第一章

1. 分布式系统的另一个定义，它是各自独立的计算机的集合，这些计算机看起来像是一个单的系统，就是说，它对用户是完全隐藏的，即使他有多个计算机也是如此。请给出一个实例。

答：并行计算。一个程序在一个分布式的系统中运行，但看起来是在单个系统中运行的。

2.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，即使种类各异的计算机和网络都呈现为单个系统。

3. 很多网络系统组织成后端办公系统和前端办公系统。这种组织方式是如何满足分布式系统要求的？

答：一个比较容易犯错的地方就是假设运行在一个组织下的分布式系统，应该运行在系统的整个组织框架下。实际上，分布式系统被安装在一个分离的组织中。从这层意义上讲，我们的分布式系

统可以支持独立的后端处理和前端处理。当然，这两部分可能是耦合的，并不需要要求这个耦合的部分完全透明。

4. 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：分布式系统中的多个计算机系统运行可能是不同的操作系统，这些操作系统的文件命名方式不同，命名方式的差异以及由此引发的文件操作方式的差异应该对用户和应用程序隐藏起来。 位置透明：从 http://www.prenhal.com/index.htm 这个 url 看不出 parentice hall 的主 web 服务器所在的位置，同时也看不出 index.html 的位置情况。 移植透明：分布式系统中的资源移动不会影响该资源的访问方式。 重定位透明：资源可以在接受访问的同时进行重新定位，而不引起用户和应用系统的注意。移动通信用户从一个点到另一个点，可以一直使用移动设备，而无中断连接。 复制透明：对同一个资源存在多个副本这样一个事实的隐藏。所有的副本同名。 并发透明：访问位于同一个共享数据库中的一批表。 故障透明：用户不会注意到某个资源无法正常工作，以及系统随后的恢复过程。 持久性透明：指对转位于易失性的存储器还是在磁盘上的隐藏。许多面向对象的数据库提供直接调用存储对象的方法的功能。

5. 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。例如，连接一台繁忙的 web 服务器，浏览器超时，报告该 web 页不可用，这种情况下，用户无法判断该服务器是否真的崩溃了。

6. 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。 例如，许多 web 程序会不断尝试连接某台服务器，多次失败后才放弃。这种在用户转向另一台服

务器之前竭力隐藏服务器短暂故障的企图会导致整个系统变慢。

7. 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。

8. 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。 系统要能在规模上可扩展，即方便的把更多的用户和资源加入到系统中。 地域上可扩展，系统中的用户和资源相隔极远，但仍可访问和使用。 管理上可扩展，分布式系统跨越多个独立的管理机构，仍可方便对其进行管理。

9. 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：可扩展性可以通过隐藏通信等待时间，分布式技术、复制和缓存来获得。

10. 请解释一下什么是虚拟组织，并给出如何实现这种组织的提示。
答：属于同一虚拟组织的人或应用，具有访问提供给该组织的资源的权限。这些资源可能分布在不同的计算机、服务器、数据工具和数据库中。虚拟组织定义了谁能访问什么。 资源应该保持一个账户用户和访问权限，这通常使用了标准的访问控制机制（类似 UNIX 中的 rwx）。

11. 当一个事务处理被异常中断，将会恢复到它以前的状态，就像这个事务处理从没有发生过一样。我们这么说其实是不对的。请给出一个无法恢复以前状态的示例。
答：任何情况下，物理端的 I/O 中断，不能重置。例如，如果一个进程是来打印文件，文件中的墨汁不能从纸中移除。

12. 运行嵌套式事务处理需要某种形式的协作，请解释一下协作者真正应该做什么事情？
答：协作者只需要保证嵌套式的事务中止后，所有的子事务都要被中止。同样，如果事务都能够被提交，那么就应该协调他们全部都提交。当协调者告诉嵌套的事务该提交时，事务理解进行提交操作。

13. 我们讨论过，对普适系统来说，分布式透明性可能并不存在。这句话并不是对所有的透明性都正确，请给出一个示例。
答：考虑迁移透明性。在普适系统中，组件是可移动的，当从一个接入点移动到另一个接入点时，它可被重新连接。这样的处理应该对用户完全透明。同样，许多其他类型的透明性也会被支持，然后不该隐藏的是用户可能访问的资源与用户本身所处的环境是耦合的。

