序言 14
1. 网络操作系统
2. 分布式操作系统
3. 并行操作系统
4. 实时操作系统 RTOS
5. 3 种系统的不同之处
6. 推动操作系统发展的因素：
第一章 分布式计算机系统
1. 分布式计算机系统 3
2. 分布式系统的特征
(1) 资源共享
(2) 并发性
(3) 容错性
(4) 透明性
3. 分布式系统的优点
4. 分布式系统的不足 5
5. 分布式系统的资源管理
(1) 全集中管理方式
(2) 分担管理方式
(3) 轮流管理方式
(4) 全分散管理方式
6. 分布式系统的拓扑结构
(1) 全互连结构
(2) 部分互连结构
(3) 层次结构
(4) 星形结构
(5) 环形结构
(6) 总线结构
(7) 立方体结构
7. 计算机支持的协同工作系统
(1) CSCW 特点
(2) CSCW 具体类型
(3) 多机 OS 的基本结构
8. 分布式 OS
9. 操作系统的形成和发展阶段 4
(1) 手工操作阶段
(2) 批量处理阶段
(3) 操作系统形成阶段
10. 操作系统分类
(1) 单用户操作系统
(2) 批处理操作系统
(3) 分时操作系统
(4) 实时操体系统
(5) 网络操作系统
(6) 分布式操作系统
(7) 多处理机操作系统
11. 分布式 OS 的控制策略
12. 分布式 OS 设计中的关键问题 2
13. 分布式操作系统主要特点 3
基于总线的多处理机
第二章 分布式通信
1 分布式系统实现进程间通信
2 进程间通信
(1) 进程间通信的实现方法
(2) 进程间通信方法主要有
(3) 对象之间的通信手段
(4) 选择进程间通信方法
消息传递
远程过程调用 RPC
(1) RPC 与本地调用的区别
(2) RPC 调用语义
(3) RPC 系统的实现问题
(4) 失败情况
(5) RPC 存在的问题
第三章 分布式协同处理
1 事件定序与时间戳
(1) 难点
(2) 定序
2 分布式互斥
(1) 要求：
(2) 集中式算法
(3) 分布式算法
(4) 令牌算法
(5) 几种算法对比
3 选择算法
(1) Bully（欺负算法）
(2) 基于环结构的算法
4 评价分布式同步互斥算法标准
(1) 响应时间和吞吐量
(2) 容错性
(3) 开销大小
(4) 收敛和公平
(5) 可扩充性
(6) 确定性
(7) 恢复
(8) 实用性
(9) 可理解性
第四章 资源管理
1 资源共享
(1) 资源共享的方法
2 资源管理
3 死锁处理
(1) 处理死锁问题的策略
(2) 资源分配图
(3) 计算机系统提供的资源
(4) 还可分为
(5) 资源管理的原则
(6) 资源管理的内容
(7) 资源管理的任务
(8) 资源死锁、通信死锁
第五章 进程与处理机管理
1 进程和线程
(1) 创建新进程
(2) 进程和线程的区别
(3) 多个线程的 3 种模式
(4) 线程的调度
2 进程管理
3 处理机管理
(1) 工作站模型
(2) 处理机池模型
(3) 混合模型
第六章 任务分配与负载平衡
1 任务分配
(1) 任务分解
(2) IMC
(3) IPC
(4) 任务划分粒度
(5) 划分方法
(6) 任务复制
(7) 任务分配
(8) 遗传算法
2 负载平衡
(1) 负载
(2) 目的
(3) 负载平衡算法分类
(4) 负载平衡算法的组成
(5) 负载平衡使用的参数
(6) 负载不平衡主要有
(7) 动态负载平衡算法（影响效率的 3 个主要因素）
(8) 负载平衡其他相关因素
3 Linux Virtual Server（LVS）
(1) 负载调度算法
(2) 应用
第七章 分布式文件系统
(1) 支持的透明性
2 分布式文件系统的组成
3 设计策略
4 接口
5 文件系统的实现技术
(1)

(3) 非结构化文件 结构化文件 ..... (3) 锁 ..... 1 分布式共享内存 23
(4) 对共享文件的修改： ..... (4) 可串行性（化） ..... (1) DSM 主要处理方式 24
(5) 文件共享语义 ..... (5) 常用的三种并发控制 ..... 2 设计和应用 24
(6) 被测文件系统的特性 ..... 2 锁机制 ..... (1) 数据结构 24
(7) 设计实例 ..... (1) 读锁 ..... (2) 同步模型 24
(8) 分布式文件系统的新技术 ..... (2) 写锁 ..... (3) 一致性模型 25
(9) Cache 管理 ..... (4) 锁的粒度 ..... 分布式系统副本复制和一致性 25
(10) 系统结构 ..... 3 乐观并发控制 ..... 1 系统级一致性 25
第八章 命名服务 ..... (1) 基础 ..... (1) 严格一致性 25
1. 名字 ..... (2) 三个阶段 ..... (2) 顺序一致性 25
(1) 名字、地址、路由 ..... (3) 优点 ..... (3) 因果一致性 25
(3) 命名机制的作用 ..... (4) 缺点 ..... (4) 同步化操作 26
(4) 命名机制的目标 ..... 4 时间戳定序 ..... (5) 最终一致性 26
(5) DEC 系统研究中心开发的全 第十章 分布式事务 ..... 2 用户级一致性
球名字服务目标 ..... 1 事务 ..... (1) 单调读一致性 26
2. 一般的命名方式 ..... (1) ACID 特性 ..... (2) 单调写一致性 26
3. 分布式系统中的命名方式 ..... (2) 分布式事务 ..... (3) 写后读一致性 26
(1) 名字管理部分的主要功能 ..... (3) 嵌套事务 ..... (4) 读后写一致性 26
(2) 分布式系统的命名方案 ..... (4) 分布式事务的协调者 ..... 3 副本复制和内容分发 26
4. 名字服务器的设计 ..... (5) 原子提交协议 ..... (1) 副本在哪里？ 26
(1) 功能 ..... (6) 两阶段提交协议中的超时 ..... (2) 传播什么？ 26
(2) 中央方式 ..... (7) 事务实现中的问题 ..... (3) 谁来发起传播？ 26
(3) 复制方式 ..... (8) 分布式事务的并发控制 ..... (4) 如何传播？ 27
(4) 分划方式 ..... (9) 分布式死锁 ..... 4 一致性的工程实现 27
5. 分布式系统的透明性 ..... (10) 死锁检测方案 ..... (1) 基于主备份的协议 27
(2) 与几点要求相冲突 ..... 第十一章 恢复与容错 ..... (2) 基于复制的写协议 27
第九章 事务的并发控制 ..... 1 具有容错能力的应用程序 ..... (3) 高速缓存一致性协议 27
1. 信号量 ..... 2 事务恢复 ..... (4) 用户级一致性协议 27
(1) 信号量 ..... 3 容错 27
(2) 管程 ..... 第十二章 分布式共享内存 27

序言

1. 网络操作系统

具有网络功能的操作系统，无严格定义。MS-DOS

(1) 网络通信能力

(2) 提供网络服务

网络上各节点的主机运行自身的操作系统，它不仅要保证本机的系统进程或用户进程能简便、有效地使用网络中各种资源；同时，也为网中其它用户使用本机资源提供服务。

OS+网络协议

2. 分布式操作系统

每台计算机没有各自独立的 OS，用户不了解其文件存储在什么地方，也不了解其程序是由远程处理机执行的，分布式 OS 自动管理文件的放置；

网络 OS 每台计算机均有自己的 OS；网络 OS 的用户要访问资源，用户必须了解资源的位置，用"文件传输"命令在计算机之间移动文件。

分布式操作系统是为分布式计算机系统配置的一种操作系统。

分布式 OS 在这种多机系统环境下，负责控制和管理以协同方式工作的各类系统资源；负责分布式进程的同步与执行，处理机间的通信、调度与分配等控制事务，自动实行全系统范围内的任务分配和负载平衡；具有高度并行性以及故障检测和重构能力。

3. 并行操作系统

并行机 ——> 并行操作系统

并行 DBMS ——>

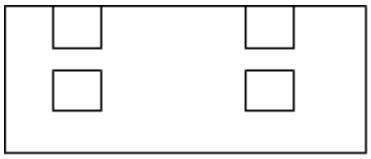
并行算法 ——>

并行程序设计语言及其开发环境（并行编译）

国内的：银河机、曙光机等；

PVM (parallel machine)

NOW 工作站机群系统



CPU

10 ——> 40 ——> 20

20 ——> 80 ——> 40

4. 实时操作系统 RTOS

（Real- Time OperatingSystem, RTOS）

支持实时系统工作的操作系统，响应时间有明确的规定；主要应用于实时控制领域；

(1) 执行效率高、快速、实时性强

(2) 系统小，可剪裁，核心部分更小；

5. 3 种系统的不同之处

项目	网络操作系统	分布式操作系统	多处理机操作系统
看起来是否象一个虚拟的单处理机系统?	否	是	是
所有的机器只运行相同的操作系统?	否	是	是

有多少操作系统的拷贝?	N	N	1
怎样通信?	共享文件	消息	共享存储器
需要共同一致的网络协议?	是	是	否
是否只有一个运行队列?	否	否	是
文件共享是否有良好的语义定义?	通常没有	是	是

6. 推动操作系统发展的因素：

(1) 硬件升级、或者出现了新的硬件类型；GUI 取代字符界面
(2) 用户、系统管理者的需求，新的功能、工具不断加入到 OS 中；
(3) bug 维护、修补；

第一章 分布式计算机系统

下一步的技术发展很**难准确预测**，我们要在网络、分布式环境下开发，需要掌握分布式计算机系统的原理，也需要了解他们的实现原理。
分布式操作系统是为分布式计算机系统配置的一种操作系统。
分布式系统需要与集中式系统完全不同的软件。

1. 分布式计算机系统

第一，从**硬件**角度来讲，各个计算机都是自治的；
第二，从**软件**角度来讲，用户将整个系统看作是一台计算机。
这两者都是必需的，缺一不可。

分布式系统由许多独立的 CPU 组成，它们在一起工作使得整个系统看上去像一台计算机。
任务分布：把一个**任务分解**成多个可并行执行的子任务，分散给各场点协同完成。
功能分布：把系统的总功能划分成若干子功能，分配给各场点分别承担。

2. 分布式系统的特征

(1) 资源共享
硬件资源、软件资源。
开放性
可伸缩性、可移植性、互操作性；数据是可以交换的、对外接口是公开的、系统提供统一的通信机制、提供统一的用户界面。
(2) 并发性
同时工作没有冲突；有冲突，通过相应算法解决；并发控制；
(3) 容错性
两个基本方法，硬件冗余、软件恢复（数据备份、日志）；
(4) 透明性
实际上比其表面要微妙得多的含糊概念之一

种 类	含 义
位置透明	用户不知道资源位于何处
迁移透明	资源可以不改名地随意移动 b
复制透明	用户不知道有多少个拷贝存在
并发透明	多个用户可以自动的共享资源
并行透明	系统活动可以在用户没有感觉的情况下并行发生

3. 分布式系统的优点

(1) 性能价格比高
(2) 速度
(3) 内在的分布性

(4) 可扩充性
(5) 可靠性
(6) 适用于多种环境

4. 分布式系统的不足

(1) 管理复杂
(2) 性能和可靠性依赖于网络
(3) 保密性差
(4) 应用软件少

项目	描 述
软件	目前为分布式系统开发的软件还很少
网络	网络可能饱和和引起其它的问题
安全	容易造成对保密数据的访问

5. 分布式系统的资源管理

(1) 全集中管理方式
一个资源由一个管理机制管理。
(2) 分担管理方式
一个资源虽由几个管理机制管理，但各分担一种管理职能。
(3) 轮流管理方式
一个资源可由几个管理机制管理，但轮流执行管理职责。
(4) 全分散管理方式
一个资源由多个管理机制在协商一致的原则下共同管理。

