第一章分布式系统概述

1.1 什么是分布式系统?

分布式系统是若干独立计算机的集合，它们对于用户来说就像一个系统。

1.2 分布式系统中透明性的种类、定义。

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。

1.3 分布式系统中的扩展技术有哪些？

(1)隐藏通信等待时间:包括异步通信和减少通信量

(2)分布技术：即分割组件，然后分散到系统中，例如 DNS 和 WWW

(3)复制技术: 多拷贝

1.4 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)

第二章体系结构

2.1 四种体系结构样式。

分层体系结构（Layered architectures）（网络通信广泛应用）

基于对象的体系结构（Object-based architectures）（特点：松散的组织结构；通过远程过程调用进行通信）

以数据为中心的体系结构（Data-centered architectures）

基于事件的体系结构（Event-based architectures）（优点：进程松散耦合）

2.2 客户端-服务器模型。

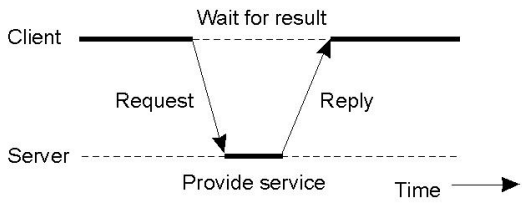
服务器（server）：实现某个特定服务的进程

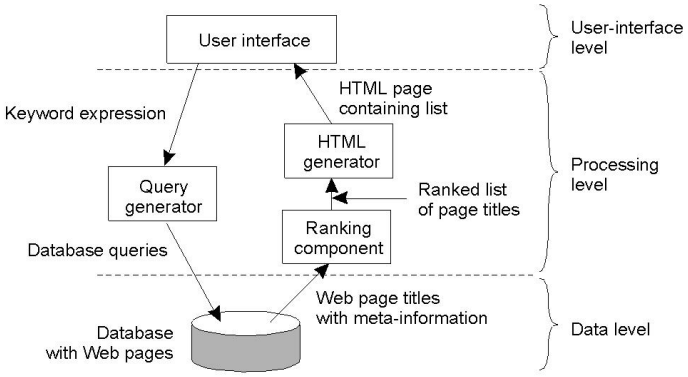
客户（client）：向服务器请求服务的进程

客户端-服务器之间的一般交互：请求/回复（如下左图）

基于无连接协议的客户和服务器通信：高效，但是易受传输故障的影响（无法检测消息是否丢失也无法解释是否发生传输故障）。适合局域网。

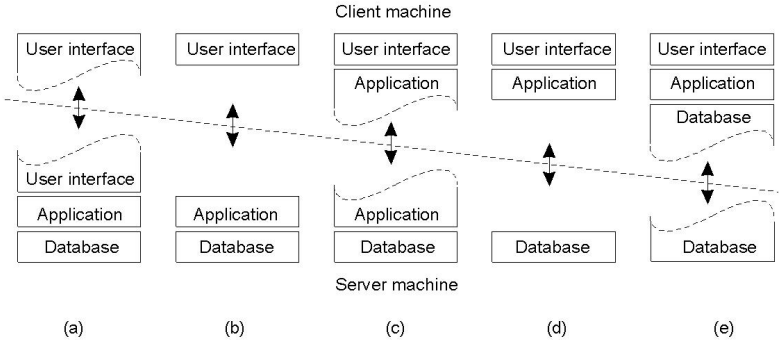
基于连接的协议：性能相对较低，不适合局域网，适合广域网（基于可靠的 TCP/IP）。





客户服务器应用程序通常组织为三个层次(如上右图)：(1)用户界面层：含有直接与用户交互所需的一切；(2)处理层：含有应用程序核心功能；(3)数据层：操作数据或文件系统，保持不同应用程序之间的数据一致性。

客户端-服务器模型可能的组织结构如下图：



(a)只有与终端有关的用户接口部分位于客户机器上；(b)把整个用户接口软件放在客户端
(c)部分应用程序移到前端；(d)大多数的应用程序基本是运行在客户机上，但所有对文件或数据库项目的操作都是在服务器上；(e)同(d)，本地硬盘含有部分数据。

第三章分布式进程管理

3.1 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。

3.2 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。

3.3 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。

3.4 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。

3.5 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。

如下图：掌握迁移代码时，根据引用本地资源方式不同应采取的做法。

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

3.6 处理机分配的超载者启动的分布式启发式算法思想。

算法描述：

当一个进程创建时，若创建该进程的机器发现自己超载，就将询问消息发送给一个随机选择的机器，询问该机器的负载是否低于一个阀值。

1）如果是，那么该进程就被传送到该机器上去运行。

2）否则，就再随机地选择一台机器进行询问。

这个过程最多执行 N 次，若仍然找不到一台合适的机器，那么算法将终止，新创建的进程就在创建它的机器上运行。

算法分析：

当整个系统负载很重的时候，每一个机器都不断地向其他机器发送询问消息以便找到一台机器愿意接收外来的工作。在这种情况下，所有机器的负载都很重，没有一台机器能够接收其它机器的工作，所以，大量的询问消息不仅毫无意义，而且还给系统增添了巨大的额外开销。

3.7 处理机分配的欠载者启动的分布式启发式算法思想。

算法描述：

在这个算法中，当一个进程结束时，系统就检查自己是否欠载。如果是，它就随机地向一台机器发送询问消息。如果被询问的机器也欠载，则再随机地向第二台、第三台机器发送询问消息。如果连续 N 个询问之后仍然没有找到超载的机器，就暂时停止询问的发送，开始处理本地进程就绪队列中的一个等待进程，处理完毕后，再开始新一轮的询问。如果既没有本地工作也没有外来的工作，这台机器就进入空闲状态。在一定的时间间隔以后，它又开始随机地询问远程机器。

算法分析：

在欠载者启动的分布式启发式算法中，当系统繁忙时，一台机器欠载的可能性很小。即使有机器欠载，它也能很快地找到外来的工作。在系统几乎无事可做时，算法会让每一台空闲机器都不间断地发送询问消息
去寻找其它超载机器上的工作，造成大量的系统额外开销。但是，在系统欠载时产生大量额外开销要比在系统过载时产生大量额外开销好得多。

第四章分布式系统通信

4.1 什么是远程过程调用？远程过程调用的步骤。

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

4.2 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。

4.3 消息持久通信与暂时通信的区别？

消息持久通信指的是，需要传输的消息在提交之后由通信系统来储存，直到将其交付给接受者为止，在将消息成功交付给下一个服务器之前消息一直储存在通信服务器上，因此发送消息的程序不必在发送消息后保持运行，同样要接受消息的应用程序在消息提交的时候可以不处于运行状态。即，不需要消息发送方和接收方在消息的传输过程中都保持激活状态。提供消息的中介存储(如，消息队列系统，面向消息的中间件)，实时性要求较低，允许几分钟完成的传输。

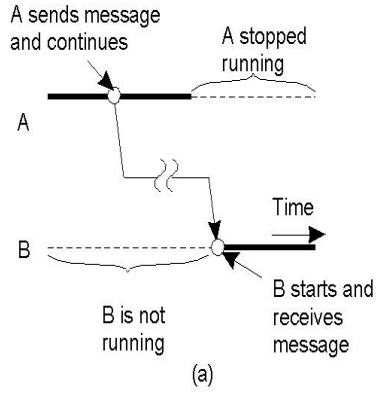
消息暂时通信指的是通信系统只是在发送和接收消息的应用程序运行期间存储消息，否则消息就会被丢弃。不提供消息的中介存储，实时性要求较高，几秒甚至几毫秒完成。如 Berkeley Sockets(套接字)，Message-Passing Interface 消息传输接口。

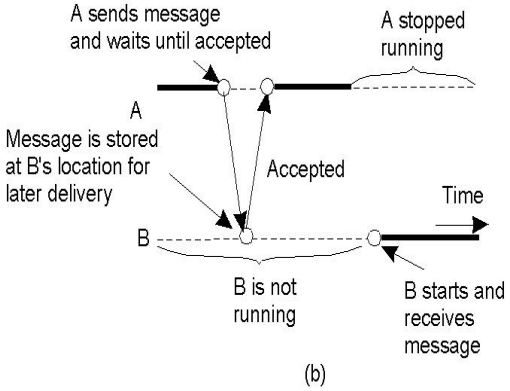
4.4 消息同步通信与异步通信的区别？

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。

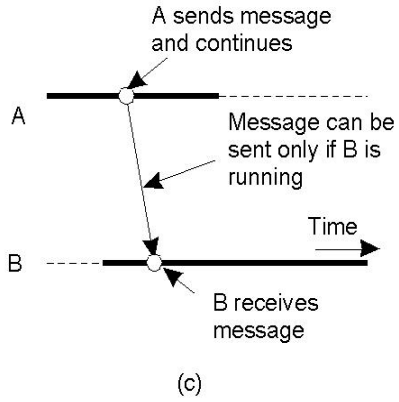
而对于同步通信来说，发送者在提交信息之后会被阻塞直到消息已经到达并储存在接收主机的本地缓冲区中以后也就是消息确实已经传到接收者之后，才会继续执行其他程序。