14. 我们已经给出了一些分布式普适系统的示例：家庭系统、电子健保系统和传感器网络，请再给出这样的一个示例。
答：考虑在城市或社区中提供大规模的无线网状网络服务的普适系统，它提供无线上网服务，当然这个也为其他服务，比如新闻系统提供了通信基础。栖息地的监控系统、监狱中的电子监控系统、大型综合体育系统、了解员工状态的办公激励系统等。

第二章

1. 如果客户和服务器相隔很远，我们可以看到，网络延时将主导整个性能。我们如何处理这个问题？

答：这实际上取决于客户端是如何组织起来的。首先，将客户端的代码切割成更小的部分，这样它们可以单独运行。这种情况下，当一部分正在等待服务器的响应时，我们可以安排另一部分去做其他的事。或者，我们可以重新安排客户端的处理过程，让他们在向服务器发送客户请求之后去做其他的事情。　最后有效的解决方案是用单程异步通信取代了客户端-服务器的同步通信。

2. 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：用户接口层、处理层和数据层，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。

3. 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，通过按逻辑把不同组件放在不同的机器上来获得。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，客户或者服务器可能在物理上被分隔成逻辑上相对的几个部分，每个部分都操作在整个数据集中自己共享的部分。例如一个数据库的分布。

4. 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。

5. 在结构化的覆盖网络中，消息是根据覆盖的拓扑结构来路由的，这种方法的主要缺点是什么？

答：问题是我们处理的是逻辑路径。很有可能出现的情况是，在覆盖网络中的两个邻居节点 A 和 B，在物理距离上是相距很远的。因此，A 和 B 之间的逻辑最短路径实际上在底层物理网络上是很长的一条路径。

6. 图 2.8 中的 CAN 网络，如何把坐标（0.2,0.3）的节点的消息路由到坐标为（0.9，0.6）的节点？

答：有几种可能性。如果我们想要根据欧几里得距离得到一条最短路径需要进过的路径是：（0.2,0.3）----（0.6,0.7）----（0.9,0.2），距离是 0.882. 另一条路径：(0.2,0.3)---(0.7,0.2) ----(0.9,0.6)，距离是 0.957

7.假设 CAN 中的一个节点知道其紧邻节点的坐标，一个合理的路由策略是把消息路由给离目的地最近的节点，这样的策略的好处是什么？

答：从先前的问题可以看出，它不需要去寻找整体最优的路径。如果节点（ 0.2,0.3）按照题目要求的策略进行，传送到（0.9,06），它先会将节点发送给（0.7,0.2）。

8. 在非结构化的覆盖网络中，每个节点随机选择c 个邻接节点。如果P 和 Q 都是 R 的邻接节点，那么 P 和 Q 互为邻接节点的概率是多少？

答： 如果网路中有 N 个节点，如果每个节点随机选择 c 个邻居，那么 P 选择 Q 或 Q 选择 p 的概率就是 2c/(N-1)。

9. 在非结构化的覆盖网络中，每个节点随机的选择c 个邻节点。要查找一个文件，节点将泛洪一个请求给他的邻接点，这些请求又将再次泛洪，该请求将到达多少个节点？

答： 可到达节点的上界是 c*(c-1)，但是这我们忽略了节点 p 同样可以成为其他邻居节点的邻居。P 的一个邻居泛洪信息给不是 P 邻居节点的概率是 1 减去发送给至少 P 的一个邻居节点的概率。

$$q = 1 - \sum_{k=1}^{c-1} \binom{c-1}{k} \left(\frac{c}{N-1}\right)^{k} \left(1 - \frac{c}{N-1}\right)^{c-1-k}$$

这个情况下，泛洪策略将会到达 c*(q-1)个节点。例如，c=20, N= 10000 ,那么一次查询将会到达的节点数是 365.817 个节点。

10. 在点对点的网络中，并不是每个节点都能成为超级对等体。满足超级对等体的合理要求是什么？

答：首先，节点是高度可用的，因为其他节点需要依赖该节点。并且，它应该有足够的能力处理各种请求。最重要的是依靠它能够高效快速的处理任务。

11. 在 Bitgtorren 系统中，每个节点具有一个带宽为Bout 的输出链接，以及一个带宽为Bin 的输入链接。这些节点（称为种子节点）中的一些可以自愿的为其他节点提供下载文件。如果假设在某个时候，一个BitTorrent 的系统最多只有一个种子节点，那么他的最大下载带宽是多少？