性能比较：
基本开销：连接系统中的各个站点要多少花费？
通信开销：从站点 A 发送信息到站点 B 需要多少时间?
可靠性：

6. 分布式系统的拓扑结构

(1) 全互连结构
优点：各场点间消息传递快，可靠性较高。缺点：开销高。
(2) 部分互连结构
其开销比全互连结构低，但通信速度较全互连结构慢。可靠性也相对较低。
(3) 层次结构
通常情况下，其中的任何中间节点故障都可能将这种结构分割成若干不相交的子树。因此，可靠性较低。
(4) 星形结构
这种结构的基本开销与场点个数成正比，这种通信速度却是没有保障的，因为中央场点可能变成瓶颈。
(5) 环形结构
基本开销较低，但通信代价可能较高。
(6) 总线结构
这类结构的开销同场点成正比，通信代价也很低。
(7) 立方体结构

7. 计算机支持的协同工作系统 CSCW

（CSCW，Computer Supported Cooperative Work），也是一种分布式系统。
(1) CSCW 特点
群体性、交互性、分布性、协同性。

(2) CSCW 具体类型

a) 电子邮件系统

b) 电子布告栏系统（BBS， Bulletin Board System）

c) 群体决策支持系统

d) 协同编辑系统

e) 计算机会议系统

f) 协同计算机开发环境

(3) 多机 OS 的基本结构

主从式　　独立式　　分布式

8. 分布式 OS

分布式计算机系统(Distributed Computing Systems)是由多个分散的计算机经互连网络连结而成的计算机系统。其中各个资源单元(物理或逻辑的)既相互协同又高度自治。能在全系统范围内实现资源管理，动态地进行任务分配或功能分配而且能够并行地运行分布式程序。

分布式操作系统是为分布式计算机系统配置的操作系统。系统任务可以在系统中任何别的处理机上运行。并提供高度的并行性和有效地同步算法和通信机制，自动实行全系统范围的任务分配并自动调节各处理机的工作负载．为用户提供一个方便、友善的用机环境。

分布式系统与网络系统是有区别的。从操作系统的角度来看，网络操作系统是为计算机网络配置的操作系统，网络中的各台计算机配置各自的操作系统，而网络操作系统把它们有机地联系起来。

9. 操作系统的形成和发展阶段

(1) 手工操作阶段

每个程序员都必须亲自动手操作计算机：装入卡片或纸带，按电钮，查看存储单元等。

(2) 批量处理阶段

用户不用与计算机直接打交道，而是通过专门的操作员来完成作业的输入和输出。

(3) 操作系统形成阶段

多道程序和**分时系统**的出现，标志着**操作系统的正式形成。**

a) 多道程序设计的定义

所谓多道程序设计，是指同时把若干个作业存放在内存中，并且同时处于执行过程中。但是在某时刻只能有一个程序占用 CPU 执行。

b) 分时系统

所谓分时系统，就是在一台计算机上，连接若干个终端，用户通过这些联机终端设备采用交互方式把他的程序和数据输入到计算机中，并同时控制程序的执行。

10. 操作系统分类

(1) 单用户操作系统

在这种操作系统控制下，计算机系统串行地执行用户程序，即在执行完一个用户程序后才接受另一个用户程序。一些微机上配置的操作系统大多数就属这种类型。

(2) 批处理操作系统

在这种操作系统的控制下，计算机系统可以同时接受多个多用户程序，一批批地进行处理。批处理操作系统一般都提供多道程序设计功能，允许多个程序同时装入内存执行。

(3) 分时操作系统

分时操作系统又称多用户操作系统，在这种操作系统的控制下，多个用户可以通过各自的终端同时使用一台计算机。分时操作系统有三个明显的特点：多路性，交互性和独占性。

(4) 实时操体系统

实时操作系统是为实时计算机系统配置的一种操作系统，在这种操作系统的控制下，计算机系统能及时地响应外部事件的请求，在规定的时间内尽快地完成对该事件的处理，并有效地控制所有实时设备和实时任务协调地进行。在设计这类操作系统时，首先要考虑系统的

实时性和可靠性，其次才是效率。

(5) 网络操作系统

网络操作系统是为计算机网络配置的操作系统。网络中的各台计算机配置有各自的操作系统，而网络操作系统把它们有机地联系起来，因此，它除了具有常规操作系统所应具备的存贮管理、处理机管理、设备管理、信息管理和作业管理等功能外，还具有以下网络管理功能：高效可靠地网络通信能力以及多种网络服务功能。

(6) 分布式操作系统

分布式操作系统是为分布式计算机系统配置的操作系统。系统任务可以在系统中任何别的处理机上运行。并提供高度的并行性和有效地同步算法和通信机制，**自动实行全系统范围的任务分配并自动调节各处理机的工作负载。**为用户提供一个方便、友善的用机环境。

(7) 多处理机操作系统

多处理机系统是由多台处理器组成的计算机系统。多处理机系统可分成两大类：基于共享存储的多处理机系统和基于分布存储的多处理机系统。前者称为紧耦合多处理机系统，而后者称为松耦合多处理机系统。

多处理机系统也称为并行计算机系统。并行机上使用的操作系统称为并行操作系统。

11. 分布式 OS 的控制策略

集中决策　　分布决策

信息交换　　合作

12. 分布式 OS 设计中的关键问题（目标）

(1) 透明性

(2) 灵活性

(3) 可靠性

(4) 性能

(5) 可扩展性

13. 分布式操作系统主要特点

(1) 进程通信不能借助于公共存储器，常采用信息传递方式；

(2) 系统中的资源分布于多个站点，进程调度、资源分配、系统管理必须满足分布式处理要求，采用一致性、强健性的分布式算法；

(3) 适时地协调各站点的负载；

(4) 故障检测、恢复、系统重构

(1) 分布式系统，首先必须有一个单一的、全局的进程间的通信机制，从而使任何进程都可以和其它进程进行通信。

(2) 不同机器上，进程管理也相同。进程建立、撤消、启动、停止都相同。

(3) 文件系统也必须看起来是相同的。同时，每个文件应该是在所有地方都是可见的，当然，这必须遵守保护和安全性约束的限制。需要一个全局的文件系统。

(4) 在系统的所有地方都使用相同的系统调用接口。

14. 基于总线的多处理机

在 CPU 和总线之间增加一个高速缓冲存储器（cache memory），如图 1-5 所示。缓冲存储器保留着最近刚存取过的字。所有的内存访问请求都要经过它。如果请求的字在缓冲存储器中，缓冲存储器就会直接响应 CPU，而不产生总线请求。如果缓冲存储器足够大的话，那么成功的可能性，称为命中率，将是很高的。而且每个 CPU 的总线通信量也会急剧下降，系统中也就能够容纳更多的 CPU。通常，缓冲存储器的大小从 64K 到 1M，命中率经常可以达到 90%或更高。

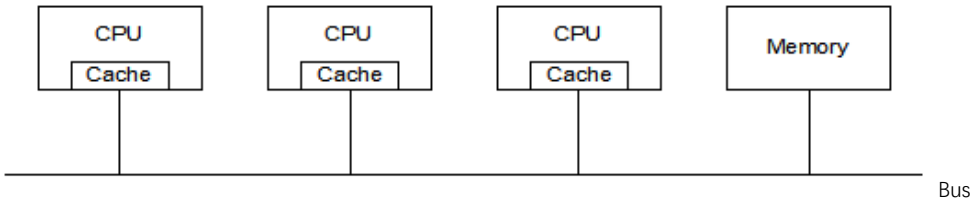


图　基于总线的多处理机

Cache 的一致性问题

第二章 分布式通信

单处理机系统中
共享存储器

1. 分布式系统实现进程间通信注意的问题：

(1) 无共享存储器，不能借助共享变量的方法；
(2) 机器间消息传递的可靠性低于机器内信息传递的可靠性；
(3) 系统内任意两台机器未必直接连接，往往需要中转；
(4) 系统内的各台机器型号可能不同；
(5) 通信的实现与系统结构、通信线路结构、通信介质的物理性能等有密切关系。

2. 进程间通信

(1) 进程间通信的实现方法
可以是低级的，涉及系统调用，或者通过语言级的支持实现
(2) 进程间通信方法主要有
a) 消息传递
b) 管道
c) sockets
d) RPC
e) 供享内存
(3) 对象之间的通信手段
CORBA， DCOM
(4) 选择进程间通信方法主要考虑的问题
a) 程序员对所选方法的熟悉程度
b) 进程间通信机制的透明性，程序员知道得细节越少，出错得机会也就越少。
c) 系统所支持的方法
d) 考虑系统的扩充
e) 支持进程的迁移，不同文件系统的进程间通信
f) 通信机制的标准化问题
g) 通信机制的有效性

3. 消息传递

消息传递，物理上复制要共享的数据到另外一个进程的地址空间。
下列情况，一般不常用消息传递
(1) 两进程不共享内存空间
(2) 在不同的系统中
在同一系统中，每个进程有自己的内存消息通常是用消息包或帧的形式发送的，通过 OS 提供的基本通信**原语**。
异步型 同步型
阻塞原语实现进程不再阻塞，一般有 2 种方法：
(1) 轮询
利用测试原语，测试缓冲区的相关信息（状态），忙等待。
(2) 中断
也可以在非阻塞原语种利用。轮询一直不成功，或者一直无中断，这样会无限阻塞下去。
要有计时器，缺省的设置，或者程序员控制
阻塞 send & receive
Procedure A
Begin
Instructions

```
... ...
send (B, message)
// where B is the destination
  // waiting for acknowledgment
  //    received send acknowledgment
next instructions
... ...
End
Procedure B
Begin
Instructions
... ...
receive (A, message)
// where A is the source
// waiting for message
//    received message
next instructions
... ...
End
```

消息传递（同步）适合于 C/S 模型

(3) C/S 模型的几个设计问题

a) 寻址

b) 阻塞和非阻塞原语

c) 有缓冲和无缓冲原语

d) 可靠和非可靠原语

管道

两进程间的通信通过内核在有限大小的缓冲区上实现，这类原语通过系统调用实现。当缓冲区满时，引起阻塞。

Sockets

通过网络的通信，不是共享数据结构或文件。

4. 组通信

(1) 用途

a) 具有冗余结构的系统

b) 在分布式系统中查找

c) 多副本的更新

d) 各种通知

(2) 组通信的特性

a) 原子性

b) 定序

组通信最简单的实现方式就是**不可靠组播**，即简单地向每个目标发送一条消息。

可靠组播：一种实现方式是发送者向一个组中所有成员发送消息，然后等待每一个成员的回复。

5. 远程过程调用　RPC

RPC 使用过程调用实现远程通信，

在传统的过程化程序设计语言环境中，它的语义类似于本地过程调用的语义，因此，它可向应用层用户提供良好的接口。

Client 进程 ⟷ Client's Stub ⟷ Server's Stub ⟷ Server 进程

程序员不知道调用的是一个远程过程，还是一个本地过程，这需要有相应的支持机制，将一台计算机上语言级调用自动转化为另一台计算机上相应的语言级调用，实现变量和结果的传送。

调用者阻塞，等待返回值，而不是仅仅一个确认值。

与各种程序设计语言一样，对参数的数目和数据类型有限制。

(1) RPC 与本地调用的区别

a) 数据表示问题

如果 RPC 是在两种异构的机器上进行的，不同机器数据表示可能不同，包括机器的字长等。

b) 指针

在不具备共享地址空间的情况下，RPC 不可能允许在网络范围内传递指针。

c) 故障

调用者和被调用者都可能在调用期间发生故障。

对于故障，由于调用者无法知道到底出现了那种情况，因此，系统需要提供一些基本的保护机制来确保 RPC 的正确效果。

不同 RPC 实现方案定义的这种效果或 RPC 语义是有差别的。

以下是几种常用的 RPC 调用语义。

(2) RPC 调用语义

a) At- Most -Once （最多一次）

相同 RPC 的重复调用，服务器不处理。

b) At- least -Once （至少一次）

RPC 将被执行至少一次，可能多次。

c) Last -of-Many-Call （最近调用）

每个调用包含一个标识，client 接收最近调用者的返回值。

(3) RPC 系统的实现问题

a) RPC 协议族

- ✧ 面向连接的 面向非连接的
- ✧ 选择标准的通用协议，还是专门为 RPC 设计的协议
- ✧ 信包和报文的长度

b) 确认

- ✧ 停等协议（stop and wait protocol）
- ✧ 爆发协议（blast protocol）

c) 缓冲区 缓冲池

d) 计时管理

(4) 失败情况下的 RPC 语义，可能出现的问题及其解决方法：

a) Client 无法定位 Server

b) 客户请求消息丢失

c) Server 应答消息丢失

d) Server 崩溃

e) Client 崩溃

(5) RPC 存在的问题

a) 服务器必须被正确定位；

b) 指针与复杂的数据结构难以传送；

c) 全局变量很难使用；

d) 很难有**精确的** RPC 语义；

第三章 分布式协同处理

全局时间为进程和数据提供时间戳。

1. 事件定序与时间戳

计算机上运行的应用程序只关心事件发生的次序，而不是事件发生的绝对时间，即只需要用计数器的值去给事件打上相应的时间戳。

对于集中的物理时间：请求类和广播类

(1) 难点

request for time

---------->

client time server

<----------

current time (delay)

