# 分 布 式 系 统

宋斌　　计算机学院

电话：13814017756　　邮箱：bin_s163@163.com

# 课程参考书

分布式操作系统　　陆丽娜等译
　电子工业出版社

分布式系统设计　　高传善译
　机械工业出版社

分布计算系统　　徐高潮等译
　高等教育出版社

WEB技术导论　　郝兴伟编
　清华大学出版社

WEB服务实用技术教程　　励志编　　清华
　大学出版社

WEB工程理论与实践　　霍秋艳编　　清华
　大学出版社

# 第一章　概　述

# 1．1　什么是分布式系统

分布式系统是由多个相互连接的处理资源组成的计算机系统，这些资源可以合作执行一个共同的任务，最少依赖于集中的程序、数据和硬件等资源。

其具有以下特点：

(1)分布式系统是由多个处理机或多个计算机组成

(2)这些计算机或处理机可以物理相邻，也可在地理上分散，用计算机网络互连。

(3)这些计算机或处理机组成一个整体，对用户是透明的

(4)一个程序可分散到多个计算机或处理机上运行

(5)系统的表现与单一系统一样

# 分布式系统的发展简史

| 系统名称 | 组织机构 | 网络要求 | 计算机 | 研制日期 |
|---|---|---|---|---|
| CM* | 卡内基.·梅隆大学 | 层次总线 | PDP | 1975 |
| Cambridge DCS | 剑桥大学 | 剑桥环 | LSI-4 | 1979 |
| Locus | 加州大学洛杉矶分校 | 以太网 | PC | 1980 |
| V System | 斯坦佛大学 | 以太网 | Sun | 1982 |
| Mach | 卡内基.·梅隆大学 | 以太网 | Sun, PC | 1985 |
| CORBA | OMG | 互联网 | 任何机器 | 1990 |
| Distributed COM | 微软公司 | 互联网 | PC | 1996 |
| JINI | Sun Microsystems | 互联网 | 任何机器 | 2000 |

# 1．2　硬件观点

1．按体系结构分类：

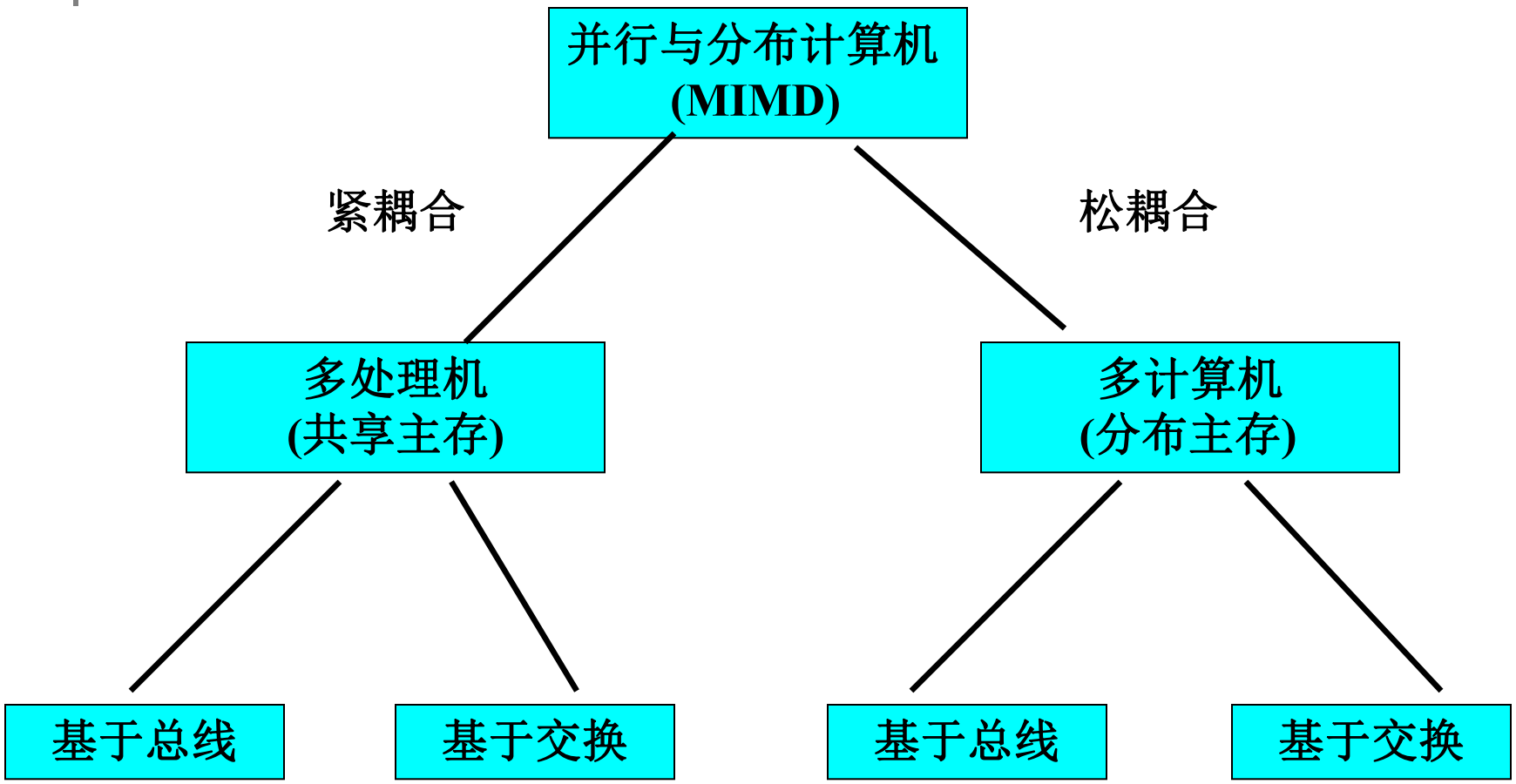
1)单指令流多数据流(SIMD)：它由一个指令部件取得指令，然后将指令同时发往多个数据操作部件并行操作.

典型的结构:阵列处理机

2)多指令流多数据流(MIMD)：由独立的处理机执行各自得到的指令对各自的数据进行操作。

我们讨论的分布式系统均属于此类系统(MIMD)，其又分为紧耦合系统和松耦合系统.

其中:紧耦合是主要为共享主存;松耦合相反主要通过通信和协调




分布式计算机系统的分类

3)两类分布式系统又分为基于总线的结构和基于交换的结构

总线:有共享的总线;

交换:CPU之间有专用的数据通路

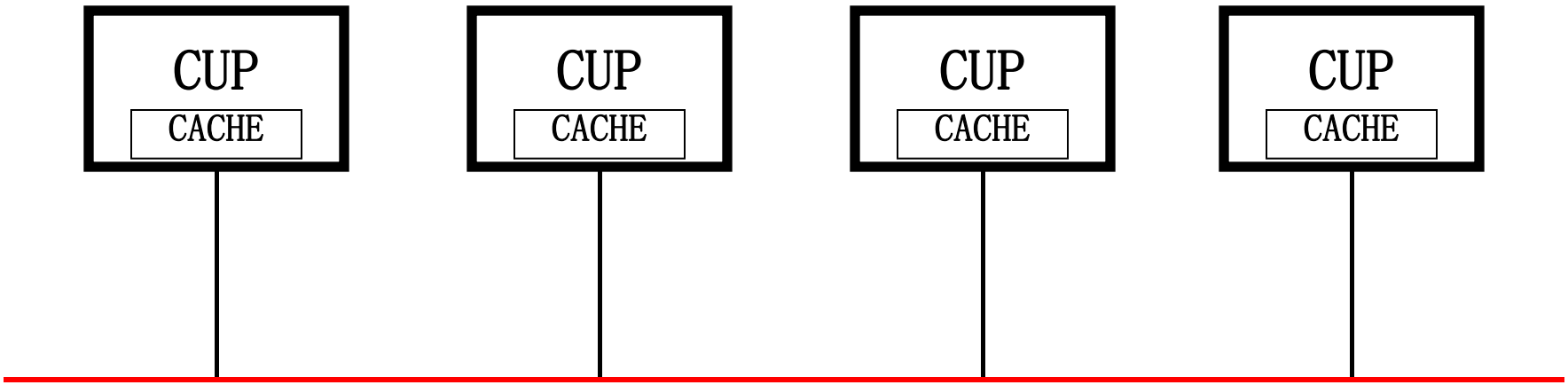
4)具体有四种形式

基于总线的多处理机:每个CPU都与总线直接相连;存储器也是如此

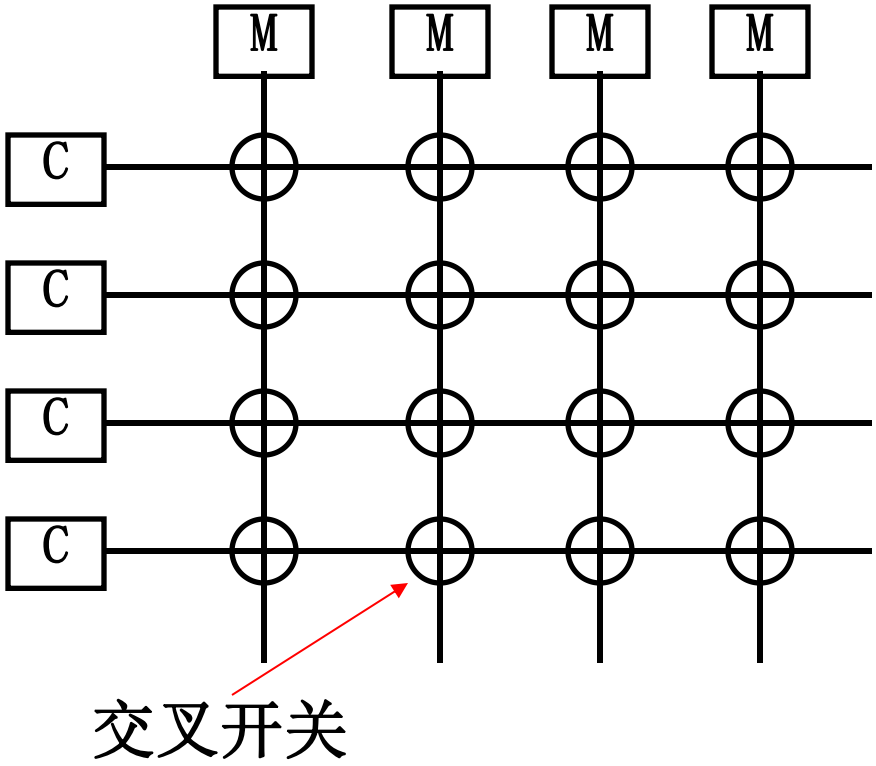
基于交换的多处理机:采用不同的组织方法来连接CPU和存储器

基于总线的多计算机:通过局域网互连

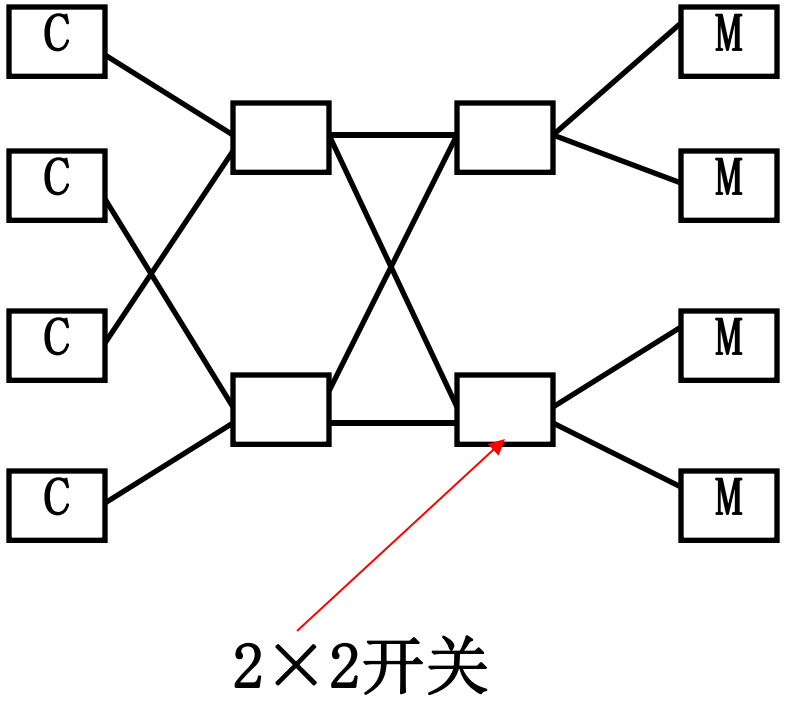
基于交换的多计算机:要保持CPU只与特定的局部存储器相连




基于总线的多处理机

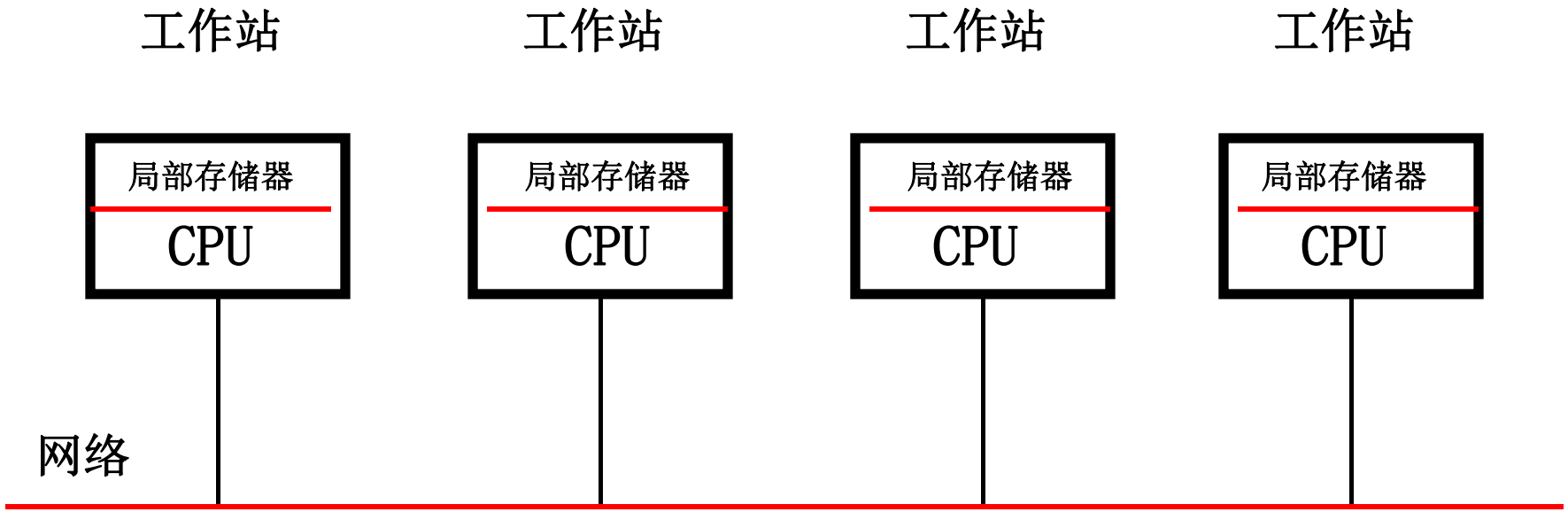



(A)交叉开关

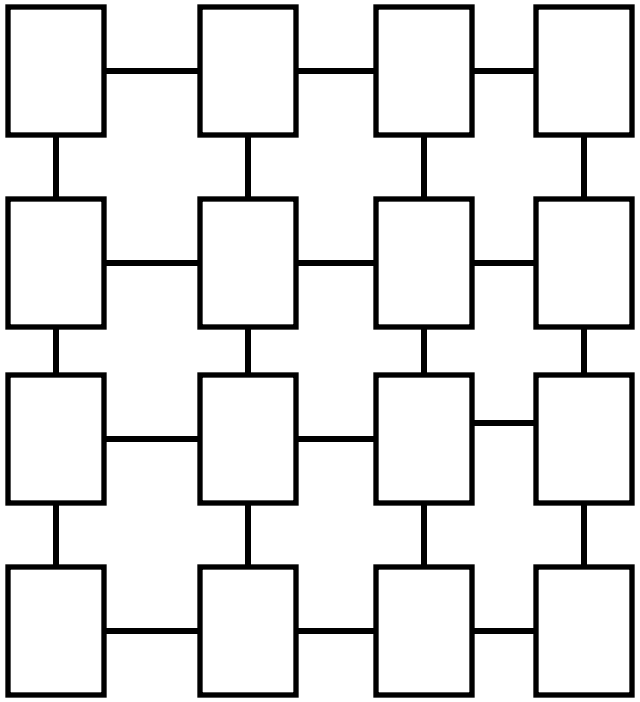



(B)Omegea开关网络

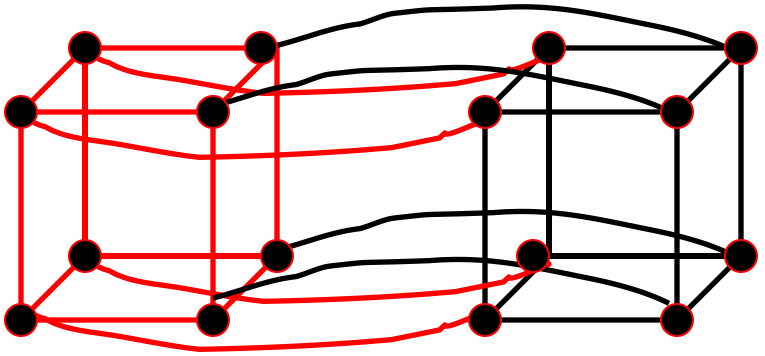
基于交换的多处理机




由局域网和工作站组成的多计算机



(A)栅格结构



(B)超立方体结构

基于交换的多计算机

# 1．3　软件观点

软件观点分两类:

紧耦合的软件系统：独立工作

松耦合的软件系统：合作完成任务

理论上软硬结合共有八种系统,但只有四种有实际意义,

因为多处理机硬件无论使用总线还是交换开关都只能配备紧耦合的软件系统.

常见的软硬件的组织形式:

1．网络操作系统

是一种典型的松耦合的软件与松耦合的硬件相结合形成的系统。网络操作系统的特点就是系统中的每台机器高度自治。它给用户的支持是最低级.

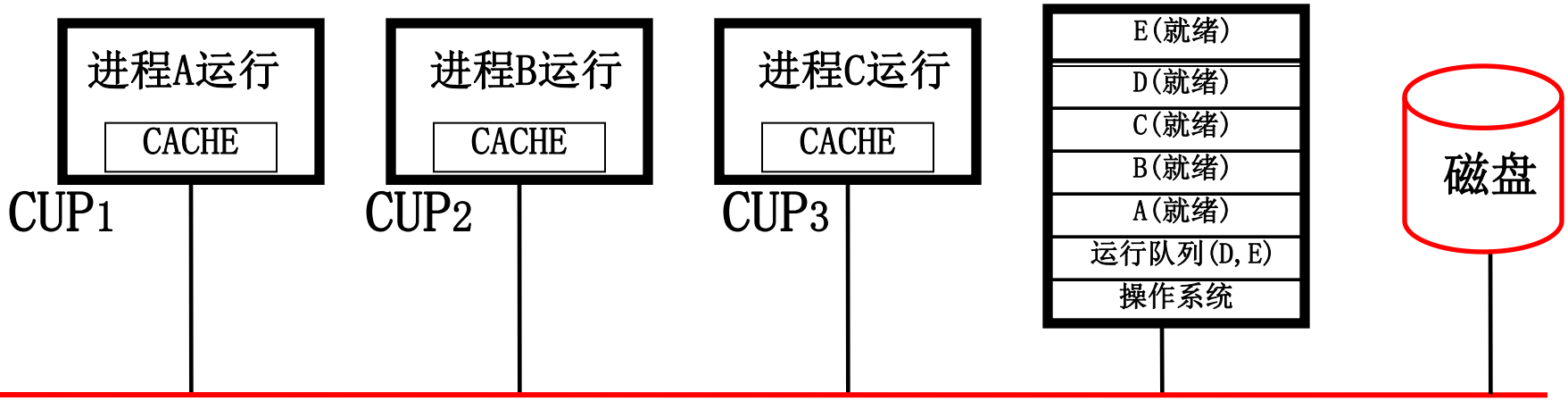
2．分布式操作系统

可提供进程间的通信机理；要具有全局性的保护机制，进程管理；文件系统表现一致；各机器间必需保持相同的操作顺序.建立统一的视图

### 3．处理机分时系统

是一种典型的紧耦合的软件与紧耦合的硬件相结合形成的系统。其主要特征是存在一个运行队列，表示系统中的一组逻辑上无阻塞的，准备运行的进程。

它与前面两种系统的主要不同在于文件系统的组织




拥有一个运行队列的多处理机系统

# 1．4　关键特征

注意:不一定每个分步式系统都具有以下特征,它们只是评价系统大依据,是分布式系统追求的目标.部件的分布是分布式系统的内在特征

通常按以下五个方面讨论分布式系统的主要特征：

1) 资源共享

两种方法：

A)客户/服务器模型，注意:客户机和服务器本身并不一定需是计算机,可为各种处理进程

B)面向对象模型, 注意:将独立存在的资源作为对象处理.

2) 开放性

①可伸缩性:删除系统中的某些软件或硬件单元,系统仍可正常工作.

②可移植性:软件上可用多种版本,硬件即插即用.

③互操作性:数据格式可互换.

3) 并发性

并发性和并行性在分布式系统中是一种内在的特征。

4) 容错性

容错的基本方法为：硬件冗余和软件恢复.表现为故障不显性,分布式系统的冗余颗粒较小,不必进行大系统的备份

注意:分布式系统的基础是网络,但网络是没有冗余,因而系统必须有等待故障的修复.

