的 **url?**

假如每个 url 大小为 10bytes，那么可以估计每个文件的大小为 50G× 64=320G，远远大于内存限制的 4G，所以不可能将其完全加载到内存中处理， 可以采用分治的思想来解决。

Step1：遍历文件 a，对每个 url 求取 hash(url)%1000，然后根据所取得的 值将 url 分别存储到 1000 个小文件(记为 a0,a1,...,a999，每个小文件约 300M);

Step2:遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 个小文件(记 为 b0,b1,...,b999);

巧妙之处：这样处理后，所有可能相同的 url 都被保存在对应的小文件 (a0vsb0,a1vsb1,...,a999vsb999)中， 不对应的小文件不可能有相同的 url。然后 我们只要求出这个 1000 对小文件中相同的 url 即可。

Step3：求每对小文件 ai 和 bi 中相同的 url 时，可以把 ai 的 url 存储到 hash_set/hash_map 中。然后遍历 bi 的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

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