有延迟，而且是不确定的，因为网络故障，可能会传送多次。

Cristian 算法

Berkeley 算法

网络时间协议（NTP）

UTC 统一协调时间

时间的质量、精确性与时间提供者的价格等相关；

物理时间：人的时间；

逻辑时钟：是一种单调增长的软件计数器，对事件集进行部分排序。

相对时间，逻辑上是一致的。

(2) 定序

若两个事件发生在同一进程中，则可用观察到的次序来确定它们发生的次序；

无论何时在进程间传递消息，发送消息的事件先于接收同一消息的事件。

先决条件：两个事件之间，逻辑时钟至少变一次，两个事件不会精确地同时发生。

2. 分布式互斥

(1) 要求：

e) 安全性

f) 可用性

g) 定序

在单机系统中，使用信号量（semaphores）、管程（monitors）等来保护临界区。

S P(S) V(S)

Wait(s) Singal(s)

(2) 集中式算法

(3) 分布式算法

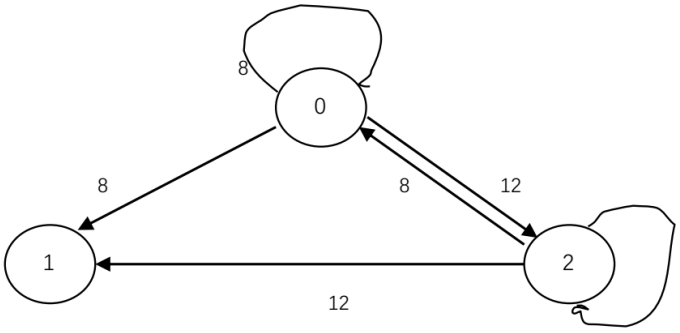
要求：系统中所有的事件都是全序的

当一个进程接受到另一个进程请求消息（Request）时,

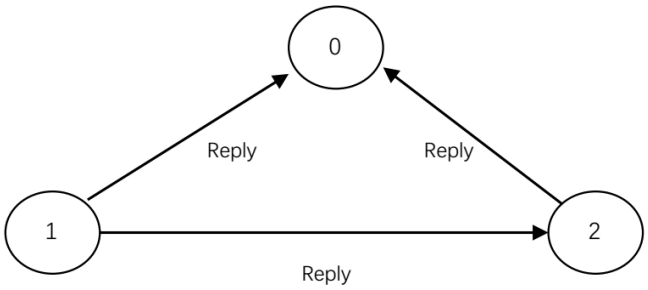
a) 若接受者不在临界区中，也不想进入临界区，它就向发送者送 Reply 消息

b) 若接受者已在临界区中，它就不想进入临界区，它就向发送者送回答（推迟）消息

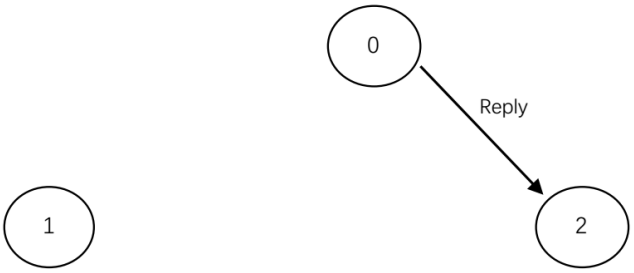
c) 若接受者要进入临界区中，但还没有进入，它就将自己的请求消息（Request）时间戳与收到的时间戳比较，若收到的小，回 Reply 消息，否则，推迟



8 12 为时间戳



当进程 0 完成时，进程 0 返回 Reply



在产生请求冲突时，遵守按时间戳排序，小时间戳优先的规则。

每次进入临界区需要 2(n-1)条消息， n 为系统中的进程数目。

相对集中式算法，慢、复杂、贵、不健壮

(4) 令牌算法

系统中所有的进程可组成一个虚拟或逻辑环，每个进程要知道谁在它的下一个位置。

算法：令牌环被初始化后，进程 0 首先获得令牌，这样令牌开始绕环运动，它从进程 k 传递 k+1，以点到点方式若一个进程得到了它相邻进程传递来的令牌，但它并不想进入临界区，就将该令牌往下传递。

仅拥有令牌的进程才有权进入临界区。

(5) 几种算法对比

	每次进出需要的消息	进入前的延迟	问题
集中式	3	2	协调者崩溃
分布式	2(n-1)	2(n-1)	任一进程崩溃
令牌	1 到 ∞不定	0 到 n-1	丢失令牌，进程崩溃

3. 选择算法

如果这个协调者进程由于它驻留的处理机故障，而无法正常工作（称为故障），系统只得通过在另一个处理机上重新开始一个新的协调者副本才能运行，确定在何处重新开始一个新的协调者算法，就称为**选择算法**。

(1) Bully（欺负算法）

a) Pi 给所有比它优先数大的进程发送消息；

b) 若无进程响应，Pi 获胜成为协调者；

c) 若有优先数比 Pi 大的进程响应 Pk，响应者 Pk 接管 Pi 的工作完成；

(2) 基于环结构的算法 （基于没有令牌的环）

a) 当任何一个进程发现协调者进程不起作用时，构造一个包含它自身进程号的消息给后继者；若后继者失败，继续下一个

b) 消息到达了始发者的手中，始发者接收者接收到包含它自身进程号的消息后，将其转化为协调者消息；

c) 该消息将再一次绕环运行，向所有进程通知谁是协调者；具有最大优先数的进程，将它作为新的协调者；

4. 评价分布式同步互斥算法标准

(1) 响应时间和吞吐量
充分利用系统的分布性，获得高的吞吐量和小的响应时间。
(2) 容错性
具有幸免于故障的能力。
(3) 开销大小
算法需要的一些额外开销，消息的数目和大小等。
(4) 收敛和公平
请求进入临界段的进程终将进入，在临界段执行的进程终将离开临界段。
并且对各进程公平。
(5) 可扩充性
容易加入新的结点和新的进程。
(6) 确定性
一定能保证同步、互斥，还是可能保证，如果不是确定的，就是概率性的。
(7) 恢复
对错误恢复的能力如何。
(8) 实用性
对其使用作了多少限制。
(9) 可理解性
如果一个算法是简单的，则很容易给出规范的正确证明。
简单是很重要的，包括：算法的实现、测试、维护、修改等等。

第四章 资源管理

1. 资源共享

(1) 资源共享的方法
a) 数据迁移
整个文件
部分文件 通过文件或数据库的 水平分割、垂直分割 但分割较麻烦。
b) 计算迁移
传递计算比传递数据更有效
c) 作业迁移
隐式： 作业迁移最终由系统实现；
显式： 用户指明作业如何迁移；