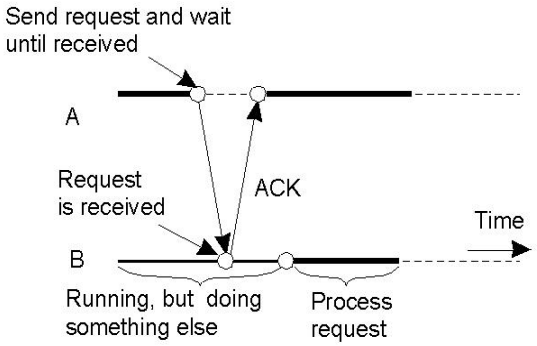
4.5 给出示意图，能够判断消息通信的类型



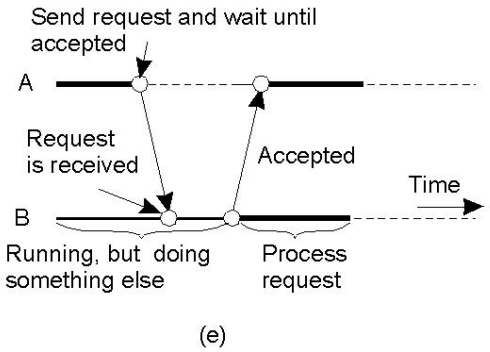


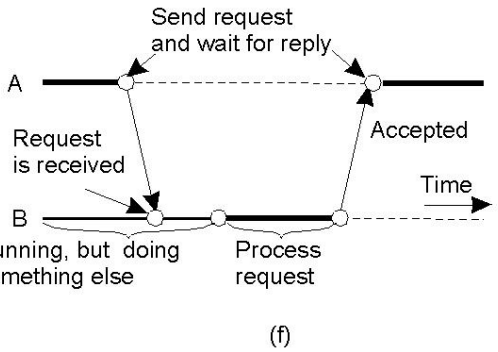
a)持久异步通信　　b)持久同步通信





c)暂时异步通信　　d)基于接收的暂时同步通信





e)基于交付的暂时同步通信　　f)基于响应的暂时同步通信

4.6 多播通信：反熵和 gossiping。（2009）

多播：服务器向其他 N 台服务器发送更新时，底层的网络负责向多个接收者发送一个消息，高效

Epidemic 协议使用本地信息在大型节点集中快速地传播信息。

提供最终一致性：保证所有的副本最终是一致的

一个服务器可以是：

传染性的：持有愿意向其他服务器散布的更新

易感的：尚未更新的服务器

隔离的：已更新的服务器如果不愿意或不能扩散其更新

反熵传播模型：服务器 P 周期的随机选取一台服务器 Q 交换更新，方式包括：

P 只把自己的更新推入 Q：较差的选择

P 只从 Q 拉出新的更新

P 和 Q 相互发送更新

可以证明：如果初始只有一台服务器具有传染性，无论采用哪种形式，更新最终将被传播到所有服务器上。

Gossiping 思想：

如果服务器 P 刚刚因为数据项 x 而被更新，那么它联系任意一个其他服务器 Q，并试图将更新推入 Q。

如果 Q 已经被其他服务器更新了， P 可能会失去继续扩散的兴趣，变成隔离的（这种可能性是 1/k）

评价：快速传播更新的方法；但不能保证所有的服务器都被更新了。

$s=e^{-(k+1)(1-s)}$,k=3 时，s 小于 2%；k=4 时，s 小于 0.7%

第五章命名

5.1DNS 名称解析的方法有哪两种？各自优缺点？

(1)迭代名称解析：每个服务器只能解析自己下面的路径服务器，把服务器的地址给解析程序，然后解析程序继续访问下一个服务器一直到实体。

优点：名称服务器负担小

缺点：通信开销大

处理过程如下图：解析 root::<nl,vu,cs,ftp,pub,globe,intex.txt>,URL 标示对应于 ftp.cs.vu.nl/pub/globe/index.txt 将该待解析的地址 root::<nl,vu,cs,ftp>输入到名称解析程序。

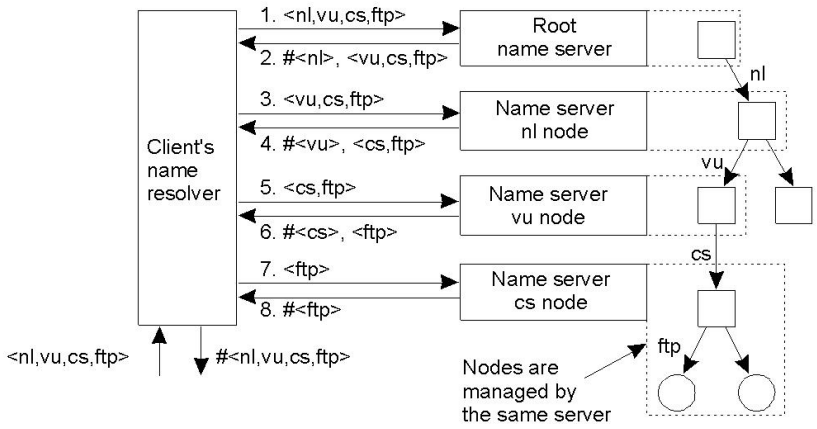
首先，名称解析程序把完整的名称(root::<nl,vu,cs,ftp,pub,globe,intex.txt>)转发给根名称服务器。根服务器会尽量深入解析路径名，然后把结果返回给客户。图中服务器只能解析标示符 nl，然后根服务器将返回与 nl 相关的名称服务器所用的地址。(#<nl>表示 nl 节点地址。)

然后，客户把剩下的路径名(即：nl::<vu,cs,ftp,pub,globe,intex.txt>)发送给一台名称服务器(nl 节结点)。而该服务器只能解析标示符 vu，然后返回相关服务器名称地址，同时剩下路径名 vu::<cs,ftp,pub,globe,intex.txt>。

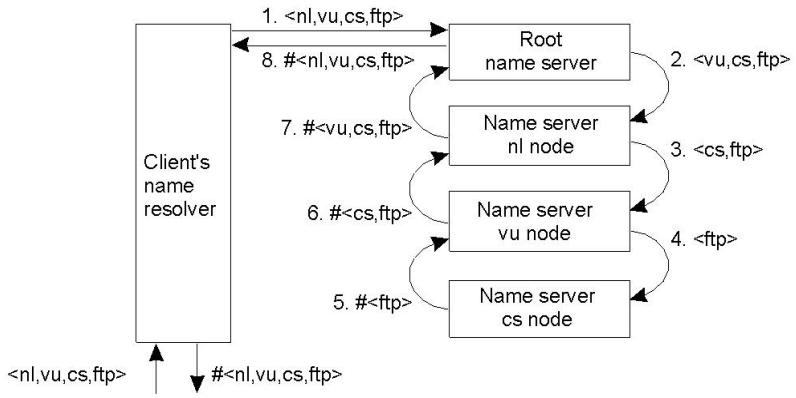
以下，客户名称解析程序继续同下面的名称服务器联系...

最后一步，名称解析程序找到由 vu 节点返回的 cs 服务器地址，cs 服务器解析到 ftp 地址。将其返回给名称解析程序。名称解析程序将该地址输出，即可以输出地址#<nl,vu,cs,ftp>。至此，名称解析程序完成工作。

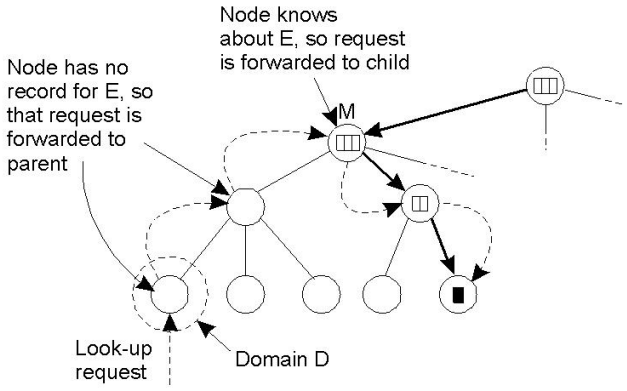
(得到 ftp 服务器请求发送路径名所指的文件，由客户进程独立完成。



(2)递归名称解析：在每一台服务器都使用递归解析。
优点：缓存效果更有效；减少通信开销
缺点：要求名称服务器有较高性能
如下图解析 root::<nl,vu,cs,ftp,pub,globe,intex.txt>。不同于迭代名称解析，每次名称服务器找到下一台服务器会把结果传给下一台服务器。如，根名称服务器找到使用 nl 节点的名称服务器所用的地址后，会请求该名称服务器解析路径名 nl::<vu,cs,ftp,pub,globe,intex.txt>，依次向下解析，最终会把 ftp 所在地址依次返回，直至传递给客户的名称解析程序。
例：



5.2 移动实体定位的方法有哪些？
(1)广播和多播
广播：(可提供多播的网络)主机将包含该实体标示符的消息广播到每台机器上，并且请求每台机器查看是否拥有该实体。只有能够为该实体提供访问点的机器才会发送回复消息，回复消息中包含访问点的地址。
多播：(点对点网络使用)主机向一个多播地址发送消息，会发送给该多播组中的所有成员。
(2)转发指针：当实体从 A 移动到 B 时，它将在后面留下一个指针，这个指针指向它在 B 中的新位置。一旦查找到实体，客户可顺着转发指针形成的链来查找实体的当前地址。
(3)基于起始位置的方法：客户必须与起始位置联系，起始位置返回客户所要查找的主机的地址。(4)分布式散列表。
(5)分层方法：见 5.3。
5.3 描述分层方法中查找一实体的过程
（1）希望定位实体 E 的客户向它所在的叶域 D 的目录节点发送了一个查找请求
（2）如果叶域 D 的目录节点中没有存储该实体的位置记录，那么就说明该实体现在不在 D 中。因此这个节点会把请求转发给它的父节点。
（3）如果父节点也没有 E 的位置记录，那么就会把查找请求转发给更高一层的域，依次类推
（4）如果节点 M 存储了 E 的位置记录，那么一旦请求到达 M 后，就可以知道 E 位于节点 M 代表的域中，M 存储了一条位置记录，其中包含了一个指向其子域的指针
（5）然后 M 就把请求转发给那个子域的目录节点，那个子域会依次向树的下方转发请求，直到请求最终到达叶节点为止。存储在叶节点的位置记录会包含 E 在哪一个叶域中的地址。
（6）将该地址返回给发送请求的客户。