## 5)透明性

美国国家标准协会(ANSA)定义了八种透明性：

①访问透明性　②位置透明性　⑧并发透明性

④副本透明性　⑤故障透明性　⑥迁移透明性

⑦性能透明性　⑧规模透明性

其中最重要的是访问透明和位置透明，直接影响到分布式系统的表现，前述网络操作系统就没有支持这两种透明性。电子邮件系统支持这两种透明性

# 1．5　用户需求

1)功能：

不仅要完成集中式系统的功能，还必须能完成一些分布式的功能。可通过鉴定改造,革新和演变来实现从集中式系统到分布式系统

2)可重构性：

主要有两种重构需求，一是短期调整，一是中长期改动。

3)服务质量与传统的评价不同,分布式系统从以下角度:

①性能　　②可靠性和可用性

③安全性　　④一致性

# 1．6　分布式系统的优缺点

与集中式系统相比主要具有一下几方面优点

1) 经济：具有较高的性价比与大型机相比

2) 速度：较快的平均响应时间,

3) 内在的分布式：支持新型应用,如计算机协同工作(CSCW)

4) 可扩充性：

5) 系统的可靠性

与分散的工作站和个人机相比具有一下几方面优点：

1)资源共享：分布式系统的目标

2)通信得到加强 ：合作方便

3)可扩充能力：提高运行效率

主要具有一下几方面缺点：

1) 分配处理和存储资源时灵活性不足,

2) 性能和可靠性依赖于网络

3) 安全保密性不足

4) 软件不足

# 1．7　分布式系统的应用

**以计算为主的问题（分布式计算）：**

数学计算，环境模拟，生物和仿生，经济和财政模型，气象预报，动画制作，…

**以数据为主的问题（分布式数据）：**

数据挖掘，信息检索，保险分析，图像处理，…

**以通信为主的问题（网络应用）：**

事务处理，电子商务，远程文件交换，电子信件，…

**标准系统：**

网络文件系统

ATM （银行自动取款机）

分布式数据库

WWW （万维网）

全球定位系统

自动售货终端机

航空管制系统

网络拍卖系统

… …

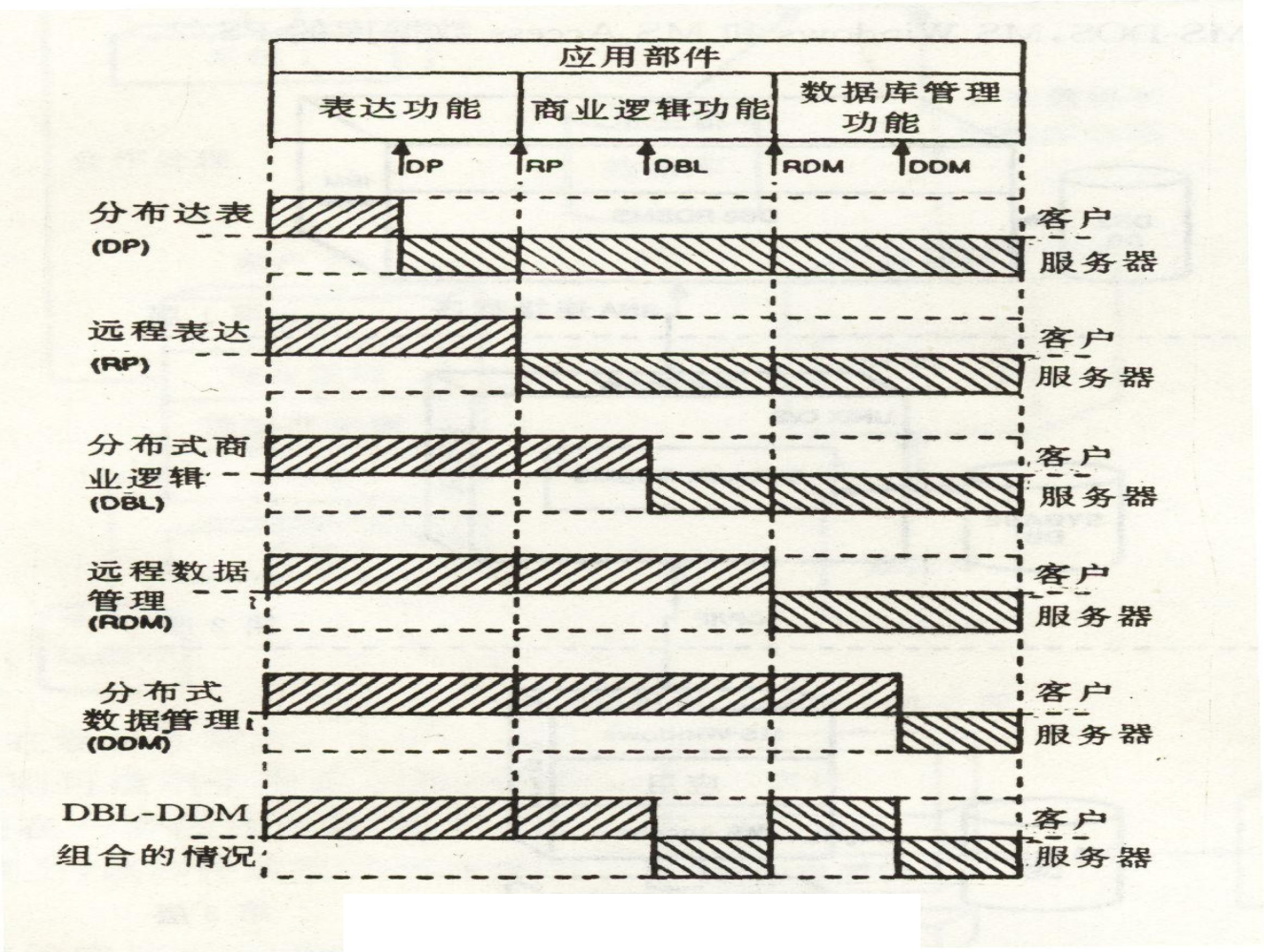
# 第二章 RPC与组通信

分布式系统和单处理机系统的最重要的区别是进程间的通信，单处理机系统的通信可利用共享存储器

当然要完成进程间通信就必须遵循规则即协议

协议的发展：**OSI模型 、ATM模式、客户-服务器模型、RPC（远程过程调用）、WEB服务**

# 分布模型

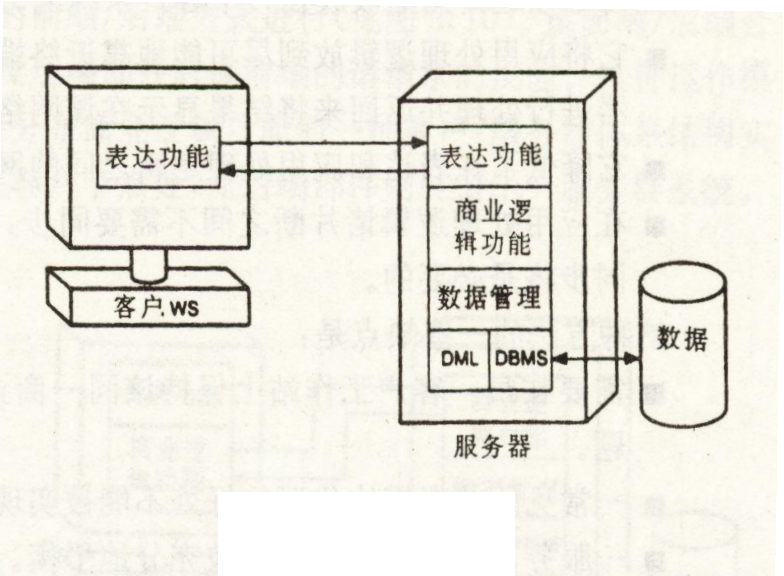

应用部件
表达功能
商业逻辑功能
数据库管理功能
DP
RP
DBL
RDM
DDM
分布达表
(DP)
远程表达
(RP)
分布式商业逻辑
(DBL)
远程数据管理
(RDM)
分布式数据管理
(DDM)
DBL-DDM
组合的情况
客户
服务器

# 表达逻辑

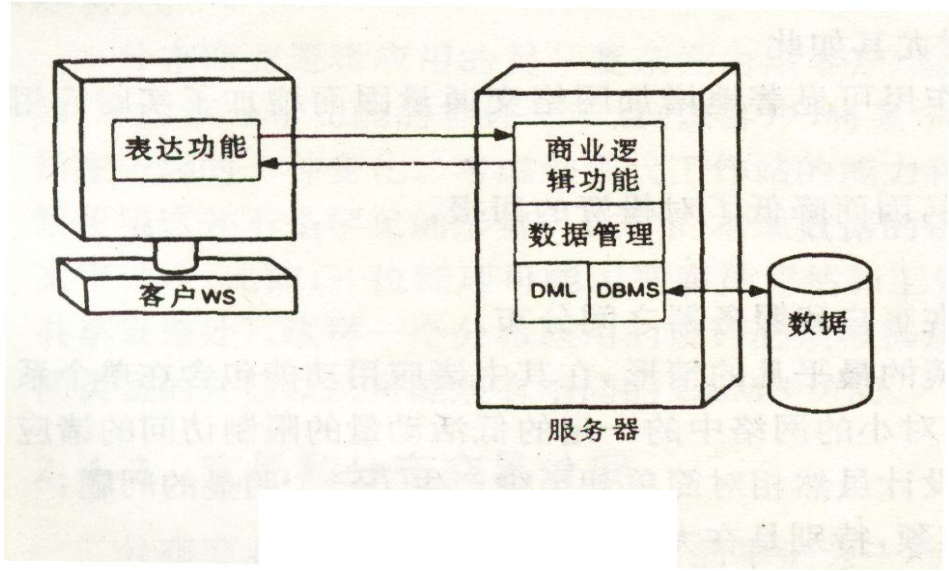
表达逻辑是应用中直接面向用户的部分，主要完成应用的前端界面的处理，如屏幕格式、对话管理、窗口管理等涉及人机交互的工作。

表达功能的划分存在两种风格：

1）分布表达：应用代码的表达部分在两个或多个网络节点之间被划分，由前端和后端组成



**2）远程表达：**应用代码的表达部分被完整地放在一节点上，而应用的其余部分位于另一节点上时，则称此表达为远程表达。远程表达处理是各表达功能和其他应用功能之间的合作处理，通过**RPC**（ Remote Procedure Call）进行。



# 2．1 概 述

尽管客户-服务器模式为分布式系统提供了一种便利的方法，但它存在无法克服的缺陷：其所有的通信是建立的基础都是输入/输出，以它为基础构建分布式会产生应用问题。

RPC （Remote Procedure Call）是一种分布式系统的构造技术，RPC操作一般在本地进程进行过程调用，而在异地计算机上执行调用。采用RPC技术简化了分布式程序的设计，设计者可不必考虑程序间的通信问题，只须考虑程序单元间的同步和出错等问题。

# 2．2 RPC的设计问题

RPC的基本原理为：

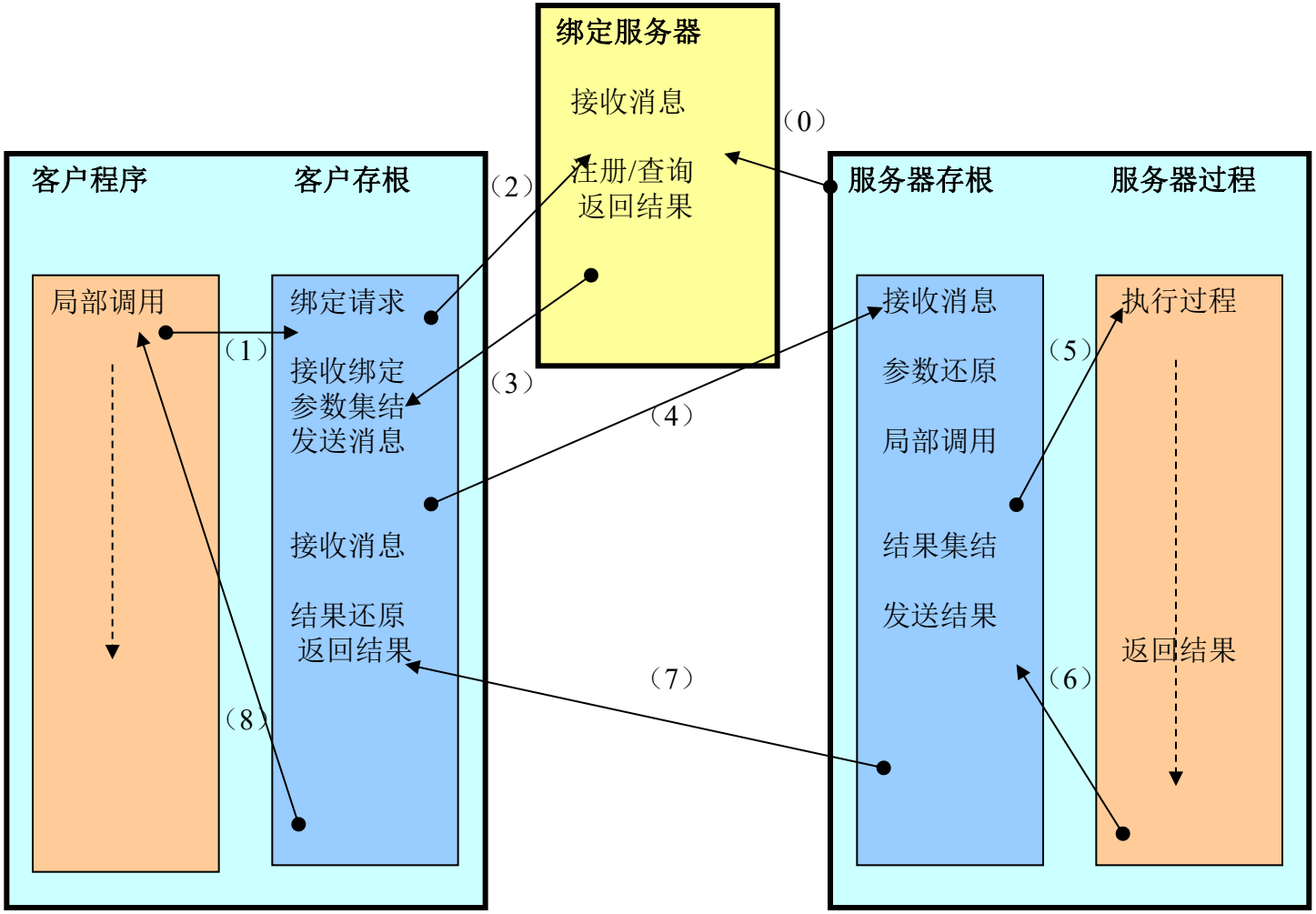
让本地计算机中的某个过程调用远地计算机的另外一个过程.

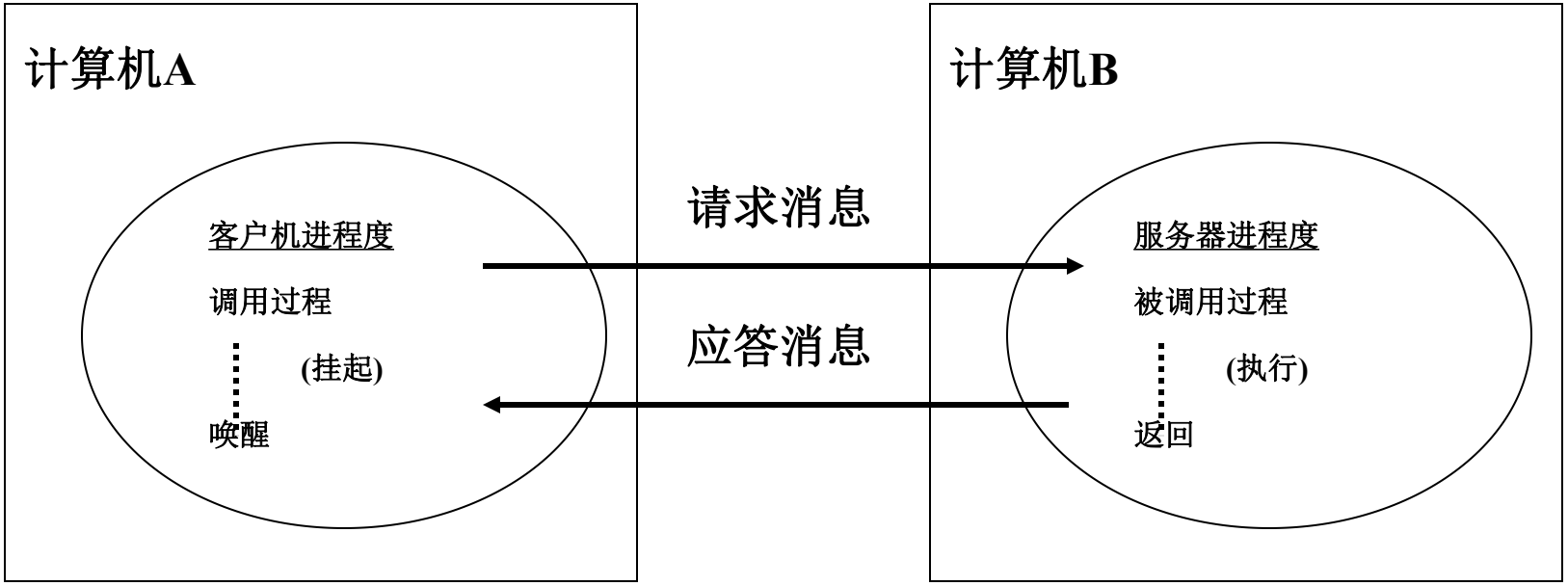
其执行过程是：

本地过程通过消息传递机制将过程调用请求消息传送到远程的一个进程，该过程收到此调用消息，执行被调用过程，然后通过应答消息返回执行结果给调用过程。

与传统的过程调用不同的是参数调用;原因是两台机器的操作系统管理自己不同的地址空间.见图示

# RPC远程过程调用


绑定服务器
接收消息
注册/查询
返回结果
(0)
(2)
(3)
客户程序
客户存根
局部调用
绑定请求
(1)
接收绑定
参数集结
发送消息
接收消息
结果还原
返回结果
(8)
(4)
(7)
服务器存根
服务器过程
接收消息
参数还原
局部调用
结果集结
发送结果
(5)
执行过程
返回结果
(6)




远程过程调用RPC

1） RPC的参数传递

输入参数被放在请求消息中，输出参数置于应答消息中。

2） 参数与结果的装配。

3） 动态联接

动态联接的三方面的工作:

①服务器定位:客户机须指明执行远程过程所在的服务器

②类型一致性检查:通过使用通用RPC界面解决

③保证版本的一致性:因客户机程序和服务器程序是分别编译的

处理①③的办法是使用联接器。联接器是一个程序。

4)RPC调用的语义

可能导致RPC的失败：

①服务器找不到　②请求消息丢失

③应答消息丢失　④服务器崩溃并重新启动

⑤客户机崩溃并重新启动

RPC的目的就是隐藏机器之间的通信

注意:至少一次调用语义;最多一次调用语义

追求的目的:恰好一次调用语义

5）RPC的透明性

RPC虽与本地过程调用相似，但RPC比本地过程调用更易出错。因此需要透明性

6） 异常处理

其异常处理与传统操作系统中的异常处理是一致的。具有异常处理机制，包括异常产生、异常处理。

# 2．3　RPC界面

一个RPC界面是客户机可见的由服务器提供过程的特征，包括过程名及参数类型，参数需注明输入输出以便使RPC软件将参数装配形成消息。既其主要作用就是作为stub生成器(也叫界面编译器)的输入。

1．RPC界面设计的基本原理

RPC界面定义的基本原理就是数据抽象。在分布式系统的设计过程中，RPC界面是客户机程序和服务器程序分别编译和连接的基础。具体来说：RPC系统将一个界面语言提供给用户（程序员），用户用界面语言来定义服务界面。

提供界面语言，就需提供语言的编译器，主要是参数装配，界面编译器又叫STUB生成器。

2．界面定义的处理

1）装配 分客户机和服务器两部分，

即请求消息和应答消息

2）分发 赋于唯一的过程标识符

3）输入和输出参数 服务器程序，界面名是输出型的，客户机程序，界面名是输入型的

3．界面编译(STUB生成器)

stub过程用于将RPC集成到用户的过程中，其在客户机和服务器程序中处理界面时说明。用户程序通过调用stub过程来完成对远程过程的调用。

注意:客户机STUB过程的任务是装配参数并打包于请求消息中,服务器STUB过程通过分析请求消息获取输入参数.

# 2．4 RPC实现

1. RPC协议

选择面向连接协议,目的是使通信更容易;
用标准的还是专用的

目前有两种方案:

A 使用IP协议 原因:

1） IP协议已存在.

2）已在许多系统中使用.

3）UNIX系统支持.