2. 资源管理

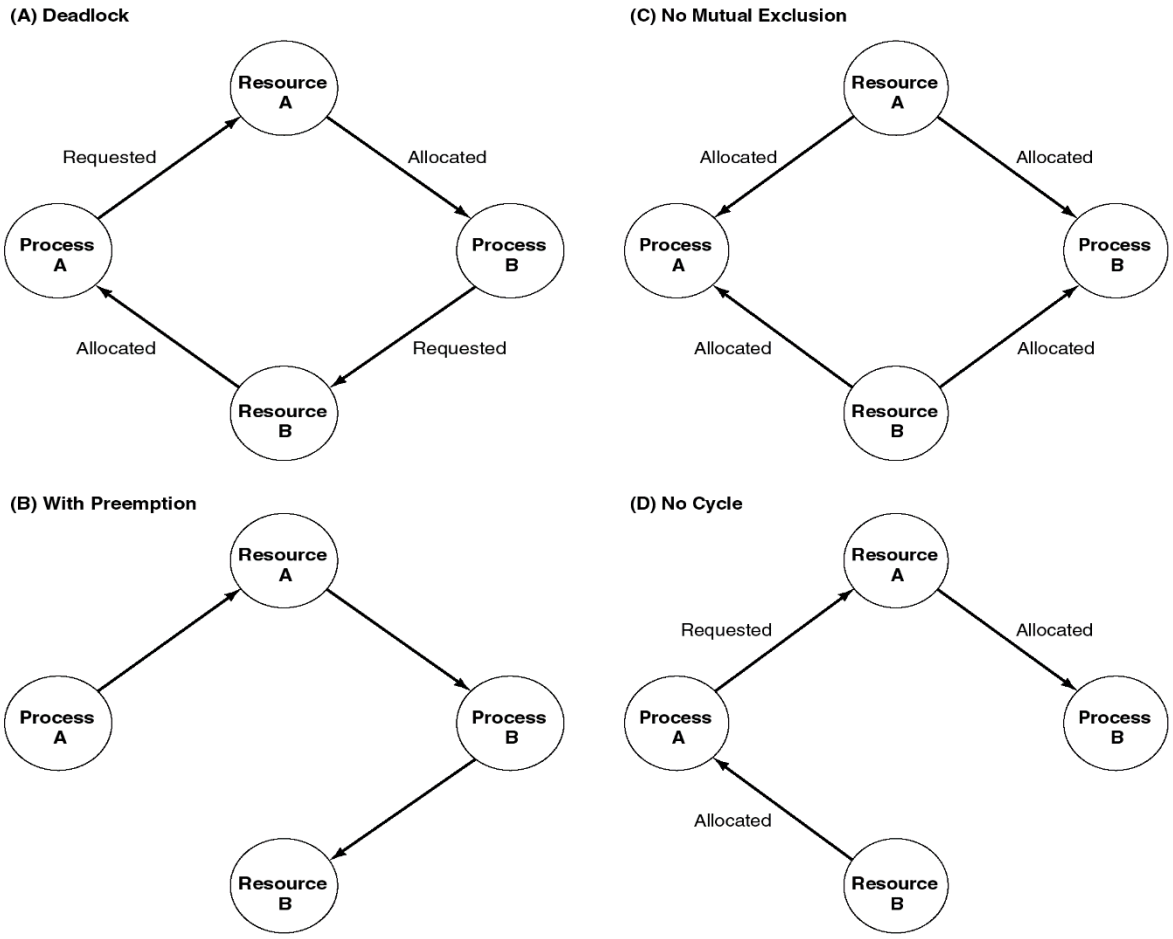
局部集中管理
分散式管理
分级式管理

3. 死锁处理

死锁的 4 个条件：
(1) 互斥
(2) 非抢占资源分配
(3) 持有和等待
(4) 循环等待

如果不存在上述的任一条件，就不会发生死锁。

死锁预防、避免、检测算法。



(1) 处理死锁问题的 4 种著名策略

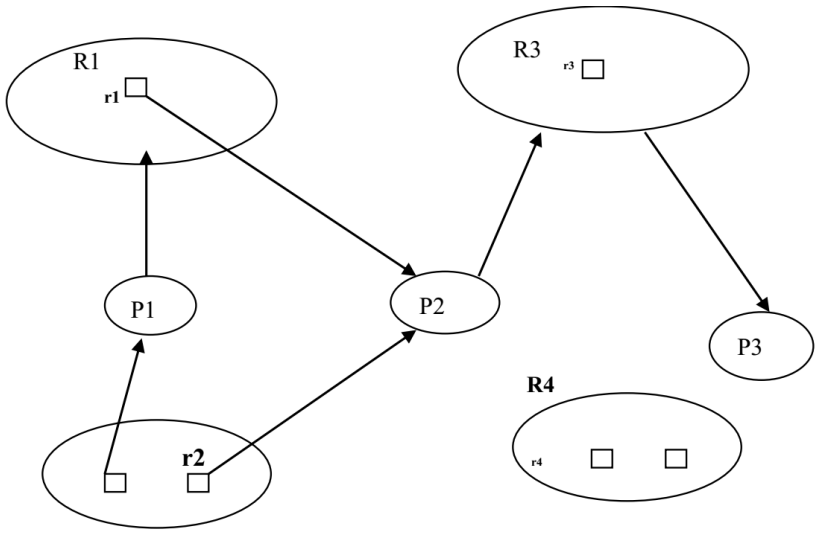
a) 死锁忽略：不考虑死锁问题。

b) 死锁检测：允许死锁发生，在检测后想办法恢复。

c) 死锁预防：静态的使死锁在结构上是不可能发生的。

d) 死锁避免：通过仔细的分配资源以避免死锁。

(2) 资源分配图（resource allocation graph）



real time 系统：很难如此构造"资源分配图"。

(3) 计算机系统提供的资源包括

a) 物理资源

CPU、主存、I/O 设备、内部设备、外存等。

b) 逻辑资源

进程、文件、共享的程序和数据。

(4) 在分布式系统中，所有这些资源在物理上是分布的。还可分为

a) 底层资源和高层资源

b) 可共享和不可共享的

(5) 资源管理的原则是

方便、高效、公平

(6) 资源管理的内容

a) 配置管理（Configuration）

通过配置管理，系统资源被放在合适的位置，调整成合适的状态。

b) 故障管理（Fault）

处理各种错误。

c) 安全管理（Security）

提供安全机制，对系统资源进行安全的访问和使用

d) 性能管理（Performance）

对系统资源进行协调、优化，以获得最大的性能和利用率。

e) 帐户管理（Account）

收集资源的使用情况等。

(7) 资源管理的任务

a) 接受来自客户方（用户、进程）申请资源的请求，并从资源中选择适当的资源进行分配。

b) 接受系统提供的资源，并能组成资源池（资源库）。具有一定的监控，最终可以收回资源。

(8) 资源死锁、通信死锁

通信死锁发生于一组直接通信的进程之间，当它们受阻于等待来自其它进程的消息以开始执行，但它们之间没有消息传递时就发生死锁。

第五章 进程与处理机管理

1. 进程和线程

进程都包括一个执行环境，其中有一个或多个线程。

一个线程是操作系统中对一个处理的抽象。

(1) 创建新进程

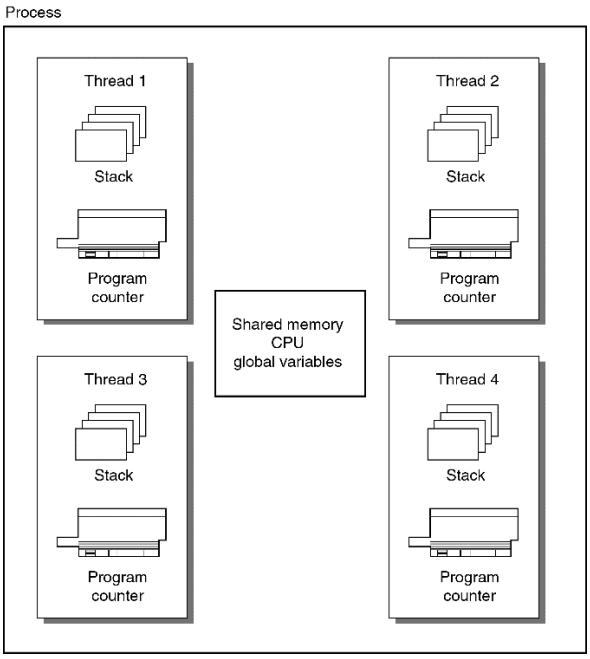
c) 选择目标机；

d) 创建一个新的执行环境；

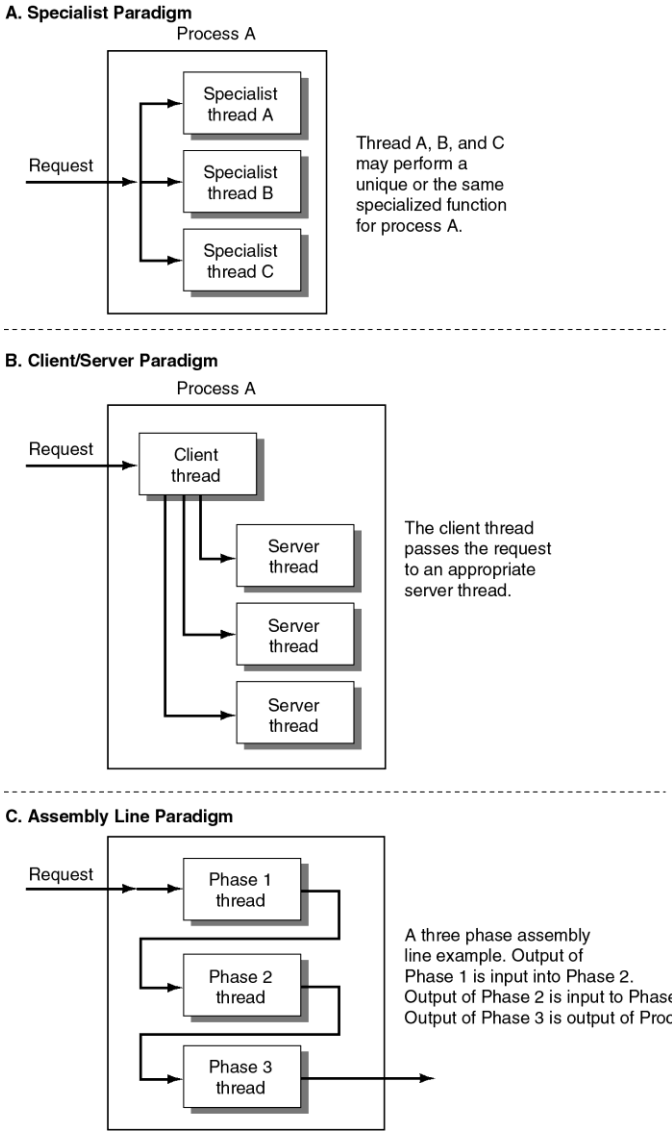
e) 在执行环境中创建一个线程；

(2) 进程和线程的区别

线程的创建和管理开销小，线程间共享资源比进程间共享资源更有效地实现。



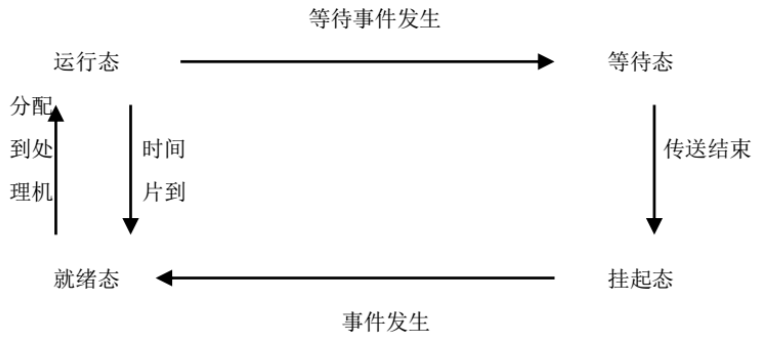
(3) 同一个进程的多个线程的 3 种模式



(4) 线程的调度

抢占式调度， 非抢占式调度

2. 进程管理



3. 处理机管理

(1) 工作站模型

整个分布式系统是通过局域网连接的工作站构成的。

a) 无盘工作站（无本地磁盘）

文件系统由一个或多个远程文件服务器实现，读写文件的请求将发送到文件服务器，由它执行并返回结果。

优点：维护容易，使用灵活，共享信息容易，成本低

缺点：网络通信频繁

b) 有盘工作站

- ✧ 本地磁盘仅用于分页和存储是临时的、不能共享的、并在登录会话结束后丢弃文件，减少网络通信。
- ✧ 本地磁盘可以保存二进制（可执行）程序，如：编译程序、文本编辑程序等。

  需要更新二进制（可执行）程序。
- ✧ 本地磁盘用来作缓冲区，用户能从文件服务器下载文件到本地磁盘，在本地读写这些文件，然后在登录会话结束之前上载修改后的文件。

  一致性问题。
- ✧ 每台机器能有自己独立完备的文件系统，而且能登录或存取其它机器上的文件系统。

  缺点：共享难，更接近网络 OS

(2) 处理机池模型

在机柜中放满 CPU，它们可以根据需要动态地分配给用户。

适用于：大规模的并行计算。

(3) 混合模型

每个用户一个私有工作站，并附加有处理机池。

第六章 任务分配与负载平衡

1. 任务分配（task allocation）

若干个**模块**构成一个任务，一个任务是单一的处理实体。

(1) 任务分解：把一个提交的任务划分成若干个独立的，具有最小 IMC 的模型。

(2) IMC：每对模块间的数据传递。

(3) IPC：处理机间的通信。

(4) 任务划分：粒度大，降低并行度，粒度小，进程切换和通信的开销就会增加。

(5) 划分方法：

水平或垂直划分：在给定的任务优先图中水平或垂直划分。关键路径（最长路径）

通信延迟最小划分：把通信频繁的节点归成一类。

(6) 任务复制：在各个处理机节点上复制任务来降低通信开销。

(7) 任务分配：把这些模块分配给处理机，使得它们由于处理机间的通信引起的开销最小。

a) 一般算法假设：

- ✧ 存储容量无限
- ✧ 每个处理机节点有相同的处理能力
- ✧ 忽略网络拥塞

b) CPU 的利用率最大化，平均响应时间最小化

a) 基于图论的分配策略
b) 0-1 程序设计策略
c) 合一 ---- 阈值 启发式分配算法
d) 进化算法（演化）

(8) 遗传算法求解方法

a) 编码方法(Encoding)

用一个 n+1 位的二进制串来表示。

00000 11111

b) 初始化群体

用过程 Initialize 来实现，方法是**随机生成初始化**的串群体。在串群体中，串长度都是相同的，串长为需要分配的文件数。群体的大小根据需要(要求的分配时间等)，按经验或实验给出。

分布均匀的二进制串能使算法更加有效。

c) 选择 (Selection)

借用达尔文的生物进化论中的自然选择(Natural Selection)思想，按照"适者生存"的原则对串进行复制。用适应度函数计算每个串的适应值，选择适应值高的串，生成下一代，去掉适应值差的串。

d) 交叉 (Crossover)

交叉是两个串按照一定的概率(交叉概率 P_c)从某一位开始逐位互换。这里先在串群体中，随机的选择两个串，成为一对串，变成多对串后，对每对串随机的选择一个交叉点，例如串长为 n+1，则可选择一整数 i，0≤i ≤n，i 为交叉点，对两个串从第 0 位到第 i 位进行互换，形成两个新串。是否发生交叉操作，还要受交叉概率的控制，选择好一对串后，在 0、1 之间产生一个随机数，若该随机数大于 P_c 则发生交叉，否则保持原状。P_c 也是根据经验或实验确定，一般可为 0.5 左右。

1010 0101 **1110 0111**

1011 1111 **1011 1101**

1010 0101 **1011 1101**

1011 1111 **1110 0111**

e) 突变 (Mutation)

二进制串的某一位按照一定的概率(突变概率 P_m)发生反转，0 变 1，1 变 0。这里 P_m 较小，P_m 可小于 0.001.

f) 适应度函数(Fitness Function)

这里我们用 Evaluation 过程来实现。

g) 停止条件

可以是以下几种或其组合：

- ✧ 规定进化代数，也就是最大迭代次数。
- ✧ 群体中某个解的适应值达到某一预先规定的范围内。
- ✧ 连续若干代，群体中的个体不再变化。

h) 相应的遗传算法描述

```
Procedure  GA  Program
  Begin
    Initialize;
    Evaluation;
    While (not termination-condition) do
          Begin
            Selection;
            Crossover;
            Mutation;
            Evaluation
```

End

End.

i) 方法有以下几个优点:

- ✧ 具有一定的规律和随机性，不确定性。为了处理这种特性引入了概率分析。
- ✧ 适用于变化的环境。
- ✧ 能得到多个解，即可得到多个分配方案可供选择。
- ✧ 算法具有良好的并行性，进化过程中的群体是一个可行解的集合。适合于并行计算。