图：在分层组织的定位服务中的位置查找

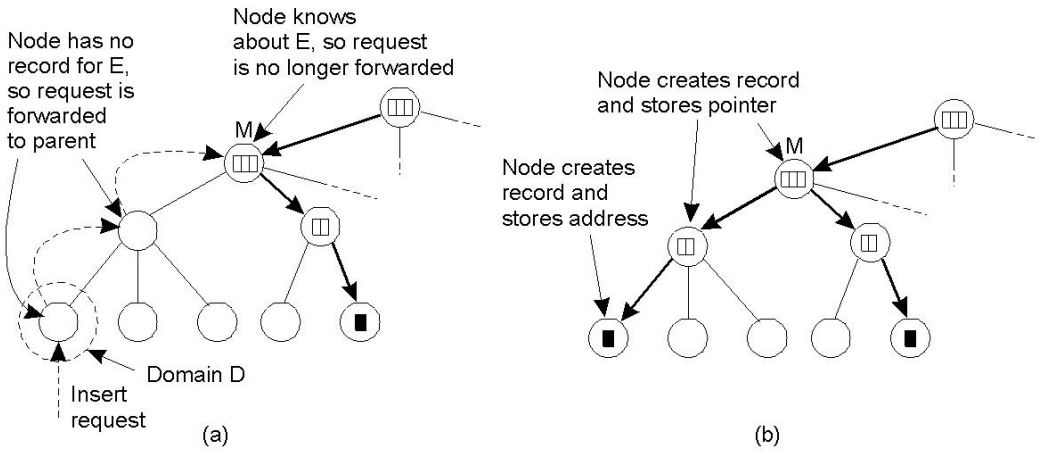
5.4 描述分层方法中插入一实体的过程

（1）实体 E 在叶域 D 中创建了一个复制实体，需要在这个复制实体中插入 E 的地址。插入操作从 D 的叶节点开始，然后 D 会立即把插入请求转发给它的父节点。

（2）父节点也转发插入请求，直到插入请求到达已经为 E 存储了位置记录的目录节点 M 为止。

（3）节点 M 在 E 的位置记录中存储了一个指针，这个指针指向转发插入请求的那个子节点，该子节点会建立一条关于 E 的位置记录，这条位置记录中包含一个指针，该指针指向转发请求的下一层节点。这个过程会连续进行，直到到达发起请求的叶节点为止

（4）最后，那个叶节点会建立一条记录，这条记录包含实体在关联叶域中的位置。



图：更新操作

- 插入请求被转发到第一个知道实体 E 的节点
- 转发指向叶节点的指针所形成的链

第六章 同步

6.1 Lamport 时间戳算法的思想

- 网络上的每个系统（站点）维护一个计数器，起时钟的作用
- 每个站点有一个数字型标识，消息的格式为（m，Ti，i），m 为消息内容，Ti 为时间戳，i 为站点标识
- 当系统发送消息时，将时钟加一
- 当系统接收消息时，将它的时钟设为当前值和到达的时间戳这两者的最大者加一
- 在每个站点，时间的排序遵循以下规则

---对来自站点 i 的消息 x 和站点 j 的消息 y，如果

- Ti<Tj 或
- Ti=Tj，且 i<j

---则说消息 x 早于消息 y

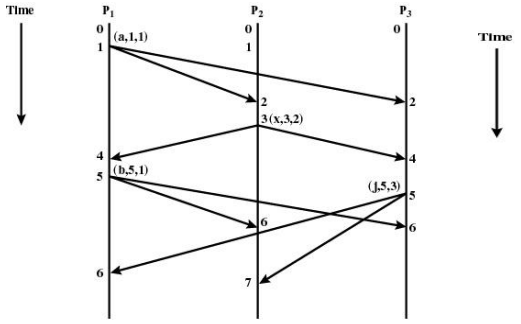


Figure 14.8 Example of Operation of Timestamping Algorithm

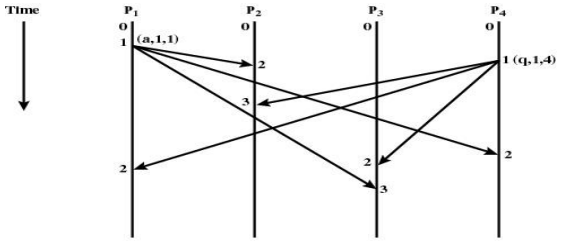


Figure 14.9 Another Example of Operation of Timestamping Algorithm

6.3 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：

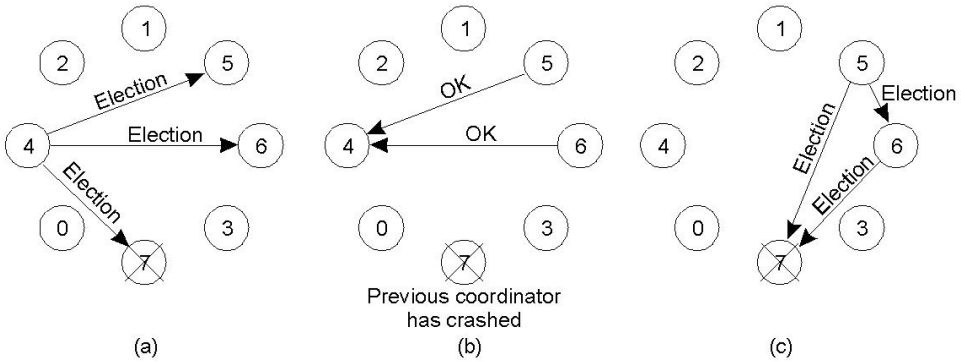
(1)向所有进程号比它高的进程发 ELECTION 消息

(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

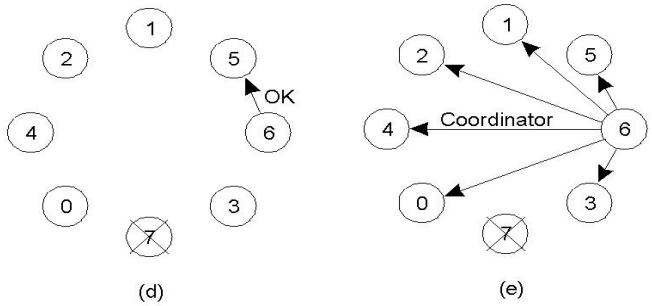
(4)当其他进程都放弃，只剩一个进程时，该进程成为协调者

(5)一个以前被中止的进程恢复后也有选举权



进程 4 启动选举

进程 5 和进程 6 响应，接管选举，成为协调者



进程 6 响应进程 5 的消息，接管选举，进程 6 成为协调者，通知所有进程

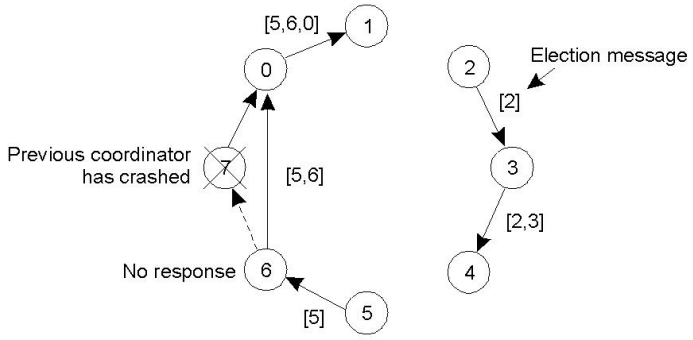
6.4 选举算法中环算法的思想

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。



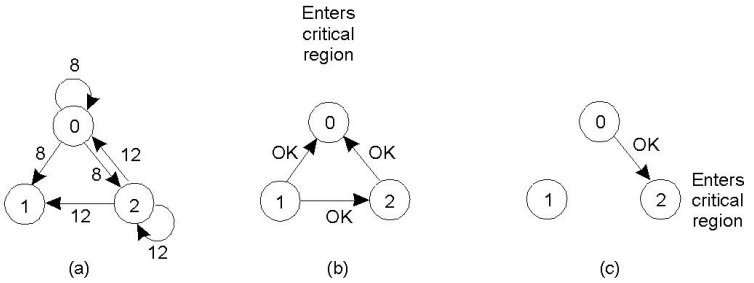
6.6 分布式互斥算法

（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



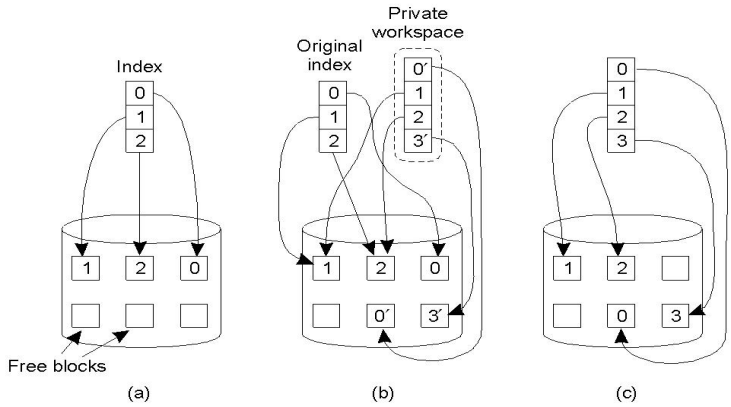
a） 进程 0 和 2 都想进入临界区

b） 进程 0 的时间戳低，抢先进入临界区

c） 进程 0 退出临界区后，发应答给进程 2，进程 2 随后进入临界区

6.7 实现事物的方法

（1）**私有工作空间**：为进程提供一个私有工作空间，包含进程有权访问的所有对象。进程的读写操作在私有工作空间进行，而不对实际的文件系统进行。如果事务中止，私有工作空间被释放，指向的私有块被删除。如果事务提交，私有索引被移到父辈空间，不再被访问的块被释放掉。



a）包含三个块的文件及其索引　b）块 0 被修改，块 3 被添加后的情况　c）事务提交之后

（2）**写前日志**（writeahead log）：先写日志，再做实际修改。

日志内容：哪个事务在对文件进行修改，哪个文件和数据被改动，新值和旧值是什么等。日志写入后，改动才被写入文件。若事务中止，则使用写前日志回退到原来的状态。

借助稳定存储器中的写前日志：当系统崩溃后，完成事务或取消事务

x = 0;
y = 0;
BEGIN_TRANSACTION;
  x = x + 1;
  y = y + 2
  x = y * y;
END_TRANSACTION;
(a)
	Log
	