4）目前的网络支持

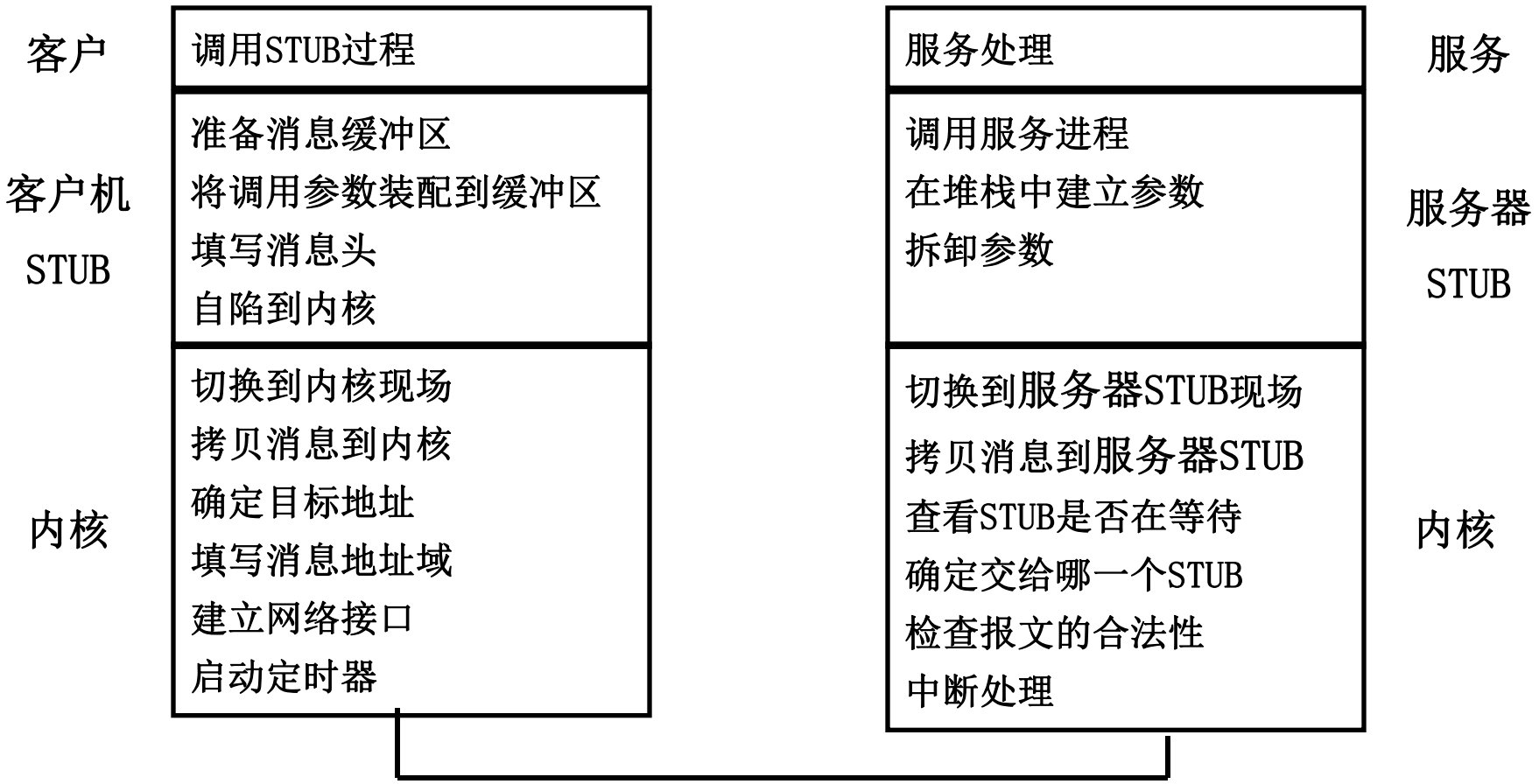
B 特定的RPC协议

报文和消息的长度,尽量的使用大数据量

2．RPC的关键路径

每个RPC执行的指令序列称为关键路径，其一般是：客户机stub，客户机stub自陷进内核，内核发送消息；服务器被中断，内核将消息交服务器stub，服务器stub将消息交给被调用进程，然后被调用的服务器进程执行调用并将按相反的路径发送应答消息。

注意:两个RPC操作在时间花费上的不同




客户机到服务器的关键路径

# RPC实例：SUNRPC

在密码学中凯撒密码作为一种最为古老的对称加密体制，它是一种代换密码。据说恺撒是率先使用加密函的古代将领之一，因此这种加密方法被称为恺撒密码。最简单且最广为人知的加密技术。

基本思想是：通过把字母移动一定的位数来实现加密和解密。例如，如果密匙是把明文字母的位数向后移动三位，那么明文字母B就变成了密文的E，依次类推，X将变成A，Y变成B，Z变成C，由此可见，位数就是凯撒密码加密和解密的密钥。

```
/* 凯撒服务接口(XDR)定义，文件名：caesar.x */

const MAX = 100;
typedef struct {/* 返回值类型 */
          int len;
          char code[MAX];
} Data;
typedef struct {/* 参数类型 */
          int key;
          char cipher[MAX];
} Args;
program CAESAR {/* 凯撒程序 */
          version VERSION {
                    Data DECRYPT(Args) = 1; /* 解密过程 */
                    Data ENCRYPT(Args) = 2; /* 加密过程 */
          } = 5;
} = 8888;
```

启动 XDR 编译程序 *rpcgen*，产生下述文件:

- 客户存根子程序
- 服务器主程序及服务器存根子程序
- 客户/服务器所需的参数集结和参数还原过程
- 程序头文件，caesar.h。该文件包含程序的常数、类型、以及远程过程所匹配的C函数原型(function prototype)

客户及服务器程序：

```
/* 客户程序，文件名：client.c */
#include <rpc/rpc.h>
#include “caesar.h”

main(){
        CLIENT *cp;
        char *serverName = “Caesar_server”;
        Args arg;
        Data * plaintext;
        /* 创建客户指针 */
        cp = clnt_create(serverName, CRESAR, VERSION, “udp”);
        if (cp == NULL) exit(1);
        arg.key = 1;  /* 构造调用参数 */
        arg.cipher = “Buubdl!bu!ebxo”;
        plaintext = decrypt_2(&arg, cp);     /* 远程过程调用 */
        /* 其它处理 */
        …
        clnt_destroy(cp);/* 删除客户指针 */
}
```

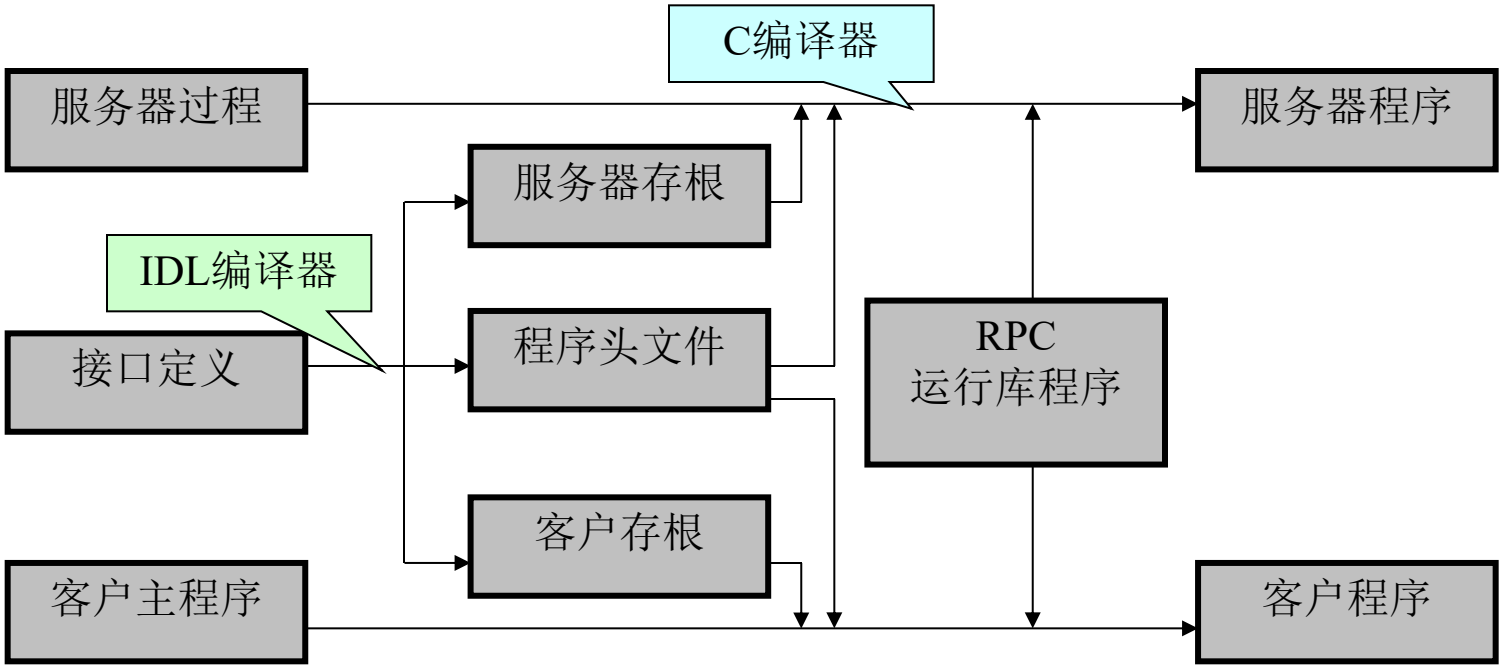
```
/* 服务器程序，文件名：  server.c */

#include <rpc/rpc.h>
#include  “ceasar.h”

Data* decrypt_2(Args *a){/* 解密函数 */
          static Data output;/* 必须静态变量 */
          char s = a->cipher;/* 密码指针 */
          int i = 0;
          while( *s) { output.code[i] = *s - key; i++; s++;}
          output.len = i;
          return &output;/* 返回结果 */
}

Data* encrypt_2(args *a){/* 加密函数 */
           /* 省略 */
}
```

# SUN RPC 流程:


C编译器
服务器过程
服务器程序
服务器存根
IDL编译器
接口定义
程序头文件
RPC
运行库程序
客户存根
客户主程序
客户程序

# 2．5 组 通 信

组通信：在这种机制下一个消息可以一次被送到多个接收者。而RPC只涉及客户机与服务器点到点。

1. 引言

组是多个进程的集合，这些进程可共同工作或以专门设计的方式工作。主要介绍操作系统(进程)组。组通信的实现主要依赖硬件。

例如:多路广播,全广播,单一地址投递

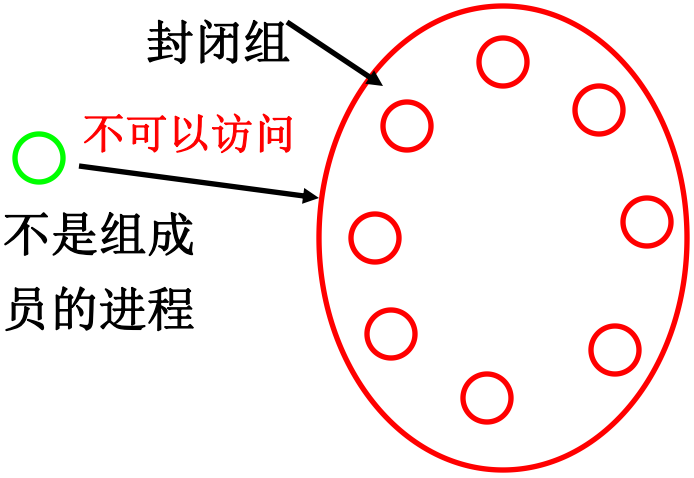
## 2. 设计要点

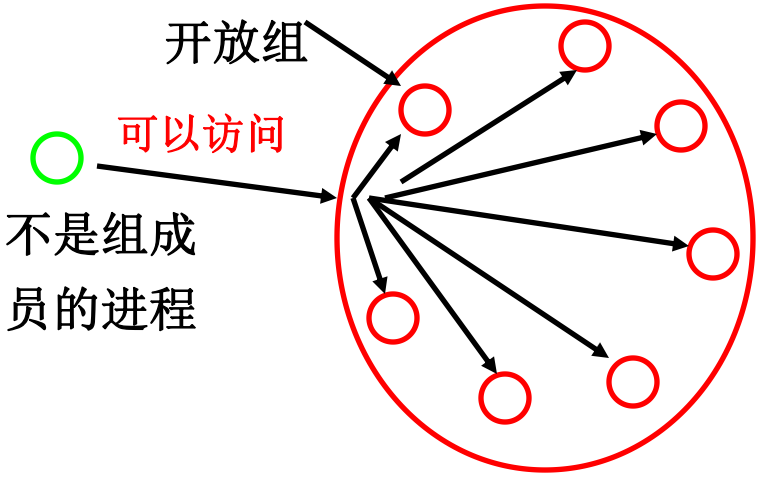
①封闭组与开放组

封闭组:只有组内成员才能发送消息到本组;

开放组:无此限制.

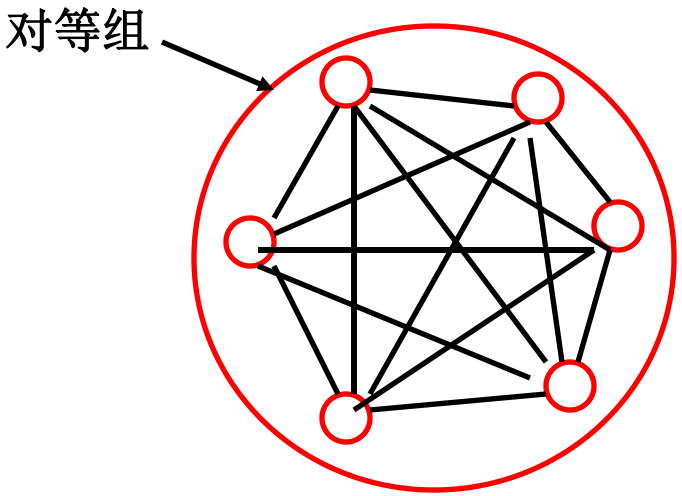
封闭组用于并行处理,或用来支持下多重服务器

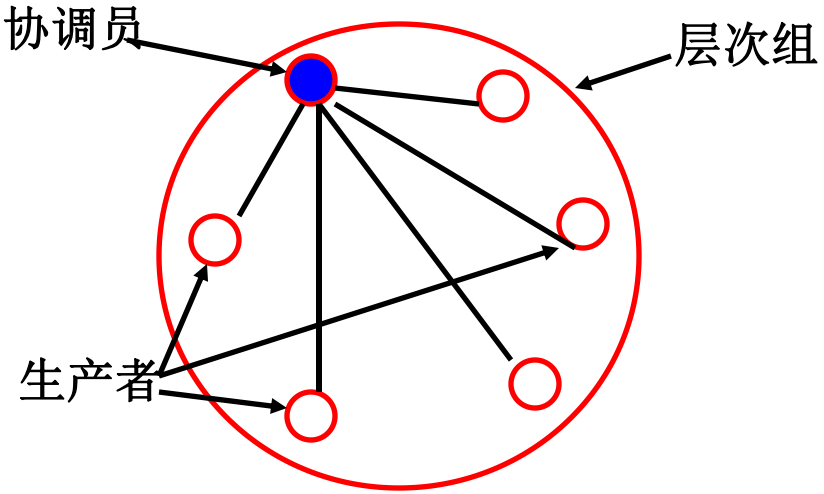


②对等组与层次组

对等组内的进程是对称的，不存在单点失效的问题。某个进程失败，只是组规模变小，组仍可工作，但决策过程复杂；层次组正好相反



③组的成员

A 管理的方法是使用组服务器

B 分布式机制管理

④组寻址

方法:

A 给每个组一个唯一地址;

B 发送者保持一份所有目的站点地址清单

⑤发送和接收原语

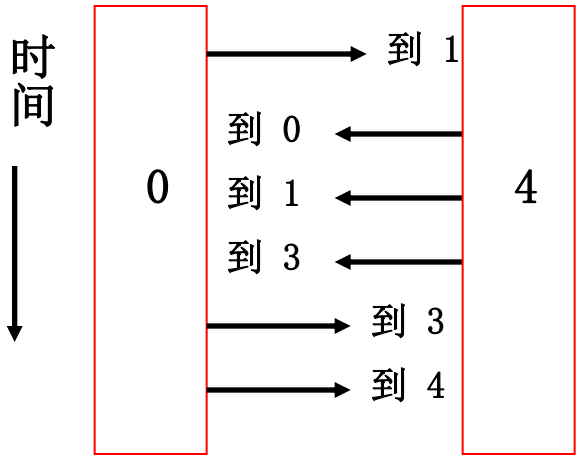
理想的方法是点到点通信和组通信合并具有单一的原语集合。

⑥原子性

组通信的一个特征是全要或不要(A11—or—Nothing)，这种A11-or-Nothing投递特性叫原子性或称原子广播。

⑦消息排序

它是组通信的第二个特性,采用全局时间排序保证传递的正确性.即按被传送的顺序投递到目的地。绝对的时间顺序并不容易实现.



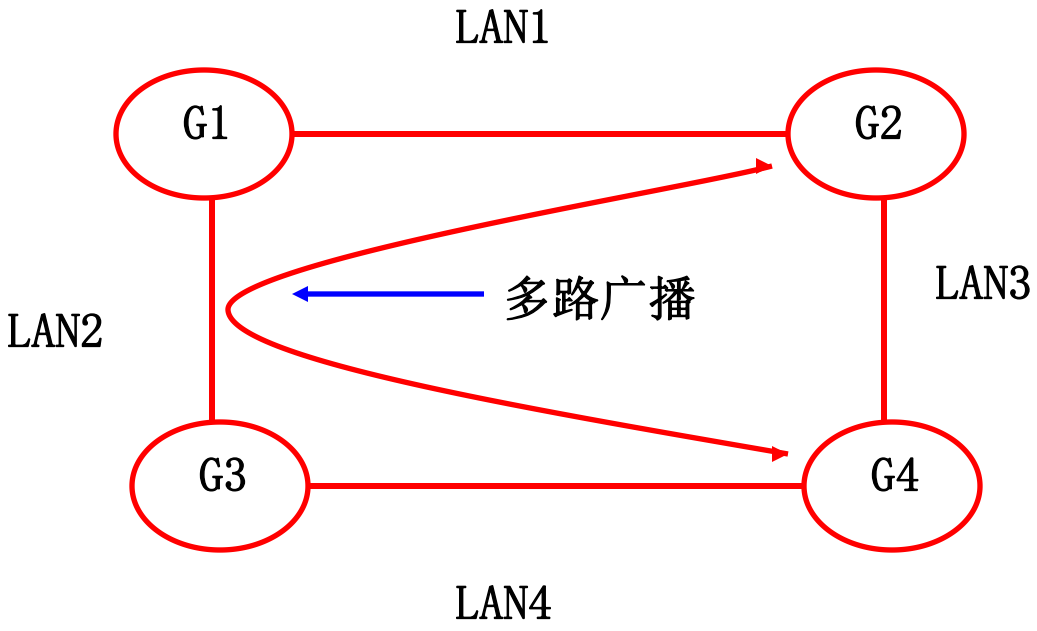

进程0和4发送的六个消息在时间上的交叉

⑧组的重叠

问题的解决相对困难需要组间协调，故一般避免组重叠。

⑨伸缩性　　组的规模的改变时算法的工作情况

注意:报文风暴




报文风暴图示

3．实例：ISIS（组合软件调用系统）中的组通信

ISIS本身并不是一个完整的操作系统，而是一组能在UNIX或其它操作系统之上运行的一个程序集合。

其关键思想是同步，采用不同形式的原子广播实现

最先用于华尔街的股票交易系统

4．ISIS中的通信原语

(略)

# 第三章 分布式系统的同步和进程

南京理工大学计算机学院

# 3．1 时钟同步

分布式算法的主要特征：

①相关信息分布在多台机器上

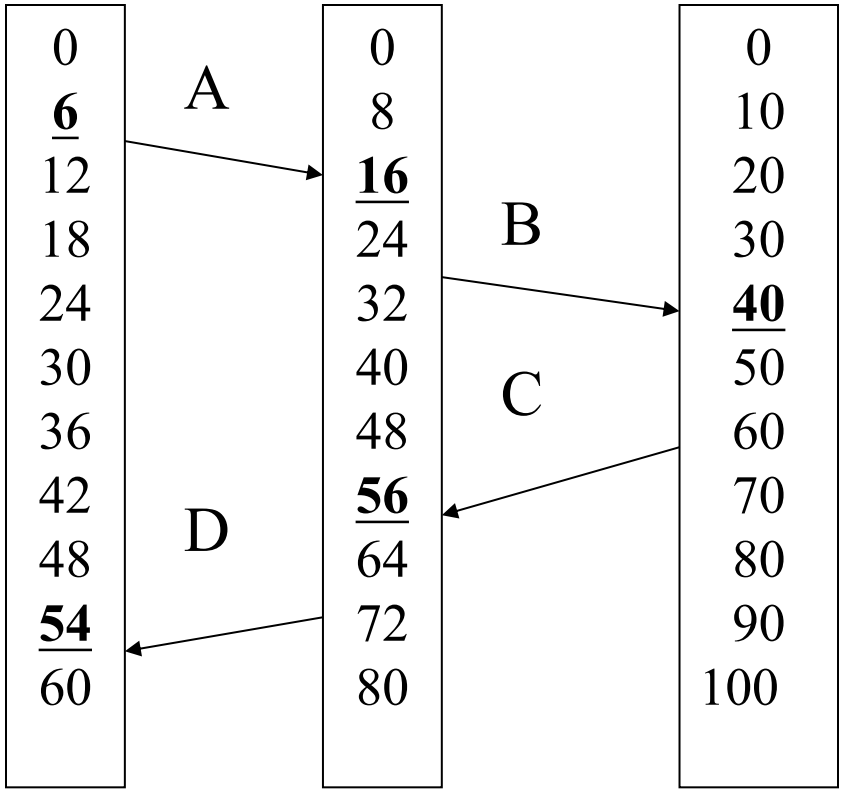
②进程仅依据局部的信息作出决定

③一台机器的故障不应引起整个系统的失败

④没有共用的全局时钟。

## 1．逻辑时钟

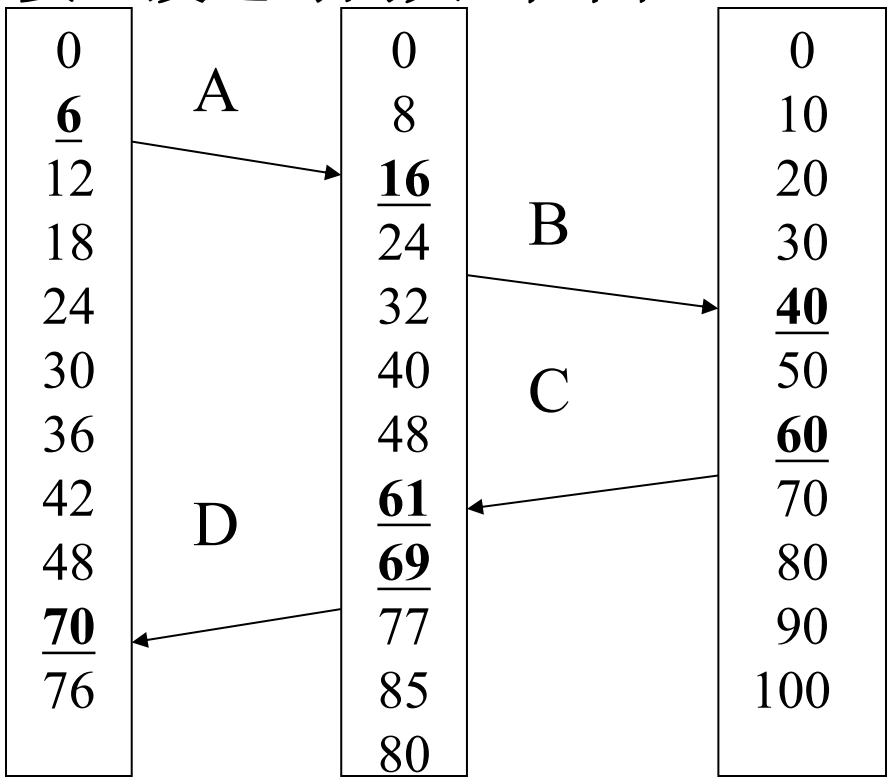
先看一个例子：




运行的结果是消息C在时间60上被发送，却在时间点54上到达

三个有自己时钟的进程,时钟不同

Lamport的算法以”先于”关系为基础，每个消息都携带它的发送时间,当它到达目的地时，如果目的地的时间早于它的发送时间，那么就把目的地的时间向前拔,至少要比发送时间大1个单位.