答：这里需要考虑种子节点的输出带宽需要在客户端间共享。假设有 s 个种子，N 个客户端，每个客户端随机的选择种子。种子节点结合的输出带宽就是 S*Bout ， 每个客户端有 S*Bout/N 的直接下载带宽。另外，如果这些客户端互相协作，每个能够在下载数据块时达到 Bout，假定 Bin > Bout。因为 tit-for-tat 策略，一个 BitTorrent 客户端的下载带宽主要取决于他的输出带宽。所以说，整体的下载带宽能够达到 S*Bout/N + Bout.

12. 请从技术的角度来解释，为什么 BitTorrent 中的 tit-for-tat 策略比因特网中的文件共享要好得多？！

答：大多数的 BitTorrent 客户端是被例如 ADSL 或调制解调器所提供的不对称链接所处理的。一般来说，BitTorrent 会提供给客户端比较高的进入带宽，但是并不希望客户端可以提供服务。BitTorrent 不会使用这种假设，把客户端变成协作服务器。对称连接会比 tit-for-tat 策略提供更好的匹配。

13. 我们给出了在自适应中间件中使用中断器的示例，请问还有其他什么示例？

答：我们利用拦截器来支持移动性。这种情况下，一个 request-level 拦截器在转发请求前，

首先将查看它所引用的对象的当前位置。同样，当安全出现问题时，一个拦截器可以用来透明的加密消息。另一个例子，如果记录日志是必须的，我们可以简单的插入一 method-specic 拦截器在将消息传递给引用对象前，记录该事件。

14. 中断器与其部署所在的中间件的依赖程度如何？
答：一般来说，它们之间是高度依赖的。客户端很有可能被中间件所提供的下层接口所约束，正如消息级中断器，它高度依赖中间件和本地操作系统的相互作用。然后，这些接口也可以标准化，便于开发可移植的拦截器，虽然它作为一种特殊类型的中间件。这些中间件需要满足 CORBA 的体系结构。

15. 现代的汽车都装备有电子设备，请给出在汽车中的一些反馈控制系统的示例。
答：巡航控制系统。一方面，巡航控制的子系统掌控着当前的速度，当它达到设定要求时，汽车就会加速或减速。巡航控制系统的设置使驾驶员可以将车速设定在一个固定的速度上，车辆准确地按照所设定的速度行驶。驾驶员可以不必踩加速踏板，从而大大减轻长途驾车的疲劳，同时匀速行驶也可以减少燃油的消耗。 防抱死制动系统（ABS），当刹车时，它是利用阀体内的一个橡胶气囊，在踩下刹车时，给予刹车油压力，充斥到 ABS 的阀体中，此时气囊利用中间的空气隔层将压力返回，使车轮避过锁死点。防抱死刹车系统可以提高行车时，车辆紧急制动的安全系数。换句话说，没有 ABS 的车，汽车在遇紧急情况采取紧急刹车时，容易出现轮胎暴死，也就是方向盘不能转动，这样危险系数就会随之增加，很容易造成严重后果。 车用传感器是汽车计算机系统的输入装置，它把汽车运行中各种工况信息，如车速、各种介质的温度、发动机运转工况等，转化成电讯号输给计算机，以便发动机处于最佳工作状态。

16. 请给出自我管理系统的一个示例，其中的分析组件为完全分布式或隐藏的。
答：我们已经遇到很多这种类型的系统，在非结构化的点对点系统中，节点间在进行信息交换，如何生成一个拓扑结构。其分析组件包含放弃那些不会帮助目标拓扑结构收敛的链接。

17. 请描述一个解决办法，为Globule 中的预测复制自动确定最佳跟踪路径长度。
答：源服务器需要利用 Ti 到 Ti+1 的迹线来定期评价策略的预测。判断该策略是否是实际访问模式所选择的在 Ti-1 到 Ti 阶段所使用的策略。这样可以帮助服务器计算预测误差。通过改变迹线长度，源服务器能够自动找到预测长度最小的迹线。这样，就可以自动的确定最佳跟踪路径长度。

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 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. 略

8. 略

9.代理可以通过调用所有副本来支持引用透明性，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 调用另一个对象副本 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。这种复制应该采取一些特殊的措施。

10.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有点也有缺点。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。

