第一章 绪论

1、中间件在分布式系统中扮演什么角色？

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、解释（分布）透明性的含义，并且给出各种类型透明性的例子。

答：分布透明性是一种现象，即一个系统的分布情况对于用户和应用来说是隐藏的。包括：访问透明、位置透明、移植透明、重定位透明、复制透明、并发透明、故障透明和持久性透明。

3、在分布式系统中，为什么有时难以隐藏故障的发生以及故障恢复过程？

答：通常，要探测一个服务器是停止服务还是该服务器的反应变慢这些情况是不可能的。因此，一个系统可能在服务响应变慢的时候报告该服务已经停止了。

4、为什么有时候要求最大程度地实现透明性并不好？

答：最大程度地实现透明性可能导致相当大的性能损失，从而导致用户无法接受。

5、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。

6、请对可扩展系统的含义做出准确描述

答：一个系统的可扩展包含下面几个方面：组件的数量、几何尺寸、管理域的数量与尺寸，前提是这个系统可以在上面几个方面进行增加而不会导致不可接受的性能损失。

7、可以通过应用多种技术来取得可扩展性。请说出这些技术。

答：可扩展性可以通过分布式、复制和缓存来获得。

8、多处理器系统与多计算机系统有什么不同？

答：在多处理器系统中，多个 CPU 访问共享的主存储器。在多计算机系统中没有共享存储器，CPU 之间只能通过消息传递来进行通信。

9、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。

10、现在考虑包含 256 个 CPU 的超立方体，最坏情况下消息的延迟有多长？

答：在具有 256 个 CPUs 的超立方体中，每个结点可以用一个二进制地址，范围从 00000000~1111,1111。一个 hop 表示地址改变了一位。因此从 0000,0000~0000,0001 表示一个 hop，而从 0000,0001~0000,0011 也是表示一个 hop。所以最长的路由有 8 个 hop。

11、分布式操作系统与网络操作系统有什么不同？

答：分布式操作系统管理多处理器和同种类的多计算机。网络操作系统连接不同的、独立的计算机，这些计算机有自己的操作系统以便用户可以容易地使用每台计算机所提供的服务。

12、请解释如何使用微内核将操作系统组织成客户-服务器的方式。

答：微内核可以把客户端应用从操作系统服务中分离出来，方法是通过强迫每个请求都通过内核来传递。因此，操作系统服务能够通过用户级的服务器来实现，这些用户级的服务器是作为普通的进程来运行的。如果微内核有网络功能，这些服务器也能作为远端机器。

13、请解释基于分页的分布式共享存储器系统主要有哪些操作。

答：基于分页的 DSM 利用了操作系统的虚拟存储器的功能。当一个应用程序对内存寻址时，如果该内存位置没有映射到当前物理存储器的存储器空间的时候，页错误就会发生，并将控制权转交给操作系统。操作系统定位到适当的页，通过网络传递它的内容，并映射到物理存储器中。从而，应用可以继续运行。

14、为什么要开发分布式共享存储器系统？你认为是什么问题给这种系统的高效率实现造成了困难？

答：主要的原因是，写基于消息传递的并行和分布式程序来进行通信要比使用共享存储器难得多。不管进行什么操作，网络中的页面传输都会发生，这导致了 DSM 系统效率的下降。如果页面被不同的处理器共享，在虚拟存储系统中很容易进入和“陷入”类似的状态。最后，DSM 系统比消息传递的解决方案要慢，而且由于需要跟踪页面而变得更慢。

15、请解释什么是分布式共享存储器系统中的伪共享。你有没有针对这个问题的解决方案？

答：当数据属于两个不同且独立的处理器（可能在不同的机器上）时会发生伪共享，这时数据被映射到相同的逻辑页上。效果是这个页面会在两个处理器之间发生交换，从而导致不明显且不必要的依赖。解决方案是让页面更小并禁止独立的处理器共享一个页面。

16、由于存在错误，某个实验性的文件服务器有 3 / 4 的时间能够正常工作，而另外 1 / 4 的时间无法工作。如果要确保服务至少在 99%的时间可用，需要将该文件服务器复制多少次？

答：令 k 为服务器的数量，则有$(1/4)^k<0.01$。即在最坏的情况下，这时所有的服务器都已关闭，发生这种情况的概率是 1/100。因此 k=4。

17、什么是三层客户-服务器体系结构？

答：三层客户——服务器体系结构包括三个逻辑层，每一层在理论上来说都在一台单独的机器上实现。最高层包括了客户的用户界面，中间层包括实际的应用程序，最底层包含了被使用的数据。

18、纵向分布与横向分布有什么不同？

答：纵向分布指的是多台机器组成的多层架构中各个层的分布。从理论上说，每一层都在一台不同的机器上实现。横向分布则处理多台机器上的一个层的分布，例如一个数据库的分布。

19、考虑一个进程链，该进程链由进程 P1,P2,…,Pn 构成，实现了一个多层客户-服务器体系结构。进程 Pi 是进程 Pi+1 的客户，Pi 只有得到 Pi+1 的应答之后才能向 Pi-1 发出应答。如果考虑到进程 P1 的请求-应答性能，这种组织结构主要存在什么问题？

答：如果 n 很大的话性能会很差。从理论上来说，两个邻接层之间的通信应该在两台不同的机器之间发生。因此，P1 和 P2 之间的性能由 n-2 次其它层之间的请求——应答过程决定。另一个问题是如果链中的一台机器性能很差甚至临时不可达，这会立刻使最高层的性能降低。

第二章 通信

1、在许多分层协议中，每一层都有自己的报头。如果每个消息前部都只有单个报头，其中包含了所有控制信息，无疑会比使用单独的多个报头具有更高的效率。为什么不这么做？

答：协议的每一层都必须和其它层相独立。从第 k+1 层传送至第 k 层的数据同时包含了报头和数据，但是第 k 层协议不能对它们进行辨别。如果使用单个大的报头来包含所有信息的话将会破坏透明性，使得一个协议层的变动会影响到其它层，这显然不是我们所希望的。