用Lamport的算法纠正时钟

该算法解决了全局时钟问题，它的条件就是两个相关事件之间必须至少相差一个时间

2．时钟同步算法

逻辑时钟只给出事物的相对时间,与真实时间并不对应,故要引入外部物理时钟,常用的时钟同步算法:

①克里司帝安(CRISTIAN)算法

具有国家标致时间接收器的机器,注意:调整的方法,通过调节单位时间内的中断的时间,来逐步完成时钟的调整.

② 伯克利算法

计算平均时间, 它是一个集中式算法, 不能在大规模的分布式系统中使用

原理:定期轮询每台机器的时间, 由结果计算平均时间, 各机器依此调整时间.

3. 同步时钟的具体使用

① 至多一次消息传送

消息的时间戳比存储的时间戳的值早, 就拒绝接受该消息

② 基于时钟的缓冲存储器（CACHE）的一致性

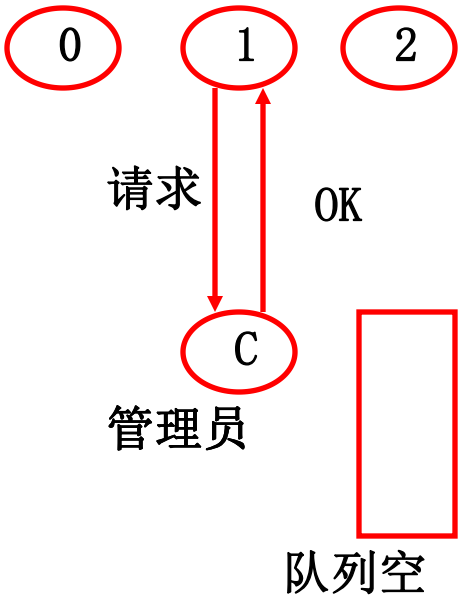
使用CACHE会出现一致性问题，原解决的方法:区分读写缓存,现在可用同步时钟来解决。即通过提供有效期（利用有效的租约）来控制CACHE一致性问题。

# 3.2 互斥操作

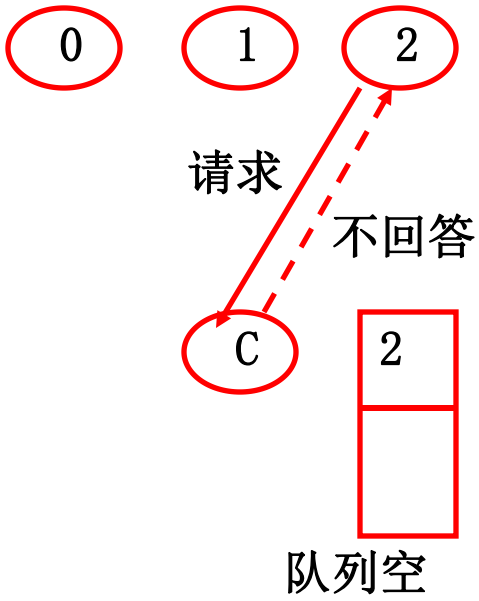
有多个进程的系统经常会碰到临界区的操作。当一个进程要访问某个共享数据时，它要先进人临界区进行互斥操作，确保没有别的进程同时访问该数据。在单机系统中，临界区可以用信号灯、管程来实现。那么在分布式系统中，由于不能共享主存，怎么实现临界区和互斥操作呢?下面我们讨论几种算法。

① 集中式算法

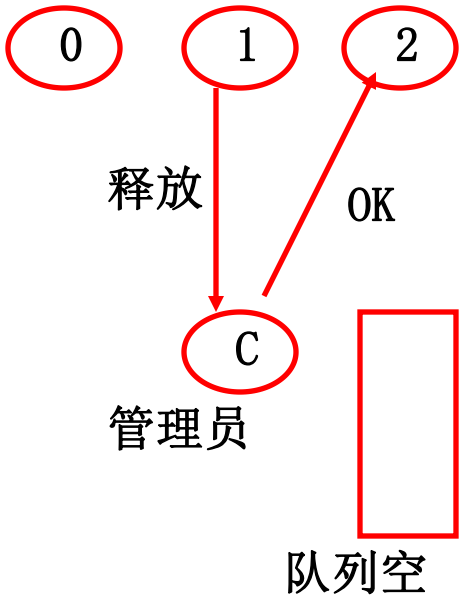
该方法就是模拟单机系统,指定一个管理员进行许可管理,该算法保证了互斥,每个临界区只需三条消息(申请,许可,释放)




(A)进程1向管理员请求进入临界区,得到许可




(B)进程2向管理员请求进入临界区,管理员不回答




(C)进程2向管理员请求进入临界区,管理员许可

缺点:集中式算法管理员是系统的瓶颈

## ② 分布式算法

算法的条件:系统中的事件必须有全局的顺序

算法的工作过程如下:

当一个进程要进入临界区时，它构造一条包括临界区名、进程号和当前时间的请求消息，然后把该消息广播给其他的所有进程。这里假设，消息的发送是可靠的。

当一个进程收到请求后，根据它的状态采取相应的动作：

(1)当接收者不在临界区，并且也不想进入临界区，就应答发送者OK消息。

(2)如果接收者已经在临界区中，它不回答．仅仅把请求加入队列。

(3)如果接收者不在临界区，但它也想进入临界区，就要将收到消息的时间戳和它广播消息的时间戳比较.如果到来的消息时间戳早，接收者回答发送者OK消息；反之接收者把到来的请求加入队列，不回答.

在发完进入临界区请求后，进程将等待所有的允许消息，一旦得到许可，就可进入临界区，当退出时向队列中的所有进程发OK消息，并将它从队列中删除。

所有进程都要参与决定是否进入临界区，若有进程不能做，算法将出错。

由于所有进程都要参加算法，所以比集中式算法慢，复杂，开销大。

改进算法：不需所有进程同意，部分回答OK即可

③令牌环算法

进程只有拥有令牌时,才能进入临界区,一个进程从相邻的进程收到令牌时先检查自己是非要进入临界区;如果要,就进入,否则就将令牌传递给下一个进程.

缺点:令牌可能丢失且检测困难，一个要进入临界区的进程最差情况下要等待其他进程进入和退出临界区所用时间总和

## ④三种算法的特点比较

集中式算法简单、高效，每次进入和离开临界区只需3次消息传递：请求、许可；释放，分布式算法中，n个进程需要(n-1)个请求和(n-1)个许可，总共要2(n—1)个消息。在令牌环算法中，所需的消息数是不固定的。如果每个进程都要进入临界区，那么每个令牌都有一次进人和退出，平均每次进入有—个消息传递；如果令牌被一个进程占有很长时间，那么进入临界区需要的消息就不确定。

进程从它发出进入临界区的请求到它进入临界区的时间延迟在三个算法中是不同的,当临界区很短并且使用频率很低时，进入临界区时间延迟的决定因素是进人临界区的控制机制．当临界区很长并且使用的频率很高时，决定因素在于等待时间，消息数就能说明这一点。

这三种算法出现故障时,都会有很大影响,要避免系统的崩溃,须注意:在容错系统中,次三种算法都不适用.

# 3.3选举算法

在分布式系统中，大多数算法要求有一个进程充当管理员或初始启动者等特殊角色。前面几个算法就有这样的例子。一般来说，由哪个进程充当特定角色无关紧要，但是必须有一个进程做这个工作。下面我们来看如何选一个进程担当特定角色。

选举算法的目的是当选举完成后，能够让所有的进程知道谁是管理员.

## 1. 霸道算法

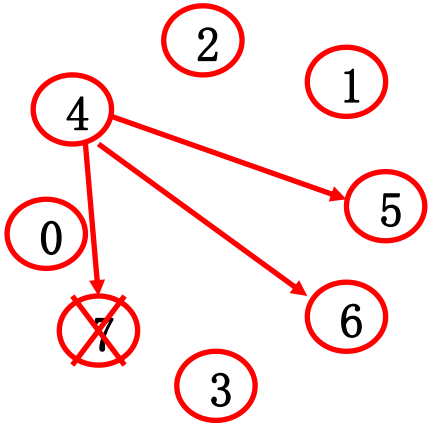
该算法提出。当一个进程P注意到管理员不再对请求作出反应时，它就开始选举．进程P执行下列步骤：

(1)向所有进程号比它大的进程发送ELECTION消息；

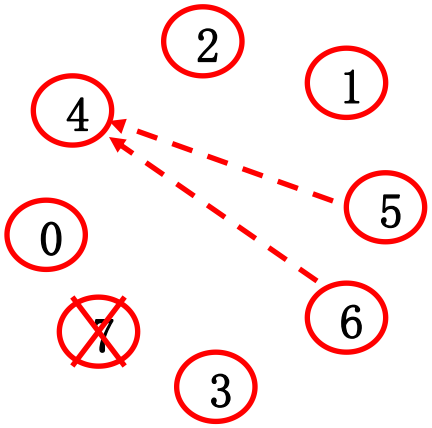
(2)如果没有进程响应，进程P成为管理员；

(3)如果有进程响应，该响应进程成为管理员，P结束选举。

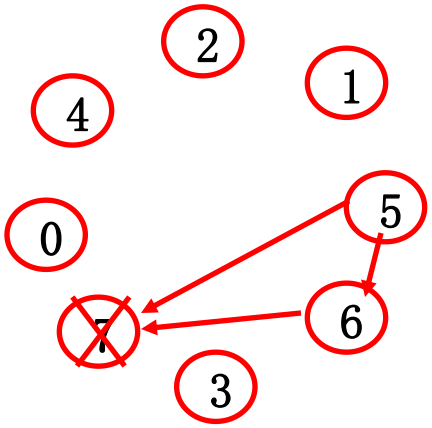
注意：一个进程只能从号码比它小的进程处得到一个选择消息



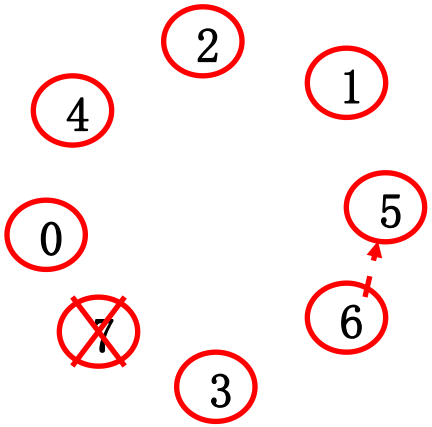

(A)进程4启动选举



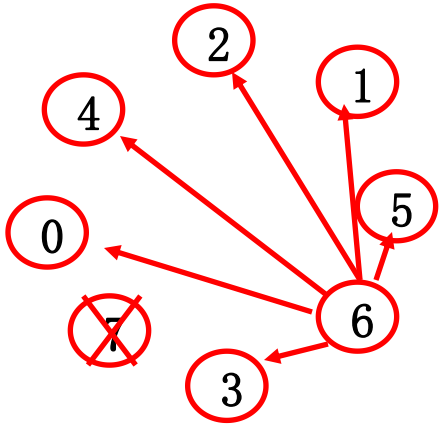

(B)进程5,6响应,让4停止




(C)进程5,6,都启动选举




(D)进程6让进程5停止选举

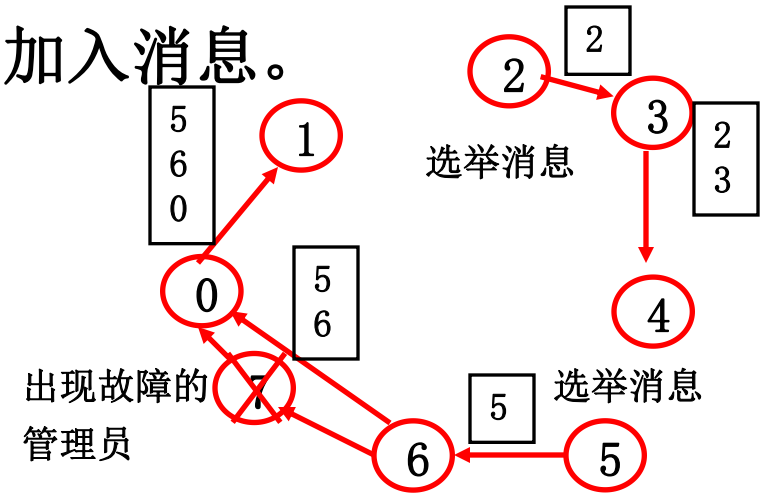



(E)进程6成为管理员,发通知

霸道算法图示

## 2. 环形算法

这个算法使用环，但不是令牌环。这里假定所有的进程都是有序号的，即每个进程都知道它的后继进程。当一个进程发现管理员不能工作时，它把包含其进程号的ELECTION消息发给它的后继进程；接收消息的进程再向后继进程发送ELECTION消息。如果接收进程没有反应，发送消息的进程便向下一个进程发消息。每一次发送ELECTION，进程都要将自己的进程号加入消息。




使用环形选举算法

最后，第一个发出该选举（ELECTION）消息进程收到该消息，再将其转换为协调（COORDINATOR）消息后，再循环一周，通知谁是管理员，谁是组成员，这时消息包中进程号最高的进程将成为管理员。当这个消息循环一周后，由发送进程把它从网上清除。

图中2和5都发现7失效，分别建立选举消息，两条消息都沿环运行，结果是一样的，只是浪费了带宽。

# 3．4 线 程

进程因等待而挂起，使进程中可并行部分不能执行，从而使系统性能下降，故引入---线程.

1. 线程：就是可以共享地址空间的轻型进程，它有自己的程序寄记数器栈和寄存器集合。它与进程的主要区别是它的地址空间是共享的。

线程相对于进程，犹如进程相对于机器

2. 线程的使用，将并行性引入到顺序执行的系统.

多线程组织的常用模型：

1)分配器/工作者模型

有一个分配器线程唤醒工作者线程可用信号灯

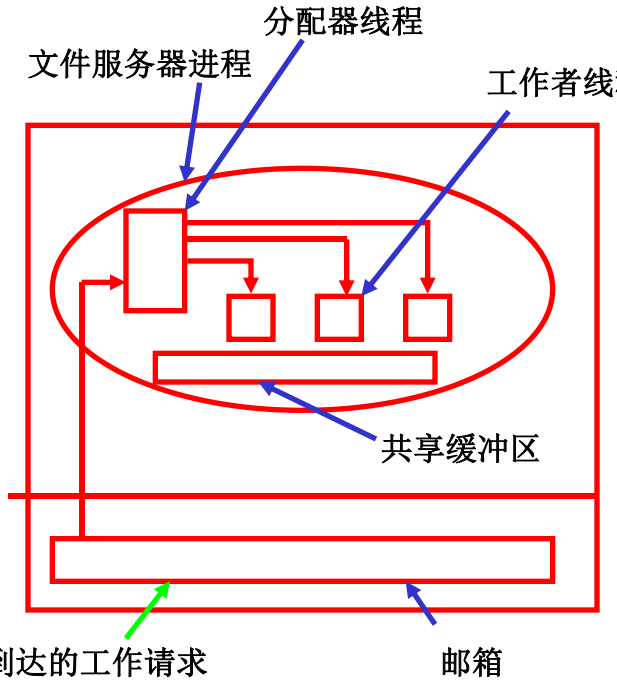
2)团队模型

地位平等 线程各自获取和处理自己的请求.

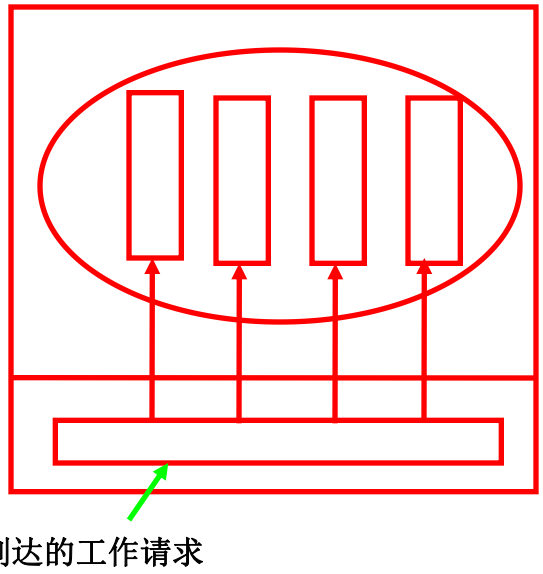
3)流水线模型

管道线模型，第一个线程产生一些数据传给下一个线程处理，且持续下去。

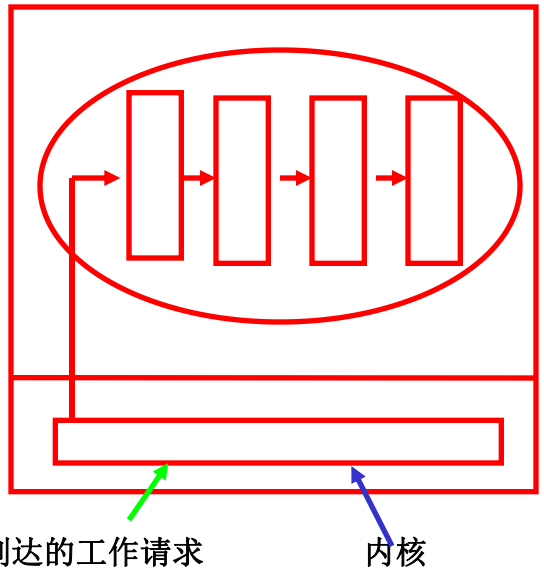
多线程可分时工作在单CPU上也可工作在多处理器系统上,而且多线程系统设计的好将可与多处理机工作性能相当.




分配器/工作者模型




团体模型




流水线模型

进程中线程三种组织方式

## 3. 线程包的设计的相关问题

线程包就是供用户或程序员调用的关于线程的一组原语。

线程的管理方法有静态和动态方法两种。

静态即开始就确定好线程的个数，动态个数不定

线程的代码与进程一样，由多个过程组成。

线程包中临界区控制利用互斥体（mutex),其总处于：

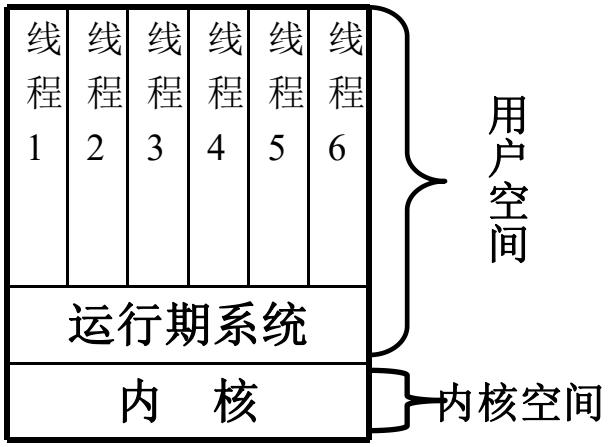
打开和锁住两种状态

```
lock mutex;
    check data structures
    while(resource busy)
        wait(condition varable);
    mark resource as busy;
unlock mutex;
```

```
lock mutex
    mark resource as free;
unlock mutex;
wakeuo (condition variable);
```

互斥变量与条件变量的使用

线程可由算法进行调度，如优先级调度、循环调度等




4. 线程包的实现

1) 在用户空间实现线程（如图）

这种方法是将线程包完全放在用户空间，内核对此一无所知，在这种方法中，内核只是管理普通的单线程进程。这种方法最明显的优点是它可以在一个不支持多线程的操作系统上实现用户线程包。同时它还允许每个进程有自己特定的调度算法.

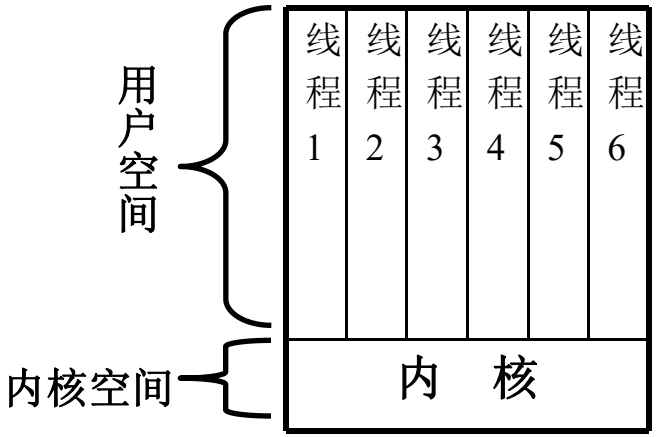
缺点是：

数量太多会引起资源紧张. 同时

（1）它也难实现阻塞系统的调用.

（2）它的调度是非抢先式的. 进程内部无时钟中断，无法进行时间片的调度.

2）在操作系统内核实现（如图）




不需要运行系统，线程创建或撤销，只需一次系统调用，比在用户空间实现线程开销大. 可通过在撤销时加标记，当做为新线程创建时仅需激活即可。

3)调度员活动方法

结合前两种的优点(用户线程的高性能和内核线程的实现简单)

原理：当使用调度员活动方法时，内核给每个进分配一些“虚处理器”，并由运行系统分给线程，同时通过避免在用户空间和内核空间之间的不必要的转换来实现高效率。如：线程在局部信号上阻塞并不去涉及内核，而是由用户运行期系统去完成阻塞和调度。具体是由内核激活用户运行期系统

4)线程和RPC

希望使一个进程中的线程可以有效的调用同一台机器上的另一个进程中的线程，具体采用与RPC 相似的过程完成。

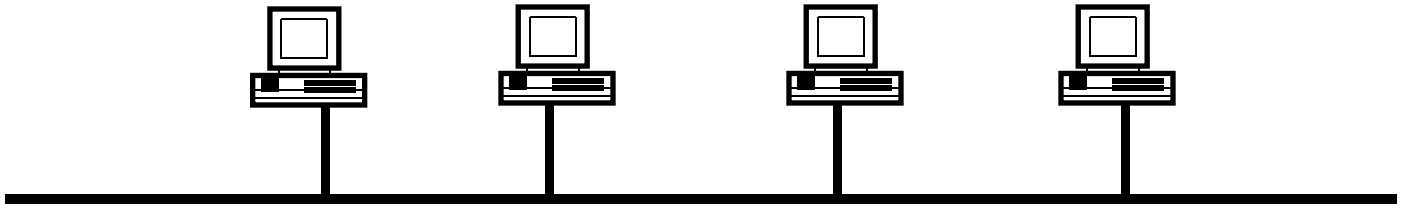
加速RPC执行的技术，具体即为当一个线程执行请求时，它将消失并且它的堆栈和环境信息也丢弃，当有新的消息进入时，内核动态创建一个新线程去为其服务。优点首先线程不用等待所以不保留环境，其次创建新线程比存储一个存在的线程花费少，节约了时间减少了开销，从而提高了速度。

# 3．5 分布式系统模型

## 1．工作站模型

其主要指：一组工作站通过高速局域网互连，在某一时刻,每台机器只有一个用户登录，即拥有者，要么空闲。

优点：清晰、易于理解，计算能力固定、系统反应时间固定、有一定的自主性、同时增强了独立性。缺点:资源存在浪费，资源利用率不高.