	[x = 0/1]
	
	(b)
		Log
		
		[x = 0/1]
		[y = 0/2]
		
		(c)
			Log
			
			[x = 0/1]
			[y = 0/2]
			[x = 1/4]
			
			(d)

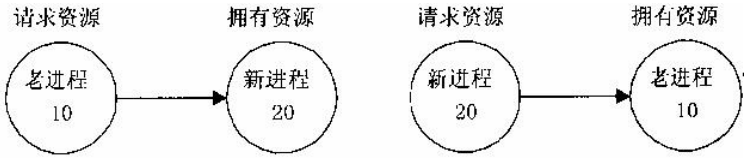
a) 一个事务 b) - d) 语句执行前的日志

6.8 死锁预防的等-死算法和伤-等算法

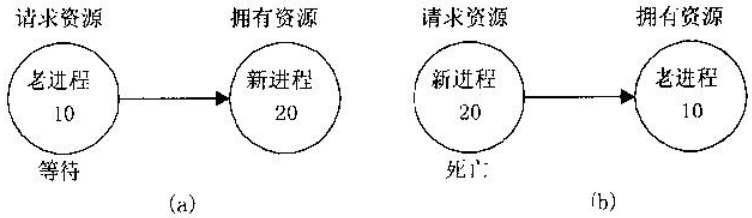
（1）等-死算法：由于使用了时间戳，当请求被占用的资源时只可能有两种情况：

- 老进程请求被新进程占用的资源，
- 或者，新进程请求被老进程占用的资源

一种情况应该允许进程等待，另一种情况应该中止进程。也即等-死算法



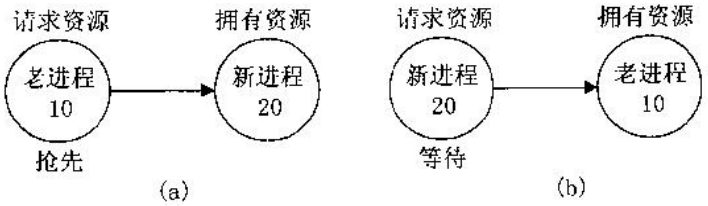
假设标记 a 为中止，b 为等待。那么就应该中止掉老进程，但这样的效率较低（老进程只会变得更老）。
所以应该以相反的方式进行标记，如下图



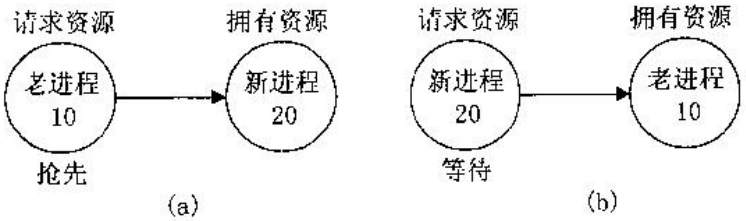
在这种情况下，箭头总是指向事务编号增长的方向，使得环路不可能出现。

（2）伤-等算法：允许抢占：假设只允许老进程抢占新进程，

因此图 a 被标记为抢先，图 b 为等待。



这种算法称为伤-等算法（wound-wait），因为一个事务可能会受到伤害（实际是被中止）而其他的事务等待。若一个老进程希望得到一个被新进程占用的资源，那么老进程将会抢占，于是新事务被中止，如 a 所示。随后新事务可能会立即重新开始，并试着请求资源，如 b 所示，然后被迫等待。



- 等-死算法中，

  若一个老事务想得到一个正被新事务占用的资源，那么它会很礼貌的等待。

  反之，若一个新事务想得到一个被老事务占用的资源，它将被中止。尽管它还会重新开始，但很可能又会立即被中止。在老事务释放资源之前，这个循环可能要重复多次。

• 伤-等算法没有这么糟糕的特性。

6.10 分布式的死锁检测 Chandy-Misra-Haas 算法的思想？

Chandy-Misra-Haas 算法允许进程一次请求多个资源（如锁）而不是一次一个。通过允许多个请求同时进行使得事务的增长阶段加速。这使得一个进程可以同时等待两个或多个进程。

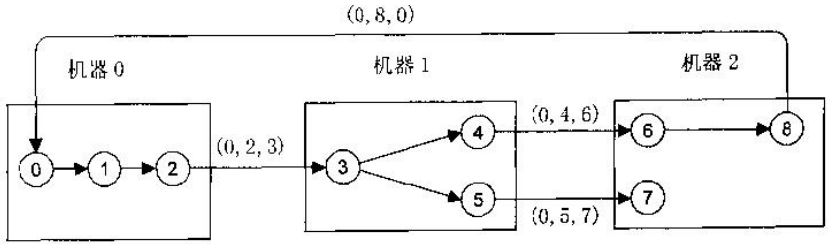


图 3-23 Chandy-Misra-Haas 分布式死锁检测算法

机器 1 上的进程 3 正在等待两个资源，一个由进程 4 占有，一个由进程 5 占有。一些进程正在等待本地资源，例如进程 1。一些进程，如进程 2 在等待其他机器上的资源。

当某个进程等待资源时，例如 P0 等待 P1，将调用 Chandy-Misra-Haas 算法。生成一个探测消息并发送给占用资源的进程。消息由三个数字构成：阻塞的进程，发送消息的进程，接受消息的进程。由 P0 到 P1 的初始消息包含三元组（0，0，1）。消息到达后，接受者检查以确认它自己是否也在等待其他进程。若是，就更新消息，字段 1 保持不变，字段 2 改成当前进程号，字段 3 改为等待的进程号。然后消息接着被发送到等待的进程。若存在多个等待进程，就要发送多个不同的消息。

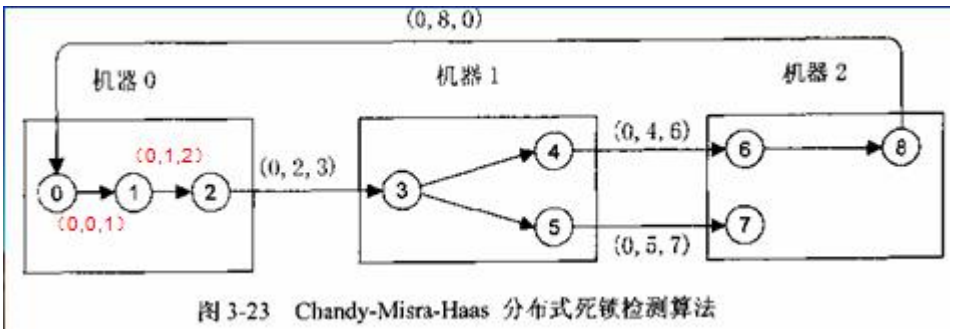


图 3-23 Chandy-Misra-Haas 分布式死锁检测算法

不论资源在本地还是在远程，该算法一直继续下去。图中（0，2，3），（0，4，6），（0，5，7）和（0，8，0）都是远程消息。若消息转了一圈后又回到最初的发送者，即字段 1 所列的进程，就说明存在一个有死锁的环路系统。

第七章一致性和复制

7.1 复制的目的和代价。

目的：提高可靠性和提高性能。

代价：(1)引起服务器数量扩展以及地理区域扩展，那么在一致性上的代价就高了。

(2)网络通信开销。

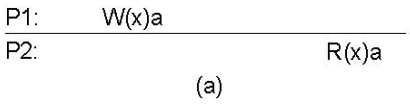
(3)强一致性导致要求的原子操作很难快速完成。

(解决办法：放宽一致性方面的限制，放宽程度取决于复制数据的访问和更新模式以及数据的用途。）

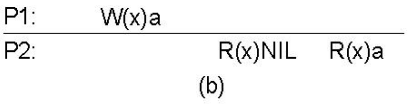
7.2 能区分是否符合严格一致性、顺序一致性、因果一致性、FIFO 一致性、弱一致性、释放一致性、入口一致性。/*原题只有严格一致性、顺序一致性、因果一致性和 FIFO 一致性，其他选看*/

共享数据读操作和写操作时的一致性问题：一致性模型实质上是进程和数据存储间的约定，如果进程同意遵守某些规则，数据存储将正常进行。正常情况下，进程的读操作应该返回最后一次写操作的结果。没有全局时钟，精确定义哪次写操作是最后一次写操作是困难的。作为全局时钟的替代，产生了一系列一致性模型，每种模型都有效地限制了一个数据项上执行一次读操作所应返回的值。