11.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。

12.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。

13.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。

14.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。

15.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。

16.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。

17.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。

第四章　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。

10.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。

11.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。

12.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。

13.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。

14.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。

15.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。

16.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。

17.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。

18.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。

19.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。

20.略

21.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。

22.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。

23.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 10 Δ 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 50 Δ tMB，等于 (1+10 Δ t)。因此， Δ t 等于 25 毫秒。

第四章

17： 如果由用户来指定缓冲区的大小，可以让缓冲区的管理变得更加简单。系统创建了一个指定大小的缓冲区，缓冲区管理变得简单。但是一旦缓冲区被占满，消息就可能丢失。可选的方案就是通过消息传递机制来管理缓冲区空间，最开始创建的时候有一个默认的大小，但可以根据需要扩张或者缩小空间。这个方法减少了由于缺少空间而丢失消息的情况的发生，但是需要系统做更多的工作。

18： 问题在于有限的地理空间上的可扩展性。由于同步通信要求在发送的消息被接受之前，发送方会处于阻塞状态，如果接收方离得很远，那么发送方在能继续工作之前会等待很久。解决这个问题的唯一方法是设计一个发送应用，这样就可以在等待过程中执行其他的操作，这样就建立起了一个有效的同步通信机制。

19： 将一份大文件传递给多个用户就是这种情况，比如说，为 web service 或者软件分发更新镜像网站。

20： 不一定，如果我们假定操作者仍然需要知道数据是何时测量的，那么在这种情况下，当测量数据接收的时候就会被加上时间戳，但是这意味着我们仍然需要确保最小化端到端的延迟。

21： 用一个令牌在环上循环，只有当计算机获得令牌的时候，才能通过环发送消息。同时，持有令牌的时间不超过 T，假设邻接计算机之间的通信有限制，那么令牌就会有一个最大化的循环时间，这对应着最大化的端到端延迟。

22： 奇怪的是，这要比确保最大化延迟难得多。问题在于，原则上来说，接收方计算机不应该在已播时间之前接收数据。唯一的问题在于需要尽可能长时间地缓存数据包。缓存可以在发送方、接收方、或者两者之间的位置，比如说中间站。暂存数据的最好的位置应该是在接收方，因为在这个点上没有其他的不可预知的阻塞数据传输的情况发生。接收方需要做的仅仅是将数据从缓冲区中移除，并且根据一个简单的时间机制来将它传递给应用。不足之处在于需要提供足够的缓存空间。

23： 这个问题主要是由 ISP（即网络服务提供商）造成的，因为他们没有理由去节省带宽（反正他们的用户已经为此付费）。不过，在下述场景中情况肯能会有所改变。网络多播服务为 ISP 所承诺的服务质量而付费。如果 ISP 无法满足质量服务的要求，那么他们的收益就会减少。这样的话，他们就有动力提供多播服务，来满足用户的需求。

24： 使用关系图的根本假设是通信延迟是影响性能的最大因素。不过在视频多播的例子中，可用性才是占主导的因素。在这种情况下，我们会构建成本最大化的树（通过带宽来衡量）。

25： 在 gossiping 中，节点之间相互交换信息，每一个节点最终都将获取其他节点的信息。每当发现了一个新的节点，都可以用与它在语义上的邻接点来评估，比如，通过计算共有的文件数目。那些语义上最邻近的节点将被用来提交查询请求。

第五章

1： IP 地址被用来定位主机。不过访问一个主机，它的 IP 地址需要被解析成，比如说，一个以太网地址。

2： 两个名字都是位置独立的，虽然第一个给出了关于位置的些许暗示。位置独立意味着实体的名称是独立于它的地址的，仅仅考虑一个名称，无法得到关联实体的地址信息。

3： 比如：书籍的 ISBN 码，软硬件产品的标识符，单个组织内的员工号，以太网地址（虽然一些地址是用来标识一台机器而不仅仅是以太网站点）。

4： 是的，但信息不允许修改，因为那将导致标识符的改变。旧的标识符需要保持有效，这样改变它将导致一个实体有两个标识符，违反了标识符的第二个属性。

5： 将机器的网络地址加上当地时间，产生一个伪随机数。尽管理论上来说，仍然有可能存

在着拥有相同数字的机器，但这几率可以忽略不计。

6： 节点 7：【9,9,11,18,28】。比如，2 号入口是这样计算的：succ(7+21)=succ(9)=9。更多的指状表需要进行改变，节点 4：【7,7,9,14,28】，节点 21：【28,28,28,1,7】，节点 1：【4,4，7,9,18】。