2. 空闲工作站的使用

1）找出空闲工作站

两种定位方式：服务器驱动和客户端驱动

2）透明运行远程进程

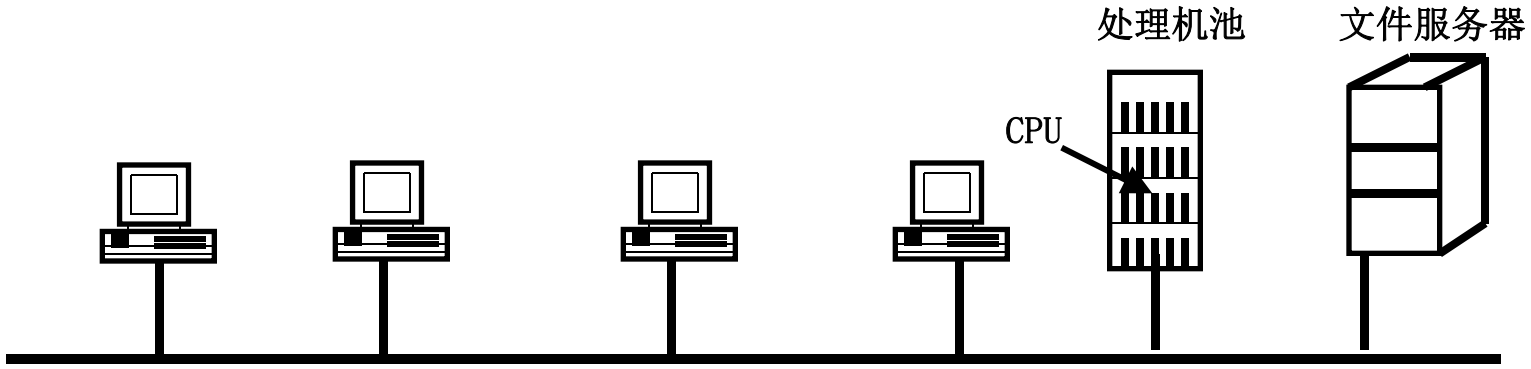
由内核完成

3）机主人重新使用时如何处理

直接杀死进程、移植到另一台机器上（进程全部包括子进程）

### 3. 处理机池模型

处理机池是无盘工作站的进一步发展。理论依据是排队论,具体有理发师模型、超市收款等。CPU根据需求动态的分配给用户。模型如图




虽然有证明：用一个是小系统功能N倍的大系统来代替N个独立的小系统的资源可把平均响应时间减少为原来的1/N倍。

然而平均响应时间不代表一切，从费用角度来看，不惜代价建造巨型机甚至是不可能的。

同时也并不是有N个处理器的处理机池就可比单个处理器的性能高N倍，而是决定于任务可分几部分。

4. 混合模型

用那种模型要依情况而定，若是偶发的事件为主，则个人工作站模型机可，若是仿真或工程计算则用处理机池较好，也可两者结合.

# 3．6 处理机分配与调度

如何决定进程运行在那台机器上，即为处理机的分配。

1．分配模型

1）分配策略有两大类:

非迁移的（进程创建后，一直呆到进程结束）

迁移的（进程可在运行中迁移到其它机器上运行）

2）分配算法的目标：

(1)最大限度地提高CPU利用率

(2)尽可能缩短平均响应时间

(3)最小化响应率

例如：

有两个处理机1及2 和 两个进程A及B

| | 处理机1<br>10MIPS | 处理机2<br>100MIPS<br>5秒队列 |
|---|---|---|
| A（1亿条指令） | 10秒 | 6秒 |
| B（3亿条指令） | 30秒 | 8秒 |

分配1： 处理机1运行进程A，处理机2运行进程B，平均响应时间为（10+8）/2=9秒 （较优）

分配2： 处理机2运行进程A，处理机1运行进程B，平均响应时间为（30+6）/2=18秒

2．设计分配算法的主要相关问题

1）确定式还是启发式

确定式适用进程是否可预知;启发是指以经验性规则和信息指导.

2）集中式还是分布式

依全局信息做判定,以局部信息做判断.

3）最优的还是次优的

4）局部的还是全局的

解决转移策略

5）发送者初启还是接收者初启

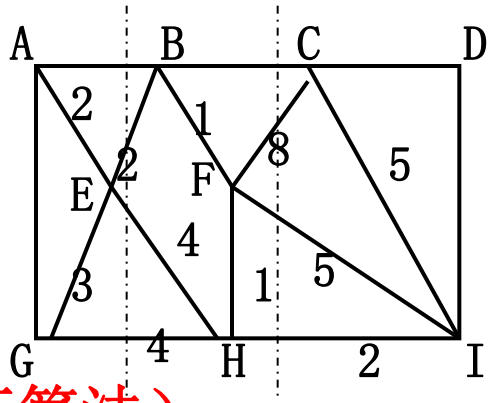
解决定位策略

3．处理机分配算法实现中的问题

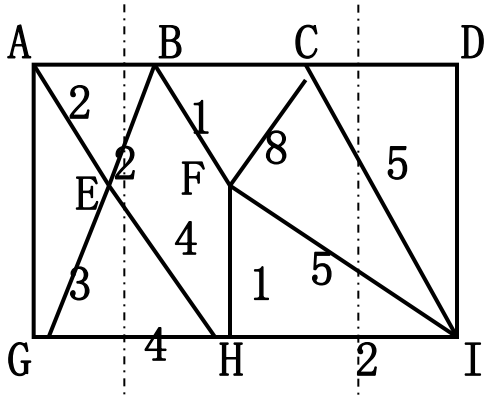
1）对负载的测量　2）额外开销的处理

3）问题的复杂度　4）稳定性问题

4．典型的处理机分配算法




30单位通信量




28单位通信量

1)图论决策算法

2)集中式算法（上—下算法）

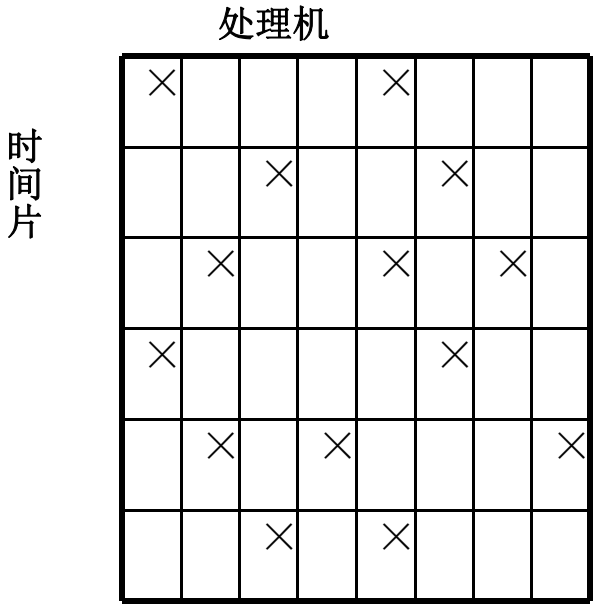
3)层次算法　逻辑上构成分层结构

4)投标算法　把系统中的资源和任务作为买方和卖方以寻求平衡.

## 5. 调度

假设进程成组创建,组间通信比组内通信要少,基于协同调度概念的算法.

算法使用一个矩阵




列是处理机的进程表

行是时间段的进程表

主要思想:每个处理机采用时间片轮转法,因此可将进程组的所有成员放在不同处理机上的同一时间段

实现文件服务的技术对于分布式系统的设计非常重要。

在分布式系统中，必须提供与传统文件系统相当的性能和可靠性。实现分布式文件服务器的技术源自传统系统，但由于要将许多设施分布在多个不同的服务器中，系统结构有所不同，因此，其技术要求及高于传统系统。

| | 共享 | 持久性 | 分布式缓存/副本 | 维护一致性 | 例子 |
|---|---|---|---|---|---|
| 主存 | × | × | × | 1 | RAM |
| 文件系统 | × | √ | × | 1 | UNIX文件系统 |
| 分布式文件系统 | √ | √ | √ | √ | SUN NFS |
| WEB | √ | √ | √ | × | WEB服务器 |
| 分布式共享内存 | √ | × | √ | √ | IVY |
| 远程对象 | √ | × | × | 1 | CORBA |
| 持久对象存储 | √ | √ | × | 1 | CORBA持久对象存储 |
| 持久分布式对象存储 | √ | √ | √ | √ | PERDIS |

# 第四章 分布式文件服务

# 4．1　引 言

文件被认为是一个基于磁盘或其它二级存储器的存储结构的抽象。最简单情况下，文件定义为数据项的序列，文件系统提供按指定位置读／写文件数据项或子序列的操作。有些文件系统将文件定义成复杂数据结构的集合(如记录等)，可以用记录的键字来定位。

在集中式文件系统中，文件系统可以管理大量的文件，包括读、写或删除等。而目录是一种抽象的文件管理设施，它能将文件名映射到一个文件指针，这种映射(即目录)一般以特殊类型的文件存储，若要访问文件，需要首先打开目录。

文件系统要负责文件共享和访问控制的管理，限定用户的访问权限，为每个文件定义访问类型等。

文件系统一般提供以下服务：

文件组织　　文件存储

文件检索　　文件命名

文件共享　　文件保护。

设计文件系统的目的，是使程序员可以使用抽象操作来访问磁盘空间，传统文件系统的层次结构如下图所示：

| |
|---|
| 目录模块:负责文件名到文件标识的映射 |
| 文件模块:负责文件标识到文件的映射 |
| 访问控制:操作的许可性检查 |
| 文件访问:文件数据、属性的读写 |
| 盘块模块:磁盘块的分配及访问 |
| 设备模块:磁盘I/O缓冲 |

文件系统模块图

以上也同样是分布式文件系统的基础.

另外加上：

1. 支持共享信息的管理

不必先拷贝，再访问，永久性数据管理

2. 为用户提供透明的文件访问服务

3. 提供目录服务

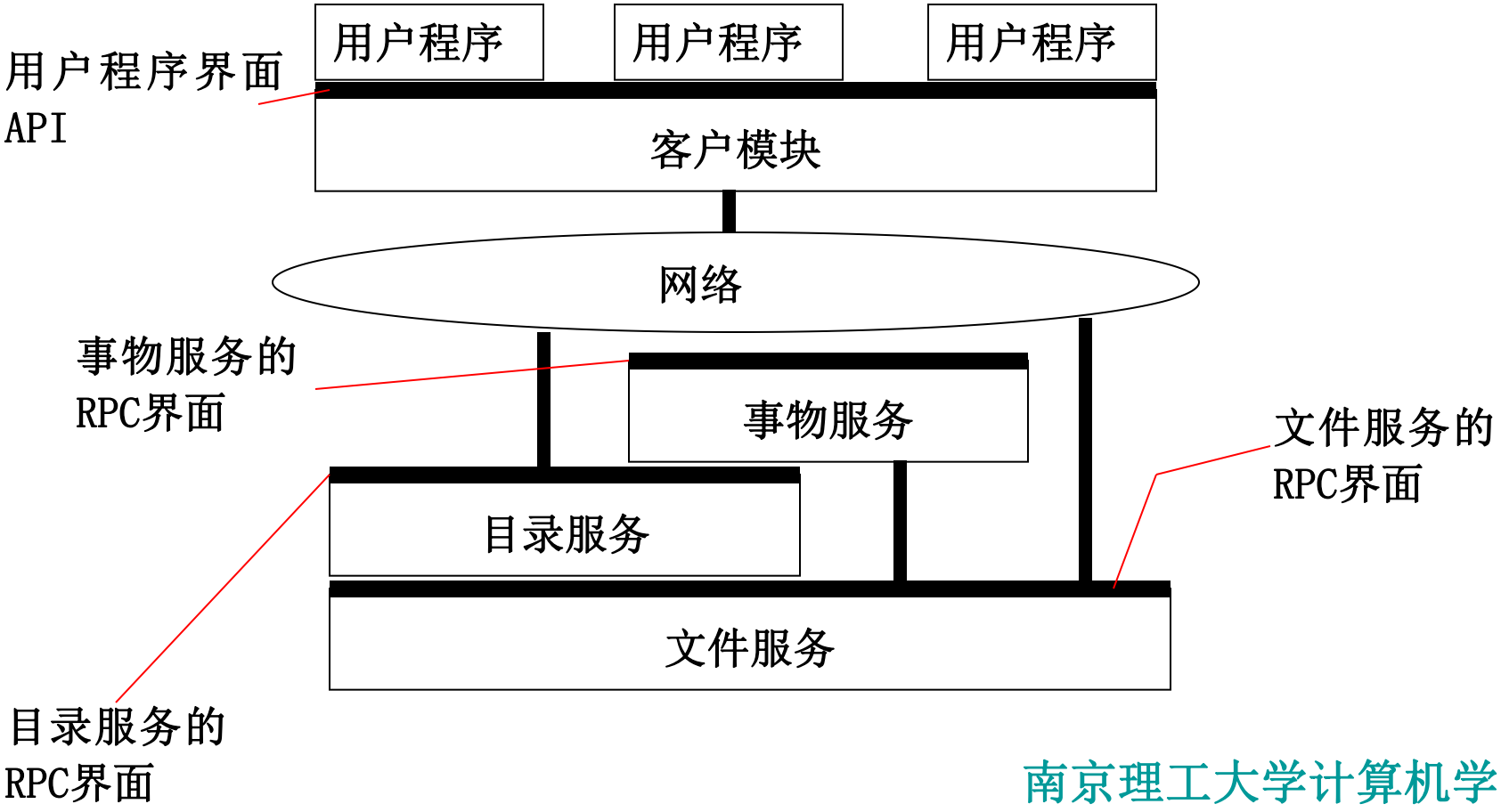
提供文件命名及文件名到文件标识的映射功能

4. 事务服务

为用户提供的最高层服务，提供并发服务，保证文件的一致性

综合而言,分布式文件可分为三种服务:

目录服务,文件服务,事务服务



# 4.2 文件服务

在UNIX和DOS系统中，文件是没有任何解释的字节序列，文件的结构和意义完全由应用程序决定，操作系统不关心其结构和意义。但在某些大型机的操作系统中，有许多不同类型的文件，每种文件具有不同的特性。

文件可以是一组记录构成的序列，该记录又可以是复杂的结构，这个结构的意义是由操作系统解释的。这样，文件便由数据和属性构成，其中，属性包括文件长度、访问时间、访问控制表和拥有者，文件长度和数据可为用户使用，其它属性均为目录服务使用，可以将这种文件看作一种文件对象。

关于文件的另一个重要问题是文件的可修改性。如果文件创建之后，不能再修改，则称为不可修改(Immutable)文件，否则称为可修改(Mutable)文件。这里主要讨论可修改的文件。

## 4.2.1 文件服务的模型和任务

对于一般的文件服务而言，有两种服务模型

1) 装载 / 卸载(Up Load / Down Load)模型

2) 远程访问(Remote Access)模型

在装载／卸载模型中，文件服务仅提供读文件和写文件两种操作。

读操作用于将整个文件从服务器中完全读出，写操作用于将整个文件从客户机写入服务器。在这种模型下，可以将文件缓冲于客户机的主存或磁盘中，其优点是概念简单，应用程序可以局部拥有并访问文件，程序结束后，必须将新建或修改过的文件写回服务器。在此，不需要复杂的文件服务界面，整个文件的传输是高效的。但是，客户机必须有足够的空间来存储所需要的文件，另外，如果实际上应用程序作用于文件的一小部分，就会造成很大的浪费。

在远程访问模型中，文件服务将提供大量的文件操作，包括打开、关闭 读和写等，其优点是不需要客户机具有大量空间。另外，当应用程序只需处理文件中少量数据时，不需要将整个文件全部拷贝到客户机上去。

文件服务的任务

两个任务：文件存储和文件寻址。其中，文件寻址利用文件位置映射方法完成文件的定位，文件访问通过调用外存访问功能完成文件的存储，外存访问功能通常以块服务的形式提供。所以，文件服务的基本任务就是，提供通用和有效的操作集合，为多个客户提供大容量的外存设备。

另外，文件服务还要解决保护问题，即严格控制用户对于文件的访问，从而保证文件的一致性。文件服务可以同时支持多个目录服务，这样，目录服务可以采用多种不同语法，从而使文件服务可以用于不同的目录服务。一个文件服务可能同时管理几个目录服务支持的多个文件集合，每个集合中的文件属性记录和结构可能还有差异。

## 4.2.2 文件服务界面

所谓界面，主要提供服务的基本操作名和调用格式及简单功能描述。

文件的UFID；参数；　数据；出错信息

# 4.3 目录服务

目录服务定义了构成文件(或目录)名的某种字母表示语法。文件名可以是由一个或多个字符、数字及特殊字符构成的字符串。有些系统通过句点将文件名分成两部分，如prog．c表示C语言文件，maim．txt表示文本文牛，第二部分一般叫做文件扩展名。

分布式系统目录一般也都包含子目录，从而使用户可以给相关文件：分组，形成层次文件系统。因此，目录服务就包括创建、删除、进入、退出和查找文件等功能。有的系统还提供建立目录指针的功能，指针可以设置到任何目录中，这样构成的目录和文件就不再只是树型结构，而是图(网状)结构，从而使表示能力更强，当然，管理也就更复杂。

设计分布式文件服务的关键在于，是否支持所有机器对于目录层次具有同样的视图。

## 4.3.1 目录服务的任务

大多数分布式系统都使用两级命名方式，即每个文件(或其它对象)都有一个符号名(常称为文件名)和一个内部二进制名(用UFID表示)。符号名一般供程序员或用户使用，如prog.c；内部二进制名供系统使用。所谓目录，就是一组文件名到UFID的映射表。目录服务最基本的任务就是将文件名映射到UFID，通过这一映射可以取代传统文件系统中的打开(Open)文件操作，客户一旦拿到UFID，便可以操作文件.

目录服务本身要确保UFID正确送到请求客户。目录服务还要检查客户的合法性，通过访问控制表做合法性检查，最终决定是否发出一个UFID.

目录服务的功能也可以分解为基本功能和辅助功能两类，基本功能提供文件名到文件标识的映射，它是构造面向用户的目录服务的基础，包括名字、结构和访问控制方法。根据授权客户的请求允许用户执行UFID所指出的文件增、删操作。辅助功能包括文件名的词法分析、建立目录形成层次结构和其它复杂操作。因此，目录服务可以在给定目录中查找文件名，然后得到UFIO，每个UFID都指出对于文件的相关访问权限，如对文件拥有者可读、写和删除，对于其它用户只读。

目录通常以文件形式存储，因此，目录服务本身是文件服务的一个客户，每个目录本身有一个UFID。

## 4.3.2 目录服务界面

目录服务定义了一组目录服务操作。不同系统有响应的约定.目录服务的定义遵循约定记号。

## 4.3.3 文件属性与目录访问

目录服务要进行访问控制，可以通过访问和更新文件属性完成。如可以在新文件名加入目录时，在文件属性中设置访问许可；提供查看文件属性的操作；还可以提供允许文件拥有者认定或重设其它用户的许可权限、及将所有权转让给其它用户的操作。

文件系统除管理数据外，还要管理许多相关信息，如创建日期、最后访问日期和最后修改日期、文件类型、文件所属目录和访问控制信息等，我们称这些信息为文件属性。文件系统将文件属性作为一组不可分解的数据，文件属性可以执行读写访问，但没有长度或结构等操作

目录服务不但要确定属性中的内部结构和所存数据值，还要负责文件属性的访问，包括读取和修改。如改变访问日期、修改文件所属关系和管理访问控制表等。

目录存储着所有文件的拥有者，文件拥有者可以对其文件执行所有操作，如创建、读、写和删除等.

问题是让谁来执行目录服务中的LookUp、AddName、ReName和UnName操作，查看或修改文件属性?当用户具有各自独立、分离的目录时，目录的UFID可能认定为一种权能，不需要进一步的访问控制；但是要访问共享目录时，则需要访问控制。一个简单的解决办法是，让目录拥有者的访问许可模式与文件拥有者的访问许可模式取得一致。

另外，为用户同时提供文件服务和目录服务操作可以克服文件的丢失问题。对于文件创建，用户通过创建操作指定的文件名和目录名，然后依靠文件服务Create操作和目录服务AddName操作完成。对于文件删除，通过删除操作指定一个UFID和目录名.