2、为什么传输层通信服务常常不适于构建分布式应用程序？

答：它们通常不提供分布透明性，这意味着应用程序开发人员需要注意通信的实现，从而导致解决方案的可扩展性很差。分布式应用程序，例如基于套接字构建的分布式应用程序，将很难移植或者和其它应用程序交互。

3、一种可靠的多播服务允许发送者向一组接收者可靠地传递消息。这种服务是属于中间件层还是更低层的一部分？

答：从理论上来说，一种可靠的多播服务可以很容易的成为传输层，甚至是网络层的一部分。例如，不可靠的 IP 多播服务是在网络层实现的。但是，由于这些服务目前尚无法应用，它们通常使用传输层的服务来实现，传输层的服务将它们放在中间件中。如果把可扩展性加以考虑的话，只有充分考虑应用程序的需求时可靠性才能得到保证。用更高、更特殊的网络层来实现这些服务存在一个很大的争议。

4、考虑一个带有两个整型参数的过程 incr。该过程将两个参数的值分别增加 1.现在假定调用它时使用的两个参数是同一个变量，比如 incr(i, i)。如果 i 的初始值是 0，在执行引用调用之后 i 将变为什么值？如果使用复制——还原调用呢？

答：如果执行引用调用，指向 i 的指针被传入 incr。i 将会被增加两次，因此最终的结果是 2。而使用复制——还原调用时，i 会被两次传值，每次的初始值均为 0。两次都会增加 1，因此结果均为 1。最后都复制到 i，第二次的复制会覆盖第一次的，因此最终 i 的值为 1，而不是 2。

5、C 语言中有一种称为联合（union）的构造，其中的记录（在 C 语言中称作结构）的字段可以用来保存几种可能值中的一个。在运行时，没有可靠的办法来分辨其中保存的是那一个值。C 的这种特性是否与远程过程调用有某些相似之处？请说明理由。

答：如果运行时系统不能分辨一个字段的值类型，它就不能对该字段进行正确的封送处理。除非有一个标签字段用来清楚的表明一个字段的值类型，联合不能在远程过程调用中使用。这个标签字段不能被用户所控制。

6、处理 RPC 系统中参数转换的一种方法是，每台机器以自己系统使用的表示方式来发送参数，由另一方在必要的情况下进行转换。可以通过首字节中的代码来表示发送消息机器所用的系统。然而，由于要在首个字中找到开头的字节这本身也是一个问题，这种方法能行得通吗？

答：首先，当一台机器发送字节 0 时，消息肯定已经送到。因此目标机器可以访问字节 0，而代码就在消息里面。这种方法不考虑字节是高位优先还是低位优先的字节。另一个方法是将代码放在第一个单词的所有字节中。因此不管检查的是哪一个字节，代码都能被找到。

7、假定客户通过异步 RPC 对服务器进行调用，随后等待服务器使用另一异步 RPC 返回结果。这种方法与客户端执行常规的 RPC 有没有什么不同？如果使用的是同步 RPC 而不是异步 RPC，情况又如何呢？

答：二者并不相同。异步 RPC 向调用者返回一个通知，这意味着客户第一次调用之后，有一个额外的消息会被发送。类似地，服务端接收到它的响应消息已经发送到客户端的通知。如果保证通信可靠的话，两次异步 RPC 调用是一样的。

8、在 DCE 中，服务器在守护程序中注册自身。如果换一种方法，也可以总是为它分配一个固定的端点，然后再指向服务器地址空间中对象的引用中就可以使用该端点。这种方法的缺陷在哪里？

答：这种方法的主要缺陷是向服务器分配对象变得很难。另外，许多端点而不止一个需要被修复。如果机器中很多都是服务器，分配固定端点不是一个好办法。

9、给出一种用来让客户端绑定到暂时远程对象的对象应用的实现示例。

答：使用 Java 实现的类如下：

```
public class Object_reference {
    InetAddress server3address; // network address of object's server
    int server3endpoint; // endpoint to which server is listening
    int object3identifier; // identifier for this object
```

URL client3code; // (remote) file containing client-side stub
byte[] init3data; // possible additional initialization data
}

Object_reference 类至少需要包含对象所属的服务器的传输层地址。在具体实现中，使用了一个 URL 来标识包含了所有必需的客户端代码文件，用一个字节数组来保存进一步初始化后的代码。另一种实现可以直接保存客户端代码而不是一个 URL。这种方法将代理对象作为引用传递，Java RMI 采用了这种做法。

10、Java 和其他语言支持异常处理，当错误发生时会引发异常。如何在 RPC 和 RMI 中实现异常处理？

答：由于异常通常在服务端发生，服务器存根只能捕获这个异常并把它作为一个特殊的错误响应传送给客户端。另一方面，客户存接收这个消息并抛出这个异常以保持对服务器访问的透明性。因此，接口定义语言中也需要有对异常的描述。

11、将静态和动态 RPC 区分开来有用吗？

答：有用。和远程对象调用有用的理由一样：它会带来更多的弹性。缺点是很多的分布透明性会被牺牲掉，其程度视哪种 RPC 先被使用而定。

12、某些分布式中间件系统的实现是完全基于动态方法调用的。甚至连静态的调用也被编译成动态的调用。这种方法的优点在哪里？

答：动态调用的实现能够处理所有的调用，而静态调用仅仅是其中的一种特殊情况。优点是仅仅需要实现一种机制。一个可能的缺点是性能并不是总比使用静态调用的情况要好。

13、描述一下客户端和服务器端之间使用套接字的无连接通信是如何进行的。

答：客户端和服务端都需要创建一个套接字，但是只有服务器把套接字绑定到本地的端点上。然后，服务器可以执行一个阻塞的 read 调用以等待从客户端发送的数据。类似地，在创建套接字之后，客户端仅仅执行一个阻塞调用以向服务端写数据。关闭连接是没有必要的。

14、说明 MPI 中 mp_based 原语和 mp_isend 原语之间的区别。

答：mpi_based 原语使用有缓冲的通信，调用者将包含了信息的整个缓冲传送到本地的 MPI 运行时系统。当调用完成时，这些信息要么被已被发送，要么被拷贝到一个本地的缓冲区。如果使用 mpi_isend，调用者仅仅将指向信息的指针传送给本地的 MPI 运行时系统，然后继续往下执行。调用者需要保证在消息被拷贝或被传送之前不能覆盖它。