7： 在 N 个节点的系统中，超级体的个数等于 min{2k-mN,2k}。

8： 不需要。考虑前一个问题，假定只有节点 7 的指状表，而余下的保持原状。最坏的情况是，一个查询节点 7 的请求被导向了节点 9。不过，节点 9 知道节点 7 已经加入了系统，所以可以采取正确的动作。

9： 问题在于，如果没有结果返回，那么请求客户端将无法发现问题出在哪里。在解析键值的过程中，可能发生消息丢失或者节点失败的情况。因此动态查询有时更受欢迎，这样客户端将知道查询的哪部分出了问题，并可能求助另一个可用的节点。

10： 每一个服务都有一个唯一的标识符，任何执行这一服务的服务器，都将把自己的网络地址插入到它所属的子区域的目录节点的位置服务列表中。查询请求使用服务的标识符，并且将会自动返回执行这一服务的最近服务器。

11： 所有的主地址组合在一起，形成了根，并且按照相应的方法分离，即每一个移动的实体都有它自己的根服务器。

12： 对由实体标识符组成的域 D 进行简单编码，用于查询操作。操作可以快速定位到目录节点 dir(D)，从此处开始搜索。

13： 改变位置其实就是插入操作以及删除操作的结合。一次插入操作最坏情况下需要改变 k+1 条位置记录。同样的，一次删除操作也需要改变 K+1 条记录，并且根记录被两种操作共享。这导致 2K+1 条记录的改变。

14： 将分层的位置服务和前置指针结合起来。当实体移动的时候，它在 A 点留下了前置指针指向下一个（中间）位置。每次它移动，一个前置指针就留了下来。到达 B 点之后，实体将它的新地址插入分层位置服务中。指针链随后就被清除，并且 A 点的指针也被删除。

15： 一个重要的前提是我们仅仅使用随机的比特字符串来作为标识符。结果是，我们可以轻松地分割标识符空间，并且为每一部分分别生成相应的根节点。此外，分离的根节点需要跨越不同的网络，这样就可以分别访问根节点。

16： 假设已知一个过程是处理 URL 的，那么它会首先从 URL 中提取协议标识符，比如 ftp:。它随后在表中查找这个字符串，进而找到在本地执行 FTP 协议的接口。此闭合机制的另一工作就是从 URL 中提取主机名，比如 www.cs.vu.nl，并且将主机名传给本地的 DNS 名字服务器。了解从何处以及如何与 DNS 服务器通信也是此闭合机制的重要部分。这通常硬编码进 URL 名字解析器，解析器执行整个解析过程。最后，URL 的最后一部分，指向着查找的文件，被传给标识符主机。后者通过自己的本地闭合机制来开始文件名的名字解析。

17：硬链接是目录文件的一个条目，指向了同样的文件描述器，并成为另一个条目（可能是在不同的目录上）。符号链接是包含了另一个文件名的文件。而通过软链接，则可以链接到不同的磁盘分区甚至不同的机器。

18：可能不会。由于高层名称服务器构建了 DNS 名称空间的全局层级，可以认为对这一名称空间的改变很少发生。结果是，从高层名称服务器获取信息非常高效，并且可以避免很多耗时的通信。注意对低层名称服务器来说，递归名称解析是很重要的，因为在这种情况下，在较低层次的服务器域中，名字解析是在本地进行的。

19：移动主机的 DNS 名字将被用作标识符（虽然不是很好）。每当解析名称的时候，它将返回主机的当前 IP 地址。这意味着负责提供 IP 地址的 DNS 服务器将作为主机的名称服务器。每当主机移动，它就与主服务器通信，将自己的当前地址发送出去。注意需要设置一个避免抓取地址的机制，换句话说，其他的名称服务器不应该获取查找到的地址。

20：使用挂载表，它包含了一个指向挂载点的入口。这意味着当查找一个挂载点的时候，

我们需要遍历挂载表来找到与挂载点相匹配的入口。

21：是的，单用户名称空间所提供的名称可以解析为共享的全局名称空间的名称。比如，原则上来说，来自不同名称空间的两个相同的名称是完全独立的，并且可能指代不同的实体。为了共享实体，有必要使用共享名称空间的名称来访问它们。比如， jade 依赖于 DNS 名称和 IP 地址来访问共享实体（比如 FTP 站点）。