## 4.3.4 树型结构

在UNIX提供的树型文件系统中，包含组织成树型结构的许多目录，通过链使文件具有多个名字，可以使用基本的文件和目录服务来实现。树型结构的目录集合可以通过叶、根和内部结点构成。树根是一个目录，其UFID是共用的。

树中任何结点或叶结点就是文件或目录，可以通过路径名命名文件或目录，路径是由多部分组成的表示路径的名字。根具有特殊名，并且每个文件或目录都有自己的名字。

在树型结构的目录服务中，文件属性应当包括一个类型属性用来区分文件和目录。这可用于沿树结构按路径名检索时，保证名字除最后一个名字外一定是一个目录名。当创建文件或目录时，文件属性中要设置适当的数值，包括访问控制表、创建日期等等。目录一旦创建了，可以使用AddName操作将它的名字和UFID插入到另一个目录中。

## 4.3.5 命名透明

透明命名与位置透明性相关，其意义是路径名并不表示文件的位置，如／serverl／dirl／dir2／x并不说明文件x一定处于server1上。服务器可以随意地迁移和切换，而路径名则保持不变。这种系统称为位置透明的系统。

但是，假设文件x非常之大，其空间的确处于serverl上，再假设在server2上也有大量的空间。系统有可能自动地将文件x迁移到server2上．但是，当路径名中第一部分是服务器的时候，系统是不能自动将文件迁移走的，即使diri和dir2在两个服务器上都有也不行，因为迁移文件将把路径名／serverl／dirl／dir2／x改变为／server2／dirl／dir2／x。

具有前一路径名的程序在路径名改变时必须停下来重新编译。如果文件可以迁移，其命名不必改变，则称其具有位置独立性。将机器名或服务器名嵌入到路径名中的分布式系统不是位置独立的，基于远程安装的系统也不是位置独立的，因为将一个文件从一个文件组(安装单位)迁移到另一个文件组后，不可能仍然使用原来的路径名。位置独立性不易实现，却是分布式系统追求的目标。

通常，在分布式系统中有三种普遍的命名文件和目录的方法：

①机器+路径命名法，如／machine／path或machine：path；

②在本地文件层次结构中安装远程文件系统；

③在所有机器上表现一致的统一命名空间。

前两种方法容易实现，特别是在使用没有特殊设计分布式服务的系统中非常有效。第三种方法非常难于实现，必须精心设计，但是，如果要求分布式系统的表现像集中式系统一样，则必须做到这一点。

# 4．4文件服务的实现

## 4.4.1 系统结构

要实现文件服务器，首要问题是客户机与服务器是异构的还是同构的?第二个问题是文件服务和目录服务的结构如何?当然，对这两个问题还没有定论的方法。

一．同构型／异构型

有些系统没有区分客户机和服务器。所有的机器都运行相同的软件，任何机器都需要提供文件服务。在有些系统中，文件服务器和目录服务器只是一些用户程序，所以，在同一机器上，可以将系统组织成运行客户机软件的系统或运行服务器软件的系统，也可以同时运行客户机软件和服务器软件。这些都是同构的客户机和服务器。

还有一种极端的情况，就是客户机和服务器从根本上(包括系统硬件和系统软件)就是不同的机器，服务器可以运行与客户机不同的操作系统或同一操作系统的不同版本。这就是异构的客户机和服务器。

## 二、文件服务和目录服务的结构

一种方法是将两部分结合处理，即将文件和目录服务统一处理；另一种方法是分开处理，即，当打开一个文件时要求先请求目录服务器将符号名映射为内部二进制名，然后再用二进制名访问文件服务器，完成文件的读写。

第二种方法的优点是两种功能互相独立、灵活性好。但由于要有两个服务器，因而会增加额外的开销。通常情况下，客户机向目录服务器发送一个符号名，服务器返回一个二进制名供在文件服务器上访问文件用。但目录层次可能在多个服务器上分开存储。

## 三、可重复操作与文件服务的状态

如果一个操作被执行多次，其效果与执行一次是一样的，这就是可重复操作。在分布式系统中，通信上的错误、计算机的故障和延迟都可能导致一次远程调用的多次执行。如果重复执行可能对服务器产生不良影响，就必须实现重复抑制机制来防止重复操作。如果操作是可重复的，则不需要进行重复抑制的开销。另一方面，操作可重复会引起性能的降低，例如，服务器的超载可能引起客户程序等待超时，从而引起不必要的重复调用。CFS件服务器支持可重复的操作，而XDFS则同时支持可重复操作和重复抑制机制，而UNIX文件访问操作是不可重复的，因为其文件的读写指针在每次操作中都要增值。

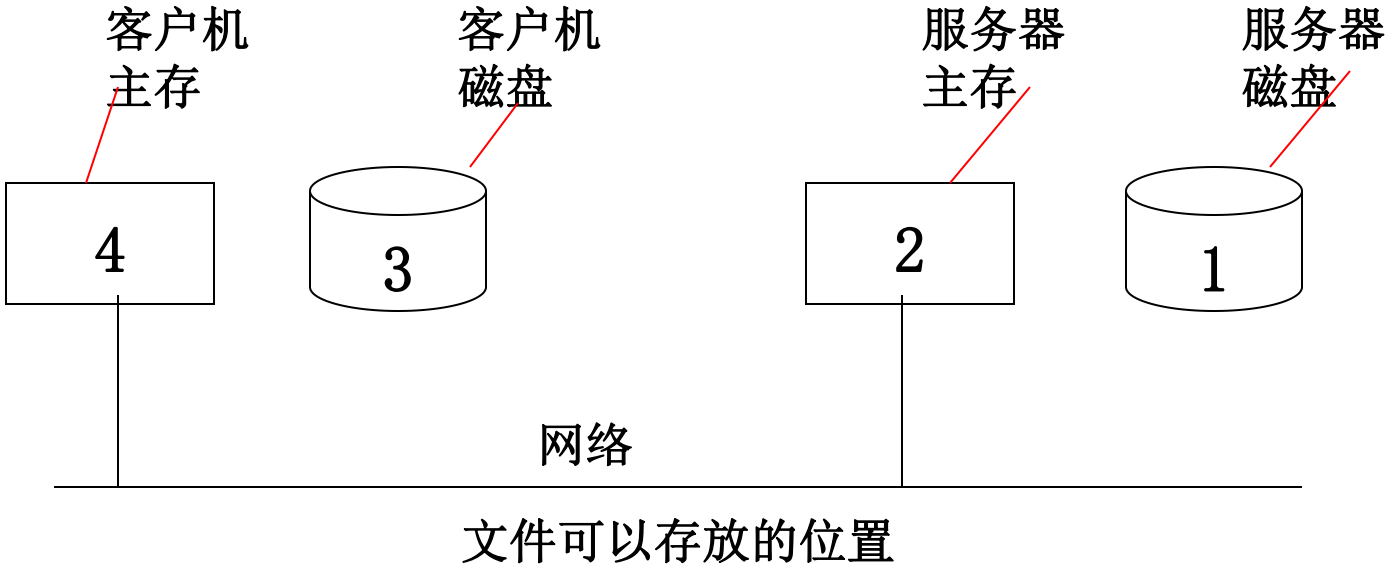
在传统的操作系统中，内核为每个进程打开的文件保持一个读写指针，但可能会产生两种情况：如果服务器故障后重新启动，所持有的状态信息将会全部丢失，而客户可能不知道服务器的故障而继续工作，从而产生不一致的结果；同样，当一个客户程序故樟而重新启动工作时，服务器仍继续持有原状态信息，同时又不易恢复到初态。因此，操作在不依赖于服务器的存储状态时，可以简化文件服务器的设计。因此，对于分布式文件服务器，无状态的文件服务更为优越。

## 四、原子性

简单的无状态服务器不能满足所有应用的需要。如果多个客户进程同时修改文件，按照一致性约束，服务器必须能够保证一个客户对某数据项所做的完整修改应在另一个客户进程访问这一数据项之前完成。

## 五、Cache的应用

Cache(高速缓存)的应用非常广泛，其意义在于提高访问效率，减少文件的访问开销。在文件服务中，文件可以在四个位置上存储，就是客户机的主存、客户机硬盘、服务器主存和服务器硬盘(如图所示)




在Cache的设置时必须解决几个重要问题：

首先是Cache中可交换单元的大小，如可以用磁盘块作为单位，也可以将整个文件作为缓冲单位。以磁盘块为单位，可以增加Cache的利用率；以整个文件为单位，可以提高文件的访问速度.

其次是Cache中文件或文件块的替换算法，

这些算法在操作系统和体系结构课程中均有论述，常用算法有随机选取、FIFO(先进先出）和LRU（最近最小使用）等。其中最一般方法是将文件放在服务器的硬盘上，此方法最大问题是效率即访问文件非常慢。

Cache设置于何处?

第一种方法是设置在服务器的主存中；

第二种是将其设置在客户机的磁盘中；

第三种是将其设置在客户机的内存中。

第一种方法直接、简单，对于客户机是透明。完全由服务器来处理Cache与磁盘的数据一致性和同步关系，且可对客户透明，可保持主存和磁盘拷贝同步，不产生一致性问题。但这种方法在网络上仍然有大量消息需要传送。

第二种方法可以消除大量的网络传输，对于大量数据的访问有优势，但对于少量数据反而不如第一种方法，此种方法还存在Cache的一致性问题。

第三种方法既可以消除大量的网络传输，又可以获得较高的性能。但是这种方法需要系统给予必须的支持(如内存锁定)，若无此功能，用户是无法控制数据是否处于内存之中的。与第二种方法一样，必须考虑Cache的一致性问题。

## 六、Cache的一致性

值得注意的是，在计算机中任何设计方法都是“有得必有失”，比如，Cache的设置带来了访问的高效性，同时又带来了访问数据的一致性问题。

解决一致性问题的一般方法有写穿透(WriteThrough)，也就是当客户机程序要修改文件时，直接修改服务器文件，而读文件时直接读取Cache中的文件。但是，如果当客户机1上的A进程读取文件F后暂时挂起，另一个客户机上的进程B此时修改文件F，当进程A重新执行时，会直接使用其Cache中的数据，那么两个客户机进程使用的数据便是不一致的。

解决这一问题，可以让Cache管理程序定期检查服务器是否已经修改，可以通过比较Cache的读取时间和服务器上的文件最后修改时间得知其修改状况。如果时间一样，则Cache可用，否则必须从服务器中重新读取数据。

写穿透算法只有助于读文件时提高性能：对于写文件，则没有任何性能收益。许多设计人员认为这是不能接受的，于是提出一种修改方案，就是客户程序在写文件时只在Cache中做记号，过一段时间(如30s)之后，再将这些有记号的数据一起写人服务器，这种方法叫做延迟写(Delayed Write)。

但是许多程序都产生一些临时文件，在短时间内先写、后读，最后删除，假如这个时间段小于延迟写方法中的写时间段，那么，这些文件就不会写入服务器，这样，对于临时文件既不必使用服务器，也提高了访问效率。

当然，延迟写带来的问题是语义问题。因为若有另一进程读文件，则读取文件的正确性取决于读的时刻。因此，延迟写策略是在性能和清晰语义之间的权衡。

沿此思路就有了使用----对话语义方法（关闭时写），即仅在文件关闭时才写服务器。

这样就会出现问题，若两个进程都读取了文件，并先后关闭文件，那么第一个关闭文件的进程：不会对文件有影响，文件的最后版本完全决定于笔二个关闭文件的进程，即出现前述的“修改丢失”问题。

还有一种完全不同的处理一致性的方法是集中控制方法

文件服务器记录客户打开文件和读、写文件的情况。如果文件是以读方式打开的，则可以让其它进程读，但必须阻止其它进程写；如果某个进程以写方式打开文件，则必须防止其它进程的任何访问。当一个文件关闭时，可以将修改后的文件写回服务器，服务器修改文件访问记录。

## 4.4.2 访问控制

UNIX系统使用的访问控制方法比较通用和高效，目录服务的访问控制可以参照这种方法实现。可以访问文件的对象一般有四类：

①文件拥有者；

②目录服务负责命名和访问控制；

③标记为系统管理员的客户进程，必须经特殊授权来管理文件内容；

④所有其它客户。

在UNIX中，同一文件可以出现在许多目录中，使用有向图让用户在自己的目录中记录共享文件的名字。如果要使文件可存放到不同目录中，访问控制表就不能只与特定文件相关，需要将访问控制表存在文件属性中，而不是目录文件中。

## 4.4.3 权能(Capability)

对于文件的访问必须加以控制，权能就是一种基本的文件访问控制方法。

一、权能定义

权能是对文件的一种标识，该标识含有授予该文件执行一定操作的权限。权能用位串来描述，该位串由以下三部分：

①标识符：标识符能唯一标识权能所涉及的文件。

②访问权限：访问权限是进程对所操作文件拥有的操作权限的详细说明，每个权限一般用一个二进制位来描述。

③随机数：随机数使得权能难以猜测，通常将访问权限与随机数组合在一起使用。

## 二、建立权能

权能可由专门管理权能的服务操作建立，以相应权限为用户所使用。当为某客户建立新文件时，服务器建造一个具有所有权限的权能返回该用户。如果该文件是一个独占个人文件和目录，则该权能不需修改，也不必传给另外的用户；

## 三、修改权能

任何一个为文件授予权能的服务都可以包含修改访问权限的部分，访问权限的修改允许客户把现有权能与其所需权限结合在一起并最终返回一个具有这些权限的新权能

## 四、取消权能

具有权限的用户可以请求服务器收回权能从而取消对该文件的访问。权能的副本可分布在用户的多个进程中，用户和服务器都不必知道这些副本在什么地方

## 五、保存权能

由于文件、目录的生存期一般比具有其权能的用户进程生存期要长，用户为了今后能重新得到这些文件的权能，需要用某种方法保存权能。

## 六、权能的保护

如果权能被当成保护机制，其自身必须被保护

## 4.4.4 文件的存储

### 一、文件的逻辑结构

在文件服务器中文件逻辑上是数据项序列。设F为文件，Pi为第i页，Ii为页中的一个数据项，E为二串数据项称为文件尾。文件可定义为页的有序集F=<P1，P2，…，Pn-1, E>

### 二、块服务

文件服务器的一个主要任务是记录文件的地址以及与之相应的UFID，故需要一个文件位置映射表(FLM，即File Location Map)．该映射表存储一组<UFID，文件地址>映射，用于将内部名直接对应到物理地址上。

当创建和扩充文件时，必须为该文件分配空间；当删除或截断一个文件时，必须回收空间。这些操作需要存储空间的动态分配和按不连续存储块进行管理的机制，以便于文件的扩充和缩小。

要完成文件的这些基本操作，其基础就是块服务。块服务主要用于管理外存储器，包括分配、读、写等操作。在分布式文件系统中，块服务一般是嵌入在文件服务中的模块，也可以做成单独的块服务层次。

以块服务支持文件服务的优点有以下几点：

①不同服务共享同一磁盘空间

②可使用不同的存储介质；

③将文件服务与磁盘和其它存储介质的优化访问问题分开考虑。

三、文件索引

文件索引主要包括文件长度、文件属性和物理块的指针，主要用于支持文件逻辑页到物理块的映射，另外，文件索引可组织成树型索引结构，即，一个索引中物理块指针指向的是一个包括另一个子索引的物理块，而不是一个具体的数据块。

## 四、文件定位

在文件请求中，对于无状态的文件服务，可以通过文件的UFID和数据段的起始地址来定位(以页为单位时，一般为页号)；对于有状态的文件服务，可以只给出文件的UFID进行文件定位，页的定位由状态确定。

具体定位方法有两种。一种是两步定位，第一步，首先通过UFID在文件映射表中找到文件索引所占物理块的指针；第二步，用逻辑页号通过文件索引中的内容查找到物理块指针，完成定位。另一种是一步定位，即将UFID和逻辑页号通过查找服务器中的映射表一次取得物理块指针完成定位。

两种方法的区别在于，两步定位方法中文件映射表中查找的键字是UFID，查找结果是文件索引的物理块指针；而在一步定位方法中文件映射表中查找的键字是UFID和逻辑页号，结果是该页的物理块指针。

## 4.4.5 分布式文件系统的实现原则

第一，尽可能地利用工作站的能力，特别是要在进行实际工作前选用工作站进行前期开发比较好，因为工作站使用方便，价格也便宜。

第二，尽可能利用Cache，可以大量减少计算时间和网络负担。

第三，开发应用特点，就是“具体情况具体分析”。例如，在典型的UNIX系统中，有三分之一的文件是临时文件，既省时间，又不必共享，这样可以获得很高的性能。

第四，尽可能减少系统层次的规模，利用层次性结构来设计可以提高系统的应变性。

第五，安全性原则。要防止“万一”的可能，通俗地讲，就是“不怕一万，就怕万一”。例如，以客户是不搞欺骗为基本假设而设计的系统(一般情况下是这样)肯定存在相当大的安全问题。

最后，批处理可以获得高性能。显然传送1个50K的文件要比传送50个1K的文件要快得多。

# 4．5 分布式文件系统实例SUN NFS

NFS最初由SUN公司在工作站上提供。它支持异构系统，例如，MS-DOS客户机和UNIX服务器，硬件也不需要一样。显然，MS-DOS的硬件平台一般是intelx86的CPU，而UNIX服务器一般都是Motorola或SPA尽C的CPU。

## 4.5.1 NFS的结构

NFS的基本思想是允许任意客户机和服务器共享文件。虽然，客户机和服务器大都处于同一局域网上，但NFS也可以运行于广域网上。

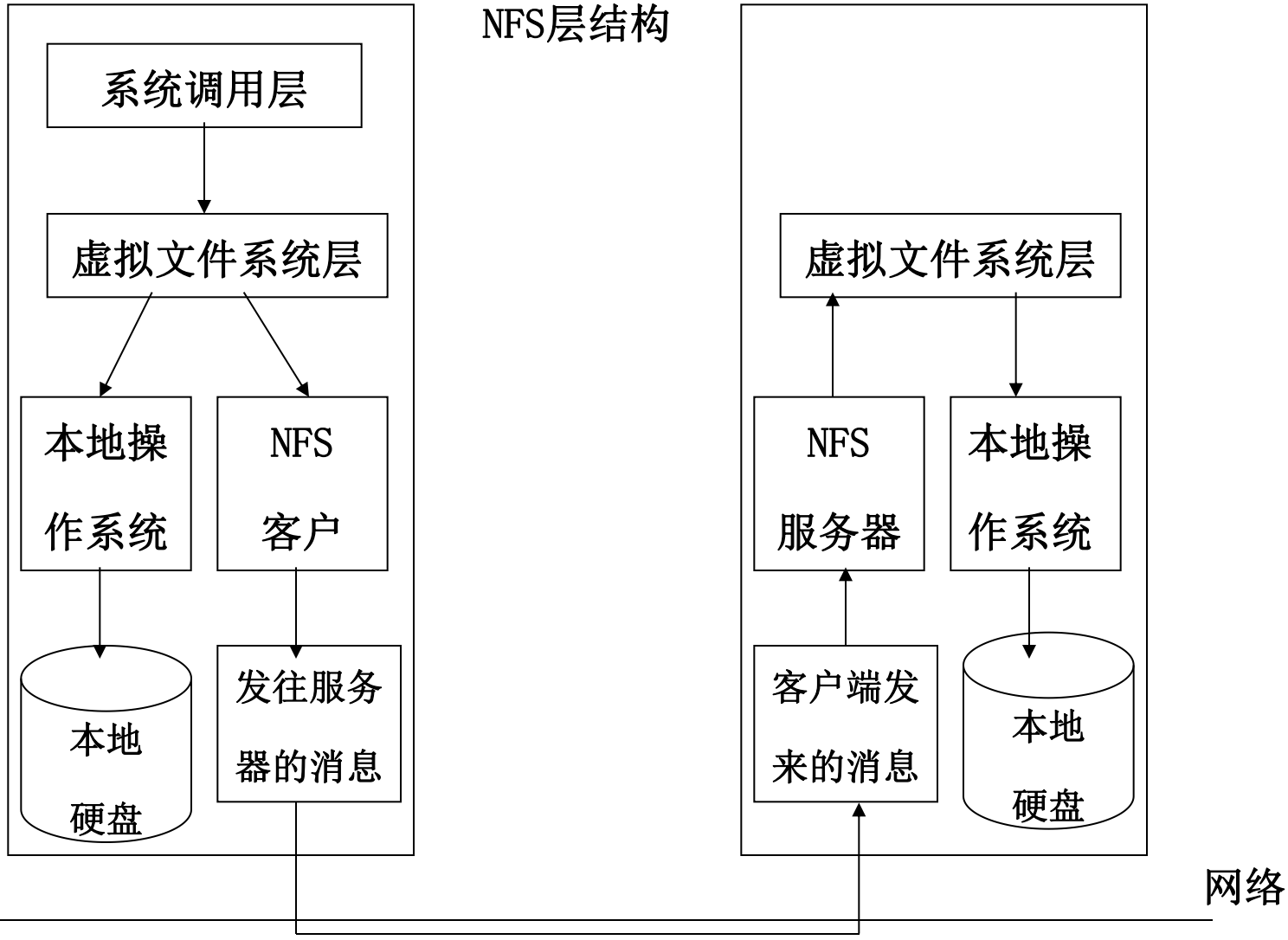
为最简单起见，我们一般认为，客户机和服务器驻留于不同的机器上，但NFS支持逻辑上的客户／服务器模型，即客户和服务器均处于一台机器上。

## 4.5.2 NFS协议

NSF定义了两个协议

第一个NFS协议用于安装

第二个NFS协议是目录和文件访问.


NFS层结构
系统调用层
虚拟文件系统层
本地操作系统
NFS客户
本地硬盘
发往服务器的消息
虚拟文件系统层
NFS服务器
本地操作系统
客户端发来的消息
本地硬盘
网络

# 4．6 分布式文件系统的发展趋势

4.6.1 硬件

4.6.2 规模

4.6.3 广域网

4.6.4 移动用户

4.6.5 容错

4.6.4 多媒体

# 第五章 分布式系统文件共享

# 5.1 共享文件的语义

两个以上的用户共享同一个文件时，会产生多种情况，从而产生不同的语义．故文件服务时必须精确定义服务的读写语义。

## 一.UNIX语义(时间顺序)

对于单处理机而言，在UNIX系统中，其读操作的语义是，读取的结果是它前面最近一次写操作形成的结果。写操作的语义是，若先后连续有两个写操作，则文件结果决定于后面的写操作。因此，最后形成的语义是严格意义下的时间序操作。

在对分布式文件系统中的文件进行读操作时，能看到以前所有对该文件执行写操作的效果。特别是，客户对于已打开文件的写操作可立即为其它打开此文件的客户所见。客户可共享文件当前位置的指针。这样，一个客户将指针向前推进时将影响所有共享客户的视图。

此种语义的特点是易于理解和实现。

二. 会话语义

对于打开文件的写操作可以立即为本地客户所见，远程的客户也同时打开该文件，但却不可见。一旦文件关闭，对此文件所作的修改仅为后面进行的操作所见，该文件已经打开的各副本不表现这些修改.

## 三. 不可改变文件语义

一但文件为共享文件，则所有用户均不能再修改它。这里的不可改变有两个含义：一是其名字不可再变；二是其内容不可改变。这样，不可改变的文件的名字代表该文件的固定内容，而不再是信息存储机制。这一语义非常简单，易于实现，但应用起来，很不灵活.

