分布计算环境

北京邮电大学计算机学院

Chapter 2

分布式系统的基本原理

主要内容

- 体系结构
- 进程
- 通信
- 命名
- 一致性和复制
- 容错
- 安全

命名

- 名称在所有计算机系统中都起着非常重要的作用
  - 共享资源、唯一标识实体等
- 命名系统的主要功能是把名称解析为它所标识的实体的地址
- 分布式系统中，命名系统的实现本身通常是分布在多台计算机上的
  - 其分布机制对命名系统的效率和可靠性起着关键的作用

基本概念

- 实体：系统中的任何事物
  - 主机、打印机、磁盘、文件、进程、用户、邮箱、Web页面、消息
- 实体的名字（Name）:一个位串或字符串，可唯一地标识一个实体
  - 如主机名、文件名、进程名、用户名
- 实体访问点（access point）：用于访问该实体的接口
  - 可以有多个访问点
  - 实体可以改变访问点
  - 访问点可赋给另一个实体

可能的命名方法（1）

- 地址就是一种特殊类型的名字，指向实体的访问点

- 用地址作为名称？
  - IP地址、电话号码
  - 看上去用地址作为名称似乎很方便，但实际上，这种命名非常不灵活且用户不友好。如，服务器运行的主机经常会变化，提供多个访问点的实体（Web 服务）

可能的命名方法（2）

- 标识符：具有以下属性的名字
  - 一个标识符最多引用一个实体
  - 每个实体被一个标识符所引用
  - 一个标识符总是引用同一个实体（不准重用）
- 使用标识符来作为实体的名称？
  - 不错，但不同进程使用的标识符会不会重名或者不一致？
- 在大多数计算机底层系统中，地址和标识符仅使用计算机可读的方式，如以太网地址、内存地址等位串形式
- 使用用户友好的名称
  - 用户理解的字符串，如“temp”，“bank”

命名系统

- 在分布式系统中，一般使用位置无关的命名方法
- 因此，命名系统的主要工作就是把名称解析为地址

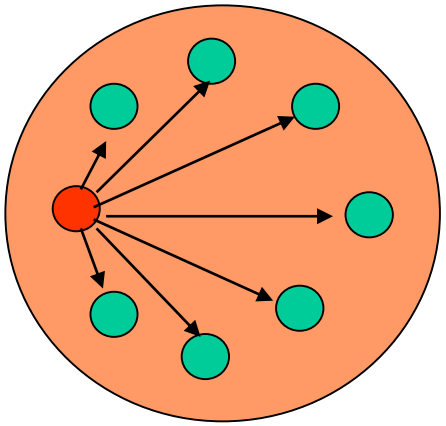
- 简单地，命名系统中可以有一个名称到地址的绑定表
  - 如（Name，address）表
    - ➔大型分布式系统，资源众多，一个表可能无法满足要求

- 分布式系统常见的三类命名系统
  - 无结构命名
  - 结构化命名
  - 基于属性的命名

无结构命名

- 无结构名称：名称中不包含任何有关如何定位其相关实体的访问点的信息

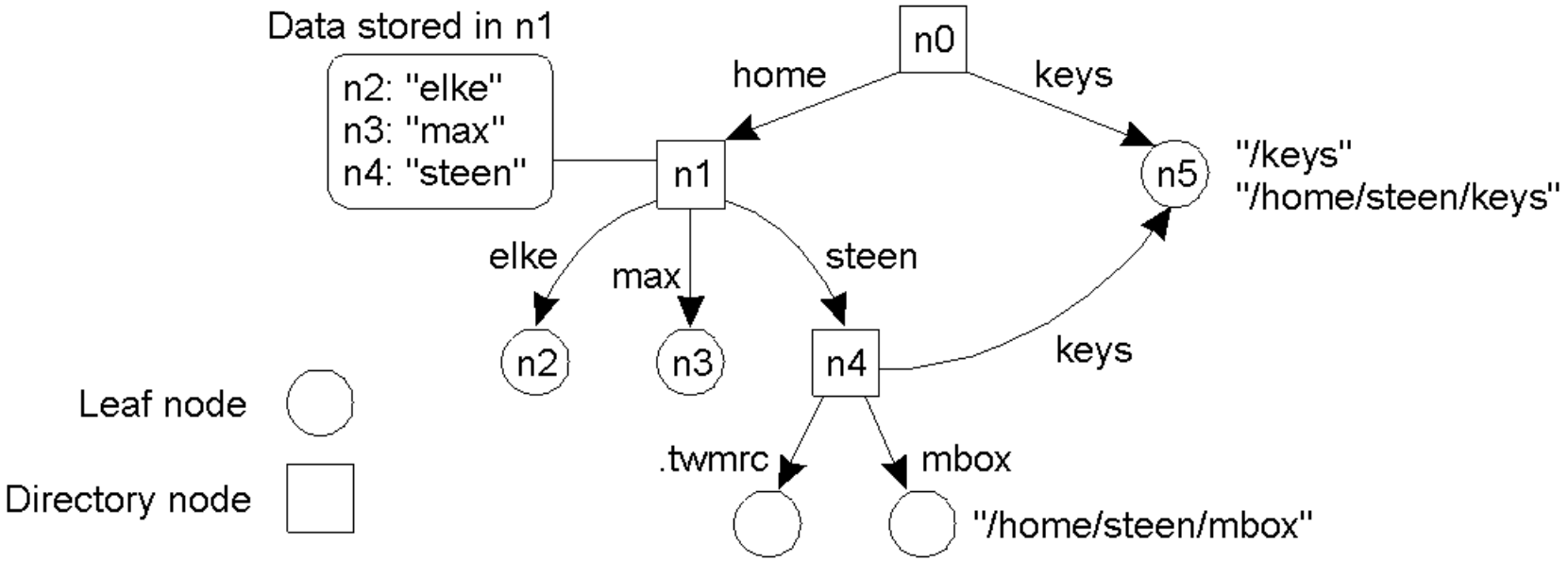
- 多种方法支持无结构名称的解析
  - 广播和多播
    - ➔Internet地址解析协议ARP
  - 分布式散列表
  - ……