22：当代表名称服务器的节点 NS 和节点 N 不在一个子域中，仅仅提供域名就可以了。在这种情况下，可以通过分离的 DNS 查询找到名称。当 NS 和 N 在同一个子域中时，需要从名称服务器获取它的地址。　　23：可以在可能的时候查询实际的内容。在文档的情况下，可以试试信息提取的一些方法，比如矢量空间模型（ VSM）。

24：略

第六章

1：首先大气层存在信号传播延迟。其次由于以太网上有 WWV 接收器的机器会相互竞争，产生冲突，进而导致延迟。第三，LAN 上存在数据包传播延迟。最后，当数据包到达处理器，由于打断进程以及内部的队列延迟，每一个处理器内部也都有延迟。

2. 答：第 2 台计算机始终每秒钟发送 990000 次脉冲，误差为 10 毫秒，一分钟，这个误差会增至 600 毫秒。

3 .答: One typical example that comes to mind is sports and health care. There are now GPS-based body-area networks that allow a person to keep track of his pace while exercising an outdoors sports. These networks are often augmented with heart rate monitors and can be hooked up to a computer to downlaod the sensored data for further analysis. Another example is

formed by car-navigation equipment, which is generally based on GPS receivers. Hooked up to a (portable) personal computer that can be connected to the Internet, maps and such can be continously updated. Related are GPS-based distributed systems for tracking the movement of cars and trucks.

4. 答:The obvious reason is that there may be an error in the current reading. Assuming that clocks need only be gradually adjusted, one possibility is to consider the last N values and compute a median or average. If the measured value falls outside a current interval, it is not taken into account (but is added to the list). Likewise, a newvalue can be computed by taking a weighted average, or an aging algorithm.

5.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。

6 .答：不需要，因为只要发送消息的事件的时间戳在接收消息的事件的时间戳之前，就可以广播发送任意类型的消息。向某一应用进行发送消息 m 的条件是，当前进程已经收到了来自于其他进行时间戳更大的消息。这个条件保证了没有正在发送着的消息的有更小的时间戳。

7.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。

8.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.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.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.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.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.答：这取决于基础规则，如果说各个进程严格的按照线性时间顺序来访问资源，也就是说一个已经获取某种资源的进程不会在尝试获取另外一种资源，这样的活他也就没有办法在拥有另外一个进程打算获取的资源的时候去阻塞了。那么，系统就不会产生死锁了。另一方面，如果进程 0 在已经获取资源 A 的情况下又尝试获取资源 B，那么如果另外一个进程此时按照相反的顺序进行操作就有可能产生死锁。因此，算法本身并不会产生死锁，因为每一种资源的处理过程相互之间都是相互独立的。

14.答：当一个进程得到了一个 ELECTION 消息的时候，它将会核对是谁发送了这个消息。如果是它自己发起的，那么他将会把这个消息转换成一个 COORDINATOR 消息。如果它没有发起任何 ELECTION 消息，它将会增加自己的进程号然后沿着环路去传递它。然而，如果它确实较早的发送了他自己的 ELECTION 消息，并且同时发现了一个竞争者，那么它将会比较二者的进程号。如果说另一个进程有一个较小的进程号，他将会把这个消息删除而不是继续传递它。如果说另一个进程的进程号更大，这条消息将会按照正常的机制去传递出去。按照这种方式，如果多个 ELECTION 消息开始了，那么进程号最大的将会得到响应，剩余的将会被杀死。

第七章

1.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.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。

3. 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。

4. 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。

5.答: In this case it is relatively simple: if A and B exchange their list of tentative operations and subsequently order them according the time, then both would get to see the same result.

6.答：是的，如果这个进程按照（a），（c），（b），，这个结果就可以获取了。

7.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。

8. 答: Yes. The end-to-end argument states that problems should be solved at the same level in which they occur.In this case, we are dealing with the problem of totally ordered multicasting for achieving consistency inactive replication. In primary-based protocols, consistency is achieved by

first forwarding all operations to the primary.Using a sequencer, we are actually doing the same but at a lower level of abstraction. In this case, it may have been better to use a primary-based protocol in which updates are propagated by sending operations.