## 四. 事务语义

用户若要访问一个文件或了组文件，首先要执行一个启动事务的操作,表示下面的操作必须独立执行，然后对文件进行读写操作，当工作完成后，再执行一个结束事务的操作。

其关键特性是,保证事务期间的所有文件操作按序执行，而不受其它用户的干扰，也就是说，在事务内部严格具有UNIX语义、显然，事务语义是一种比较实用的文件语义。事务的完成要求一个客户机与一个或几个服务器进行协作。

# 5．2　原子事务

在分布式系统中，原子事物又简称事物，事务实际上就是一组逻辑上连续执行的操作，其具有动态性，有三种状态：

①提交 事务中的文件数据项的修改永久保存

②中止 由于同其他事务冲突或硬件故障导致事务中止

③临时 事务执行中的存在的临时状态

## 5.2.1 事务的特性

事务具有以下四个特性,简称ACID特性

①原子性(Atomic):即事务的作用要么完整，要么没有。

②一致性(Consistent)：事务处理不影响系统中的不变性：意思是，当系统具有某种不变特性需要保持时，在事务执行前后该不变性一定要保持。例如，银行业务系统中有一个关键的不变特性是“金钱不灭”，经过内部任何转帐之后，银行的总钱数是不变的。

③孤立性(Isolated)：并发的事务不会相互影响，多个事务处理可并发执行，其结果和各事务处理串行执行结果一样，也叫串行等价性。

三个事务A、B、C被三个独立的进程同时执行,若顺序执行其结果为1、2或3

```
BEGIN_TRANSACTION A     BEGIN_TRANSACTION B     BEGIN_TRANSACTION C
    X=0;                    X=0;                    X=0;
  X=X+1;                  X=X+2;                  X=X+3;
END_TRANSACTION         END_TRANSACTION         END_TRANSACTION
```

时间 →

| | | |
|---|---|---|
| 调度1 | x=0;x=x+1;x=0;x=x+2;x=0;x=x+3; | 合法 |
| 调度2 | x=0; x=0;x=x+1; x=x+2;x=0;x=x+3; | 合法 |
| 调度3 | x=0; x=0;x=x+1; x=0;x=x+2; x=x+3; | 不合法 |

④持久性(Durable)：如果事务处理成功完成、则结果将永不消失，除非发生硬故障。

## 5.2.2 事务需求

| 服务过程 | 解释 |
| --- | --- |
| 存款(账号，数额) | 将指定数额的款项存入给定账号 |
| 取款(账号，数额) | 从给定账号取出指定数额的款项 |
| 平衡(账号) | 返回给定账号的当前平衡 |
| 总平衡() | 返回该客户所有账号的总平衡 |
| 开始事务处理(标号) | 开始指定标号的事务处理 |
| 结束事务处理(标号) | 结束指定标号的事务处理 |
| 流产事务处理(标号) | 迫使指定标号的事务处理流产 |

银行基本业务服务

# 银行服务的例子

开始事务处理(T)；
K：取款(A，100)；
K：存款(B，100)；
K：取款(C，200)；
K：存款(B，200)；
结束事务处理(T)

我们将用T、U、V代表事务处理标号，用K、M、N代表不同的银行分行，用A、B、C代表客户的分行账号，一个客户发出的一系列服务过程调用就可以合并为一次事务处理。

# 5．3 并发控制

并发控制的主要目标是满足事务处理的一致性(串行等价性),最早的方法:

A.某一时刻只允许执行一个事务

B 在启动多个事物操作之前先检查是否满足一致性

缺点:

解决的不好.为弥补不足.提出下面三种方法.

## 5.3.1 加锁

当某一事务访问一共享数据项时，由服务器对该数据项加锁，当完成访问时，再由服务器开锁，以便于其它事务访问。在上锁期间，只有锁定该数据项的事务才能对其访问，这样就保证了在某一时刻访问数据进程的唯一性和确定性。

一. 基本原理

一个锁可由三都分组成：

①一个二值逻辑变量，用以指示上锁 / 开锁；

②一个类似于信号灯的条件变量；

③访问该锁的宿主事务标识符

实现上锁机制时，需要注意锁的粒度。粒度是指被加锁的数据项的大小，粒度越细，则并行度越高，反之，并行度越低。对整个文件加锁是一种极端情况，这时候，事务串行执行。在下面的讨论中，上锁一般施加于文件中的数据项上。

锁定机制是分两个阶段进行的。一个事务在工作过程中，可分为“生长”和“消亡”两个阶段。生长阶段需要上锁，消亡阶段需要开锁，这就是两阶段锁定机制。在生长阶段，事务处于临时状态，其临时数据不为其它事务所见。在消亡阶段，临时数据要变成永久数据，为了保持事务的特性，必须在事务关闭的最后，才能开锁。

## 二、几种加锁方案

### 1. 最简单的加锁方法

在这种方案中，文件服务器对客户事务访问的每一个数据项加锁，而在事务完成(或中止)时打开所有的锁，当另一事务试图访问已上锁的数据项时，它必须等待到开锁为止。

### 2. 读 / 写锁方案

由于简单锁定机制不必要地将所有访问到的数据项锁定，从而降低了事务的并发性。特别是当事务中均是读操作时，便没有必要上锁。

基于这种分析，提出了读／写锁方案，即允许多个事务并发读同一数据项，只允许一个事务写一个数据项。也称为“多读／单写”方法。在这种方法中，对于读操作，还不能放弃上锁，因为不上锁，可能会有其它事务修改它，造成不一致。为此，要采用两种不同的锁，即读锁和写锁 对于访问的所有数据项均可上读锁，只对写操作访问的数据项上写锁。上写锁的数据项不能被其它事务所访问，上读锁的数据项只能为其它事务读，但不能写。

上锁和开锁的基本规则如示：

1．当客户在事务中访问数据项时，有如下情况：

①如果数据项还未上锁，服务器将其锁定，并让客户防问该数据项；

②如果数据项已被其它事务上锁，客户必须等待该锁打开：

③如果服务器已经锁定了本事务中的一个数据项，客户可以继续防问。

④如果事务想要写自己已上有读锁的数据项，应当将读锁改为写锁。

2. 当事务提交或中止时，服务器打开它为该事务锁定的所有数据项。

### 3. 读写锁的死锁问题

以上两种方法都在一定程度上提高了并发性，但与此同时也会带来另一个问题——死锁。所谓死锁就是一组事务中的每个操作都处于上锁且又等待开锁的状态，例如以下两个事务U和T，在时间顺序上依次采取如下动作，结果将导致死锁。

T等待事务U释放读锁b，而它本身又对其加读锁引起事务U对其解锁的等待，由此，便导致了互相牵制。

解决方法有如下4种

①在事务开始执行前便对其所要访问的数据加锁，这虽能预防死锁，但却降低了资源共享率。

②给资源规定一个序号，申请资源时必须按序号单调递增或递减的方向申请，这种方法也降低了并行性。

③通过资源申请占有图来检测有无死锁，一旦发现死锁便由服务器中止一个事务来打破循环占有等待，解决死锁。

④“时限”控制，是文件系统中较常用的方法，即给每个锁规定一个时间段。在此时段内，该锁是稳定的，若超出此时限后，该锁便变成易损锁，若此时没有别的事务对上锁数据项竞争，则该锁继续保持；否则的话，便打破此锁，与此同时，原上锁事务中止。这种方法也有两个不足，第一是增加了系统开销；第二是“时限”的取值问题

4．意向写锁

读／写锁中读锁的存在阻止了其它事务对其进行写操作，在一定程度上降低了并发性。然而事务的执行要经过两个阶段，在临时阶段，写操作实际上只是将改写的内容写到一个临时缓冲区中，并未改写实际的数据项。只有在提交阶段才写回数据项，基于此原理可把读／写锁改成意向写锁和提交锁来提高并发性.

## 5.3.2 乐观的并发控制方法

### 一. 问题的提出

使用锁机制处理并发控制时存在一些缺陷：

①分布式系统中的锁机制是一种额外的开销。例如，在只有读操作的事务中，锁可以保证所读的数据项不被别的事务修改，但这种锁只有在最坏的情况下才有必要。又例如，两个客户进程并发地对n个数据项进行增值运算，若它们同时启动，执行时间量也相同，以互不相关的序列访问数据项，并且各自使用一个事务来访问和增值数据项，则这两个程序试图同时访问同一数据项的机会仅有1／n，也即每n个事务中实际有用的锁只有一次。

②使用锁机制会导致死锁，并且没有令人满意的死锁解决算法。在锁机制中，只有在一个事务终止时才释放它的所有锁，这明显有损于并发性。正是基于以上原因，有人提出另一种算法——乐观的并发控制方法。之所以称其为“乐观”，是基于这样一种假设，两个客户的事务同时访问某一数据的可能性很小，因此两个事务可以执行下去，直至发出CloseTransaction请求。当产生冲突时，一般要中止一些事务，并由客户重新启动。这样，每个事务便分为以下三个阶段：

1. 读阶段：在这一阶段中，每个事务有一个待更新数据的临时版本。读请求可以立即执行，如果有临时版本存在，则要访问最近提交的数据值。而写请求以一种其它事务不可见的形式缓存起来，若有几个并发事务，可能会同时存在同一数据项的几个不同的临时值。另外，针对于每一个事务需要设置两个集合：读集合和写集合，读集合列出事务所读的数据项的集合，而写集合则列出事务创建、修改、删除的数据项集合。

2. 确认阶段：当服务器收到CloseTransaction请求之后，进入这个阶段，在该阶段中，对该事务进行确认是否可以将该事务的写操作结果永久保存下来。

如果事务确认成功，则进入写阶段(写操作结果记录到相关文件中，事务成功完成，发出commit);否则，要解决冲突，需要中止某些事务。确认阶段是建立在一致性基础上的，即如果事务执行的结果等价于各个事务顺序执行的结果，则该事务视为确认成功。

3.写阶段：如果一个事务确认成功，则临时版本记录的所有修改均可以变为永久性修改。只读事务可以在确认通过后立即提交。写事务在临时版本中的数据变为永久数据之后立即提交。

## 二、事务的确认

确认是利用读写冲突规则来保证一组重叠事务(即当前事务还未提交便已开始的事务)的调度符合一致性，当一个事务完成第一阶段工作后,为其指定一个事务号，若该事务确认成功完成,则事务号被保留下来：否则，若事务未被确认，或事务是只读事务，则释放该事务号.

确认工作主要基于两个事务操作的冲突来完成的　对于两个重叠事务Ti和TJ，必须满足下列规则。

确认方法有两种，一种叫做向后确认(Backward Validation)，以正在执行确认的事务为基准，检查已经进入确认阶段的事务。一种叫做向前确认(Forward Validation)，以正在执行确认的事务为基准，检查后续进人确认阶段的事务.

三. 饿死现象

事务中止后，通常由客户程序重新启动，但有可能该事务仍然无法通过确认，于是其又被中止，重启，再中止.如此，该事务则被剥夺了提交能力　此现象即为饿死

## 5.3.3 时间戳

要利用时间戳完成并发控制，需要对每个事务的操作进行有效性检查，若检查未能通过,则该事务立即中止并重新启动

基本的时间规则:

①事务对数据项的写请求，仅当该数据项最近由前一个事务(有冲突)读和写，才能有效。

②事务对数据项的读请求，仅当该数据项刚刚由前一个事务(有冲突)写，才能有效。

该规则允许并发事务共享临时数据项，从而确保每个数据项的临时值按时间戳顺序提交.

# 5．4　恢复

事务的原子性要求事务要么提供完整的运行结果，要么么作用都没有，即持久性和失效原子性。这两个需要并不是独立的，可以由服务器上的独立机制来管理，我们称这个机制叫做恢复管理器。

恢复管理器的主要任务是；

①将提交事务的数据保存到永久性存储介质(恢复文件)上；

②故障重启后，恢复服务器的数据；

③组织恢复文件，改进恢复性能；

④回收恢复文件涉及到的空间。

# 5．5　事务服务中的文件版本

## 5.5.1 文件版本的实现

通过在每个文件的索引表中扩充一项,即版本号,通过对影子页的操作,到事务提交时,若无版本冲突,则合并临时版本与当前版本得到最新版本.若有冲突则放弃临时版本.

## 5.5.2 意向表的实现

也可通过对影子页的操作实现意向表,

意向表记录:操作类型、事务标识符、文件标识符、页号、影子页面的指针

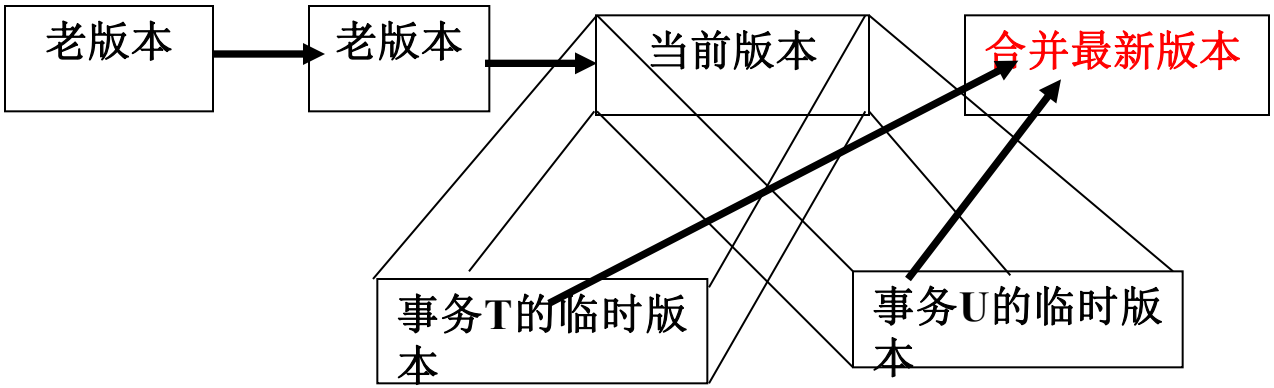
## 文件版本方法

解决两类问题:版本冲突和串行冲突

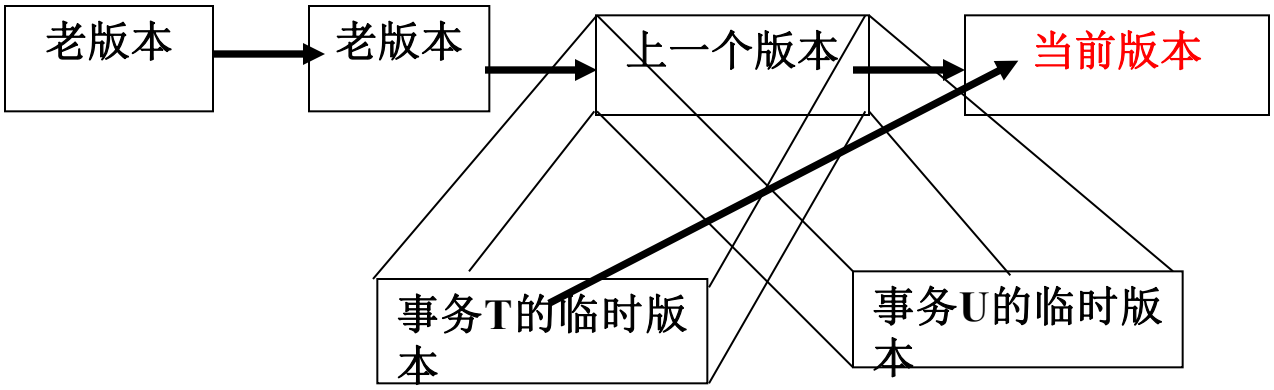
版本冲突:并发事务访问同一个文件的不同数据段,从而产生不同的版本,但无一版本包含所有的修改.

串行冲突:并发事务访问同一数据段,从而有多个写操作,导致数据项决定于最后的版本.

版本冲突解决如图:



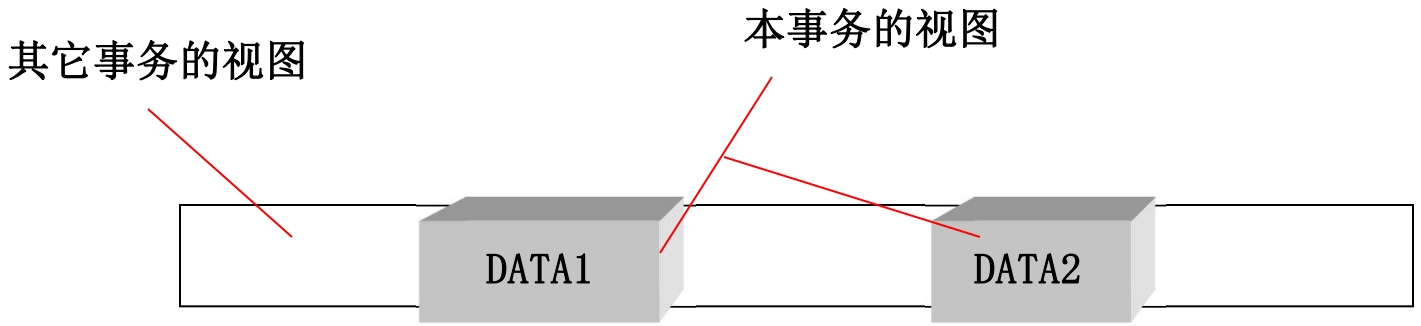

版本的合并




串行冲突的解决

# 意向表方法

意向表实际上是一个事务操作的日志记录,是两阶段提交的机制.即:第一阶段,事务处于临时状态,第二阶段,事务进入提交阶段.如图




DATA为服务器为待修改的数据的临时拷贝.意向操作只是记录到意向表并不是真的对文件操作,一个意向只有给出足够的信息,才能到第二阶段执行.

# 第六章 分布事务与文件备份

# 6．1 合作服务器

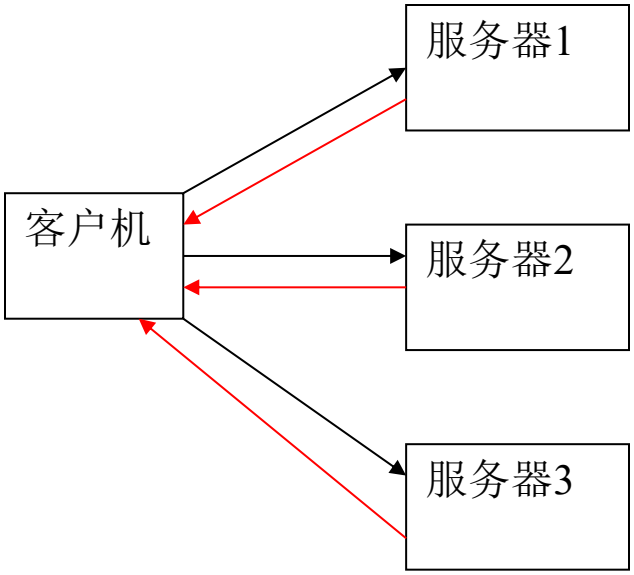
合作服务器是由多个物理服务器合作完成一个逻辑服务器的功能，各个服务器由网络互连，每个服务器可具备不同性能，可位于不同地点，并持有整个合作服务器中所有文件的一部分

# 6．2　分布事务

分布事务是指一个事务将涉及到多个服务器的操作，即该事务是由合作服务器完成的，构造分布事务的方法有简单分布事务和嵌套分布事务两种

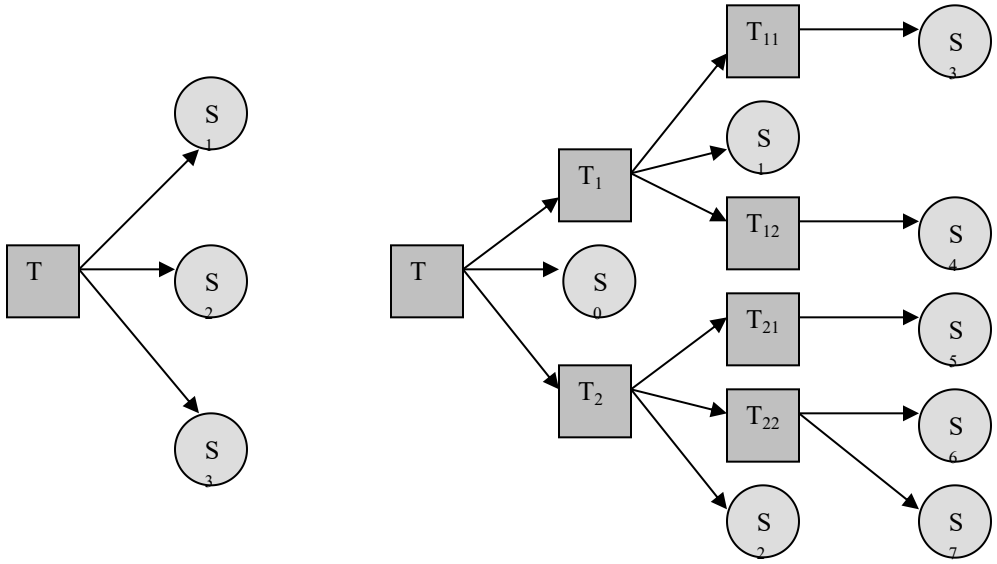
简单分布事务:客户机可以多次访问不同的服务器,服务器仅响应客户机的请求,不引发其它服务器的操作

嵌套分布事务:一个服务器上的操作可能引发其它服务器操作




在分布事务中,多个服务器需要相互通信和合作,各自完成部分工作,最终是事务提交完成.在分布事务处理中,第一个响应客户机请求的服务器为该事务的协调服务器,负责中止、提交该事务，其后加入的服务器为工作服务器。

# 事务处理分类




(a)分布式平面事务处理　　(b)分布式嵌套事务处理

其中方框代表事务处理，圆形代表执行操作的服务器

# 分布式事务处理

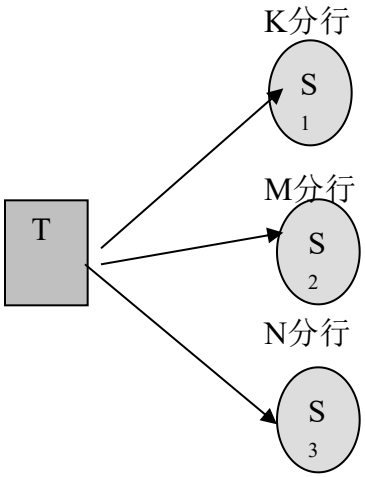
- 分布式事务处理的关键在于服务及数据的分布，即一个事务处理所需的服务与数据可能分散在不同的服务器上，因此，事务处理过程就必须在多台服务器上执行。
- 分布式事务处理的调度与同步
  - 多台服务器联合执行一个事务处理时需要彼此协调，才能做到整个事务处理的成功提交
  - 常用的方法是由一个协调者(coordinator)通过服务器之间的通信来实现最终提交