15、假定只能使用暂时异步通信原语，再加上异步 receive 原语，如何实现用于暂时同步通信的原语？

答：考虑一个同步的 send 原语。一个简单的实现是使用异步通信向服务器发送一个消息，然后让调用者不停地查询接收到的来自服务器的通知或响应。另一种实现方案是，如果假设本地操作系统将接收的消息保存在一个本地缓冲区中，那么阻塞调用程序直到接收到系统的消息到达信号，之后调用程序执行异步 receive。

16、假定只能使用暂时同步通信原语，如何实现用于暂时异步通信的原语？

答：异步 send 可以通过如下方式实现：调用者将它的消息拷贝到一个缓冲区，实际处理消息发送的进程共享该缓冲区。每当客户端将消息拷贝到缓冲区，消息发送线程被唤醒，它将该消息从缓冲区中删除并使用一个阻塞的 send 原语将其发送到目标机器。接收方的实现与此类似，它提供缓冲区，一个应用程序可以检查该缓冲区以确定是否有消息。

17、通过 RPC 实现持久化异步通信有意义吗？

答：仅仅在管理了一个队列的进程通过 RPC 将消息发送给下一个队列管理器时有意义。一个队列管理器为另一个管理器提供的服务是保存消息，调用的队列管理器可以获得远程队列的一个代理对象，并可能接收到每一个操作成功或失败的状态。使用这种方法时，队列管理器看到的仅仅是队列，而不会发生通信。

18、在本章中我们讲过，为了自动启动一个进程以从输入队列中获取消息，常常要使用守护程序来监视输入队列。请给出一种不使用守护程序的实现方法。

答：一个简单的实现方案是，每当接收端的进程将一条消息放进它的一个队列时，同时检查一下是否接收到了消息。

19、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。

20、如何将持久异步通信加入到基于远程对象 RMI 的通信模型中去？

答：RMI 应该是异步的，也就是说，在调用的时候不能即时得到调用结果。RMI 应该被保存在一个特殊的服务端，它在对象服务器中的对象启动并运行时立即将 RMI 转发到该对象。

21、在持久通信中，接收者一般拥有自己的本地缓冲区，如果接收者不在运行状态，可以将消息放入该缓冲区中去。为了创建这种缓冲区，必须指定它们的大小。分成两方进行辩论：一方认为这种指定缓冲区大小的行为是可取的，而另一方反对这种指定大小的行为。

答：用户指定缓冲区大小使得实现更加容易。系统创建一个固定大小的缓冲区，之后缓冲区管理会十分容易。但是，如果缓冲区已满，消息可能会丢失。另一种方案是由通信系统来管理缓冲区大小，首先先制定一个默认大小，然后视情况需要扩充或者压缩缓冲区。这种方法减少了由于缺少空间而导致丢失消息的机率，但是需要系统做更多的工作。

22、请说明为什么暂时同步通信在可扩展性方面存在固有的问题，以及如何解决这些问题。

答：问题在于地理上的可扩展性受到了限制。由于同步通信需要调用者被阻塞直到消息被接收，因此，如果接收方相隔较远的话调用者可能需要阻塞很长的时间才能继续运行。解决这个问题的惟一办法是设计调用程序，使得它在发生通信时可以做其它有用的工作，以有效地实现一种异步通信。

23、给出一个将多播应用于离散数据流的例子。

答：将一个大文件传送给许多用户就是一个例子。例如，更新 Web 服务或软件发布的镜像站点。

24、当一组计算机组成一个逻辑上或物理上的环时，如何确保传输延迟不超过允许的最大端到端延迟时间？

答：用一个令牌在环上循环，只有当计算机获取到该令牌的时候，它才能通过环发送数据。另外，计算机持有环的时间不能超过 T 秒。因此，如果假设两台邻接电脑之间的通信是有限制的，那么可以得到令牌循环的最大时间，这个时间就是发送一个包的最大端到端延迟时间。

25、当一组计算机组成一个（逻辑上或物理上的）环时，如何确保传输延迟不小于允许的最小端到端延迟时间？

答：从理论上来说，接收端计算机不应该在特定的时间之前接收数据。唯一的解决方案是将数据包存储在缓冲区中，缓冲可能发生在发送端、接收端或它们之间的某一个环节。缓冲数据最好的地方是在接收端，因为这里没有不可预见的、可能延迟数据传输的障碍。接收端仅需要将数据从缓冲区中删除并将它传送至使用了一个简单的计时机制的应用程序。缺点是必须提供足够的缓冲能力。

26、想象一下，某个令牌存储桶规范说明中的最大数据单元大小是 1000B，令牌存储桶速率是 10MB/s，令牌存储桶的大小是 1MB，而最大传输速率是 50MB/s。突发的传输可以以最大传输速率持续传输多长时间？

答：令最大的突发时间间隔为Δ t 秒。在极端的情况下，存储桶在开始时就已经满了，还有 10Δ tMB 的数据在该间隔内被传进来。在突发的传输过程中，输出的数据为 50Δ tMB，等于(1+10Δ t)。因此，Δ t 等于 25 毫秒。

27、在这道练习题中，要求实现一个使用 RPC 简单客户——服务器系统。服务器提供一个名为 next 的过程，该过程接受一个整型数作为输入，并且返回紧接着该整型数之后的一个数。编写一个客户使用的名为 next 的存根，它的任务是使用 UDP 将参数发送给服务器，随后等待服务器响应，如果过长时间未收到响应就认为超时了。服务器过程必须在一个公开的端口监听并接受请求，完成这些请求所要求的操作，然后返回结果。

答：无。

第三章 进程

1. 比较使用单线程文件服务器读取文件和使用多线程服务器读取文件有什么不同。花费 15ms 来接收请

求、调度该请求并且完成其它必须的处理工作，假定需要的数据存放在主存储器的缓存中。如果需要磁盘操作，就需要额外多花 75ms 在磁盘操作的过程中线程处于睡眠状态。如果服务器采用单线程的话，它每秒能处理多少个请求？如果采用单线程呢？