- 无结构名称适用于机器的理解，人难以理解

结构化命名

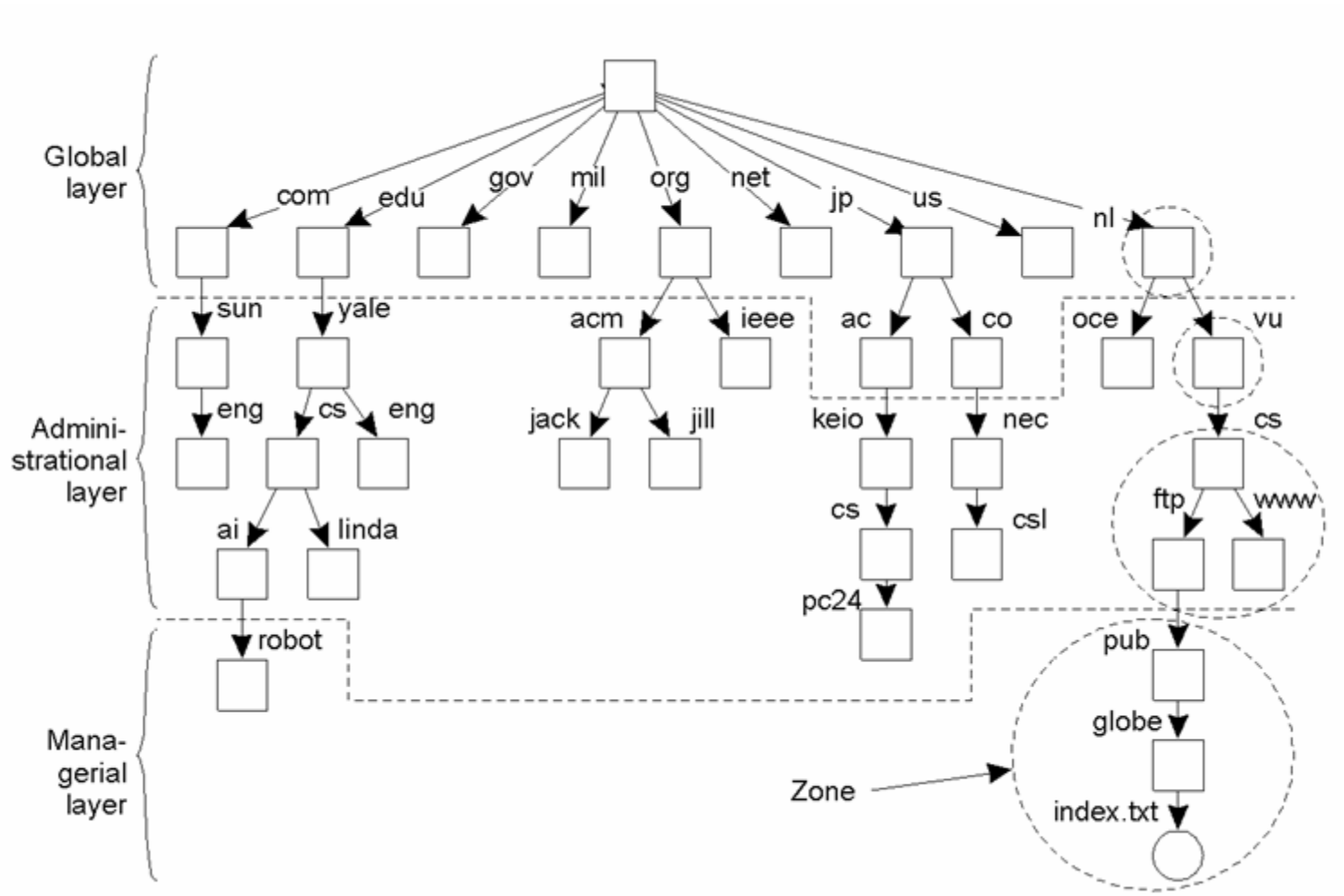
- 结构化名称：简单的、人类可理解名称
  - 文件命名：/user/lib
  - Internet上的网页地址：www.bupt.edu.cn
- 名称通常组成名字空间：根节点、目录节点、叶节点：具有某种结构



名字空间的实现

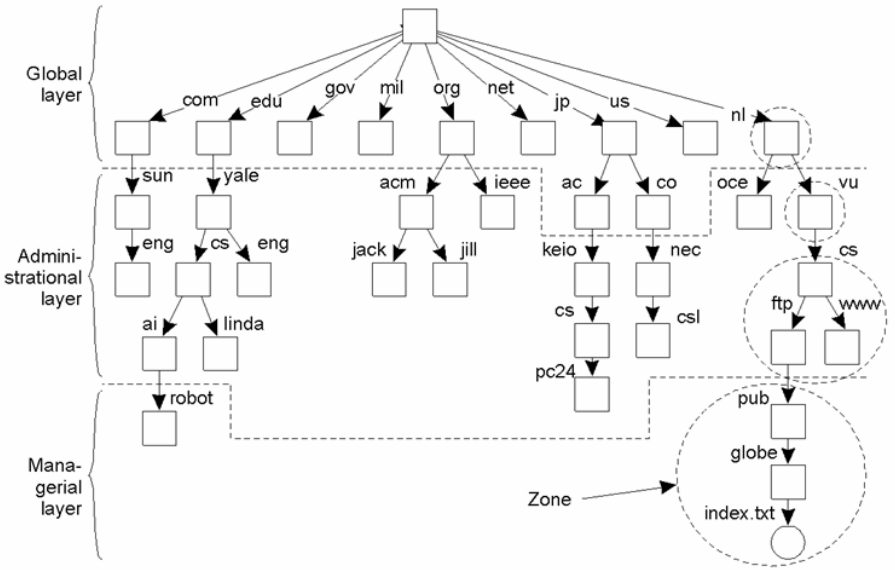
- 命名服务：管理名字空间，允许用户和进程添加、删除和查找名称的服务
- 命名服务由命名服务器实现
- 大型分布式系统包含很多实体，跨越很大的地理区域，需要使用多台命名服务器来分散名字空间的管理