2. 负载平衡

(1) 负载

CPU 队列的长度（比如进程的数目）

某段时间内 CPU 队列的平均长度

可用内存的大小

上下文切换的速率

系统调用的速率

CPU 的利用率

对系统中的负载情况进行动态调整，以尽量消除或减少系统中个场点负载不均匀的现象。

由于任务到达的**随机性**，各节点处理能力上的差异，当系统运行一段时间后，就会出现某些节点上还有很多任务没有完成，而另外一些节点处于空闲。

(2) 目的

发挥系统冗余资源

提高资源利用率

防止软件并行性和硬件并行性之间失配

(3) 负载平衡算法分类

a) 局部和全局

b) 静态和动态

动态的算法增加了系统的调度开销。

动态的算法适应了应用程序可变化、可伸缩等特点的需要，静态的算法由于在编译时可以完成调度，对于大型应用程序的性能提高也起着一定的作用。可以结合使用。

c) 最优和次优

d) 近似和启发式

e) 集中和分散式

f) 协作和非协作的

g) 针对单个应用程序 和 多个应用程序的

h) 抢占式和非抢占式的

i) 自适应和非自适应的

(4) 负载平衡算法的组成

a) 转移策略 T1 T2

b) 选择策略

c) 定位策略

d) 信息策略

收集信息的方式：集中式（多对一，一对多）和分布式（多对多的指令）

收集的时机：周期或非周期

收集的范围：全局还是局部（CPU 可以划分为大小为 K 的一些不同的组）

收集的负载信息内容：节点机的负载信息。在运行的静态和动态阶段所收集的负载信息内容应该是不同的。

(5) 负载平衡使用的参数

a) 系统大小： 如处理机的个数,

处理机多，系统容易找到负载轻的节点，但系统消息传输量大。

b) 系统负载：一般用 CPU 队列长度来衡量系统负载。

c) 系统通信速率：各个处理机上任务的到达率

d) 移动阈值

一般说：移动一个太小的任务是不合适的，对于一个太大的任务，或者涉及到大量数据和文件的任务，也最好在本地处理机节点上执行。

e) 任务大小

决定任务的大小难。

一般是：对资源的要求、任务类型（I/O 多，还是 CPU 多）、存储要求、数据文件要求等。

f) 管理成本

CPU 当前负载的测量、CPU 决策用的负载信息、决策发生的位置、CPU 间的任务传递。

矛盾：应该从足够多的 CPU 间寻找，但是寻找过多，通信成本太高。

g) 响应时间

h) 可选择的目标节点

i) 资源要求

(6) 负载不平衡主要有

a) 某些算法的迭代大小不是固定的，但迭代的大小在编译时却可以求得。

b) 某些算法的迭代大小不是固定的，但迭代的大小依赖于被处理的数据，在编译时无法求得。

c) 即使迭代大小是固定的，也会有许多不定因素导致计算速度的差异。

(7) 动态负载平衡算法（影响效率的 3 个主要因素）：

a) 算法

b) 网络拓扑结构　结点的度数

$Davg = \sum D(I,j)/N(N-1)$

c) 执行动态负载平衡代码的频率

（确定负载平衡的粒度）

(8) 负载平衡中其他相关因素

a) 编码文件和数据文件

比如地理上分布的系统，移动所需的代价。

b) 系统的稳定性

c) 系统体系结构

总线连接系统中传递文件的成本比超立方体的要高。

3. Linux Virtual Server (LVS)

Linux 虚拟服务器，负载调度是在 Linux 内核中实现的。

一组服务器通过网络连接，它们的前端有一个负载调度器（load balancer）。

负载调度器将网络请求调度到真实的服务器上。

(1) 负载调度算法

负载调度是以连接为粒度的。

a) 轮询调度（Round-Robin Scheduling）

依次将请求调度到不同的服务器上。

假定：所有服务器处理性能相同，请求服务时间变化不大。

b) 加权轮询调度（Weighted Round-Robin Scheduling）

用相应的权值表示服务器的处理性能，服务器的缺省权值为 1。

可以解决服务器间性能不一致的情况。

c) 最小连接调度（Least-Connection Scheduling）

把新的连接请求分配到当前连接最小的服务器。

假定：所有服务器处理性能相同。

d) 加权最小连接调度（Weighted Least-Connection Scheduling）

用相应的权值表示服务器的处理性能。

e) 基于**局部性**的最少连接调度（Locality-Based Least Connections Scheduling ）

将相同目标 IP 地址的请求调度到同一台服务器，提高各台服务器的访问局部性和主存 Cache 命中率，从而提高这个集群系统的处理能力。

f) 带复制的基于局部性最少连接调度（Locality-Based Least Connections with Replication Scheduling）
将相同目标 IP 地址的请求调度到同一组服务器。

g) 目标地址散列调度（Destination Hashing Scheduling）
通过一个散列（Hash）函数将一个目标 IP 地址映射到一台服务器上。

h) 源地址散列调度（Source Hashing Scheduling）
根据请求的源 IP 地址，从静态分配的散列表中找出对应的服务器。

(2) 应用于

可伸缩的 web 服务、可伸缩的媒体
服务（音频视频）、
可伸缩的邮件服务 @ B.COM.CN
@ B2.COM.CN @ B3.COM.CN

RedHat（www.readhat.com），
从 6.1 版，包含 LVS 代码。
TurboLinux 有 Linux 集群产品。

第七章 分布式文件系统

1. 允许用户程序直接存取远程文件而不需要将它们拷贝到本地场点

(1) 它支持以下部分或全部的透明性

a) 存取透明性
b) 位置透明性
c) 并发存取透明性
d) 故障透明性
e) 性能透明性
f) 复制透明性
g) 迁移透明性

2. 分布式文件系统的组成

(1) 展开文件服务
(2) 目录服务
(3) 客户组件

3. 设计策略

4. 接口

(1) Read(File, i, n)

从文件 i 位置开始读出 n 个数据项。

(2) Write(File, i, data)

把 data 数据项从 i 位置开始，按顺序写入文件。

(3) Creat()
(4) Truncate(File, l)
(5) Delete(File)
(6) GetAttributes(File)
(7) SetAttributes(File, Attr)

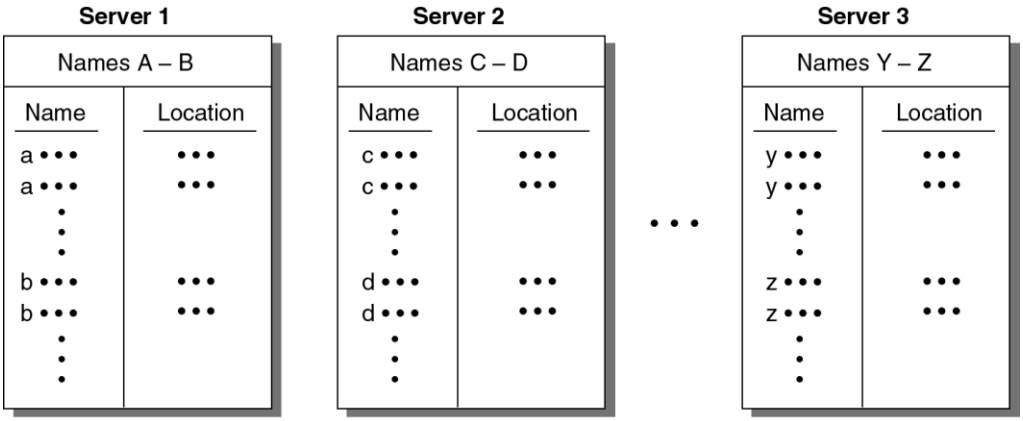
5. 文件系统的实现技术

文件组结构
权限和存取控制
文件服务：文件系统提供给客户内容的详细说明。
文件服务器：是运行在某台机器上的一个有助于实现文件服务 的进程。

(1) 分布式系统中用于文件和目录命名的三种常见方法：

a) 机器+路径
b) 安装远程文件系统到本地文件分层结构
c) 一个在所有机器上看上去都一样的单个名字空间

(2) 任何大型分布式系统都需要用带有多个位置解析服务器的分布式解决方案，每个服务器负责所有名字集合的特定子集。用一个服务器位置映射表来查询，以确定系统中哪个服务器负责哪组名字集合的解析。



Server Location Mapping Table

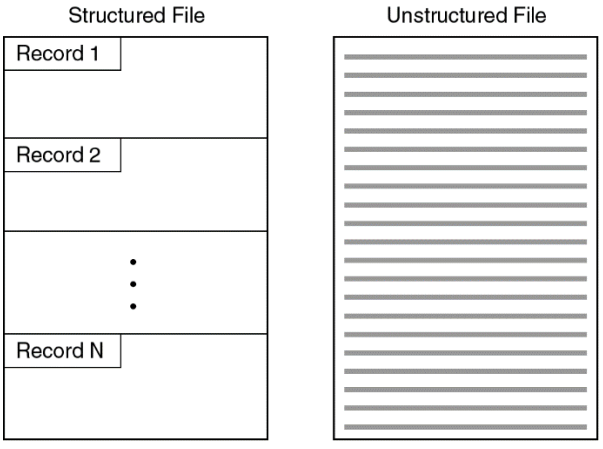
Name Range	Master Server	Additional
A – B	Server 1	
C – D	Server 2	
⋮		
Y – Z	Server N	

(3) 非结构化文件 & 结构化文件

操作系统可以用各种方式来存储文件。

非结构化文件：以连续的字节流来存放文件，在文件中没有内在的结构。

结构化文件： 以记录结构来表示数据。



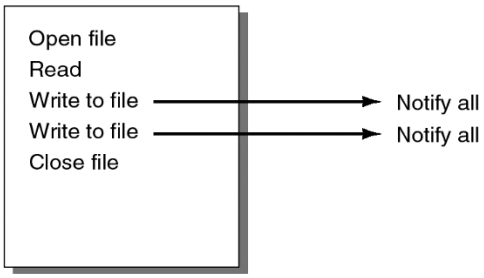
(4) 对共享文件的修改：

a) 立即通知：对文件的每次修改，立即对拥有该文件拷贝的所有参与者可见。

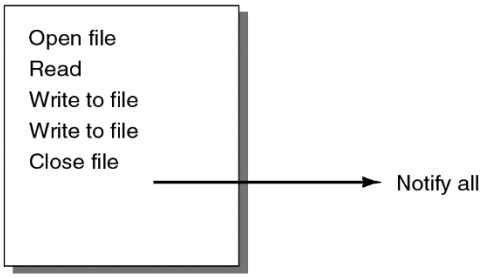
b) 关闭时通知：对将文件进行修改的参与者关闭文件，即终止对文件的访问时，其他参与者才被通知已修改文件。