9.答: Causal consistency is probably enough. The issue is that reactions to changes in stock values should be consistent. Changes in stocks that are independent can be seen in different orders.

10. 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。

11.答：最简单的实现方式是让浏览器一直去检查它是否正在展示最近版本的页面内容。这要求浏览器向 WEB 服务器发送一个请求。由于这种实现机制已经被很多系统实现了，所以这种实现机制是简单的。

12.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，只有该拥有者具有写权限。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。

13.答：不需要，如果客户端对于其与服务器端同步的等级持一个悲观的态度，那么他将会在当前令牌到期之前去尝试获得一个新的令牌。

14.答: Lamport's way totally ordered multicasting requires that all servers are up and running, effectively hindering performance when one of them turns out to be slow or has crashed. This will have to be detected by all other servers. As the number of servers grows, this problem is aggravated.

15.答:正确性公式如下所示：

$$v(t) - v_i = \left(v(0) + \sum_{k=1}^{N} TW[k,k] \right) + \left(v(0) + \sum_{k=1}^{N} TW[i,k] \right)$$

$$= \sum_{k=1}^{N} \left(TW[k,k] - TW[i,k] \right)$$

$$\le \sum_{k=1}^{N} \left[TW[k,k] - TW_k[i,k] \right] \le (N-1) \times \delta_i / (N-1) = \delta_i$$

注意:我们可以在求和过程中将（N - 1）略去。

16.答：如果我们将正值更新和负值更新分离开，并且为各自做分别记录，那么情况就会简单一些。如下所示：

$$TWN[i, j] = \sum \{weight(W) \mid weight(W) < 0 \ \& \ origin(W) = S_j \ \& \ W \in L_i\}$$

$$TWP[i, j] = \sum \{weight(W) \mid weight(W) > 0 \ \& \ origin(W) = S_j \ \& \ W \in L_i\}$$

注意：TWP[i, j]恒等于 TW[I, j]。每一个节点都记录了视图 TWPK[i, j]和 TWNK[i, j]，并且当它注意到一个新的写操作会增加|TWN[k, k] – TWNK[i, k]| 或者|TWP[k, k] – TWPK[i, k]|时就会提交它的视图。

17、A:不一定。只要更新进程接收到更新正在被处理的确认，该进程就会从数据存储断开并连接另一个副本。但不能保证更新已经到达了那个副本。相比之下，如果存在阻塞协议，更新进程只能在更新被传播到其他副本时才会断开连接。

18、A:不必要。考虑一下在不可修改的数据或者交互的写操作上执行的读操作。这种情况允许在不同副本上有不同的排序。然而，这样就无法检测到两个写操作是否是交互的。

19、A:一种方法可以组播这个操作，但是延迟投递直到序列器(sequencer)为其组播了一个序列号。后一个操作发生在这个操作被序列器接收之后。第三种方法是首先从序列器获得一个序列号，然后组播该操作。第一种方法（向序列器传递操作）包括了一个伴随操作的点对点信息传递和一条组播信息。第二种方法需要两条组播信息：一个包含操作，一个包含序列号。第三种方法消耗一条包含序列号的点对点信息，随后传递包含操作的组播信息。

20、A:下列组合是合法的。（1，10）（2，9）（3，8）（4，7）（5，6）（6，5）（7，4）（8，3）（9，2）（10，1）。

21、A:不一定。有一些简单的原因使得一些客户端有必要在更新发生时受到提醒。如果不维持任何状态的话，这些客户端可能会经常性向已经十分繁忙的服务器投票（ poll）。

22、设计题。

第八章

1、 A:例如，如果因为第三方的恶意篡改导致服务器所提供的响应不可信赖，那么这个系统 就不能认为是可靠系统。同样，服务器也应该信赖其客户端。

2、 A:在实践中，服务器在宕机时停止输出，因而检测到服务器停机时比较困难的。而从另 一个过程中可知，服务器可能变得很慢或者通信失败。

3、 A:这个问题是否是故障取决于对用户的一致性是如何保证的。如果浏览器保证提供的页 面是至多 T 个时间单位之前，这种情况可能体现出了性能故障。然而，任何浏览器都不能满足这样的保证。一种一致性的弱形式可以提供第 7 章中所提到的客户端（client-centric）模型。这种情况下，如果仅仅从缓存中返回了页面但没有检查一致性，就可能导致响应故障。