答: 在单线程情况下，命中 cache 花了 15ms，未命中 cache 花了 90ms。加权平均值为 2/3 X 15 + 1/3 X 90。这意味着请求花了 40ms，服务器没秒可以完成 25 次。对于多线程，所有磁盘等待都是交迭的，所以每个请求花了 15ms， 服务器每秒可以处理 $66\frac{2}{3}$ 个请求。

2. 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：

（1） 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。

（2） 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。

3、在文中我们描述了一个多线程的文件服务器，说明了为什么它比单线程服务器和有限状态服务器更好。有没有这样的环境，在其中使用单线程服务器会更好？给出这种环境的例子。

答：有。如果服务器完全是 CPU 绑定的，就没有必要使用多线程了。多线程可能只是增加了不必要的复杂性。例如，某个地区拥有 1 百万人口，现要建立一个数据库来保存每个人的一些信息，如（名字，电话号码），假定每个人的信息大小为 64 字节，则数据库的总大小为 64M 字节，为了快速查找，应该把这些数据保存在服务器的存储器中。

4、将轻量级进程与单个线程静态关联起来并不好，为什么？

答：这样的关联将在很大程度上迫使只有一个内核级的线程，这就意味着多线程的性能优势将会被损失掉。

5、如果每个进程只使用单个轻量级进程也不好，为什么？

答：在这种模式下，我们只能拥有用户级的线程，这意味着任何阻塞系统调用都将阻塞整个进程。

6、描述一种使用与可运行线程数目相等的轻量级进程的方法。

答：开始只有单个轻量级进程并让它选择一个可运行的线程。当发现一个可运行的线程后，轻量级进程创建另一个轻量级进程来寻找下一个线程来执行。如果没有找到可运行的线程，轻量级进程就销毁它本身。

7、本章解释过，代理可以通过调用所有副本来支持引用透明。能否对（服务器端的）对象的副本进行调用？

答：可以。假设一个副本对象 A 调用另一个对象 B（非副本），如果 A 包含 K 个副本，一个 B 的调用将会被 A 的每个副本执行，然而，B 应该只被调用一次。这种复制调用应该采取一些特殊的措施。

8、通过生成进程来构建并发服务器与使用多线程服务器相比有优点也有缺点。给出部分优点和缺点。

答：一个重要的优点是每个进程都被保护。在超级服务器处理完全独立的服务的时候是非常有必要的。另一方面，创建进程的代价也较高，同时，如果进程需要通信，则使用线程在很多情况下将更为简单，因为它避免了用内核来实现通信。

9、粗略地设想一种多线程服务器的设计，该服务器必须使用套接字作为面对底层操作系统的传输级接口，以支持多种协议。

答：一个相对简单的设计是，使一个单一的线程T等待接收传输层消息(TPDU)。如果我们假定每个TPDU的报头包含一个数字来识别更高层的协议，这个线程可以把它传递给处理特定协议的模块。每个这样的模块都有一个专门的线程来处理消息，这些模块把消息看作是一个输入请求。当处理完这个请求后，一个应答消息被传递给T，这时T将把这个应答封装成一个传输层消息并把它传递给适当的目的地。

10、如何防止应用程序绕过窗口管理器破坏屏幕显示。

答：使用微内核方法，一个包含窗口管理器的窗口系统通过这个方法运行，因此所有的窗口操作都需要通过这个微内核。从效果上看，这正是第一章中所讲的把客户端——服务器端模式转换为单计算机的本质。

11、解释对象适配器的概念。

答：对象适配器是一种普通的程序，它可以接收输入的调用请求并把它们传递给对象的服务器存根。对象适配器主要负责实现调用策略，这些策略确定是否、怎样以及多少多线程被用来调用一个对象。

12、举出几个用于支持持久性对象的对象适配器设计方面的问题。

答：最重要的问题可能是产生一个对象引用，这个引用可以被当前服务器和适配器独立地使用。这样的引用应该能被传递给一个新的服务器并可能为相应的对象指定特定的活动策略。其它的问题包括，持久性对象什么时候、怎样被写到磁盘上，以及对象保存在主存储器中的内容和保存到磁盘上的内容有什么区别。

13、改变对象适配器示例中的 thread_per_object 过程，使用单个线程来处理受该适配器控制的所有对象。

答：代码基本保持不变，除了我们需要创建一个 single 线程，该线程用修改过的版本 thread_per_object 来运行。这个线程作为 adapter_thread 来被引用。当多路输出选择器调用这个适配器的时候，它向 adapter_thread 的缓冲区中存放一条消息，然后再读取它。适配器线程调用适当的 stub 并处理响应消息。

```
#include <header.h>
#include <thread.h>
#define MAX3OBJECTS 100
#define NULL 0
#define ANY −1
METHOD3CALL invoke[MAX3OBJECTS]; /* array of pointers to stubs */
THREAD *root; /* demultiplexer thread */
THREAD *adapter3thread /* thread that runs single3thread*/
void single3thread(long object3id) {
    message *req, *res; /* request/response message*/
    unsigned size; /* size of messages */
    char **results; /* array with all results*/
    while(TRUE) {
        get3msg(&size, (char*) &req); /* block for invocation request */
        /* Pass request to the appropriate stub. The stub is assumed to */
        /* allocate memory for storing the results. */
        (invoke[req->object 3id]*)(req->size, req->data, &size, results);
        res = malloc(sizeof(message)+size); /* create response message */
        res->object3id = object3id; /* identify object */
        res->method3id = req.method3id; /* identify method */
        res->size = size; /* set size of invocation results */
        memcpy(res->data, results, size); /* copy results into response */
        put3msg(root, sizeof(res), res); /* append response to buffer */
        free(req); /* free memory of request */
        free(*results); /* free memory of results*/
    }
}
void invoke3adapter(long oid, message *request) {
    put3msg(adapter3thread, sizeof(request), request);
}
```

14、维护到客户的 TCP/IP 连接的服务器是状态相关的还是状态无关的。

答：假设服务器没有在客户端上保存其它消息，就可能认为服务器是状态无关的。问题在于不是服务器，而是服务器的传输层在客户端上保存了状态。本地操作系统所跟踪的与服务器无关。

