一、名词解释

3事务：能够使程序员将集中精力集中在处理算法和进程是如何运行的抽象。

4主机后备容错：在任何时刻都有一台服务器是主机，可以完成所有工作。当主服务器瘫痪时，后备服务器可以承担所有责任。

1分布式系统：多个独立计算机的集合，对于系统用户来说就像是一台计算机。从硬件角度，每台计算机都是自主的。从软件角度，用户将整个系统视为一台机器。

2组通信：在系统和用户的指定方式下，协同工作的多个进程的集合。当一个消息发送到该组后，组内所有进程都可以接收到这条消息。

3欺负算法：当进程p检测到协调者进程崩溃后，发起选举消息。如果有高号进程Q应答，那么P进程结束由Q进程接管选举。如果没有高号进程应答，难么P进程被选举为协调者。

二、简答

1.分布式系统相较于集中式系统优点/缺点，从长远看，推动分布式系统发展的动力

答：

相对于集中式系统，分布式系统的优点：

1. 从经济上来看，分布式系统比大型主机有较高的性价比。
2. 从算力上来看，分布式系统总的算力比大型主机要强大。
3. 从分布上看，分布式系统具有固有分布性，一个应用可能分散在多台主机上完成。
4. 从扩展性上看，分布式系统的计算能力可以主键增加。
5. 从可靠性上看，一台主机崩溃，整个系统仍然可以继续运行。

相对于集中式系统，分布式系统的缺点：

1. 软件问题，目前基于分布式系统的软件太少
2. 网络问题，一个系统一旦依赖网络，如果发生消息丢失或者饱和那么会抵消我们建立分布式系统的大部分优势

推动分布式系统发展的动力

1. 现在有大量的个人计算机
2. 基于多人协同工作的信息共享的需要，这种信息共享必须更方便的形式进行。而不受地理、人员、数据以及机器地理位置的影响。
3. RPC主要思想/RPC调用的步骤

基本思想：允许程序去调用其他机器上的过程。当机器A上的进程调用机器B上的过程时，机器A上的进程被挂起，被调用的过程在机器B上进行。机器A上的进程将消息放在参数表中发送给被调用者，被调用者将结果返回给调用者。

RPC的主要步骤：

1. 客户调用客户存根
2. 客户存根建立消息并激活内核陷阱
3. 客户内核将消息发送给远程内核
4. 远程内核将消息发送给服务器存根
5. 服务器存根取出消息中的参数后调用服务器的过程。
6. 服务器完成工作后将结果返回服务器存根
7. 服务器存根将结果打包并激活内核陷阱
8. 远程内核将消息发送给客户内核
9. 客户内核将消息传递给客户存根
10. 客户存根从消息中取出结果并返回给客户
11. 集中式算法、分布式算法、令牌环算法

集中式算法：进程进入临界区前向协调者发送临界区请求，协调者检查临界区。如果临界区没有被占用，那么协调者允许进程访问临界区。如果临界区正在被占用，那么协调者拒绝该进程并将进程放入等待队列。当临界区恢复空闲时，协调者从等待队列中取出最靠前的进程并发送应答消息。

分布式算法：进程进入临界区前，向组中的进程广播消息，该消息包括【临界区号】【进程号】【时间戳】。接受进程会有如下3中应答：

1. 接受进程不打算进入临界区，那么发送ok消息
2. 接受进程正在使用临界区，不必应答，并且将进程放入等待队列当中
3. 进程也准备使用临界区，那么对比两者的时间戳，如果发送消息的进程的时间戳小，那么发送应答消息；否则，接受者负责排列请求消息并不响应。

进程发送消息后，等待所有接受进程的消息。如果所有接受进程都返回ok消息那么该进程可以进入临界区。当从临界区退出时，向等待队列中的进程发送ok消息。

令牌环算法：通过软件对所有进程进行排列，所有进程都知道自己下一个位置的进程。初始化令牌，令牌从进程0开始传递，当进程获得令牌时检查临界区是否可以使用，如果可以使用那么进程进入临界区。从临界区中退出时，将令牌传递给临近的下一个位置的进程。不能使用同一个令牌进入第二个临界区 。

1. 文件共享四种语义 5

Unix语义：系统系统中的所有操作具有严格的时间顺序，并且总是返回最近的值，Read操作总是读取最后一个Write操作的内容，要求对一个任何操作全部进程都是及时可见的。