DNS的命名空间划分示例



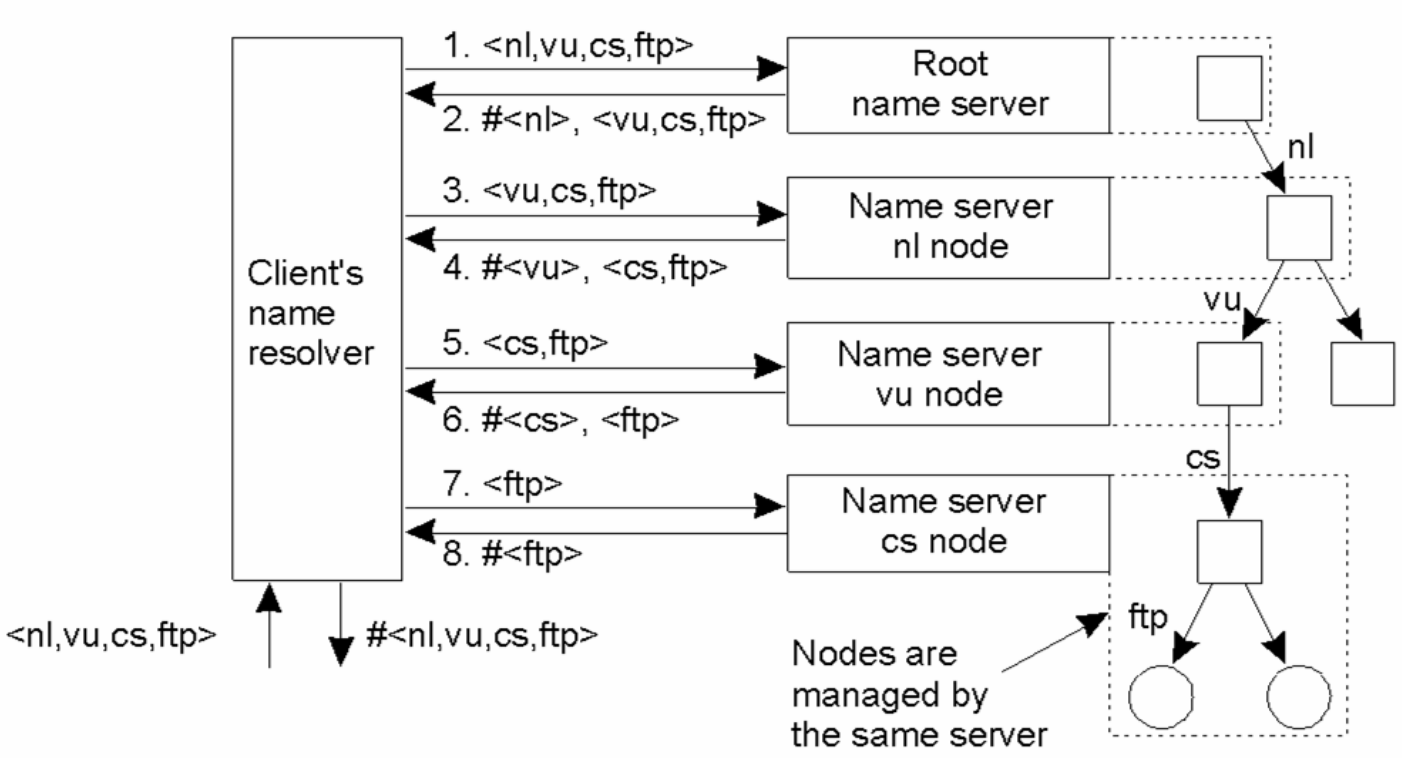
用于不同层的命名服务器比较

内容	全局层	行政层	管理层
网络的地理位置	世界范围	组织	部门
节点数量	少许	多	极多
是否适于客户端缓存	是	是	有时
查询响应	s	ms	立即
复制数量	许多	没有或很少	没有
更新传播情况	延迟	立即	立即