c) 事务完成时通知：在事务完成时，通知系统中的成员。

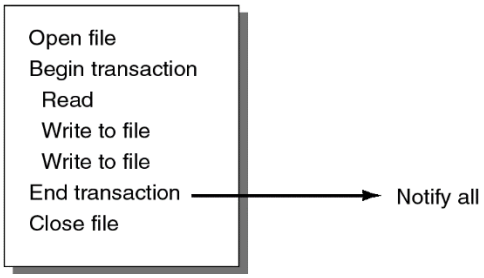
A. Immediate Notification



B. Notification on Close



C. Notification on Transaction Completion



(5) 文件共享语义

a) Unix 语义

b) 会话语义

c) 不可更改文件

d) 事务

(6) 被测文件系统的特性

a) 大多数文件比较小（小于 10K）

b) 对文件的读操作多于对文件的写操作

c) 对文件的读写是顺序进行的，随机访问非常少

d) 大多数文件寿命比较短

e) 文件共享非常少

f) 一般进程只是使用某几个文件

g) 用不同的属性区分文件类型

(7) 设计实例

目录操作：

创建目录

删除目录

重命名目录

列出目录内容

管理目录的访问权限

改变目录的访问权限

移动目录

遍历目录结构

文件操作：

创建文件

删除文件

重命名文件

移动文件

查找文件

复制文件

(8) 分布式文件系统的新技术

a) 内存文件系统

b) 文件服务器可以配置几十个 GB 的内存，这样文件系统可以永久储留在内存，而不需要磁盘。

c) 每个文件在内存中是连续存放的，而不是将它们打散成若干块，连续存储文件易于跟踪和网络传输。

d) 掉电丢失

e) UPS

f) NVRAM

g) 不断备份

(9) Cache 管理

通过 cache 可以提高系统的性能，解决一致性

(10) 分布式文件系统的系统结构

实际系统中，客户和服务器没有区别，任何机器都可以为其他机器提供文件服务

具体实现上客户和服务器往往是不同的。

文件和目录服务结构

一样：既处理目录也处理文件

分开：可以提高系统的灵活性，但是如果有多个目录服务器，则要参与多次路径名查找，开销较大。

文件、目录、其他服务器是否要维持客户的状态信息。

服务器没有状态的

服务器在客户请求之间维持它们的状态信息。

容错，请求消息短

第八章 命名服务

1. 名字

OS 负责实现各个对象（如进程、信箱、结点、I/O 设备、文件、地址）之间的消息传递，这就要求访问的系统对象都有一个名字。

一个名字（标识符），就是标识一个对象的一串符号（位或者字符）。

名字和对象之间对应是一个引用关系，通过名字对对象进行引用和标识。

对于计算机本身，用数字标识符（二进制数最适合表示）。

但用户不能高效地使用二进制表示，用户往往使用 OS 支持且易读的正文串识别对象。

因此，计算机有两类名字：用户名和系统名。

仅仅给出名字是不够的，还必须知道当前的位置。

(1) 名字、地址、路由

名字：规定了一个寻找的对象（即欲访问的对象）；

地址：定义了对象的位置；

路由：定义了到达对象的路径；是代表从源点到目的点路径的一列名字。

(2) 将名字变换为地址，将地址变换为路由。
如果名字可以变换为相关路由，就可以高效地进行通讯。这个路由变换函数称为路由选择，并由路由算法实现。

(3) 命名机制的作用：
a) 实现对资源的共享，重定向等等；
b) 从分布式 OS 和应用的角度，对资源进行访问。

现有的一些计算机系统，对不同类型的对象使用独立的**命名原则**，如文件名与其它对象名之间差别较大，分布式系统应使用统一的命名原则可以识别各个可用的对象。

(4) 命名机制的目标：
a) 命名机制应可以识别全局范围的对象；
b) 应至少支持两类不同的标识符；方便用户使用的用户级；便于机器存储表示的系统级；
以上两级的机制应各自独立，且可以高效地进行转换。
c) 在整个系统范围内的唯一标识符；
d) 在发送具体的消息前，初始化和转换标识符时交换消息的数量小；
e) 命名机制应可以对对象进行重定位；负载平衡要求对象应该是可移动的。
f) 名字和地址之间的变换不一定是一一对应的，还可以是一对多；
g) 命名机制应支持对同一对象多副本的使用；
h) 命名机制应允许用户对同一对象定义多个局部的标识符；
i) 命名机制应允许多个不同的对象共用同一个标识符（进程组）；
j) 分布式系统中使用的独立命名机制的数目必须是最小的；

(5) DEC 系统研究中心开发的全球名字服务（Global Name Service, Lampson, 1986）目标：
a) 处理任意数量的名字，并为任意数量的管理组织服务
b) 长生命期
c) 高可靠性
d) 故障隔离
e) 容忍怀疑

2. 一般的命名方式

在计算机系统中，每个对象一般有两个名字，一个是由用户识别的文本名（符号），另一个是由系统使用的内部名。
由于系统可以有多个用户，目录常常组织成层次结构。
大多数系统允许用户设置一个默认目录或当前目录。

3. 分布式系统中的命名方式

(1) 名字管理部分的主要功能
(2) 分布式系统中常用的命名方案
f) 绝对命名
全系统范围唯一的、无二义性的，通常是由时钟或计数器之值产生的位串。
g) 相对命名
依赖于使用它的上下文，对于不同的使用者，一个对象的名字可以是不同的。
h) 层次命名
系统被划分成若干组，每组给定全局唯一的组名，每组中的每个对象在组内给定唯一的名字，一个组中对象名还可按此方式进一步划分成若干组。

4. 名字服务器的设计（name server）

(1) 功能
将一个符号串映射成系统内唯一的物理地址。名字服务器管理着包含有"名字及其物理地址"的对照表。
系统中的所有服务程序都由名字服务器来寻址和定位。
(2) **中央方式**：全系统仅有一个名字服务器；

(3) **复制方式**：每个场点都有一个名字服务器的副本，用以管理该场点上的所有服务程序及本场点与其它场点间相互请求的服务信息；

(4) **分划方式**：系统由若干个子系统组成。

5. 分布式系统的透明性

(1) 命名方案与透明性问题极为相关，系统的透明性隐含了下面的事实：

a) 资源的位置不应嵌入其名字中；

b) 名字应该是全局唯一的，相同的名字应该有相同的效果。

(2) 透明性与下面几点要求相冲突

a) 局部自治区

由于强调透明性而牺牲了（部分）自治性；

b) 优化

希望知道资源位置的显示信息，并对它进行控制，这可能出于优化系统性能的要求；

c) 异构

完全透明性是很难实现的。

命名方案与系统的透明性密切相关，命名服务器可根据给定的名字来进行资源或对象的地址定位，并获取有关的属性信息。

第九章 事务的并发控制

1. 信号量

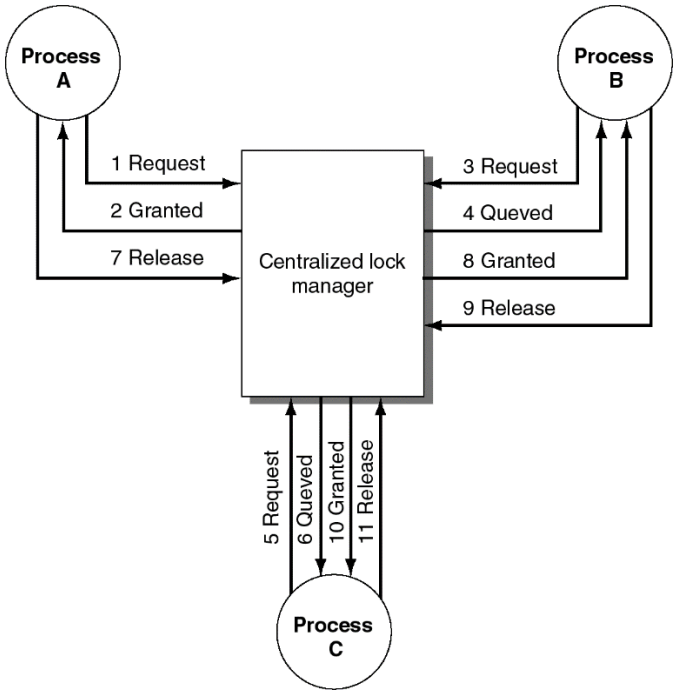
(1) **信号量**：依赖于程序员；不适当的使用会死锁。分布式环境中很难实现，必须保持信号量数据的绝对一致性。

(2) **管程**：编译器支持的编程语言结构。

编译器依靠共享内存实现信号量，没有共享内存，就不能使用管程。

对事务的调度要保证对共享数据的执行效果与其串行调度等价，服务器可通过串行访问数据项来实现串行等价。

(3) **锁**：串行结构的实现



(4) 可串行性（化）：两个事务的全部冲突操作对应相同的顺序执行。

(5) 常用的三种并发控制

为保证可串行性，常用的三种并发控制方法：加锁、乐观并发控制、时间戳定序。

a) 加锁：

当事务完成时解锁，当一数据项被加锁，则只有加锁的事务可访问它，其它事务或者等待锁被解开，或者在某种情况下共享锁。

使用锁会导致死锁，即事务彼此等待解锁。

b) 乐观法：

事务执行到提交前，在允许提交前，服务器完成一个检查，发现已完成的操作是否与相同数据项上的其它并发事务的操作发生冲突，若冲突，服务器终止它。

c) 时间戳定序：

服务器记录读写每个数据项的最近时间，且对每一操作，要比较事务的时间戳和数据项的时间戳，以决定操作是否可立即执行、或被延迟、被拒绝。

2. 锁机制

读　读　　不冲突

读　写　　冲突

写　写　　冲突

(1) 读锁：其它可读，但不能写，有一个事务即加一个锁；

(2) 写锁：写之前获得，不能读或写（再写）；

(3) Lock & UnLock 操作

(4) 锁的粒度

锁的粒度越小，加锁就可以越精确，也就能实现更大的并行度。

同时，锁的粒度越小，就需要更多的锁，这样开销也就越大，也就更容易导致死锁。

3. 乐观并发控制

(1) 基础：在大多数应用中，两客户的事务访问同一数据项的可能性很小。

(2) 三个阶段： 故意框图扩音机表面年

a) 读出阶段：

每个事务修改的每个数据项都有一个**试用性**版本，所有的读操作都是在数据项的**提交版本**上完成的，不会读出"脏"数据。

b) 验证阶段：

当接到 Close Transaction 请求，就检验事务是否存在对数据项上的操作与其它事务在**同一数据项上**的操作发生冲突，若验证通过，则事务提交，若验证失败，则要采用某种形式的**冲突解决**策略。

c) 写入阶段：

若事务通过验证，它的试用性质版本中的所有修改记录将设置成永久的。

事务验证即验证采用读/写冲突规则保证其特定事务的调度关于所有其他的重叠事务是串行等价的。

事务验证：是基于一对事务 Ti 和 Tj 的操作间冲突的，事务 Tj 关于一个重叠事务 Ti 是可串行化的，那么它们的操作必须遵守以下规则。

Level of Support	Tools Provided
Hardware	Atomic operations
	Spin locks
	Mutex hardware operations
	Cache concurrency control
System Support	System Libraries utilizing hardware support
	System support for concurrency with multi-threaded applications
	Lock Manager
	Token Passing mutual exclusion
Language Support	Semaphores.
	Critical regions
	Monitors
	Mutex operations using system libraries

(3) 优点：避免了死锁，允许最大的并行度。

(4) 缺点：有时会失效，所有的事务都必须退回重新运行一遍。
在重负载的情况下，比较严重。

4. 时间戳定序

每个事务开始时都被分配了一个唯一的时间戳，来自事务的申请就可以根据它们的时间戳获得一个**全序。**
服务器可以用自己的时钟来分配时间戳，或者它可采用当分配时间戳时就加 1 的计数器——“伪时间”。
优点：不会出现死锁。
缺点：在于实现的复杂性，这将导致降低性　硬件、系统和语言对并发支持的总结

第十章 分布式事务

1. 事务

(1) 事务的 ACID 特性：
原子性
一致性
独立性
持久性

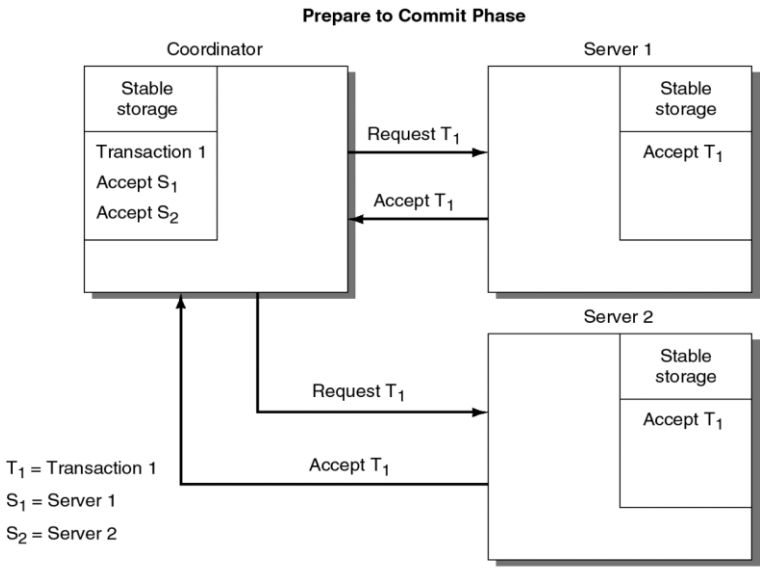
(2) 分布式事务：其活动涉及多个服务器的事务；

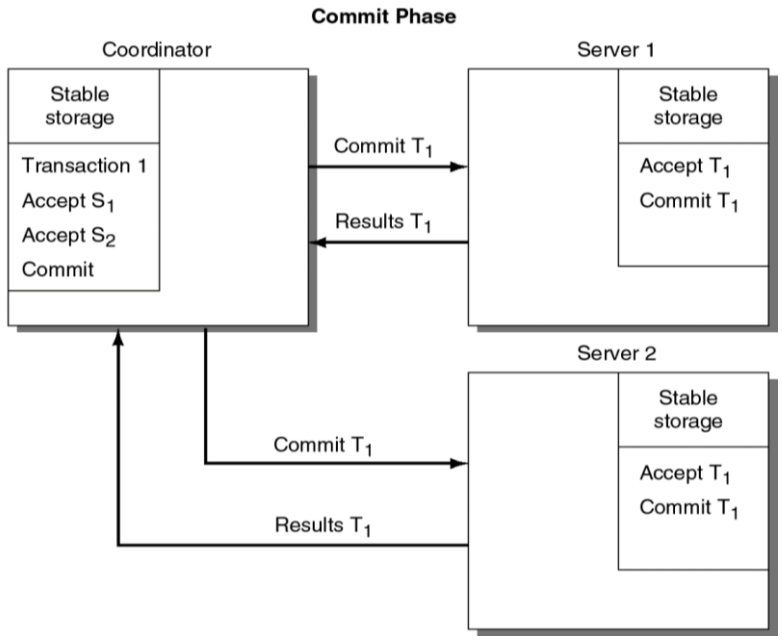
(3) 嵌套事务：间接涉及多个服务器的客户事务；

每个服务器对其本地数据实行本地并发控制，以保证事务在本地可串行，分布式事务必须在全局可串行。
在某些情况下，事务在本地服务器上是串行的，但同时可以发生不同服务器间的依赖循环，从而导致死锁。