15、想象一下，某个 web 服务器维护一个列表，该列表中的内容是 IP 地址与该地址最近访问过的 web 页的映射关系。当一个客户连接到该服务器的时候，该服务器在列表中查找该客户，如果找到的话就返回注册过的页面。这个服务器是状态相关还是状态无关的？

答：是状态无关的。状态无关设计的主要问题不是服务器是否保存了客户端的任何信息，而是所保存信息的准确度。在本例中，如果表丢失了，客户端与服务器还能正常的进行交互，就象什么事都没有发生一样。

在一个状态相关设计中，这种交互只有在服务器从错误中恢复过来以后才能进行。

16、Java RMI 对代码迁移依赖到何种程度？

答：把对象引用看作是可移动的代理，对象引用每次被传递的时候，我们实际上通过网络来迁移代码。幸运的是，代理没有执行状态，因此支持简单的移动性。

17、在 UNIX 系统中，可以通过让进程在远程机器上派生出一个子进程来支持强可移植性。请说明这种机制的工作机理。

答：UNIX 中派生的意思是把父进程的完整镜像拷贝给子进程，这意味着在调用完 fork 后，子进程继续运行。一个相似的方法可以用来做远端克隆，这里假定了目标平台与父进程的平台是一样的。第一步是让目标操作系统为新建的子进程保存资源并创建相应的进程和存储器映射。这一步完成之后，父进程的镜像可以被拷贝了，子进程也可以被激活了。

18、图 3.13 指出，强可移植性不能与在目标进程中已迁移代码的执行结合在一起。请举出一个反例。

答：如果强可移植通过线程迁移发生的话，它就应该能使一个迁移的线程在目标进程的环境中执行。

19、考虑某个进程 P，它请求访问与自己位于同一台机器上的本地文件 F。在 P 迁移到另一台机器上以后，它还需要访问 F。如果文件对机器的绑定是紧固的，如何实现对 F 的系统级引用？

答：一个简单的解决方案是，创建一个单独的进程 Q，用 Q 来处理对 F 的远端请求。提供给进程 P 与以前一样的接口。例如以代理的形式。从效果上讲，进程 Q 是作为文件服务器来工作的。

20、D'Agents 系统中的每个代理都由单独的进程实现。代理通信的主要方式是共享文件以及消息传递，而文件无法在机器之间传递。如果按照 3.4 节中给出的移动框架结构，代理的那一部分状态包含了资源段？

答：资源段包含了所有本地以及全局的资源引用。同样地，它由那些指向其它代理、本地文件等等的变量组成。在 D'Agents 系统中，这些变量主要包含在全局程序变量部分中。事实上，在 D'Agents 系统中，所有的资源都是不可转移的，这就使得事情变得简单了。只有代理能在机器之间传递。因为代理已经被全局引用命名，即：一个（address,local-id）对，在提供迁移的情况下转换引用到资源，在 D'Agents 中就相对比较简单了。

21、将 D'Agents 的体系结构与 FIPA 模型所实现的代理平台的体系结构进行比较。

答：它们之间的主要区别是，D'Agents 没有单独的目录服务，事实上，它只提供了低级的命名服务，通过这个服务，代理能被全局引用。FIPA 体系结构中的管理组件对应 D'Agents 中的服务器，然而 ACC 是由通信层实现的。与 D'Agents 相比，FIPA 没有提供代理的体系结构的更进一步的信息。

22、代理通信语言(ACL)在哪些方面与 OSI 模型相符？

答：这些语言是应用层的一部分。

23、在系统顶层实现代理通信语言以进行电子邮件处理时，这中代理通信语言又有哪些方面与 OSI 模型相符？这种方案有什么优点？

答：它应该还是应用层的一部分。在 e-mail 的顶层实现 ACL 的一个重要原因是简单。一个完整，全世界范围内的通信基础设施对于处理代理之间的异步消息传递来说是可使用的。实际上，它与第二章讨论的消息队列系统很相近。

24、为什么 ACL 消息中通常必须指定实体(ontology)？

答：在这个特定的环境下，实体最好被解释为对 ACL 消息中包含的实际数据的标准解释的一个引用。通常，消息传递系统中的数据总是被假定能被消息的发送者和接收者正确地解释。代理经常被认为是高度独立的。因此，不能总是假定接收方能正确地解释接收到的数据。当然，对实体域的解释形成一些约定是必须的。

第五章 同步

1、说出至少三种可在 WWV 广播时间和在分布式系统中处理机设置内部时钟之间引入的延迟源。

答：信号在大气中的传播延迟，当机器在协调 WWV 接受者与以太网时的碰撞延迟，包在局域网上的传播延迟，各个处理器由于中断处理延迟和内部队列延迟所产生的延迟。

2、考虑分布式系统中的两台机器的行为。这两台机器的时钟假设每毫秒滴答 1000 次。但实际上只有一台

是这样的，另一台每毫秒滴答 990 次。如果 UTC 每分钟更新一次，那么时钟的最大偏移量是多大？

答：第二台时钟每秒滴答 990，000 次，每秒的偏移量为 10ms，所以一分钟偏移量为 600ms。

3、向图 5.7 中加入一个与消息 A 并发的新消息，即它既不发生在 A 的前面也不发生在 A 的后面。

答：在 0 时刻从进程 2 发送消息，并且在 8 时刻到达进程 1，或者是在 0 时刻从进程 1 发送消息并在 10 时刻到达进程 2。这两种方法都可以满足要求。

4、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：不是。只要消息的时间戳比所接收的消息的时间戳大，就有足够的时间多播其他类型的消息。将消息 m 传送到应用程序的条件是：以一个很大的时间戳从其它所有进程那里接受到其他消息。这保证了进行中的消息都使用更小的时间戳。

5、考虑一个消息只是以它们被发送的顺序被传递的通信层。给出一个不需要这种限制的例子。

答：想象传输一个大的图像，图像被分成连续的块。每块以它在原始图像中的位置标识，可能还有高度和宽度。如果是那样的话，FIFO 的顺序就不是必要的，因为接受者可以很容易地把到来的块拼接到正确的位置。