对话语义：文件在打开修改后，之后修改该进程的文件可见。在文件关闭后，文件的修改才可以被其他进程看见。

不可更改语义：文件不能被修改，只能共享和复制

事务语义：对于文件的操作要么一次性全部完成，要么都不能完成。

1. write through write once 6

Write-throug（通写缓存一致性）：通写缓存一致性协议是一种简单，通用的协议。当CPU从存储器中读取某个字时，该字通过总线从存储区中取出并存储在提出请求的CPU缓存中。方便下一次使用该字时，不再向存储器发送请求，而是直接中缓存中存取。

Write-once：基本思想：允许正在被多个CPU读取的字保存在他们的缓存中，但是被一个CPU经常写的字只保存在他的缓存中。

1. 严格一致性 顺序一致性 因果一致性 6

严格一致性：从存储器的地址X处取出的值是最新写入X的值。写操作在任意时刻对所有进程都是可见的，同时有一个绝对的全局时间，一旦存储器中的值改变，以后读取的都是更新的值。

顺序一致性：比严格一致性稍弱，如果所有进程以一定的顺序执行操作，则任何操作的结果都是一样的（所有进程看到的相同内存的操作顺序是一样的）。对于不同机器上的并行运行的进程，任何有效的交错都是可以接受的，但所有进程必须遵守同一访问顺序。进程可以看到所有进程写，只能看到本进程读。

因果一致性：先读再执行写，两个操作可能具有因果关系。

读和提供所读数据的写有因果关系，没有因果关系的操作称为并发。

**可能因果相关的写操作应对所有进程可见**，且顺序一致。并发写操作在不同机器看来顺序可能不一样

1. 哪些技术可以提高伸缩性

集群和负载均衡

第一章 分布式系统概述

1. 什么是分布式系统
2. 分布式系统与集中式系统相比优缺点？推动分布式发展的主要动力是什么？

优：经济、速度、前景、可靠性、分布性

缺：软件、网络（信息丢失和饱和）、安全

动力：大量个人计算机的存在、多人工作信息共享的需要（地理、人员、机器和数据物理分布的影响）

1. 分布式系统的分类和说明

是否共享存储器：多处理机系统、多计算机系统

连接方式：总线型、交换型

1. 分布式系统的特点

单一的、全局的进程通信机制

全局进程管理

全局文件系统

**统一的系统调用接口**

1. 分布式系统的设计问题

透明型（位置、迁移、复制、并发、并行）、灵活性、可靠性（可用性、安全性、容错性）、可伸缩性、**性能（吞吐率、实时性、并行性）**

1. 微内核提供的四个基本服务

进程间的通信机制

某些内存管理

少量低层进程调度和管理

低层输入输出服务

1. 分布式的通信
2. 客户-服务器模式思想和实现

设计一个操作系统，由为用户提供服务的一组协同进程组成。提供服务（服务器），申请服务（客户）。

- 运行在微内核中，以进程的方式运行

- 请求/应答协议

优点 非连接、协议栈短更有效

-寻址

机器号+进程号

广播定位寻址 发送者广播定位包

ASCII进程名

-阻塞

阻塞发送：消息发送时，发送消息的进程被挂起，直到消息发送完毕才继续执行后面的指令（异步式）

非阻塞发送：客户内核先将消息copy到缓冲区中，在消息发送前将控制权返回给客户进程，消息发送和客户进程可以同时运行

-缓冲

无缓冲：

-可靠性

1. 组通信

组：在系统、用户指定方式下协同工作的多个进程

消息发送该组，组中进程都收到

1. 什么是RPC？RPC的过程
2. 分布式系统的同步
3. 时钟同步

逻辑时钟：Lamport算法

物理时钟；集中式（cristians’算法 Berkeley算法）、分布式算法（平均值算法、多个外部时钟源）

1. 临界区同步

集中式（协调者允许）

分布式（广播消息）

令牌环算法

-选举算法

Bully（欺负算法）、环算法

1. 事务：允许编程人员将精力集中在算法和进程如何运行的抽象

**- 原子事务的特征（ACID）**

原子性：对外部事件来说，事务的发生是不可分割的

一致性：事务的发生并不会破坏系统的恒定

独立性：并发事务并不会互相影响

持久性：事务一旦提交，它的更新结果并不会因为系统的崩溃而丢失

-事务模型

通过稳定存储器实现：由两个磁盘来构成，驱动器2上的每一块都是驱动器1的精确拷贝，当更新一个块时，先检查更新驱动器1再检查更新驱动器2上的块。

-事务模型的实现

1. 私有空间

一个进程的事务在开始前，先分配一个私有空间，这个私有空间中包含着进程所需访问的所有文件块的地址索引。当需要读文件时，只需要根据索引进行正常的读取。当需要对某一个块进行写操作时，需要先对这个块进行拷贝并将地址也插入索引当中，这样就可以在不影响原始块的情况下更新这个块。添加块也是这样操作。