<1>严格一致性：任何对数据项 X 的读操作将返回最近一次对 X 进行写操作的值。对所有进程来说，所有写操作都是瞬间可见的，系统维护着一个绝对的全局时间顺序。



(a)



(b)

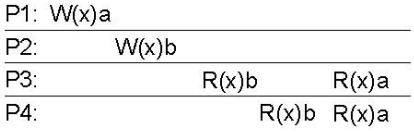
a）严格的一致性存储　　b）非严格的一致性存储()

<2>线性化和顺序一致性：

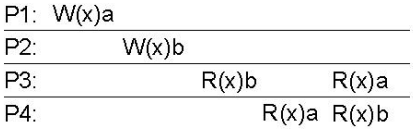
顺序一致性对存储器的限制比严格一致性要弱一些，要满足以下的条件：

（1）每个进程的内部操作顺序是确定不变的；

（2）假如所有的进程都对某一个存储单元执行操作，那么，它们的操作顺序是确定的，即任一进程都可以感知到这些进程同样的操作顺序。



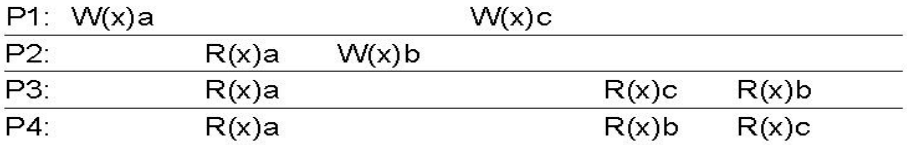
(a)



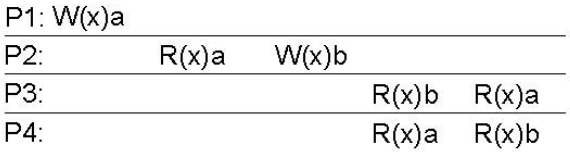
(b)

a）顺序一致的数据存储　　b）非顺序一致的数据存储(P3P4 操作顺序不同)

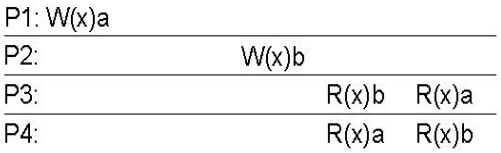
<3>因果一致性：所有进程必须以相同的顺序看到具有潜在因果关系的写操作，不同机器上的进程可以以不同的顺序看到并发的写操作。当一个读操作后面跟着一个写操作的时候，这两个事件有潜在的因果关系，同样，读操作也作为为读操作提供数据的写操作因果相关。没有因果关系的操作可以看作是并发的，并发的操作的顺序是不考虑的。



上图是因果一致性存储允许的，但顺序和严格一致性存储不允许的顺序



(a)

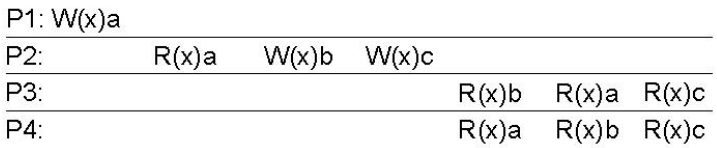


(b)

a)违背因果一致性的时间存储顺序　　b)符合因果一致性的时间存储顺序

<4>FIFO 一致性模型：是在因果一致性模型上的进一步弱化，它满足下面的条件：

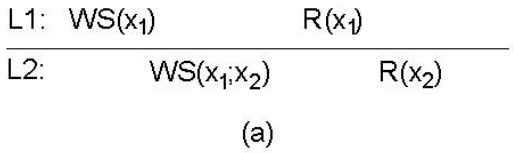
由某一个进程完成的写操作可以被其他所有的进程按照顺序感知到，而从不同进程中来的写操作对不同的进程可以有不同的顺序。一致性表现在要求任何位置都可以按顺序看到某个单一进程的写操作。



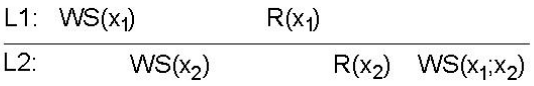
7.3 能区分是否符合单调读、单调写、写后读和读后写。

单调读：如果一个进程读取数据项 x 的值，那么该进程对 x 执行的任何后续读操作将总是得到第一次读取的那个

值或更新的值。
进程 P 对同一数据存储的两个不同本地备份执行的读操作

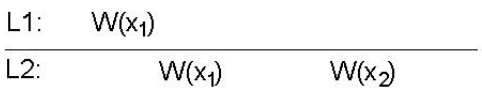


(a)



(b)

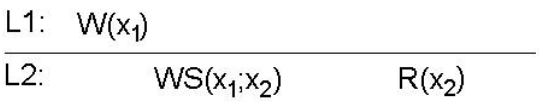
a）提供单调读一致性的数据存储　　b）不提供单调读一致性的数据存储.
单调写：如果一个进程对数据项 x 执行的写操作必须在该进程对 x 执行的任何后续写操作之前完成。
进程 P 对同一数据存储的两个不同本地备份执行的写操作，类似以数据为中心的 FIFO 一致性



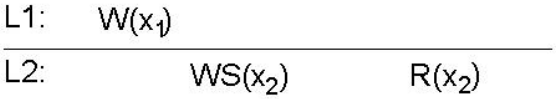
(a)

(b)

a）提供单调写一致性的数据存储　　b）不提供单调写一致性的数据存储
写后读：一个进程对数据项 x 执行的写操作结果总会被该进程对 x 执行的任何后续读操作看见。这个一致性保证，写操作的结果可以被所有的后续读操作看到。

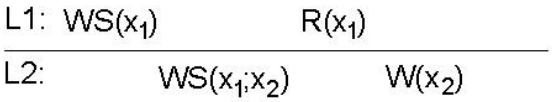


(a)

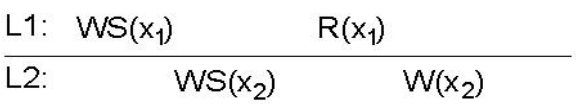


(b)

a）提供写后读一致性的数据存储　　b）不提供写后读一致性的数据存储
读后写：同一个进程对数据项 x 执行的读操作之后的写操作，保证发生在与 x 读取之相同或更新的值上。也就是说，进程对数据项 x 所执行的任何后续的写操作都会在 x 的拷贝上执行，而该拷贝是用该进程最近读取的值更新的。读后写一致性用于保证，网络新闻组的用户只有在已经看到原文章之后才能看到他的回应文章。



(a)

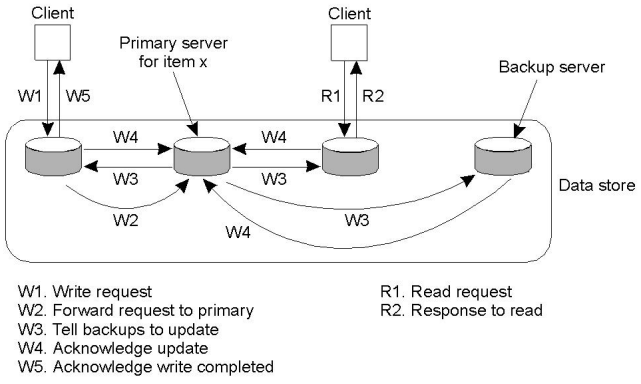


(b)

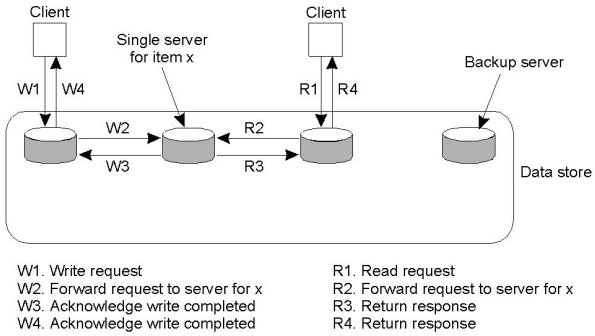
a）提供读后写一致性的数据存储　　b）不提供读后写一致性的数据存储

7.4 掌握基于主备份的写协议。P224

在基于主备份的协议中，每个数据项 x 都有一个关联的主备份，负责协调 x 的写操作，根据主备份是否被固定在一个远程服务器上，或将主备份移动到启动写操作的进程那里之后写操作是否可在进程本地执行，可以区分各种主备份协议。根据主备份的位置可将基于主备份的写协议分为远程写协议和本地写协议。
基于主备份的远程写协议：
（1）所有读操作和写操作都被转发到一个固定的远程服务器上。(如下图右)
（2）从一致性角度看，主机备份协议允许进程在本地可用的副本上执行读操作，但必须向一个固定的主拷贝上转发写操作的协议。(如下图左)



a) 主机备份协议

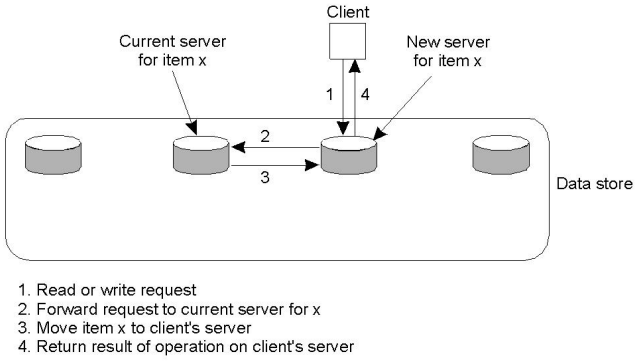


b)基于主备份的远程写协议

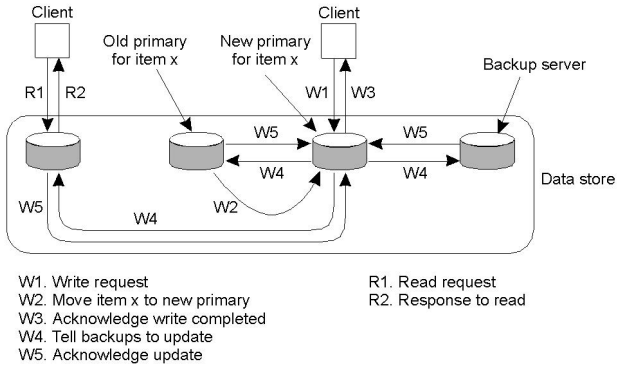
基于主备份的本地写协议：

（1）每个数据项都只有一个单一拷贝，每当一个进程要在某个数据项上执行操作时，先将那个数据项的单一拷贝传送到这个进程，然后在执行相应操作。(如下图左)