(4) 分布式事务的**协调者**
事务的第一个服务器成为事务的协调者，负责终止或提交事务，以及增加其他被称为参与者的服务器，协调者管理着一个参与者表列，而每个参与者都记录了协调者服务器的标识。

(5) 原子提交协议：
两阶段提交协议





(6) 两阶段提交协议中的超时

若协调者失败了，则参与者不能获得应答直到协调者重新启动，这会导致参与者处于不确定状态的额外延迟。

当客户发送了 close transaction 给协调者，参与者只能检测到这种情况：长时间没有特定事务的请求。

由于在这个时期没有作出决定，参与者可在一段时间后单向决定终止。

(7) 事务实现中的问题：

a) 预读写

如果一个事务对一个数据进行读操作，而这个数据正在被其他将要退出的事务操作，这时预读问题就发生了。

如果一个事务对一个数据进行写操作，而这个数据正在被另一个事务操作，预写问题。

涉及到独立事务的相互作用。

事务管理服务采用延迟执行就可以避免这些问题的发生。

延迟执行使得正在被事务操作的数据不能被其他事务读写，直到当前事务进入提交阶段或者中途退出。

延迟会降低系统的性能。

b) 中途退出（domino effect）

如果其他事务看到了与退出的事务有关的结果，这个事务的退出可能遭受 domino effect。

任何以预写的数据为执行基础的事务在预写的事务中途退出后，也都必须退出。

c) 保证恢复能力

如果另一个未完成的提交请求涉及到相同的数据，则应延迟当前的提交。

暂时拷贝可以保存在本地的易失存储器中，如果一直不进入提交阶年段，则删除暂时拷贝。如果进入提交阶段，则把暂时拷贝复制到永久的不易丢失的存储器中。

(8) 分布式事务的并发控制

a) 分布式事务的锁机制

由于服务器彼此独立地设置它们的锁，有可能不同服务器将不同的次序加于事务之上，在这种情况下，这些不同次序会导致事务间的循环依赖，出现分布式死锁情况。

在嵌套事务中，为了避免层次间的潜在冲突，父事务不允许与它们的子事务并发执行。嵌套事务从它们的祖先那里继承锁。对一个获得数据资源读锁的嵌套事务，该数据资源写锁的持有者必须是它的祖先。当一个嵌套事务提交时，它的锁被其父母继承。当嵌套事务中止时，它的锁被解除。

b) 分布式事务的时间戳定序

为实现在所有服务器上的相同次序，服务器必须在它们的时间戳次序上达成一致。

在分布式系统中，要求每一个事务可以分配到环境中**唯一的时间戳**。该事务在调用其他服务器的资源时，同样也把该时间戳发送给相应的服务器，以使该服务器对事务进行合理的调度。分布式系统中的服务器共同负责保证它们按与串行效果等价的方式完成。

但是时间戳的分配还存在着一致性的问题。一个分布式系统包含很多不同的地点和个别的计算机系统，每一个地点和系统都有其各自的本地时间，各处系统的时钟也会偏移。因此各个服务器间的时钟可能不同步，从而造成时间戳分配的不一致性。为了保证事务次序同在实际时间中开始的次序一致，通常采用**同步本地物理时钟**的方法。

c) 分布式事务的乐观并发控制

服务器在验证开始时，分配事务号，事务根据事务号的次序排序。分布式事务被一组相互独立的服务器集验证，每个服务器验证访问其数据项的事务。所有服务器的验证发生在两阶段提交协议的第一阶段。

(9) 分布式死锁

超时 解除死锁。

寻找事务等待图中的环路来实现。

(10) 死锁检测方案

伪死锁

带复制数据的事务

事务服务器上的数据项可被复制，以提高可用性和其他性能。

单副本可串行性：事务在各客户的复制数据项上执行的效果仿佛是他们一次只对单个数据项执行。

Read one/ write all

读操作只由单个复制管理者执行，而写操作则由所有的复制管理者执行。

它不是一个实用的方案，因为当某些复制管理者失效时，它无法运行，设计有效副本方案，用来允许一些复制管理者暂时失效。

其策略是：客户的请求可由任一有效复制者执行，而客户的更新请求必须由包含该数据项副本的组中所有有效复制者执行。

第十一章 恢复与容错

1. 具有容错能力的应用程序

基于事务型：

基于进程控制型：

2. 事务恢复

事务的原子化特征要求：所有已提交的事务对数据项的影响都已反应到数据项中，所有未提交或异常终止的事务对数据项的影响应全部撤消。

假定：服务器运行时，将所有的数据项都保存在易失性存储器中，并把所有已提交的数据记录到恢复文件或文件中。

3. 容错

(1) 同步故障

(2) 服务器失效故障

(3) Byzantine 故障

5000 red army

3000 blue1

3000 blue2

第十二章 分布式共享内存

1. 分布式共享内存（Distributed Shared Memory, DSM）

(1) DSM 主要处理方式

a) 基于硬件的方式

b) 基于虚拟页的方式

c) 基于库的方式

2. 设计和应用

(1) 数据结构

(2) 同步模型

(3) 一致性模型

a) 严格一致性模型

读操作所返回的值必须总是反映最近更新的结果。

必须有全局时间。

b) 顺序一致性模型

分布式系统中所有成员和它们的进程共享一个通用视图，此视图记录了对于共享内存访问操作的顺序。

不依赖一个全局时间，而是更强调各事件保持同一的顺序。

c) 偶然一致性模型 （casual consistency model）

如果事件是偶然相关的，那么它们就必须以相同而且正确的次序出现。

如果事件将修改同一个共享数据集合，那么它们就是偶然相关的。

在关系图中，没有偶然关系的事件就是无关事件，也就不受这种一致性模型的约束。

d) PRAM 一致性模型

在每个地点看来，从同一位置来的操作之间的事务顺序是相同的，而它们和系统中其它来自不同位置的操作之间的顺序则因观察地点的变化而有可能不同。

不同位置在视图中是允许并行操作的。

e) 处理器一致性模型

对于一个特定的存储位置，这个系统必须就此位置的所有写操作顺序达成一致。

分布式系统副本复制和一致性

多副本的存在是提升一个分布式系统可靠性、可用性、性能以及可扩展性的必要手段，有点像"狡兔三窟"，一个出口堵上了，还有其它的备选出口可供逃生。复制可以提高系统的可靠性显而易见，多个副本可以用于分流（如**数据库**的一主多从结构）也可以用于加快响应时间（如 cdn），这使得复制具有增强系统可用性和扩展性的效用。实现数据复制，不仅会涉及到副本的管理（包括副本的存放位置、多副本之间内容的分发），还包括如何保持多个副本之间的一致性，而复制的一致性要求同复制的可用性、可扩展性要求之间是矛盾的，对一致性的要求越高，响应的速度就会越慢，在实践中需要根据具体的应用需求进行折中。

从整个系统的角度出发来实现系统级的一致性，称为系统级一致性；从单个用户的角度出发来实现单用户自身的一致性，称为用户一致性。根据一致性的强弱程度又可分为严格一致性、强一致性和弱一致性，典型的是放宽了数据传播时间要求和一致性强度的最终一致性。

1. 系统级一致性

顺序一致性 SC (Sequential Consistency) 模型是一个最严格的也是程序员最乐于接受的存储器一致性模型，对于满足顺序一致性的多处理机中的任一执行,总可以找到同一程序在单机多进程环境下的一个执行与之对应,并且两者的结果一样。

而弱一致性模型，其主要思想是通过硬件和程序员之间建立某种约定，由程序员来负担一些维护数据一致性的责任，从而放松硬件对访存事件发生次序的限制。具体做法是把同步操作和普通访存操作区分开来，程序员必须用硬件可识别的同步操作把对可写共享数据的访问保护起来，以保证多个处理器对可写共享单元的访问是互斥的。

(1) 严格一致性

一般是指按照所有读写操作实际发生的绝对时间执行操作，读操作返回的必须是最新的写操作写入的数据。这就要求全局时钟的存在，而在一个分布式系统中很难做到将所有进程的操作按照按绝对时钟进行排序，由此引发的恶性竞争、单点故障、消息排序等不仅极大的增加了系统实现的复杂性，还会降低系统的可用性和可靠性，使得这种方案得不偿失，不适合具有一定规模的系统。

(2) 顺序一致性

所有进程看到一个相同的总体读写操作顺序，且每个进程上的操作也依序出现在这个总体操作顺序中，属于强一致性。在共享存储多处理机中，若任一处理机都严格按照访存指令在进程中出现的次序执行访存指令，且在当前访存指令彻底完成之前不能开始执行下一条访存指令，则此共享存储系统是顺序一致性的。不得不介绍顺序一致性概念的最初来源，以便于理解其原本的含义，详见本博文章翻译：http://blog.csdn.net/csq_year/article/details/46744333。

顺序一致性包含两层含义

a) 进程必须按照程序指定的顺序执行程序,不能重排操作；

b) 所有进程上的操作就好像以某种总体顺序执行且所有进程看到的总体顺序均相同，每个进程的操作在总体顺序中出现的顺序和它自身的相同。

显然，顺序一致性并未从绝对时间的角度要求对操作进行排序，因此绝对时间上最后的写操作并不一定是进程能看到的最后一个写操作，对于所有进程的任何交叉执行顺序都是可以接受的。进程必须要看到（读）其它进程的写操作，但只能看到自身的读操作，看不到其它进程的读操作。在这里有一个暗含的理想化假定：即硬件的执行环境是顺序一致性的，而事实上现代计算机为了优化性能在硬件层并不是顺序一致性的，但提供了一些机制用于支持所有进程实现顺序一致性。如果硬件层不支持顺序一致性，那么要达到顺序一致性就必须要对所有的操作进行顺序化，可以按 FIFO 的方式处理请求，这就需要一个全局的 FIFO 队列（串行化是实现顺序化的一种最简单的方法）；也可以把所有访问同一数据的请求进行顺序化，而访问不同数据的请求则可以乱序执行。

(3) 因果一致性

所有进程中有因果关系的读写操作必须是顺序一致性的，没有因果关系的操作可以乱序执行，这种一致性强度要弱于顺序一致性。但是在工程实践中确定所有操作间的因果关系并不是一件容易的事情。

(4) 同步化操作

顺序一致性和因果一致性是在读写操作层面上定义的，这些模型最初是为了共享内存的多处理器系统开发的，在硬件层面上真正实现的。这些一致性模型的粒度和应用程序提供的粒度在很多情况下并不匹配，共享数据的应用程序之间的并发往往是通过互斥和事务的同步化机制来控制的，这种机制一般是将读写共享数据的一系列操作封装成一组操作单元，并通过使用同步变量来互斥的执行这组操作单元中的操作。