6、假设两个进程同时检测到协调者崩溃了，并且它们都使用欺负算法主持一个选举。这时将发生什么。

答：每个更高位置的处理器都会收到两个 ELECTION 消息，但是会忽略掉第二个。选举会象往常一样进行下去。

7、在图 5。12 中，我们有两个 ELECTION 消息同时在循环，当对它们都没有什么不利影响时，杀掉一个将更好。设计这样一个算法，该算法在不影响基本的选举操作算法的情况下完成此项任务。

答：当一个进程接受到 ELECTION 消息，它会检查消息是谁开始发送的，如果是它自己开始的（例如它的位置在列表的首位），它会把消息变成协调者消息并在正文里描述它。如果消息不是它开始的，它会加入其进程号并沿着环向前发送。然而，如果它更早地发送了自己的选举消息并发现了竞争者，它会将创始人的进程号与自己的相比较。如果其他进程拥有较小的号码，它会丢弃那个消息而不是传递消息。如果竞争者更大，那么消息将以平常的方式发送。这样，如果多个选举消息被开始发送，那么入口最大的消息将会幸存，其他消息将沿着路由被忽略掉。

8、许多分布式算法需要使用协调进程。讨论一下，这样的算法实际上可以在什么程度上被看作是分布式的？

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。

9、在集中式互斥方法中（图 5。13），协调者在收到一个进程释放它独占访问的临界区的消息后，通常给等待队列中的第一个进程授权，允许其访问临界区。请给出协调者另一个可能的算法。

答：请求会和优先权联系在一起，优先权取决于他们的重要性。协调者应该保证最高优先级的最先请求。

10、请再考虑图 5。13，假设协调者崩溃了，这会必然导致整个系统瘫痪吗？如果不会，那么在什么情况下会呢？有什么方法避免了这个问题的发生，使系统能够忍受协调者的崩溃？

答：在允许和拒绝的条件下，假定算法是只要有请求，就立即回应。如果没有进程在临界区并且没有进行在排队，那么这个崩溃不是毁灭性的。下一个请求准许的的进程将不会获得任何回应并且初始一个协调者的选举。在发送回应之前，使用协调者存储每个到来的请求将会使系统更加健壮。这样的话，在崩溃事件中，新的协调者将会重建一个活动临界区域列表并将从磁盘读文件的行为排队。

11、Ricart 和 Agrawala 算法会有这样的问题：如果一个进程崩溃，并且没有对另一个要求进入临界区的进程的请求作应答，没有应答意味着拒绝请求，我们建议所有的请求应用立即被应答，以便很容易地检测到崩溃的进程，是否存在一些情况，即使使用这种方法也还不足够？请讨论。

答：假定一个进程在拒绝许可并在那时崩溃。请求进程会认为它是活动的，但许可永远不会到来。一种方法为是请求者并不真正阻塞，而是休眠一定的时间。在休眠后，请求者将会测试所有拒绝许可的进程是否还是活动的。

12、如果我们假设该算法可以在支持硬件广播的 LAN 中实现，那么图 5。16 中的实何时将如何改变？

答：仅仅只有分布式案例的实体改变。因为发送一个点对点的消息的花费与广播消息一样多，所以我们只

需要发送一个广播消息到所有的进程去请求实体到临界区。同样地，仅仅一个出口的广播消息是需要的。延迟变成 1+(n-1)：一个延迟来源于广播请求，再加上 n-1,因为在被允许进入临界区之前，我们仍然需要从其他每个进程接受到消息。

13、分布式系统可能有多个互相独立的临界区，假设进程 0 想进入临界区 A 而进程 1 想进入临界区 B，Ricart 和 Agrawala 的算法会导致死锁吗？请解释原因。

答：这取决于程序的基本规则。如果进程严格地依次进入临界区域，也就是说一个临界区内的进程不会试图进入其他进程的临界区，那么当它拥有其他进程进程想要获取的资源（比如临界区）的时候，它是不会阻塞的。系统于是死锁释放。另一方面，如果进程 0 进入临界区 A 并且试图进入临界区 B，如果另外的进程以相反的顺序获取它们时，死锁就会发生。因为每个临界区被所有的进程独立处理，所以 Rucart 和 Agrawala 算法不会造成死锁。

14、在图 5。17 中，我们看到一种用磁带实现对存货表自动更新的方法。因为磁带可以很容易地用磁盘上的一个文件来模拟，你认为为什么不使用这种方法（用磁盘文件代替）呢？

答：第一个原因大概是人类变得更加贪婪和想要做更多他们过去习惯的东西。如果用户满足于每天运行一次简单的保存详细目录，可以用磁盘。问题在于每个人都想即时访问数据库，这会使得将详细目录保存在磁带是不可能的。

15、在图 5。25（D）中有三个调度策略，两个合法，一个非法的。对于同一个事务处理，给出临终 x 所有的可能值和合法的与非法的状态的一个完整表。

答：合法的值为 1，2，3。非法的值为 4，5，6。4 可以通过中间的事务处理来实现，其他两个的交错是不正确的。5 在策略 3 中实现。6 可以通过首先三次设置 x 为 0，然后三次增加它实现。

16、当一个私有工作空间用于实现事务处理时，可能需要将大量的文件索引复制到父辈工作区，怎样在不引入竞争条件下实现这种操作？

答：一种办法为在系统顶层设置锁以阻止所有的活动直到所有的索引都被重写。在预防崩溃以前，创建一个目的表也许是明智的。

17、给出一个完整的算法判定一个试图锁住一个文件的操作是否成功。要同时考虑到读锁和写锁，以及文件被解锁，读锁写或写锁写的可能性。

答：算法如下：如果文件没被锁，那么操作通常是成功的。如果文件被读锁定并且操作是另一个读锁定，那么操作也是成功的。在其他情况，操作是失败的（例如，写锁定一个被锁文件会失败，当一个文件被写锁定时，读锁定会失败）