（2）主机备份的本地写协议，多个连续的写操作可在本地进行，而读操作的进程还可以访问它们的本地拷贝，其中主备份移动到要执行更新的进程那里，并且支持离线操作。(如下图右)



a) 基于主备份的本地写协议

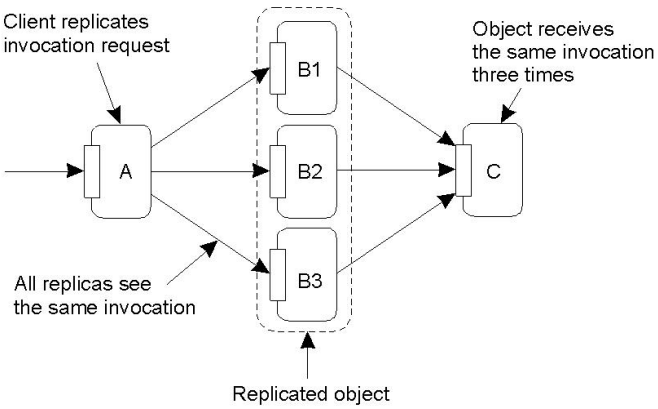


b)主机备份协议的本地写协议

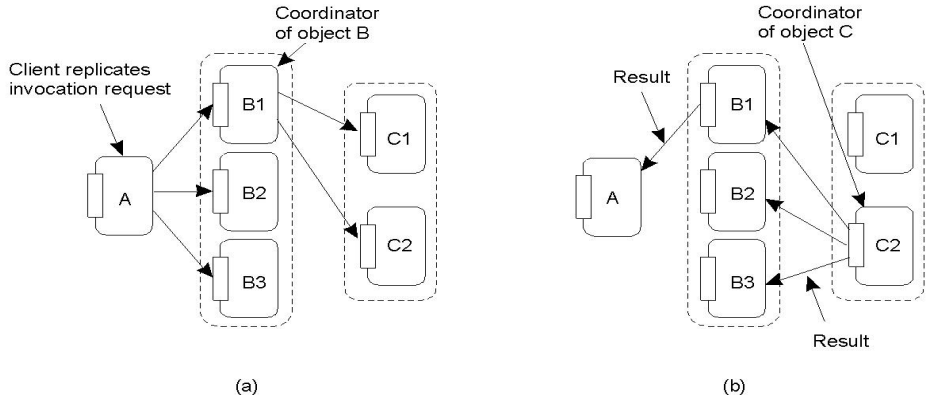
7.5 掌握基于复制的写协议。

复制的写协议：写操作可以在多个副本上执行，分为主动复制和基于法定数目的协议。

主动复制的写操作：可以在多个副本上执行，每个副本对应一个进程，该进程执行更新操作.潜在问题：(1)写操作导致更新传播，操作需要在各地按相同顺序进行，时间戳和定序器来实现。（2）被复制的调用，如下图中 C 被调用多次。



解决办法：给 b 和 b 的复本分配一个相同的，唯一的标识符，然后使用 B 的协调器将请求发送到 c 的所有复本上，c 处理完后将消息反馈到 c 的协调器上，然后由 c 的协调器将消息复制到 b 的所有复本上。



基于法定数目的协议：要求客户在读或写一个复制的数据项之前向多个服务器提出请求，并获得他们的许可。与前边的主动复制的差异在于，主动复制要将更新传递到所有的复本上去。

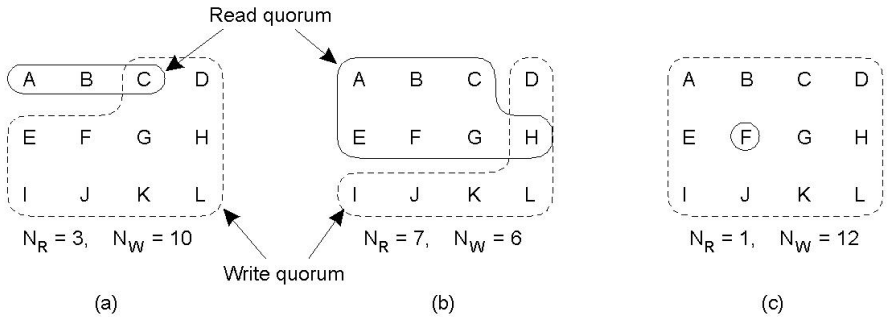
Gifford 方法(确定法定数目)：

客户在读写一个复制的数据时，先向多个服务器提出请求，获得许可;

读团体 NR 和写团体 NW, N 个副本

NR+NW>N; NW>N/2

表决算法的三个实例



a)读集合和写集合的正确选择　　b)可能导致写写冲突的读集合和写集合

c)读集合和写集合的正确选择，ROWA (read one, write all)

第八章容错性

8.1 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

8.2 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。

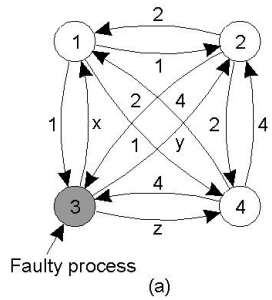
时间冗余：多次执行一个动作，如事务。

物理冗余：增加额外的设备或进程。如三倍的模块冗余。

8.3 拜占庭将军问题

Lamport 证明在具有 m 个故障进程的系统中，只有存在 2m+1 的正常工作的进程才能达成协议，即总进程为 3m+1。

三个忠诚将军和一个叛徒的问题，叛徒通过发送错误和矛盾来组织忠诚将军达成协议。如下图：假设现在将军们要相互告知自己的兵力，将军三是叛徒。每个人都将自己收到的向量发给其他所有的将军。



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

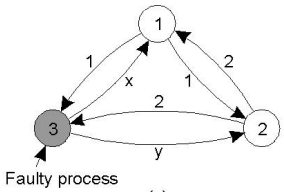
(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

(a) 将军宣布他们的兵力(b)(a)基础上每个将军的向量(c)每个将军收到的向量

这个时候，将军们把自己收到的向量中，哪一个数出现得最多就把他纪录到自己的向量中去，因此将军 124 都可以看到相同的消息：（1,2，UNKNOWN，4）。但是如果有两个忠诚将军和一个叛徒将军，就不能判断了。如下图：



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

1、2 得到的信息都没有出现大多数情况，所以，该算法不能产生协定。

8.4 什么叫原子多播？

在分布式系统中经常需要保证消息要么被发送给所有的进程，要么就不向任何进程一个也不发送。另外，如果传送的话，通常还需要所有的消息都按相同的顺序发送给所有的进程。这种方式称为**原子多播**。

原子多播确保没有故障的进程对数据库保持一致；当一个副本从故障中恢复并重新加入组时，原子多播强制它与其他组成员保持一致。

虚拟同步：保证多播到组视图的消息被传送给组中的每个正常进程，如果发送消息的进程在多播期间失败，则消息或者传递给剩余的所有消息，或者被每个进程忽略。具有这种属性的可靠多播称为虚拟同步。所有多播都在视图改变之间进行

消息排序：

（1）可靠不排序的多播：对接收不同进程发送的消息的次序不做任何保证

（2）FIFO 顺序的多播：按照消息发送的顺序传送同一进程的消息，对不同进程发送的消息的传送顺序没有约束

（3）按因果关系排序的多播：按因果关系排序多播来保留消息间的因果关系

（4）全序多播：无论消息传送是无序、 FIFO 顺序还是按因果关系排序，对所有的组成员按相同的次序传送

提供了全序的消息传送的虚拟同步可靠多播称为原子多播

8.6 分布式提交—两阶段提交的思想？

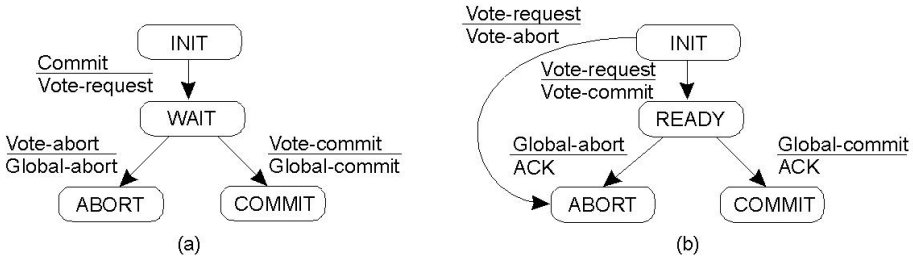
在两段提交协议中，将提交分成两个阶段，每个阶段有两步（1）（2）属于第一阶段（3）（4）属于第二阶段：

（1） 协调者向所有的参与者发送一个 VOTE_REQUEST 消息。

（2） 当参与者收到 VOTE_REQUEST 消息时，就向协调者返回一个 VOTE_COMMIT 消息通知协调者它已经准备好本地提交事务中属于它的部分，否则返回一个 VOTE_ABORT 消息。