(5) 最终一致性

当没有更新产生时，在一段时间后，所有进程最终都能看到存储对象的最新值。一个达到最终一致性的系统通常被称为是收敛的或者复制集收敛的；在到达收敛之前，系统可能返回之前的任何值。

2. 用户级一致性

用户级一致性定义中的"进程"可以认为是具体应用场合中的一个会话或客户等参与者。

(1) 单调读一致性（monotonic-read）

如果一个进程读取数据项 X 的值，那么该进程后续任何读取 X 值的操作都将得到那个值或者更新的值。单调读一致性从时间的维度上保证了如果进程在 t 时刻看到了 X 的值，那么以后不会在看到较老版本的值。

(2) 单调写一致性（monotonic-write）

一个进程对数据项 X 执行的写操作必须在该进程对数据项 X 执行的任何后续写操作之前完成。单调写一致性同以数据为中心的 FIFO 一致性类似，本质是同一进程上执行的写操作必须在任何地方以正确的顺序执行。单调写一执行保证在一个副本上执行数据更新时，在此之前（其它副本上执行的）的所有数据更新都将首先执行。

(3) 写后读一致性（read-your-write）

一个进程对数据项 X 执行的写操作总是被该进程后续对 X 的读操作看见。一个写操作总是在同一进程执行的后续读操作之前完成，不管这个读操作是发生在哪个副本。

(4) 读后写一致性（write-following-read）

同一个进程对数据项 X 执行的读操作后的写操作，保证发生在与读取的 X 值相同或更新的值上。

3. 副本复制和内容分发

(1) 副本在哪里？

这里包含两个层面的问题，一是副本服务器该放置在什么位置；二是副本的内容该如何组织存放在哪些服务器。有多种不同的方法可以度量副本服务器的最佳放置位置，比如到达一定区域内所有用户的平均响应时间最短的地理位置或者简单的按 idc 机房放置。而内容如何放置在副本服务器上不仅会涉及到数据的组织问题，还涉及到系统性能、可用性和容灾等问题。常见的是把源数据进行分片放置在不同 idc 下不同的机器上，并保证同一副本的同一分片数据不会出现在同一 idc 下的同一物理机上。

(2) 传播什么？

分发的内容有三种形式，一是仅传送发生了更新的消息通知：可以使用无效化协议（invalidation protocol）使用传播通知这种方式通知其他拷贝已经发生了更新，可以指定数据存储的哪部分发生了更新，不至于使得副本中的全部数据均失效；由于它仅传送关于哪些数据不再有效的信息，因此对带宽的消耗很小，适用于读/更新（写）比率比较小的应用场景；二是传送更新了的实际数据：适用于读/更新（写）比率比较大的应用场景，这时更新的有效性比较大，被修改的数据很可能在下一次更新之前被读取，不传播所有相关的数据而将发生修改的数据记入日志，传送这些日志可以节约带宽；三是传送更新的操作而不是数据：

不传送任何修改的数据，而是告诉其它副本该执行什么操作，也被称为主动复制（active replication ），这种情况一般只适用于那些更新是由易回放低耗时的操作引起的应用场景，当被更新的数据量比较大时比较有效，这样可以减少数据传送造成的带宽消耗和时延。

(3) 谁来发起传播？

根据更新传播是由谁发起的，可以分为推式和拉式。一是推式的方法（push-based approach）也被成为基于服务器的协议（server-based protocol ），能够主动的将更新传送至其它副本而不必等待其它副本请求更新。适用于多个副本之间需要保持较强一致性的情形，这时副本上的读/更新（写）比率一般比较高，传递的更新有效性比较高，它使得一致的数据在请求时能尽可能快的有效。二是拉式的方法（pull-based approach）也被称为基于客户的协议（client-base protocol ），主动向其它服务器请求该服务器所持有的任何更新，通常被用户客户的高速缓存，适用于读/更新（写）比例比较低的应用场景。基于推式协议中的一个重要问题是服务器需要跟踪所有的客户高速缓存，状态较多的服务器不但通常缺少容错能力，而且跟踪客户的高速缓存在服务器端的代价可能相当大。由于基于推式和拉式的方法在网络通信、更新时延以及实现代价等方面各有利弊，导致出现了一种更新传播的混合方式-基于租用的更新传播。租用（lease）是服务器所做的承诺，承诺在给定的时间里把更新推送给用户。当租用到期时，客户被迫轮询服务器并在必要的时候拉取更新。另一种方法是在上一个租用到期时客户及时请求一个新的租用以实现更新的推入。租用为基于推式的策略和基于拉式策略之间的动态转换提供了一种灵活机制，持有更新数据的一方可以根据自身的负载情况调整租期的长短和发放的租用的数量，以便在服务性能和数据一致性之间取得合理的折中。

(4) 如何传播？

传播方式可以分为单播和多播。在局域网环境下多播是可用的，对于推式协议，通过多播来推送更新的效率要明显高于通过单播推送数据。而对于拉式协议，发起方一般是单一的客户或服务器，采用单播可能是唯一可行的方法。

4. 一致性的工程实现

理论终将需要接受实践的洗礼才能落地开花结果，在工程实践中简单易理解的东西总是最受欢迎的，过于复杂的理论或设计即便很优雅，也难以广泛应用且难以证明其正确性。在一个分布式系统中，如果仅有一份共享数据而没有副本数据或本各种缓存数据的存在（即每次访问均直接访问共享数据），那么引起顺序一致性得不到满足的将是对共享数据的访问冲突；在这种情况下，一般通过使用同步变量保证互斥访问共享数据就可以达到顺序一致性。如果有副本或者各种缓存的存在，那么问题就会变得复杂起来，不仅要保证互斥的访问共享数据，还要保证各个副本之间满足某种一致性要求，而整个系统的一致性强度取决于所有环节中最弱的那个一致性强度。

基本上使用最广泛的还是操作全局串行化的一致性模型，包括顺序一致性模型、使用同步变量的弱一致性以及原子性事务处理。根据是否存在一个数据的主拷贝（即所有的写操作都**只能**被转发到该拷贝），将顺序一致性协议分为以下两类。当不存在这种主副本时，写操作可以在任何副本上启动。

(1) 基于主备份的协议

数据存储中的每个数据项 x 都有一个关联的主拷贝，该主拷贝负责协调 x 上的写操作。根据主拷贝是被固定在单一服务器上，还是可以切换至发起写操作的副本上可以区分各种主备份协议。

a) 远程写协议

最简单的基于主备份协议的是所有读写操作都在单一服务器上执行，实际上此时根本没有数据被复制，但服务可以是分布式的，退化成简单的 C/S 模型了。比较常见的是允许进程在可用的副本上进行读操作但是必须将所有写操作转发到一个固定的主拷贝上，常称为主机备份协议（primary-backup protocol）。常见的一种阻塞式更新方式是对数据项 x 的写操作都转发到主拷贝上（或 x 的主拷贝上），主拷贝在本地先进行更新，然后再将更新传播至副本，每个副本再完成更新后向主服务器发送确认消息，待所有副本服务器完成更新后，主服务器向请求方发送确认消息，完成一次更新。一种非阻塞的更新方式是主服务器在主拷贝上完成更新后就向请求方返回更新成功的消息，然后再向副本服务器发送更新通知，而不必等待副本服务器完成更新。主机备份协议提供了一种实现顺序一致性的直接方法，由单一的主服务器对所有进程的写操作进行排序。无论读请求落到哪个副本上，所有进程都将以相同的顺序看到所有的写操作。在阻塞协议中，落在各个副本上的读请求均能看到最近执行的写操作的结果。而对于非阻塞协议则必须要采用一定的措施使得更新副本在规定的时间内完成。

b) 本地写协议

第一种协议是每个数据项都只有一个单一的拷贝（即不存在副本）。每当一个进程要对数据项 x 进行更新时，首先要将数据项 x 传送到该进程，然后再执行相应操作。这个协议本质上是建立了一个完全分布式的、非复制的数据存储，其一致性是显而易见的，因为没有共享的存在。存在的问题是这种更新方式会不断的移动数据的位置，需要跟踪记录所有数据的位置。

另一种变种协议是将主拷贝在多个执行写操作的进程之间来回迁移。每当更新一个数据项 x 时，先定位其主拷贝，然后将该主拷贝迁移到进程的本地位置，主要优点是多个相继的写操作都可以在本地执行，而执行读操作的进程仍然可以访问它

们的本地拷贝（但是不能执行更新操作，否则会违背一致性）。这种变种协议可以用于以离线方式操作的移动计算机。离线前移动计算机成为它欲更新的数据的主拷贝，离线时所有操作都在本地执行，而其它进程则可以执行读操作，但不能执行写操作（以免造成数据不一致）。再次连线时，更新从主拷贝传播至所有副本服务器，使数据存储再次达到一致性。

(2) 基于复制的写协议

在复制的写协议中，写操作可以在多个副本上执行，而不像主备份协议那样只能在一个主拷贝上执行。主动复制和基于多数表决的一致性协议的区别在于：主动复制中的所有操作被转发到所有副本。

a) 主动复制

每个副本有一个相关联的进程，该进程执行更新操作；同时该操作被发送到每个副本，也可以使用前述讨论的方法来发送更新。主动复制时需要在不同的副本上以相同的顺序执行，因而需要一个全序的多播机制，可以使用中心协调器（定序器：sequencer）来实现全序，可以将所有操作转发到定序器，由定序器为每个操作分配一个唯一的序列号，接着将这些操作转发给所有副本，副本按照序列号顺序执行操作。显然，这种全序多播的实现与基于主备份的一致性协议类似。

b) 多数表决协议

基本思想是要求进程在读或写一个复制的数据项之前向多个服务器提出请求，并获得它们的许可。在 Gifford 的方案中，一个进程要读取一个具有 N 个副本的数据项，必须获得 R 个以上任意服务器（读团体：read quorum）的许可；执行写操作之前也必须获得 W 个以上任意服务器（写团体：write quorum）的许可。R 和 W 的值应该满足以下两个限制条件：

- ✧ R+W>N;
- ✧ W>N/2;

第一个限制条件用于防止读写操作冲突；第二个限制条件用于防止写写操作冲突。只有在适当个数的服务器同意参与数据的读写后，进程才能读或写。一般来说，要为每个更新的数据项标记一个新的版本号，以便于进行数据新旧程度之间的比较

(3) 高速缓存一致性协议

高速缓存形成了一类特殊的复制，因为它们通常是由客户而不是服务器控制。高速缓存一致性协议保证高速缓存与服务器启动的副本一致，与之前讨论的两种由服务器方控制的一致性协议在本质上没有多大区别，目的都是要保持数据一致性。不同的高速缓存解决方案在实际检测不一致性的时间上不同。静态解决方法假定编译器在运行前执行必要的分析，并确定哪些数据可能因为被缓存而导致了不一致，对此编译器仅插入一些避免不一致性的指令。动态解决方案是在运行时检测不一致性，比如可以检测服务器查看被缓存的数据自从被缓存后是否被修改过。

当共享数据可以被缓存时，实现高速缓存一致性有两种方法：一是每当一个数据项被修改后，服务器向所有高速缓存发送无效化消息；二是仅传播更新。当使用只读高速缓存时更新操作只能由服务器执行，然后服务器根据某个分发协议确保更新被传播到各个高速缓存。很多系统使用基于拉式的方法，即如果客户检测到高速缓存已经过时，则向服务器请求更新。另一种情况是允许用户直接修改被缓存的数据，并将更新转发给服务器。直写式高速缓存（write-through）采用这种方法，分布式文件系统常使用这种方法。这种方法实际上类似于基于主备份的本地写协议，客户的高速缓存成为一个临时的主拷贝；为了保证顺序一致性，客户必须获得独占写的权限，否则可能导致写写操作冲突。直写式高速缓存将所有的操作在本地执行，理论上性能较其它方法更好。如果允许在通知服务器更新之前执行多个写操作来推迟传播更新，可以进一步提高性能。这种方式促使回写式高速缓存（write-back）出现了，这种方法也主要应用于分布式文件系统。

(4) 用户级一致性协议

在以用户为中心的一致性协议简化实现中，每个写操作 W 都被分配一个全局唯一的标识符，服务器把该标识符赋给提交写操作的用户。对于每个用户，跟踪两个写操作集：一是用户的读操作集即由用户所执行的读操作相关的写操作组成的，二是用户的写操作集即由用户执行的写操作的标识符组成。