1. 写前日志

文件将被真正的修改，但是当块被修改前先将操作写入稳定存储器的日志

当事务终止时，则用日志备份初始状态，从日志末尾开始向回逐步读取日志，反做日志中描述的修改，回到初始状态。

1. 两阶段提交协议

发起事务的进程作为协调者，协调者先写入一份日志条目表明他要开始提交，向其他子进程发送“准备”消息，如果其他进程均回复可以提交那么事务就可以提交了。如果有进程不能提交，那么事务就必须终止。

1. 并发控制

**-乐观法**

读阶段：将文件读入私有工作区

确认阶段：提交前，检查是否在事务开始后，文件被其他事务更改。

-有，则废弃事务。重启

-无，则提交事务

写阶段：如可以提交，则将修改内容从私有工作区，写入文件

**- 时间戳法**：比较事务的时间戳与文件所带有的对他进行操作的最后一个提交的事务的读时间戳、写时间戳进行对比，如果当前事务时间戳大那么提交，如果小那么终止事务

1. 死锁问题

- 集中式检测死锁：假死锁问题（解决方法：协调者确认）

- 预防死锁（wait-die wound-wait）

1. 分布式进程管理
2. 引入线程的用途：将并行运行与顺序运行相结合
3. 分布式系统中处理机的组织方式：工作站（高性能个人终端电脑）模型和处理机池模型

如何定位空闲工作站？

- 服务器驱动

当一台工作站空闲时，就会成为一台潜在的服务器，工作站将（名字、网络地址和属性）写入到注册表中声明其可用性。当某用户希望使用某个空闲工作站时，会先查看注册表并请求一个空闲工作站，用户通过remote命令检查该工作站是否可用，如果可用那么这个工作站将自己从注册表中删除并打上正在运行记号。此时调用设射中环境并启动运行。

-客户端驱动

当用户调用remote时，它广播发送一个请求说明运行的程序、需要的内存和是否浮点运算等。每个空闲工作站在与自己负载成正比的时间延迟应答，这样负载最轻的工作站最先应答并被选中。

处理机池模型：在机柜中放满CPU，根据需要动态的分配给用户，用户拥有的是高性能图形终端而非工作站。用一个是小系统功能n倍的大系统来代替n个独立的小系统的资源，可把平均响应时间减少为原来的1/n

1. 处理机分配设计

-转发法：随机选择一台机器A，向它发送新进程。如果A已经过载，A也同样随机选择另一台机器B，向B发送进程。

-查询法：随机选择一台机器A，向它发送一个负载情况询问。如果欠载，A就会得到这个进程；否则，将重新选择一台机器。

-比较法：向k台机器发起询问，从k台机器中选择负载最轻的机器发送进程。

算法设计：

-层次式算法： 每层领导者负责监管其所管理的工作者的闲忙情况，并对分给每层的k个任务进行分配。

-发送者启发式算法：当创建进程时，创建继承的机器随机挑选一台机器询问负载是否低于某个阈值，如果是，将发送进程。否则，继续随机挑选一台机器进行询问。如果经过N次询问后没有找到空闲机器，那么算法将停止，新进程在创建它的机器上运行。

-接收者启发式算法：当一个进程结束时，结束进程的机器将检查自己是否有足够的工作要做。如果不是，那么该机器将随机的向其他的一台机器申请工作，如果这台机器没有给予工作，那么将继续询问。经过N轮询问后，将暂停申请，机器将执行系统队列中的等待进程。当这个进程结束后，机器将重新发起申请。如果系统无事可做，则进入空闲状态。一定时间后，再次发起询问。