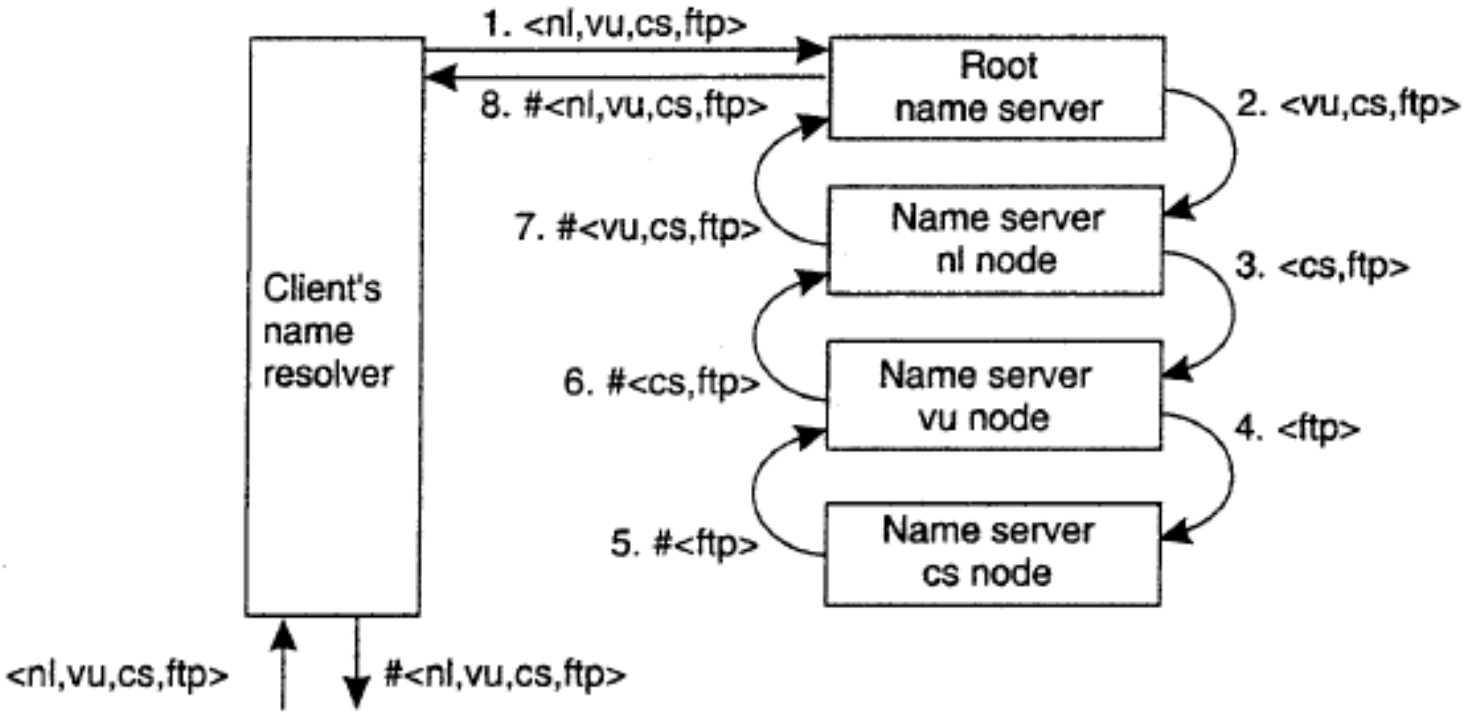
名字解析例：迭代式名字解析

- 客户侧名字解析器从根名字服务器开始，逐个与名字服务器交互，实现名字的解析。（不考虑缓存的情况下）
  - ftp://ftp.cs.vu.nl/pub/global/index.txt



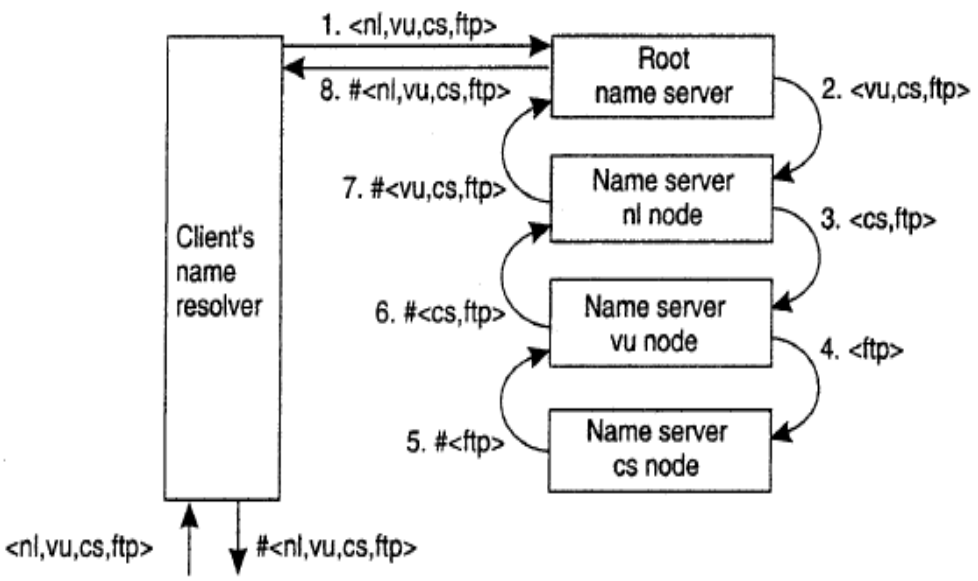
名字解析：递归式名字解析

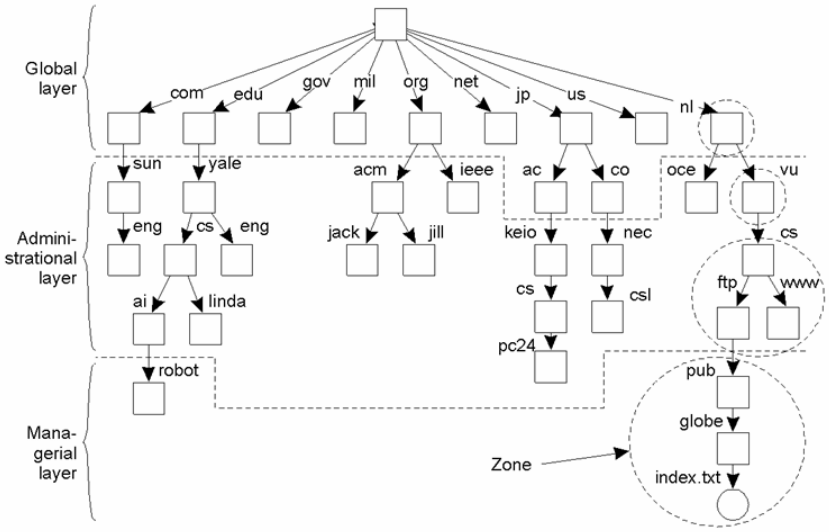
- 根服务器接到解析请求后，与其下层服务器联系，直至最下层服务器解析完毕将结果上报，然后逐级上报，最后由根服务器返回解析结果。（不考虑缓存的情况下）
  - ftp://ftp.cs.vu.nl/pub/global/index.txt



递归式名字解析优缺点

- 优点
  - 缓存结果更为有效
    - ➔例如：根节点可以缓存#<vu>、 #<vu,cs>、#<vu,cs,ftp>的地址
  - 可以减少通信开销
- 缺点
  - 要求名字服务器有较高的性能





“命名系统”总结

- 名称可用来表示实体，基本有3种类型的名称，地址、标识符、用户友好的名称
- 无结构命名系统需要将标识符解析为对应的实体地址，如Internet地址解析协议ARP
- 结构化名称可以很容易组织成名称（名字）空间，名称空间可以用命名图来表示，名称解析的过程是通过查找路径名的各个部分来贯穿该命名图
  - 命名服务：管理名字空间，允许用户和进程添加、删除和查找名称的服务
- 基于属性的命名系统中，实体是由（属性、值）对集来描述，可基于属性查找满足条件的实体