18、用锁定的方法实现并发控制的系统通常区分读锁和写锁，如果一个进程已经得到了一个读锁但现在又想将它换成写锁，将会怎么样呢？

答：一个读锁只有在没有其它读进程的情况下才可能被转换成一个写锁。一种有效的方法就是先释放这个读锁，然后马上再获得一个写锁。如果有其它的读进程的话这个操作将会失败。如果有其它进程正在等待获得一个写锁时，这个操作是否成功取决于具体的策略，而不是技术上的问题。将一个写锁降低优先级为一个读锁问题可行的，并且不会出错。

19、在分布式事务中使用时间戳排序，假设写操作 write(T1,x)被传给数据管理器，因为仅有的可能发生冲突的操作 write(T2, x)的时间戳较早，为什么让调度管理器推迟传递 write(T1,x)直到 T2 完成呢？

答：通过推迟到 T2 完成，调度器可以避免由于 T2 运行的失败而造成的其它一系列的运行失败。

20、乐观并发控制与使用时间戳相比是更严格还是更不严格呢？为什么？

答：它更加严格。因为通过使用乐观并发控制的机制，如果一个事务正在交付并且发现另外一个事务也在更改当前事务使用的文件，它问题会退出。如果使用时间戳的话，如果另一个事务的时间戳更低的话，这个事务有时候可能会成功。

21、使用时间戳来进行并发控制能保证串行性吗？请讨论。

答：可以的。它要么按照事务的时间戳全部串行地完成，要么只完成一部分。

22、我们常说事务被中止时，一切都恢复到它以前的状态，就好像事务从来没有发生一样，实现上我们在说谎。请给出一个重新恢复一切是不可能的例子。

答：物理 I/O 发生的事务不能够被重置。比如说，一个进程打印出了一些数据，这些墨不能够从纸上移掉。

同样，一个控制所有进程的系统也不大可能会恢复一切事务到它们以前的状态。

第六章 一致性和复制

1、对共享的 JAVA 对象的访问可以通过将其方法声明为同步的而被串行化.当这种对象被复制时,这种方法足以保证访问的串行化吗?

答: 不能.问题是对每一个复制对象的访问是串行化的.但是对不同的复制对象可以在同一时间进行不同的操作,使得复制的实例变量不一致.

2、对于第一章所讨论的监视器,如果允许在一个复制的监视器中阻塞多个线程,那么给条件变量发信号时,需要保证什么?

答: 在每一组复制线程中,相同的线程会被唤醒,所以所有的复制对象都发生了完全相同的流控制.实际上,这意味着运行时系统应该完全控制着每一组复制对象的线程的运行过程.

3、请用自己的语言解释实际考虑一致性模型的主要原因.

答: 弱一致性模型的出现是为了满足性能复制的需要.但是,只有当我们能够阻止全局同步, 复制才会有效。而要达到这样的目的,就得放松一致性的限制。

4、请解释 DNS 如何进行复制，以及它实际运行很好的原因。

答：基本思想是域名服务器预先查询结果。结果可以存在高速缓存中很长一段时间，因为 DNS 认定主机名——IP 地址的映射不会常常改变。

5、讨论一致性模型时，我们经常提及软件和数据存储间的合约。为什么需要这一合约？

答：如果程序期望连续的一致性的数据存储且不能缺少这些任何数据，存储必须保证连续一致。但是，为了提高性能，一些系统提供弱一点的模式。事实上软件能遵守模式强加的规则。总的说来，这意味着遵守规则的程序可以感知到什么看起来像是连续一致的数据存储。

6、线性化假设存在一个全局时钟。但是，我们已经在严格一致性中指出，这种假设对于大多数分布式系统都是不现时的。线性化可以应用于物理分布的数据存储吗？

答：可以。线性化假设放松了同步时钟，就是说，它假设几个事件发生在同一时间段。这些事件需要根据时间分类。

7、如果多处理器使用单一总路线，那么可以实现严格一致的存储器吗？

答：可以。总线连续请求因此它们以绝对的时间顺序出现在内存中。

8、对于图 6。5（B），为什么 W1（x）a R2（X）NIL R3（X）a 是非法的？

答：它违背了数据一致性。

9、在图 6。7 中，000000 是仅满足 FIFO 一致的分布式共享存储器的合法输出吗？请解释你的答案。

答：是的。假设（a）第一个运行。它打印 00。现在（B）运行。如果存入（A）的还没有到达，它也打印 00。现在（C）运行。如果前面的两个在存储的都没到达，它也打印 00。

10、在图 6。8 中，001110 是顺序一致的存储器的合法输出吗？请解释你的答案。

答：是的。如果程序以（a）（c）（b）的顺序运行，就能得到这个结果。

11、我们在 6。2。2 节的最后讨论了一个形式模型，该模型规定顺序一致的数据存储上的每个操作集合可以用一个字符串 H 表示，从这个字符串 H 可以衍生出所有的单独进程的顺序序列。对于图 6。9 中的进程 P1 和 P2，给出 H 所有可能的值。忽略进程 P3 和 P4，也不包括它们在 H 的操作。

答：很明显P2在被写之前不能读1，因此H的所有值都将在读1 前写入1。程序的顺序是有关的，因此写2一定是在写1之后。唯一可能出现的结果是：

H = *W*(*x*)*a* *R*(*x*)*a* *W*(*x*)*b* *W*(*x*)*c*

H = *W*(*x*)*a* *R*(*x*)*a* *W*(*x*)*c* *W*(*x*)*b*

12、在图 6。13 中，一个顺序一致的存储器允许 6 种可能的语句交叉。请列出这 6 种所有可能。

答：(1) a=1; if (b==0); b=1; if (a==0);

(2) a=1; b=1; if (a==0); if (b==0);

(3) a=1; b=1; if (b==0); if (a==0);

(4) b=1; if (a==0); a=1; if (b==0)

(5) b=1; a=1; if (b==0); if (a==0);

(6) b=1; a=1; if (a==0); if (b==0);

13、通常认为弱一致性模型给程序开发人员强加了负担。这一命题在哪个程度上确实是正确的？

答：这要看情况。许多程序员习惯于使用同步机制例如锁或通信的方式来保护他们的共享数据。主要意图是他们需要一种能提供只进行读或写操作的方式而不是两者同时进行。但是，程序员非常期望对同步变量的操作一致连续。