4、 A:可以。这个讨论假设了失败的元素(element)会引出随机结果，这与 Byzantine 故障 是一致的。

5、 A:在每一行圆圈中，至多有一个元素会出故障并被掩盖。进一步说，每一组中的一个 voter 可能出故障并且在下一个阶段反馈一个出故障的元素。例如，如果六列顶端的元素（element）都出故障，2/3 的最终结果会是正确的，从而避免了 6 个故障所带来的后果。

6、 A:是的，奇数都可使用。再有 5 个元素(element)和 5 个 voter 的情况下，每个设备组 中最多有两个错误可以被掩盖。

7、 A:对(a)和(b)，(at-least-once) 策略是最好的。一遍遍尝试是可行并且无害的。对 (c)，最好只进行一次尝试。如果失败，用户必须进行干预来清理混乱状况。

8、 A:语义通常以与普通 RPC 相同的方式受到影响。区别是，当客户端被阻塞时，服务器端 并不会处理请求，如果客户端同时崩溃，这样就可能导致问题产生。相反，服务器端执行操作，并且过

段时间在必要的情况下联系客户端。

9、 A:用小片段组播图像，每个片段信息中包含了（x,y）坐标。又例如，发送一个数的页 面，每个页面都被记数了.

10、 A:不必要。在很多情况下，例如传输文件，只有当数据在应用层可用时以上要求才 是必要的。通信层没有必要保留自己的备份。

11、 A:取决于一个组内包含了多少进程。重点是，如果考虑到容错需要，进程被复制， 那么只需要几个备份样本就足够了。在这个情况下，可扩展性就不算是一个议题。只有当不同组别的进程生成时，可扩展性就会真正称为一个议题。而当为了性能进行复制时，原子组播就可能被过度使用（overdone）了。

12、 A:我们无法做出类似于当一份确认信息发送到客户端时，服务器就执行了操作之类 的保证。然而，使用原子组播策略可以提高容错度，使得开发一个容错系统更加容易。

13、 A:单独组播所实现的同步，完全地，保持因果地，先进先出处理。注意：视图变换 是作为特殊的组播信息发生的，也需要适当进行处理。

14、

A: There are six orderings possible:

Order 1	Order 2	Order 3	Order 4	Order 5	Order 6
m1	m1	m1	m3	m3	m3
m2	m3	m3	m1	m1	m4
m3	m2	m4	m2	m4	m1
m4	m4	m2	m4	m2	m2

15、 A:当进程 P 接收到 Gi+k，P 首先向其之前所属的视图和 Gi+k 中的每一个进程转发 任

何不稳定信息的副本，紧接着为 Gi+k 设置 flush message。这样，就可以安全地将信息标记为稳定的。　如果进程 Q 接收到的信息 m 是在 Gj 视图中传送的（j<i+k），那么当 Q 不在 Gj 时丢弃该信息。如果最近安装的视图是 Gl（l>j），m 也会被丢弃（复制信息）。如果 l=j 并且 m 还未被接收，进程 Q 在投递 m 是会考虑所有的额外信息顺序的限制。最后，如果 l<j，信息 m 就会被储存在通信层中直到 Gj 被安装。

16、A:选举之后，新的协调器（coordinator）也可能崩溃。这种情况下，剩余的参与者可能无法达到一个最终的决定，因为最终决定需要来自于新选举出来协调器的投票。

17、A:不对，需要注意的是一个不能参与到其他进程最终决定的恢复进程，将会恢复到一个与其他进程的最终选择相容的状态中。

18、A:通常来说，更多日志信息更好。这些日志信息包含了所有的非决策事件，包括本地 I/O 和系统调用。

19、A:日志信息记录了事务中的每一次读写操作。当故障发生时，日志可以重演（replay）至上一次有记录的操作。重演日志与回滚有些不同，回滚发生在事务需要被中止时。

20、A:取决于服务器端做了什么。例如，一个已经提交了完整事务的数据库服务器会保留一份日志，保证在恢复时可以重做所有的操作。然而，考虑到分布式系统的状态，没必要执行检查点操作。检查点操作只需要在本地恢复时执行。

21.A:主要的原因是恢复过程是完全本地的。在基于发送方的日志中，恢复过程需要联系发送方重新发送信息。