主要内容

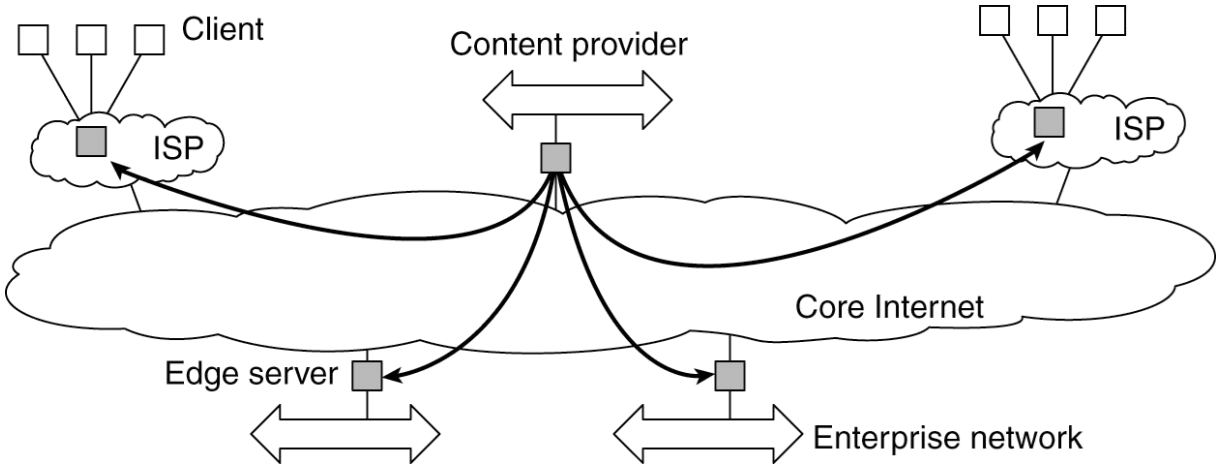
- 体系结构
- 进程
- 通信
- 命名
- **一致性和复制**
- 容错
- 安全

引言

- 进行数据复制的主要原因：
  - 可靠性：一个副本被破坏，可以切换到另外的副本，保护数据，不影响应用的运行
  - 性能：并行性、可伸缩性，（服务器数量、地域）
    - 服务器数量，可均衡负载，提高访问性能
    - 地域扩展，就近访问，提高性能
- 进行数据复制是有代价的，一致性保证
  - 例1：Web页的Cache
  - 例2：镜像网站

复制管理

- 任何支持复制的分布式系统，需解决两个问题
  - 决定何时、何处、由谁来放置副本:
    - ➔服务器放置：找到放置或托管数据存储的服务器的最佳位置
    - ➔内容放置：找到放置内容的最佳服务器
  - 何种机制保证副本的一致性

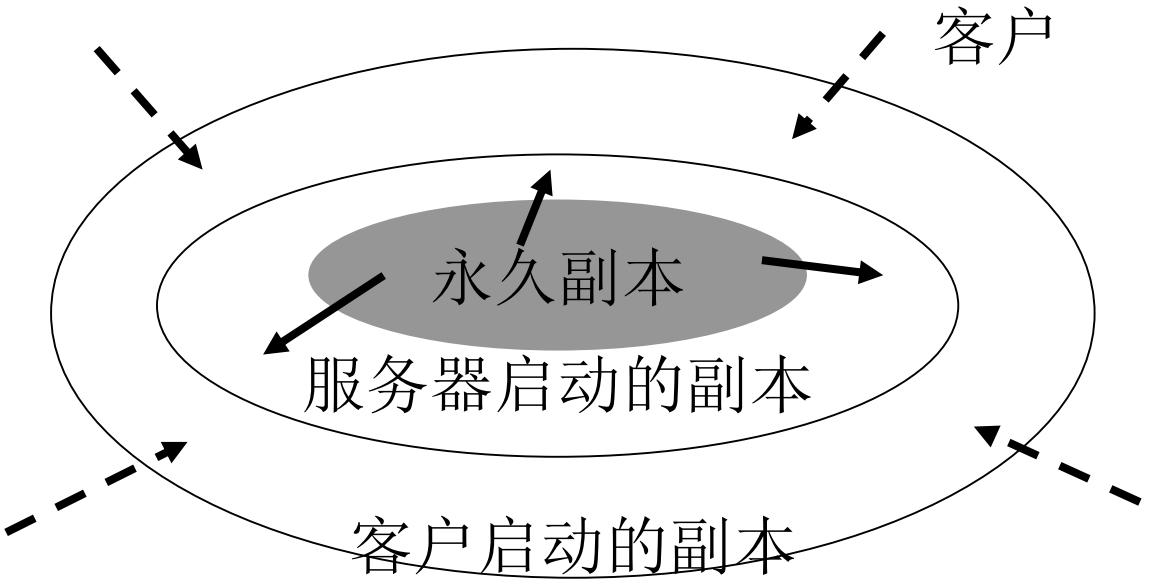


服务器放置

- 基于对客户和网络属性进行分析、优化和决策，如
  - 以客户与位置之间的距离作为计算依据，距离可通过延时或带宽来度量，在某个时间里选取服务器，使得服务器与客户的平均距离最小
  - ……