14、在分布式共享存储系统中的释放一致性的大多数实现中，共享变量是在执行释放操作时被同步的，而不是在执行获取操作时被同步的。那么，为什么还需要获取操作呢？

答：获取操作需要用于延迟进程当它试图访问共享变量时而另一个进程正在对共享变量进行此操作。

15、ORCA 提供顺序一致性还是入口一致性？请解释你的答案。

答：形式上，ORCA 只提供入口一致性，因对相同对象的并行操作是完全序列化的。但是，当使用完全顺序广播用为操作排序的实施方法时，所有操作顺序都是全局安排的，并取决于它们操作的对象。在这种情况下，它提供顺序一致性。

16、使用序列器并为维持主动复制的一致性而进行的全序的广播是否违背系统设计中的端到端理论？

答：是的。端到端理论表明了问题应该在它们发生的相同层次上去解决。既然这样，我们以全序广播的方式处理问题以达到主动复制的一致性。在基本协议里，一致性是通过趋向原始作为第一性来达到的。而如果使用一个序列器，我们可以做到相同的事但在更低的一抽象层次。在这种情况下，可能使用原始协议更好因为它的升级是通过广播发送操作。

17、您会选择哪种类型的一致性来实现电子股票市场？请解释您的答案。

答：因果一致性应该能满足需要。问题是对于股价的变化，反应应该是连续的。独立股票的变化应该能以不同的顺序显示出来。

18、如果要将一个移动用户的个人邮箱作为广域范围的分布式数据库的一部分来实现，哪种类型的以客户为中心的一致性最为合适？

答：所有的类型都可以。关键是不论是写还是更新，邮箱对于用户来说应该都是一样的。对于这种邮箱来说，最简单的实现应该是基于主备份的本地写协议，该协议中主备份总是位于用户的移动电脑上。

19、请描述出一个用于显示刚被更新的 Web 页面的写后读一致性的简单实现。

答：最简单的实现是让浏览器检查它所显示的是不是最新版本的页面，这需要发送一个请求到 Web 服务器。由于许多系统都已经实现了该方案，所有它非常简单。

20、请给出以客户为中心的一致性易于导致写操作冲突的案例。

答：在描述以客户为中心的一致性模型时，假设每一个数据项都只有一个拥有着，只有该拥有着具有写权限。如果放弃这个假设的话肯定将导致写操作冲突。例如，如果数据的两个独立用户最终都被绑定到了同一个副本上，他们各自的更新都需要被传播到这个副本上。这是，更新冲突就很明显了。

21、使用租用时，客户和服务器的时钟分别紧密同步是否是必要的？

答：不需要。如果客户端认为它的时钟和服务器的时钟同步有困难时，它会在当前服务器的租借到期之前获取一个新的租借。

22、考虑一个用于保证分布式数据存储上顺序一致性的非阻塞的主机备份协议。这样的数据存储总能提供写后读一致性吗？

答： 不能。如果更新进程接收到它的更新请求已被处理的通知，它就从数据存储断开并重新连接到另一个副本。更新是否应用在副本上则没有通知。相比来说，如果使用一个阻塞协议，更新进程仅在他的更新请求被完全应用在其它副本上时才断开连接。

23、为了使主动复制正常工作，所有操作必须以相同的顺序在每个副本处被执行。这个顺序总是必需的吗？

答：不是。对于那些对只读数据的操作或者可交换的写操作，从理论上来说，这些情况允许不同的副本有不同的顺序。但是，要检查两个写操作是否为可交换的十分困难。

24、为了使用序列器实现全序的广播，一种方法是先将操作转发给序列器，然后该序列器为这个操作分配一个惟一的号码，随后广播这个操作。请提出另外两种可选择的方法，并比较这三种方法。

答：另一种方法是对操作进行广播，但是延迟发送直到序列器为该操作广播一个 iexuliehao。后者在该操作被序列器接收后发生。第三种方法是首先从序列器获取一个序列号，然后广播这个操作。
第一种方法（将操作发送给序列器），涉及到发送包含操作信息的点到点的消息和一个广播信息。第二种方法需要两个广播消息：一个包含操作信息，另一个则包含序列号。第三种方法需要一个包含序列号的点对点的消息，和一个包含操作信息的广播消息。

25、一个文件被复制在 10 个服务器上。请列出表决算法允许的所有读团体和写团体。

答：下面这些是合法的：
(1,10), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8, 3), (9, 2), and (10, 1).

26、为防止复制的调用，我们在书中讨论了基于发送方的方法。在基于接收方的方法中，是由接收副本辨认属于同一调用的输入消息的多个拷贝。请描述基于接收方的方法如何防止复制的调用。

答：假设对象 A 调用对象 B，如以前，每一次调用指定的 A 的副本相同的唯一的标志。每一个副本多点调用并请求 B 的副本。当 B 的副本接收到调用请求时，它检查它是否已经接收到过 A 的副本的请求。如果没有，调用就会在本地副本上执行，然后对 A 的副本发一个回应的广播。如果调用已经执行，请求信息就会进一步被忽视。同样地，A 的一个副本只有在获得对象的本地副本的显著调用请求第一个引入回应时才执行，而此后相同的调用请求都会被简单地忽视。

27、考虑在因果一直的懒惰复制中，一个操作被从一个写队列中删除的确切时间。

答：当知道此操作已经在各处都被执行，这可以通过追踪每一个 Lj 中的副本最新的一次升级来检测，如果每一个副本都执行过了来自 Li 的 W 操作，LI 可在更小的时间内先移去 W2 的所有操作，（比如：每一个 $W2[k] \delta W[k]$）。因此，每个 LJ 发送一个回应给 LI 其中包含最近执行的操作的时间。

28、在实现一个支持广播 RPC 的简单系统时，我们假设系统有多个复制的服务器，每个客户可以通过 RPC 与一个服务器通信。但是，处理复制时，客户需要向每个副本发送一个 RPC 请求。设计客户程序，以便客户好像只向应用程序发送单一的 RPC。假设复制的目的是为了提高性能，而那些服务器可能是易于出故障的。

答：无。