协同调度：将在不同处理器上协同工作的进程放在不同处理器的相同的时间片中，以产生最小延迟。

1. 系统失效故障类型：fail-stop类型、拜占庭故障

fail-stop故障：失效的处理机只是停止运行，对于接下来的输入不进行任何反应，也不产生任何输出。

拜占庭故障：当系统出现错误时并不是停止运行，处理机继续运行产生错误答案。并可能与其他处理机一起“恶意”的工作，给人产生正常工作的假象。

1. 解决容错的方法有冗余技术，在物理冗余的实现有主动复制容错技术 和 主机后备容错技术

-主动复制容错技术：Fail-stop故障 需要k+1冗余度； 拜占庭错误需要2k+1 冗余度

多模冗余是如何进行容错的？

服务器就像一个有限状态机，接受请求给出应答。将每一个用户的请求发送给每一个服务器，服务器接受按照相同的顺序执行，那么处理完成后所有正确的服务器将处于同样的状态，并将给出同样的结果。客户端和表决器可以综合这些结果屏蔽掉错误。

-**主机后备技术**

基本思想，并结合例子进行具体描述？

基本思想是任何一台服务器都可以成为主机，完成所有任务。当主机失效时，后备服务器将承担起全部责任。

（写操作的简单主备份协议）当客户向主服务器发起请求时，主服务器接受消息并处理完成后给备份服务器发送更新消息。备份服务器处理完成后给主服务器发送确认消息，主服务器接收到消息后，再给客户机发送应答消息。

容错系统的协同一致问题：若要有m个处理机出错的系统实现协同一致，最少要有2m＋1个正常处理机。处理机总数为3m+1

第五章 分布式文件管理

1. 文件共享语义

Unix语义

对话语义

不可更改语义

事务语义

1. 文件服务的两种类型

-上传/下载模型

只提供两种操作：读操作和写操作。文件系统将文件服务器上的文件传送给发起请求的客户，将客户或修改的文件从客户端传送到文件服务器上。文件系统运行在客户端

-远程访问模型

文件服务器只提供物理存储和传送的功能。文件系统运行在服务器端而不是客户端。客户端不需要很大的空间，当客户只访问一小部分时，服务器不需要传送整个文件。

第六章 分布式共享存储器

1. 缓存一致性，以及两个缓存一致性协议

- 缓存一致性指的是，不同缓存对于同一存储器的同一位置具有相同的值

对于缓存一致性的研究具有两个协议 通写（write-through）与 write-once 协议

- 通写缓存一致性协议：当CPU首次读取某个字时，CPU将这个字存储到自己的缓存当中，下一次读取时不再向存储器发起请求而是直接从自己的缓存中读取。

- write-once协议：当多个CPU读取某个字时，每个CPU的缓存中都存储着这个字的拷贝。但是当某个字被一个CPU多次写时，只有对这个字读写的CPU的缓存中拥有这个字。（P217例）

2. 一致性模型

- 严格一致性：**（只能读到最后一次的写）**

存储器位置X上的值是最近一次写入的X的值，严格一致性要求所有进程拥有绝对的全局时间。并且每一个进程的写操作是对所有进程可见的。

- 顺序一致性：**（要按照写的顺序读）**

满足顺序一致性的存储器应该满足条件：如果所有进程以一定的顺序执行，每一个进程的操作都按照程序规定的顺序执行，那么任何操作得到的结果是一样的。即**多个进程的交错是可以接受的，但是要求所有进程必须按照同一顺序访问存储器**。保证所有进程都按照相同的顺序看到存储器的访问。写操作是可以被所有进程看到，每个进程只能看到自己的读操作。

- 因果一致性：**（因果关系的写要顺序，并发的可以不顺序）**

可能具有因果关系的写操作对于所有进程可见，并且顺序一致。并发的写操作对于不同的处理机可以看到不同的顺序。

先有读操作之后有写操作那么两个操作具有因果关系，读操作与提供数据的写操作同样具有因果关系。没有因果关系的称为并发。

- PRAM一致性：**（不同进程的写操作是并发的）**

同一个进程的写操作必须以指定顺序被其他进程看到，不同进程的写操作在不同进程看来顺序可以不同。

- 处理机顺序一致性：**（位置相同要顺序，位置不同可并发）**

对于存储器任意位置x，写入x的顺序要有全局约定。写入不同的地址，对于不同的进程可能看到不同的顺序。

- 弱一致性：

对同步变量的访问必须是顺序一致性的

在所有前面的写操作完成之前,不能访问同步变量

在前面所有同步变量的访问完成前,不能访问（读或写）数据