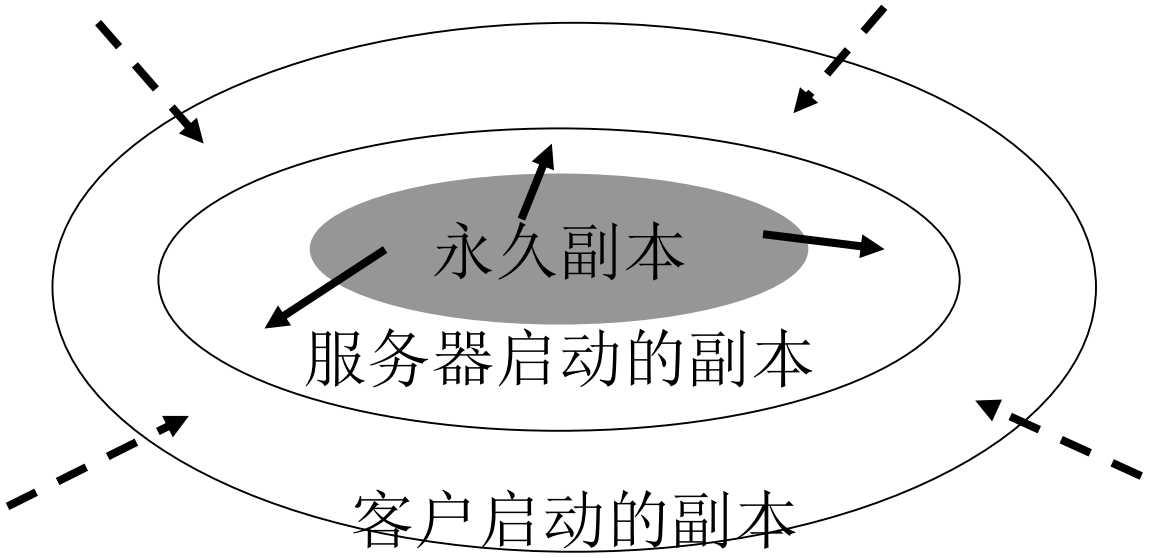
内容复制与放置

- 从逻辑上可以组织为三种不同类型的副本:
  - 永久副本
  - 服务器启动的副本
  - 客户启动的副本



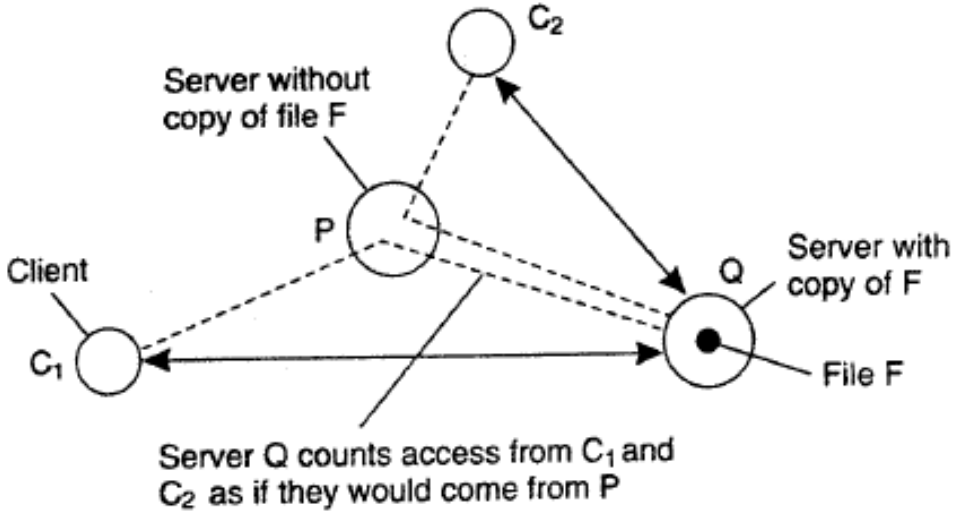
永久副本

- 构成分布式数据存储的初始集，一般数量很小，常用的分布方式：
  - 集群：在单个位置、有限数量服务器，复制
  - 镜像： 地理上分布到因特网
- 特点：静态



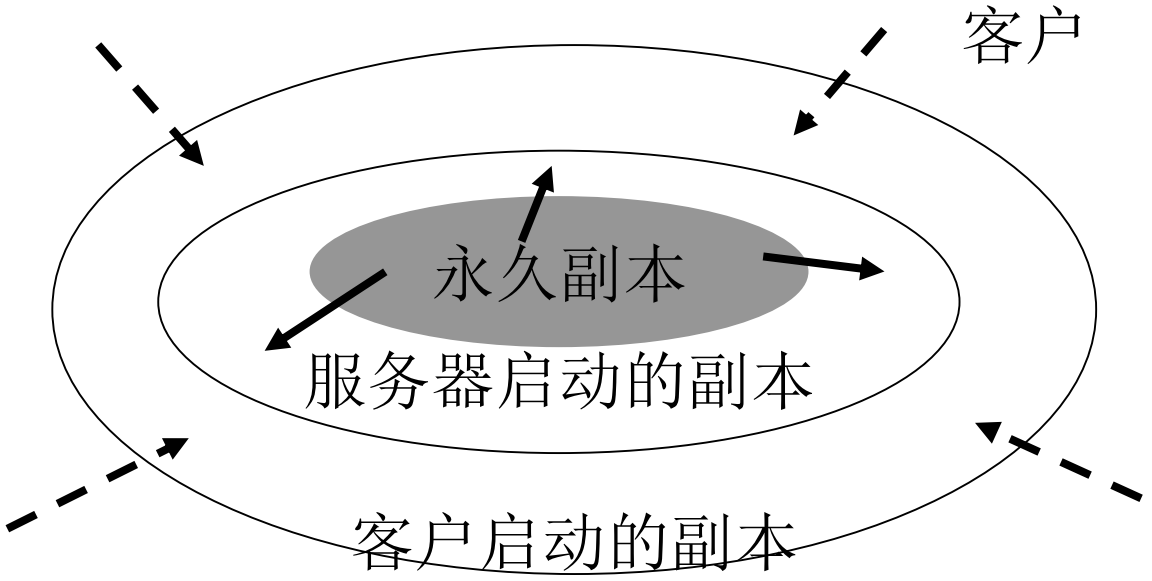
服务器启动的副本

- 是为提高性能而存放的数据存储副本
  - 例如，对于突发的www意外请求访问，可以在产生请求的地区安装一些临时的副本，可放在Web托管服务处
- 问题：如何决定创建和删除副本的位置和时间？
  - 如一种Web托管服务中实现文件动态复制的方法



客户启动的副本

- 客户创建的副本，常称为客户高速缓存。客户使用它暂时存储刚刚请求的副本，用于改善数据访问的时间
- 高速缓存的管理完全是由客户负责的
- 高速缓存可位于客户的机器中，也可位于客户所在的局域网中的其它机器



内容分发

- 复制管理也负责将（更新）内容向相关副本服务器上传播
- 重要设计问题：实际传播哪些信息？
  - 只传播更新的通知
  - 将数据从一个副本传到另一个副本
  - 将更新操作传播到其它副本

如何保证副本的一致性

- 一致性维护与可伸缩性问题
  - 保证所有的副本都是相同的，--〉紧密一致性
  - 当某个副本上执行更新操作时，需对所有副本进行全局同步，在大型系统上很难实施--〉可伸缩性问题
- 解决策略
  - 松驰一致性，放宽在一致性方面的限制，所有副本不一定保持完全相同，尽量减少立即的全局同步
  - 一致性放宽的程度主要取决于复制数据的访问和更新模式，同时还取决于这些数据的用途