（3） 协调者收集来自参与者的所有选票。如果所有的参与者都表决提交，那么就向所有的参与者发送一个 GLOBAL_COMMIT 消息，如果有一个参与者表决取消，协调者决定取消事务并多播一个 GLOBAL_ABORT 消息。

（4） 每个提交表单的参与者等待协调者的最后反应。如果参与者收到 GLOBAL_COMMIT 消息，就在本地提交事务，否则收到 GLOBAL_ABORT，就在本地取消事务。



a) 2PC 中的协调者的有限状态机　b) 2PC 中的参与者的有限状态机

参与者一旦投票，则失去自主能力，必须等待协调者的最终决定，可能造成阻塞

可能的阻塞状态：

参与者在 INIT 状态等待协调者的 VOTE_REQUEST 消息;

协调者在 WAIT 状态等待来自每个参与者的表决;

参与者在 READY 状态等待协调者发送的全局表决消息;

第九章分布式安全

9.1 什么是机密性和完整性？

机密性(confidentiality):指计算机系统的一种属性，系统凭借此属性使得信息只向授权用户公开。

完整性:是指对系统资源的更改，只能以授权方式进行。

9.2 对通信通道的安全威胁类型有哪几种？

侦听（Interception），中断（Interruption），更改（Modification），伪造（Fabrication）。

切断（Interruption）是使得系统中的资源被破坏而变得不可获得或不能用，是对可用性进行攻击。（如破坏硬件，切断通信线，使文件系统失效等。）

侦听（Interception），指的是未授权者获得对资源的访问，对机密性的攻击。（如搭线获得对资源的访问,非法复制文件或程序。）

更改（Modification），未授权者不仅获得资源的访问，还进行更改。是对完整性的攻击。（如通过改变数据文件的数值，改变程序使其执行发生变化，修改网络中传输消息的内容。）

伪造（Fabrication），未授权者将伪造的对象加入系统中，是对可靠性的攻击。（如将假消息放到网络中，在文件中添加记录等。）

9.3 对称加密系统和公钥系统的区别？

对称加密系统：加密与解密密钥相同，即 $P=D_k(E_k(P))$,也称为共享密钥系统。

非对称加密系统：加密与解密密钥不同(一个公开、一个保密)，但构成唯一的一对，即 $P=D_{Kd}(E_{Ke}(P))$,也称为公钥系统。

后者在计算上更为复杂。用 RSA 加密消息比 DES 慢大约 100-1000 倍。所以一般用 RSA 以安全方式交换共享密钥，很少用它来实际加密“标准”数据。

9.4 什么是安全通道？

在通信各方之间家里的通道，使客户与服务器之间的通信保持安全，免受对消息的窃听、修改和伪造的攻击。对防止中断地保护不是必要的。保护消息免受窃听是通过确保机密性实现的，安全通道可确保入侵者不窃听到其消息。防止入侵者的修改和伪造是通过相互身份验证和消息完整性的协议实现的。

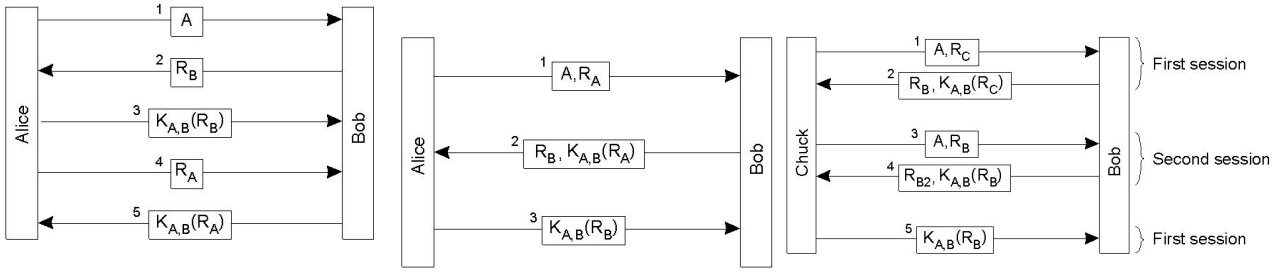
9.5 阐述基于共享密钥的身份验证的思想。

质询-响应协议：一方向另一方质询一个响应，只有对方知道共享密钥时才能给予正确的响应

如下图左：基于共享密钥的身份验证（使用 5 个消息）。Alice 将其身份（消息 1）发送给 Bob，指出她希望在两个人之间建立一个通信通道，随后 Bob 向 Alice 发送一个质询 R_B（消息 2）。（这样的一个质询可以采取随机数的形式。）需要 Alice 使用她与 Bob 共享的密钥 $K_{A,B}$ 加密该质询，返回给 Bob。此响应在途中为消息 3，包含了 $K_{A,B}(R_B)$。Bob 接收到对其质询 R_B 的响应 $K_{A,B}(R_B)$ 时，可以再次使用该共享密钥 $K_{A,B}$ 对该消息进行解密以查看其中是否包含 R_B。如果包含，那么他就知道 Alice 在另一边，Bob 现在已经检验出他确实在和 Alice 交谈。然而，Alice 还没有检验出通道的另一边确实是 Bob。一次，她发送一个质询 R_A（消息 4），Bob 通过返回 $K_{A,B}(R_A)$ 响应该消息（消息 5）。Alice 使用 $K_{A,B}$ 解密该消息并看到 R_A 时，她就知道在于 Bob 交谈。

如下图中：基于共享密钥的身份验证（使用 3 个消息）。基本思想是如果 Alice 最终希望用任何方式询问 Bob，那么可以在她建立通道时可能还会与其身份一起发送一个质询。同样，Bob 在单个消息中返回其对该质询的响应，以及自己的质询。

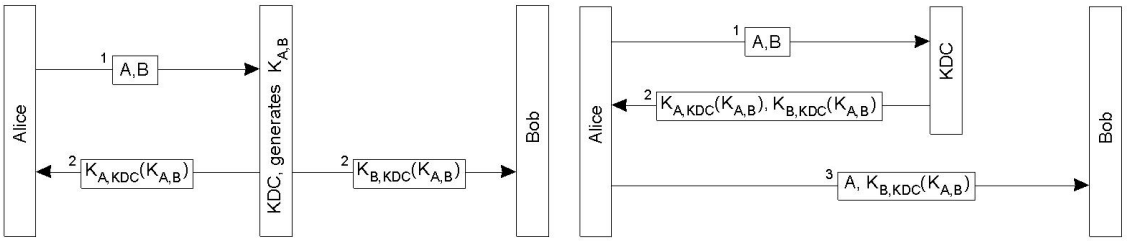
缺点：反射攻击。（如下图右）假设有一个名为 Chuck 的入侵者，即 C。Chuck 的目标是与 Bob 建立一个通道。以使 Bob 相信他在与 Alice 交谈。第一次会话，Chuck 发送 Alice 的身份标示和质询 R_C，Bob 发送响应 $K_{A,B}$（R_C）和询问 R_B，Chuck 无法返回用 $K_{A,B}$ 加密的 Bob 的询问，所以发起第二次会话，即以 Alice 身份发送询问 R_B，而这个密钥是 Bob 先前使用过的，Bob 本身没有认识到，所以，Bob 会发送响应 $K_{A,B}$（R_B）和新的询问 R_{B2}。这时 Chuck 已经具有了第一次会话需要的对 Bob 的响应 $K_{A,B}$（R_B），发送给 Bob，从而完成看了第一次会话的建立。



9.6 阐述使用密钥发布中心的身份验证的思想。

主要通过引入第三方的认证机构来确认身份和分发密钥。由于基于共享密钥的身份验证存在可扩展性问题：即对 N 台主机，任两台需要一个共享密钥，共需要 N*(N-1)/2 个密钥，每台主机管理 N-1 个。而使用密钥分发中心 KDC（key distribution center）使用集中式管理。KDC 与每个主机共享一个密钥，总共只需要管理 N 个密钥，KDC 与每台主机共享一个密钥。通过向通信的两主机分发一个密钥通信实现通信。

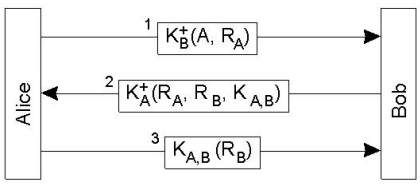
下图左：KDC 向 Alice 和 Bob 分发一个密钥，他们可使用该密钥进行通信。Alice 先向 KDC 发送一条消息，通知 KDC 她希望与 Bob 交谈。KDC 返回一个包含她能使用的共享密钥 $K_{A,B}$ 的消息。该消息使用了 KDC 好 Alice 共享的密钥 $K_{A,KDC}$ 进行加密。此外，KDC 也向 Bob 发送 $K_{A,B}$ 并使用 KDC 与 Bob 的共享密钥 $K_{B,KDC}$ 加密。从而通信双方具有了共享密钥，可进行通信。



缺点：Alice 可能希望在 Bob 接收到来自 KDC 的共享密钥前就开始与 Bob 建立一个安全通道，此外，需要 KDC 向 Bob 传密钥。解决：如上图右，Alice 发送 KDC 要与 Bob 通话的请求，把自己和 Bob 的标示发送给 KDC，KDC 返回给 Alice 的是 $K_{A,KDC}(K_{A,B})$ 和 $K_{B,KDC}(K_{A,B})$（$K_{B,KDC}(K_{A,B})$ 也称为票据）。Alice 再把 $K_{B,KDC}(K_{A,B})$ 发送给 Bob，因为票据是 KDC 与 Bob 的共享密钥加密的，所以只有 Bob 可以解开，从而使 Bob 得到他与 Alice 的共享密钥，可以发起会话。

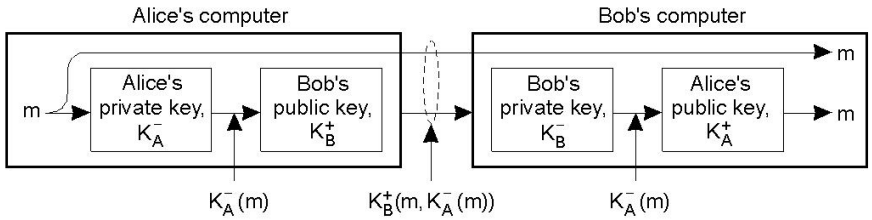
9.7 阐述使用公钥加密的身份验证的思想。

公钥加密使得，消息交互双方拥有彼此的公钥，通过用公钥加密，私钥解密。如下图。Alice 向 Bob 发送一个使用他的公钥 K_B^+ 加密的质询 R_A 开始。Bob 可以用自己的私钥 K_B^- 解密。所以 Alice 可以知道自己是在和 Bob 交谈。Bob 收到 Alice 建立通道的请求时，返回解密过的质询 R_A 以及他只身用于对 Alice 进行身份认证的质询 R_B，此外他还产生一个用于进一步通信的会话密钥 $K_{A,B}$，这些信息都用 Alice 的公钥 K_A^+ 进行加密发送给 Alice。Alice 收到消息 2 时，可以用自己私钥 K_A^- 进行解密。最后，Alice 返回七对 Bob 的质询的回答，并使用 Bob 提供的会话密钥 $K_{A,B}$ 加密，证明了她能够解密消息 2，使 Bob 可以确定在和 Alice 交谈。



9.8 使用公钥加密对消息进行数字签名的思想。

数字签名利用了，公钥，私钥对唯一对应的思想，使得一个消息先通过发送方的私钥加密，在用接收方的公钥加密，解密的时候，可以唯一确定发送方的身份。因此使用公钥加密对消息进行数字签名，如果消息签名检验为真，发送者不能否认消息签名这一事实，消息与其签名的唯一关联防止了对消息进行修改而未发现的可能。



缺点：Alice 可以声称其私钥被盗。（如果 Alice 希望在向 Bob 发送了她的确认后撤销该交易，那么她可以声称她的私钥在该消息发送前被盗了）

9.9Diffie-Hellman 建立共享密钥的原理。

Diffie-Hellman 是公钥加密系统，目的是建立共享密钥。

首先，Alice 和 Bob 双方约定 2 个大整数 n 和 g,其中 1<g<n，这两个整数无需保密，然后，执行下面的过程如下图：

Alice 随机选择一个大整数 x（保密），并计算 $X=g^x \bmod n$

Bob 随机选择一个大整数 y（保密），并计算 $Y=g^y \bmod n$

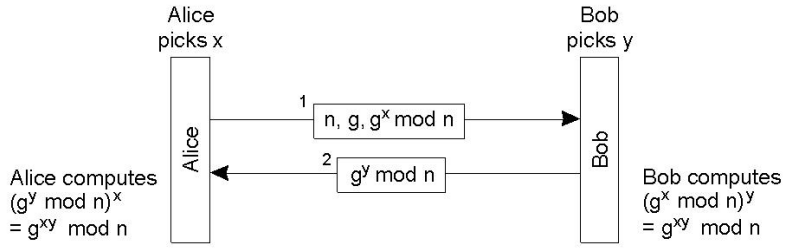
Alice 把 X 发送给 B,B 把 Y 发送给 ALICE

Alice 计算 $K=Y^x \bmod n = g^{xy} \bmod n$

Bob 计算 $K=X^y \bmod n = g^{xy} \bmod n$

K 即是共享的密钥。

监听者在网络上只能监听到 X 和 Y，但无法通过 X，Y 计算出 x 和 y，因此，无法计算出 K



K 即是共享的密钥。

9.10 权能和委派。

权能：对于指定资源的一种不可伪造的数据结构，它确切指定它的拥有者关于该资源的访问权限。

委派：将某些访问权限从一个进程传送到另一个进程，使得在多个进程间分布工作变的更容易，而又不会对资源保护产生明显影响。

第十章分布式文件系统

10.1 NFS 的共享预约

NFS(Network File System 网络文件系统)共享预约：一种锁定文件的隐含方法。客户打开文件时，指定它所需的访问类型（READ、WRITE 或 BOTH),以及服务器应该拒绝的其他客户的访问类型（ NONE、READ、WRITE 或 BOTH ）。如果服务器不能满足客户的需求，那么该客户的 open 操作失败。

当前文件的拒绝状态

请求访问	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(a)

请求的文件拒绝状态

当前访问状态	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

(b)

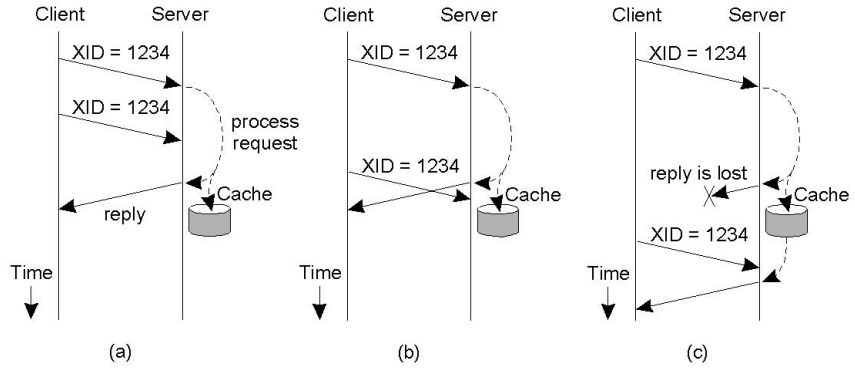
使用 NFS 共享预约实现操作的结果（新客户试图打开另一客户已成功打开的文件）。

a）当前文件的拒绝状态下客户请求共享访问的情况

b）在当前文件访问状态下，客户请求拒绝状态的情况

10.2 NFS 的重复请求高速缓存

NFS 服务器提供重复请求高速缓存：每个来自客户的 RPC 请求头部都有一个唯一的 XID（事务处理标识符）。当 RPC 请求到来时，服务器缓存其标识符。只要服务器还没有发出响应，就说明正在处理这个 RPC 请求。当服务器处理了某个请求后，也缓存该请求的关联响应，此后该响应才返回客户。现在需处理如下三种情况：



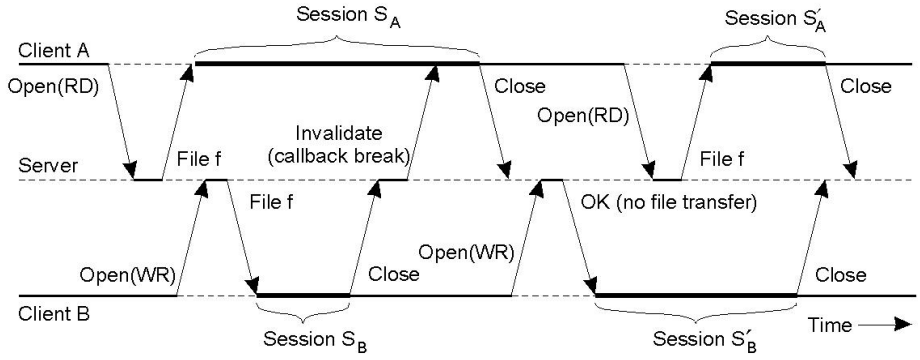
a)客户启动计时器，超时前若客户没有收到响应，就使用与原 XID 请求重传。若服务器还没处理完原先的请求，忽略此重传请求。

b)服务器刚向客户端返回响应后收到重传请求。如果重传请求到达时间与服务器发送应答时间接近，服务器忽略此重传请求。

c)响应确实丢失了，应该向客户端发送操作的缓存结果以响应重传的请求。

10.3 Coda 的回叫承诺

对于每个文件，客户从服务器获取文件时，服务器记录哪些客户在本地缓存了该文件的拷贝，此时称服务器为客户记录回叫承诺。如果文件被客户更改，会通知服务器，服务器向其他客户发无效化消息（这种无效化消息叫做回叫中断，服务器随后会废弃它刚刚想向其发送了无效消息的客户保存的回调承诺）。如果客户在服务器上有未被废弃的回叫承诺，它就可以安全地在本地访问文件。



如上图：当客户 A 开始会话 S_A 时，服务器会记录一个回调承诺，同样，B 开始绘画时也是如此 S_B。当 B 关闭 S_B 时，服务器向客户 A 发送回叫中断，从而中断了它对回调客户 A 的承诺。(当客户 A 关闭会话 S_A 时，不会发生任何特殊的事情)，这个关闭操作就像所预期的那样被简单接受。)随后，当 A 打开 S_A'时,它会发现 f 本地的副本是无效的，必须从服务器获取最新的版本。另一方面，当 B 开始会话 S_B'时，它会注意到服务器仍有一个未被废弃的回调承诺，该承诺意味着 B 可简单的重用从 S_B 获得的本地副本。

10.4 Coda 的储藏技术

Coda 允许客户在断开连接时的继续操作。使用本地备份，再次连接后回传服务器。基于事实，两个进程打开相同的文件进行写操作很罕见。为了成功地完成断开连接操作，需要解决的问题是确保客户缓存包含连接断开期间将访问的那些文件。如果采用简单的缓存方法，可以证明客户可能由于缺少必要的文件而不能继续执行。预先使

用适当的文件填充高速缓存称为储藏。