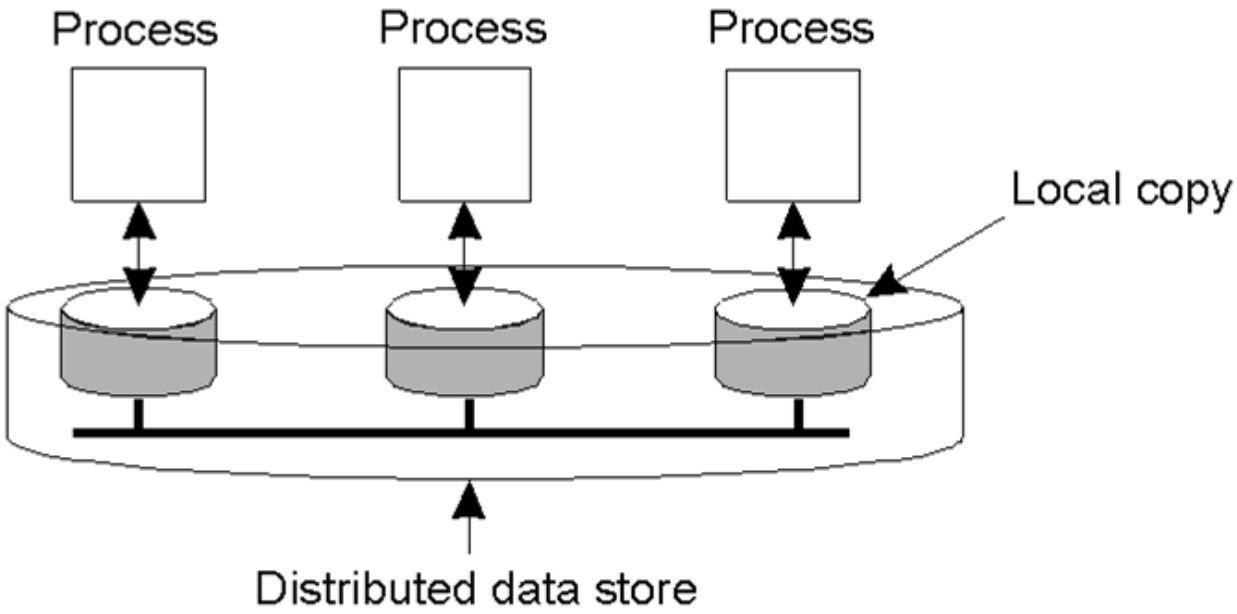
➔具体问题具体分析

常见的一致性模型

- **以数据为中心的一致性模型**
- **以用户为中心的一致性模型**

以数据为中心的一致性模型

- **分布式数据仓(data store)模型**
  - **物理上，分布的和复制的**
    - ➔例如，分布式共享内存、数据库、文件
  - **操作：每个进程可执行读操作，写操作**
    - ➔写操作在本地副本上进行，再传播给其他副本



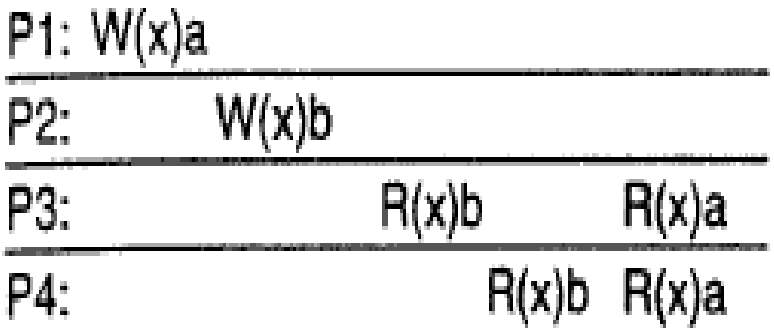
一致性模型

- 以数据为中心的一致性模型，实质上是进程和数据存储之间的一个约定
  - 如果进程遵守这些约定，那么数据存储将正常工作
- 显然，一个进程在数据项上执行读操作时，该进程肯定期待该操作返回的是在其最后一次写操作之后的结果
- 但在没有全局时钟的情况下，精确地定义哪些写操作是最后一次写操作是十分困难的
  - 作为替代方法，产生了一系列放宽了限制条件的**一致性模型**，每种模型都有效地限制了在一个数据项上执行一次读操作所应返回的值

一致性模型举例

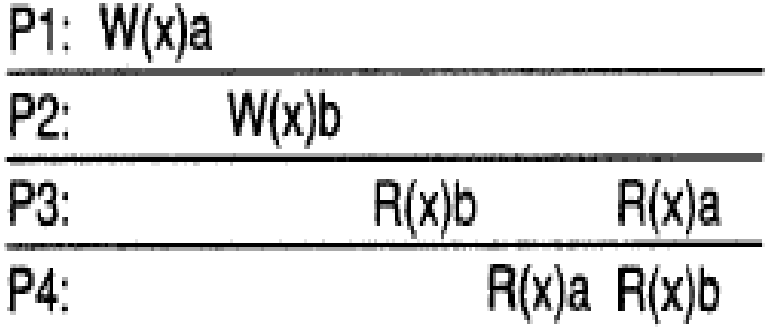
- **严格一致性：对数据项x的读操作返回的值为最近写入x的值**
  - **不可实现**
- **顺序一致性：当进程在多台（可能）不同的机器上并发执行时，任何读、写操作的有效交叉都是可以接受的行为，但所有进程看到的是相同的操作交叉顺序**
- 因果一致性：一种弱化的顺序一致性，所有进程必须以相同的顺序看到具有潜在因果关系的写操作，不同进程可以以不同顺序看到没有因果关系的并发写操作。
- ……

例：顺序一致性



(a)

满足顺序一致性

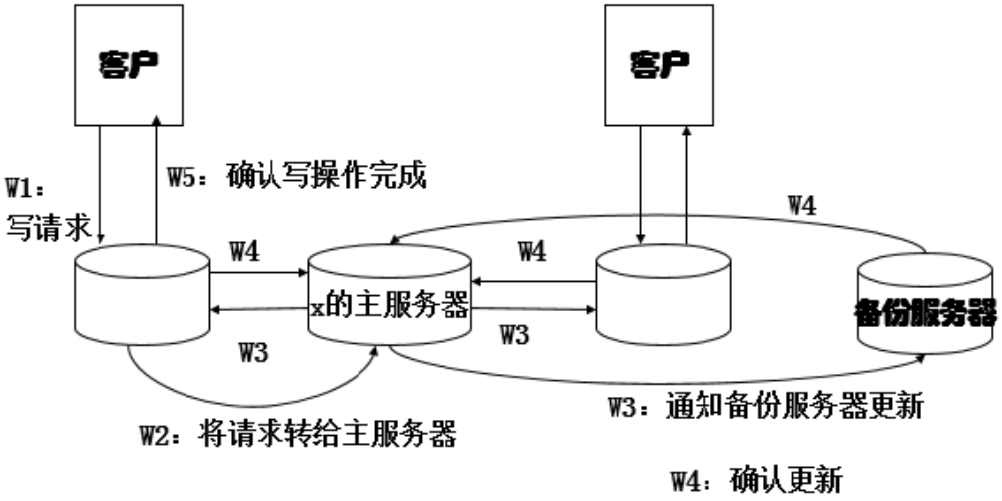


(b)

不满足顺序一致性

顺序一致性实现例

- **基于主备份的远程写协议：**数据存储中的每个数据项都有一个相关的主备份，该主备份负责协调在x上的写操作
  - **主服务器可以对所有进来的写操作进行全局排序，各个进程无论使用哪个备份服务器执行读操作，都会以相同的顺序看到所有写操作写的结果**



- **问题**
  - **若更新以阻塞的方式实现：性能问题**
  - **若更新以非阻塞的方式实现：容错问题**

常见的一致性模型

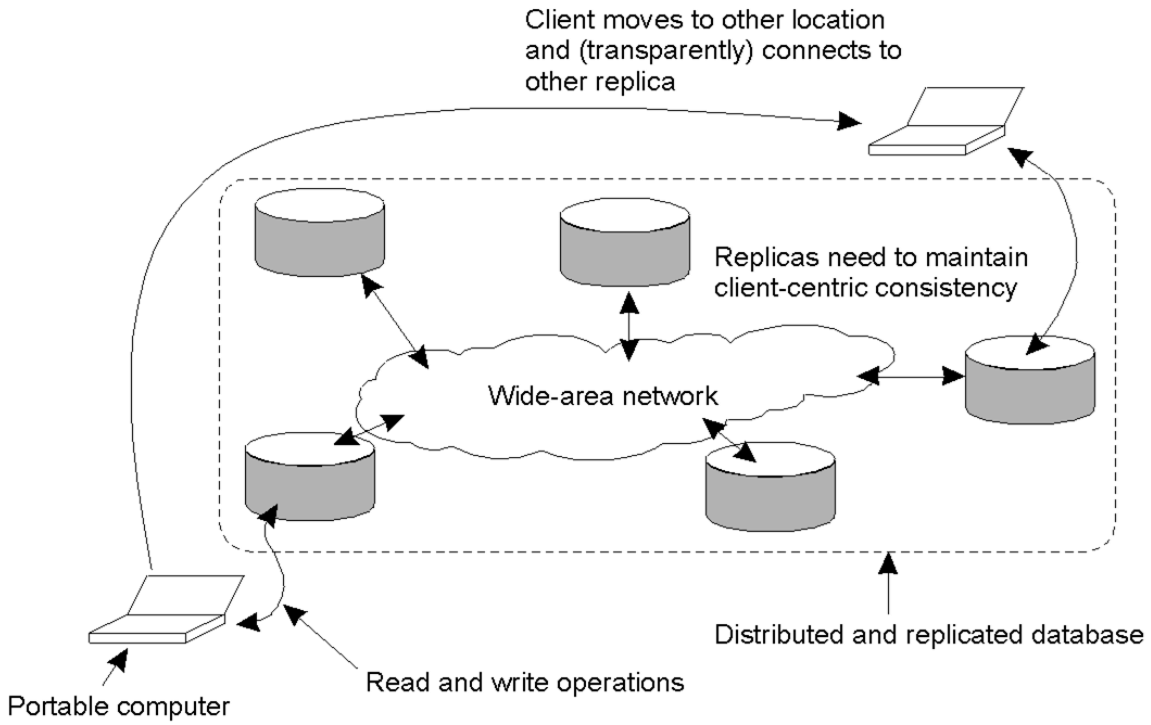
- 以数据为中心的一致性模型
- 以用户为中心的一致性模型

最终一致性

- 以数据为中心的一致性模型的一个重要假设是并发进程可能同时更新数据存储
- 这里讨论的分布式数据存储区
  - 没有同时写操作（无写-写冲突）或容易解决：如DNS
  - 写操作少，大多数操作为读操作：如一些数据库系统
  - 哪怕是旧的数据，用户也常认为是可接受的，如Web网页（服务器，代理缓存）
- **最终一致性**：如果在一段时间内没有写操作，那么所有的副本将逐渐趋于一致
  - 每次写操作并不更新所有副本
  - 有可能读到未更新数据
  - 适用于能够容忍较高不一致性的系统

以用户为中心的一致性模型

- **以客户为中心的一致性（Client-centric）**
  - 保证一个客户对数据存储的访问是一致的
  - 不考虑不同客户之间的并发访问
- **只要用户总是访问一个副本，最终一致性会工作得很好。但用户访问不同副本时，容易出现问题**
  - 如移动用户问题



以客户为中心的一致性模型例

- **单调读一致性：如果一个进程已经在t时刻看到x的值，那么以后它不会看到较老版本的x的值**
- 单调写一致性：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
- 读写一致性：一个进程对数据项x执行一次写操作的结果总是会被该进程对X执行的后续操作看见
- 写读一致性：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读取值相同或比之更新的值上。

单调读一致性实现例

- **每个写操作都被分配一个全局唯一的标识符，对于每个客户，跟踪两个写操作集**
  - 客户的读操作集：由客户所执行的读操作相关的写操作组成
  - 写操作集：客户执行的写操作标识符组成
- **当客户在一台服务器执行读操作时，该服务器获得客户的读操作集，检查所有已标识的写操作是否已在本地执行**
  - 如果没有，联系其他服务器进行副本更新，或将读操作转发到已执行这些写操作的服务器去执行
- **读操作执行后，在所选择的服务器上执行的写操作以及与读操作相关的写操作会被加入客户的相应操作集中**

“复制与一致性”总结

- 复制数据主要有两个原因
  - 提高分布式系统的性能或者可靠性
- 复制要求副本之间保持一致性
  - 严格的一致性难以实现
  - 需要放松一致性的要求
- 以数据为中心的一致性模型
  - 假设数据会被多个进程并发读写
- 以用户为中心的一致性模型
  - 关注保持一个客户的数据的一致性问题