# 分布式事务处理例子

开始事务处理(T)；
K：取款(A，100)；
M：存款(B，100)；
N：取款(D，200)；
M：存款(C，200)；
结束事务处理(T)

某客户要在K、M、N三个银行分行的A、B、C、D四个账号上执行转帐业务，即从K分行的A账号取出100元，存入M分行的B账号去，然后从N分行的D账号取出200元并存入到M分行的C账号。假定这三个分行的数据库分别位于三台服务器上，其中$S_1$管理K分行的A账号 ，$S_2$管理M分行的B、C两个账号，$S_3$管理N分行的D账号

# 分布式银行事务处理




| | |
|---|---|
| (1.1) 开始事务处理(T)；<br>(1.2) K：取款(A，100)；<br>(1.3) 结束事务处理(T)； | 协调者 |
| (2.1) 加入服务器(T，$S_1$)；<br>(2.2) M：存款(B，100)；<br>(2.3) M：存款(C，200)； | 参与者 |
| (3.1) 加入服务器(T，$S_1$)；<br>(3.2) N：取款(D，200)； | 参与者 |

由于每个服务器可能同时执行不同的分布式事务处理，因此在整个系统中事务处理标号必须是唯一的。首先启动分布式事务处理的那台服务器是整个事务处理的协调者服务器

# 6．3 分布事务的提交协议

最简单的方法:

1．一阶段原子提交协议:

即由协调服务器不断查询所有工作服务器,直到所有工作服务器都回答提交成功,完成整个事务提交

2．两阶段提交协议(2PC协议)，可以保证分布事务的原子性，在此协议中，任何服务器都可以随时中止自己的子事务而不影响客户机事务的正常提交或中止。

协调服务器分为两阶段工作(2PC)：

第一阶段　投票阶段

A协调服务器向每个工作服务期发出提交请求

B工作服务器收到请求后,回答YES或NO,如回答有NO,则终止

第二阶段　完成阶段

C协调服务器查看搜集的票数,若无反对票,协调服务器则提交该事务并通知所有工作服务器,否则,若有反对票协调服务器则终止事务,并向所有回答YES的工作服务器发出终止请求

### 3 嵌套事务的两阶段提交协议

嵌套事务的两阶段提交协议的执行过程的第一阶段与非嵌套事务不同，当工作服务器接到提交：

1)如果工作服务器具有暂时提交且是顶层事务的子事务

A.检查它有没有前辈事务处于中止事务表中，准备提交

B 中止具有中止前辈事务的事务

C 向协调服务器投票YES

2)如果工作服务器没有处于暂时提交状态的、且是顶层事务的子事务，它肯定已经失败，应向协调服务器投票NO

注意:

A.子事务的标识符可以通过扩充其父事务的标识符;

B.子事务的提交决定于其父事务的提交,但父事务的提交并不决定于其子事务的提交

# 6．4　分布事务中的并发控制

| 开始事务处理(T)：<br>K：取款(A，40)；<br>K：存入(B，40)；<br>结束事务处理(T)； | | 开始事务处理(U)：<br>K：取款(C，30)<br>K：存款(B，30)<br>结束事务处理(U)； | |
|---|---|---|---|
| **分解操作** | **平衡** | **分解操作** | **平衡** |
| A．平衡 ← A．读出() | (A) 100元 | | |
| A．写入(A．平衡 – 40) | (A) 60元 | | |
| | | C．平衡 ← C．读出() | (C) 300元 |
| | | C．写入(C．平衡 – 30) | (C) 270元 |
| B．平衡 ← B．读出() | (B) 200元 | | |
| B．写入(B．平衡 + 40) | (B) 240元 | | |
| | | B．平衡 ← B．读出() | (B) 240元 |
| | | B．写入(B．平衡 + 30) | (B) 270元 |

**当多个事务处理访问同一个数据时，顺序等价性要求必须把每一个事务处理对该数据的完整(读/写)访问一一排序，严格禁止任何冲突**

## 1. 分布事务中的锁

每个服务器都要提供锁管理机制，本地的锁管理机制可以决定是否接受事务的请求操作。

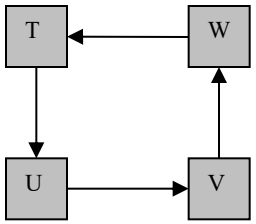
利用互斥锁进行事务处理并发控制

| 开始事务处理(T)；<br>K：取款(A， 40)；<br>K：存款(B， 40)；<br>结束事务处理(T)； | | 开始事务处理(U)；<br>K：取款(C， 30)<br>K：存款(B， 30)<br>结束事务处理(U)； | |
|---|---|---|---|
| **分解操作** | **互斥锁** | **分解操作** | **互斥锁** |
| 开始事务处理(T) | | 开始事务处理(U) | |
| A．平衡 ← A．读出() | 锁定A | C．平衡 ← C．读出() | 锁定C |
| A．写入(A．平衡 － 40) | | C．写入(C．平衡 － 30) | |
| B．平衡 ← B．读出() | 锁定B | | |
| | | B．平衡 ← B．读出() | 等待B的锁 |
| B．写入(B．平衡 ＋ 40) | | ． | |
| 结束事务处理(T) | 释放A，B | ． | |
| | | ． | 锁定B |
| | | B．写入(B．平衡 ＋ 30) | |
| | | 结束事务处理(U) | 释放B，C |

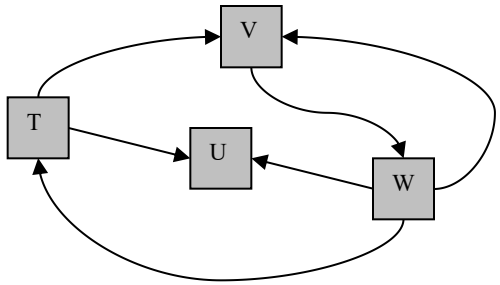
# 分布式死锁

- **在各种涉及互斥的算法中，只要算法采用互斥锁，就有可能发生“死锁” 现象。**
- 死锁的典型特征是一组事务处理形成了一条循环等待链
- 死锁处理：
  - 置之不理 （鸵鸟算法）– 由程序员对其负责
  - 预防（静态的使死锁在结构上不可能） – 不存在运行系统支持
  - 避免 （合理的分配资源）– 运行系统支持
  - 检测恢复（允许死锁发生，检测到后恢复）– 不同的算法

# 死锁 – 循环等待链




(a) 简单循环链




(b) 复杂循环链

- 互斥
- 持有并等待
- 不容抢占
- 循环链

**(a) T→U→V→W→T**

**(b) V→W→T→V**

**V→W→V**

# 分布式事务处理的死锁例子

| 事务处理U | | 事务处理V | | 事务处理W | |
|---|---|---|---|---|---|
| 存款(D，100) | 锁定D @ $S_3$ | | | | |
| | | 存款(B，300) | 锁定B @ $S_2$ | | |
| 存款(A，200) | 锁定A @ $S_1$ | | | | |
| | | | | 存款(C，500) | 锁定C @ $S_3$ |
| 取款(B，200) | 等待B @ $S_2$ | | | | |
| | | 取款(C，100) | 等待C @ $S_3$ | | |
| | | | | 取款(A，300) | 等待A @ $S_1$ |
| 死锁 | | | | | |

# 死锁检测

- 死锁是一种稳定的状态
- 尽管无法从局部状态检测分布式事务处理的死锁，但死锁依旧是环形等待链。
- 死锁检测算法：
  - 中央式 – 周期性地收集等待状态
  - 分布式 – 推出等待状态
  - 提示式 – 构造检测体系

2. 分布事务中的时间戳

利用时间戳进行读写协作

3. 分布事务中的乐观并发控制

分布事务通过一组独立的服务器进行确定，每个服务器都要确认事务访问的是自己的数据项，所有确认均通过两阶段进行。

# 乐观并发控制算法

- **[读阶段]** 在读阶段，该事务处理所访问的数据都有一套暂时版本，这套版本不对外，只由拥有者使用。采用暂时版本可以使事务处理流产而不会影响到长存数据。当执行一个读操作时，如果数据的暂时版本已经存在，则读操作立即访问之，否则，就必须访问那个数据最近提交的值。写操作把每一数据的新值作为暂时值记录在案。显然，如果若干个并发事务处理共享同一个数据，则这个数据可能有不同的暂时值。除了上述规则外，对每一个访问的数据，事务处理还要维护两个数值集合：读集合包括该事务处理读出的数据，写集合囊括该事务处理写入的数据。
- **[验证阶段]** 当事务处理试图提交时，就进入验证阶段，其目的是检测是否它所访问的数据与其它事务处理的操作有冲突。如果验证无冲突，该事务处理可以提交；否则我们就必须消解冲突。消除冲突的方法很简单：或者命令该事务处理流产，或者从卷入冲突的事务处理中选择一个令其流产。
- **[写阶段]** 如果该事务处理已经通过验证，暂时版本中所记录的数据更新就成为永久性的。如果该事务处理是只读型事务处理，则它马上提交；否则就要等到暂存数据全部写入长存存储器后，执行提交操作。

# 6．5　分布事务中的恢复

## 分布事务的恢复工作

| 服务器 | 状态 | 恢复管理器的工作 |
| --- | --- | --- |
| 协调服务器 | 准备提交 | 表示服务器故障时还未做出仟何决定．将向所有工作服务器发出AbortTransaction，将中止状态加入到恢复文件中．若状态是中止，工作相同。<br>如果没有工作服务器，将以超时判断，中止相应事务 |
| 协调服务器 | 提交 | 表示服务器故障时已做出提交决定．若是还没有做完. |
| | | 要发送DoCommit，给各工作服务器，从这一步开始恢复两阶段提交协议的执行。 |

| | | |
|---|---|---|
| 工作服务器 | 提交 | 表示已经提交．如果还未向上作服务器发送HaveCommited消息，则发送之；然后，执行整个事务中属于自身的那一部分操作(若文件操作是可重复的，这样做就不会引起不一致性)。 |
| 工作服务器 | 不确定 | 表示工作服务器在知道事务输出之前故障，必须等到协调服务器作山决定，可以有GetDecison获取。收到应答后，根据情况提交或中止。 |
| 工作服务器 | 准备提交 | 表示工作服务器还没有投票，可以中止事务. |
| 协调服务器 | 完成 | 不需要任何工作 |

# 6．6　备份

备份是与分布事务及合作服务器密切相关的问题。一个备份对象(如文件或数据项)由位于多个服务器中的许多副本组成，我们称组成一个备份对象的副本集合中的任意副本为该备份对象的一个代理。备份对象有如下特点：

(1)减少分布式系统中的通信量，并通过为用户提供本地副本而加快响应时间；

(2)可以从多台服务器上访问同一对象，从而提高系统有效性，降低服务器故障的影响及通信失败的可能性；

(3)可以在不同服务器上并行执行多个用户对同一对象的请求，从而提高系统吞吐率。

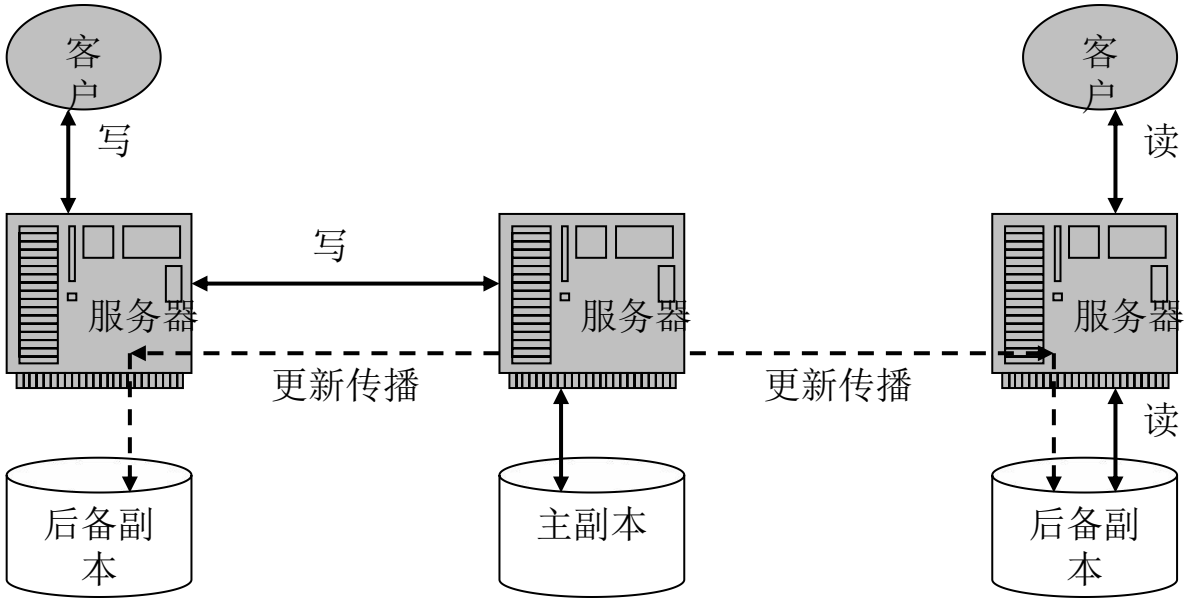
按照客户的观点，备份事务服务应当与非备份事务服务一样具有单副本可串行性，即通过并发控制，使多个事务表现为在一定顺序下的串行执行。

## 6.6.1 基本模型

最简单的策略就是读一／写全，即读时只读一个副本，写时要写所有副本。这样可以随时保证各副本的一致性。但是由于多个客户可能同时写某一副本而产生写冲突，所以必须提供并发控制机制来保证分布事务的基本特性.

## 6.6.2 主/从模型

备份对象的服务需求有一个基本服务器及多个二级服务器，基本服务器拥有一个基本对象副本并响应所有修改请求，其它的副本只响应用户的读请求，不响应用户的写请求。只接收来自基本服务器的修改消息来修改副本或直接拷贝基本对象副本。



## 6.6.3 可用副本模型

主/从模型的主要不足在于：备份文件在基本服务器失效时不能修改，并且仅适用于修改很少的对象。而读一/写全策略又是不现实的，因为当有些副本服务器因故障或通信问题不能工作时，是绝对达不到“写全”要求的

其基本思想是：即使有部分副本服务器故障时系统仍可工作，其基本策略是，客户的读请求可以在任何可用副本上执行，但客户的写请求必须在可用副本同时执行。可用副本模型要求副本服务器之间的通信无误.

## 6.6.4 具有分布控制的系统

我们要使用分布式控制机制，来完成副本管理，目的是即使一些副本失效，修改依然能进行下去。在这种方法中，任何一个副本服务器都能接收读写请求，并让客户得到有效一致的数据.

一. 文件组:备份文件的副本集合

为支持备份策略，文件组必须存储在每台副本服务器上。在完全一致的情况下，文件组中的副本有相同的初始值，并且各次修改都是针对整个文件组而言的，所有副本自始至终保持一致。

二. 版本号:

服务器在读副本之前，必须能从版本号和时间戳上判断该代理是不是最新版本。副本的初态一般被认为是第一版，以后每一次修改,就生成文件一个新版本.

三.多数一致算法:多数副本取得一致,即可完成操作

多数一致算法是最早提出的处理备份的分布控制算法，其原理可以应用于两个方面：

(1)备份：两个多数集合中至少具有一个相同元素，故多数原理能确保最新版副本的选择。

(2)备份的并发控制：它对于非当前版的文件可以作否决修改的决定。

## 6.6.5 分割与法定数

分割是指某一时刻，副本所涉及到的全部服务器之间不能两两相互通信而造成的一种难以保证副本一致性的状况，相当于将副本服务器分成若干个组，组内副本服务器可以相互通信，但组间服务器不能通信.

法定数：一个组内副本服务器的个数，目的是让处于不同组的副本服务器能够独立决定是否执行客户的请求.

## 6.6.6 法定数算法

原理:给文件组中的每个副本分配一个带权的选票,不同服务器权值不同,首先得到一个达到读法定数R(即R个选票)后,才能从最新的副本执行读操作,在写操作执行前,必须达到写法定数W(W个选票)

R和W值的设定所要遵循的规则:

① W > 选票总数的一半

② R + W >选票总数

## 6.6.7 虚拟分割算法

可用副本算法和法定数算法结合而来

实现:虚拟分割一个备份文件涉及的服务器,划分成若干组

注意这是逻辑分组

一个组中含有足够的副本服务器,能满足读法定数和写法定数则事务可在该组服务器上执行,这时候实际是采用法定数算法.若出现服务器故障便试图创建新的分割,使得分组可以满足读法定数和写法定数,从而保证单拷贝的可串行性.这种算法效率很高,在每个虚拟组内使用的是可用副本算法,此方法是基于假设(实际分割较少产生,也是一种乐观方法.)

# 第七章　容错与实时系统

# 分布式系统与WEB服务

**属性：**
- 可用性
- 可靠性
- 保险性
- 可信任性
- 完整性
- 可维护性

什么是“可依赖的系统”？

**后果：**
- 失灵
- 错误
- 故障

如何区分各种故障？

**策略：**
- 防止故障
- 故障容错
- 故障恢复
- 故障预报

如何处理故障？

容错与实时系统是分布式系统的两大领域.

一般而言，容错服务是允许系统出错的，但它可以在故障后恢复,而不丢失数据。

大多数容错应用有两种形式，一种叫做基于事务的容错;二种叫做进程控制的容错，二者的主要区别在于恢复时间。

容错具有两个方面，一是故障特征的描述,二是故障屏蔽,故障屏蔽的方法有层次式屏蔽和成组屏蔽两种.

本章中将讨论相关设计中的主要问题包括协议、调度策略和设计依据等。

# 7．1　事务的故障模型

利用事务处理服务的概念起源于数据库管理，最初的目的是提供原子性协议，用于处理消息的丢失。原子提交协议接受这样一个模型：机器在故障时，将不能做任何事情；而在故障前均能正确地做任何事。对于事务的处理，要考虑磁盘出错，服务器故障及通信不畅等出错问题，这样便需要一个故障模型，在该模型中，事务处理算法可以在可预测的故障下正常工作(含利用恢复技术)，但不保证在出现不可预测的灾难性故障下也能恢复正常工作。

故障模型描述如下：

1. 写永久存储器时可能出错，可能未写入，也可能写错

   例如在出现灾难性故障.

2. 服务器随时可能出现故障。

3. 消息传递延迟可能是无限制的。消息可能丢失、重复或受损。接收消息的服务器应该能够检测消息损坏情况。所有出错或未检测出的受损消息都会引起严重故障。

以上故障模型的特点有：

(1) 原子提交协议不能保证在有限的时间内完成，因为通信消息的延迟可能无限大；

(2) 分布式事务的恢复时间可能要比预料的时间长；

(3) 原子提交协议认为，服务器总能正确执行并能检测故障；

(4) 原子提交协议认为，永久性存储器上的受损消息和出错数据是可检测的。

以上故障模型是设计稳定存储器的基础，稳定存储器要在写操作出现故障时、或在进程故障时提供原子写操作。通信故障可以通过可靠的RPC协议缓解。下面先讨论稳定存储器，尔后更详尽地讨论容错概念与技术。

# 7．2　稳定存储

服务器(特别是文件服务器)的重要特性就在于其可恢复性,特别是文件映射表和文件索引的恢复。文件访问一般有这样几种出错情况：读块时出错、写块时出错、数据传输时出错及存储介质本身出错。

因此，需要一种检测错误的方法，一般采用计算并记录校验和的方法来解决，这一方法步骤如下：

(1)写块时，计算数据的校验和并记录下来；写块后，一般要读出一次，进行校验和比较，若不同则再写；同样，经过一定次数的重写后，若读出的校验和仍与写入的校验和不同，则记下“坏块”，并向客户机报错，从而保证写时的正确性。

(2)读块时，计算读出数据的校验和并与原记录的校验和比较，若不同，则重新读块。经过一定次数的重读后，若校验和仍然不同，则向客户机报错。这样，保证避免产生读块时的错误。

上述方法叫做精心块传输(CBT，Careful Block Transfer)方法，但是这种方法并不保证其可恢复性，因为若在写索引时出现错误，则所有文件块均将访问不到，如果在写文件映射表时出错，则无法再访问文件。针对这两种情况，文件映射表和文件索引一般要备份存储，当系统出现故障后，读操作发现校验和不一致时，则服务器可利用备份进行恢复。

这种备份一般采用两种存储结构，以减少两备份同时出现错误的可能性，一种常用的结构就是文件映射表和文件索引。

另一种结构用一个柱面映射表完成文件页到物理块的映射，柱面映射表的每一行由<分配状态， UFID，索引中的位置，块指针>组成。显然，柱面映射表的大小决定于磁盘的柱面数。

以上措施都假设在存储器可能出错的情况下，存储故障对于程序是不透明的，至少程序员认为存储器不是十分可靠的。

下面讨论的稳定存储(StableStorage)则是通过一定的手段使程序员感到所使用的存储器是非常可靠的，从而增加存储系统的故障透明性。

稳定存储其实是一种在不可靠介质上建立可靠信息存储系统的方法，它可以保证服务器系统可以从硬件或软件故障中得到恢复。

稳定存储一般采用冗余备份的方法。

由于稳定存储的空间冗余太大，故一般仅用于文件的关键信息，如前述的文件映射表和文件索引等。当然，它也用于对数据安全性要求甚高的应用中，如银行业务等。

实现稳定存储的主要原则有以下三个：

1）一般用两个相距较远的磁盘块形成一个稳定存储块，尽可能减少发生两块同时出错的可能性；

2）要同时保持以下不变性：

①至少有一个块是好块；

②若两块都是好块，则内容应当一致，若在操作过程中，可以有暂时的不同。

3）稳定块指针由两个正常的块指针提供.

# 7．3　容错

## 7.3.1　基本概念

### 一、部件故障

故障通常可分为暂时性、间歇性和永久性故障.

暂时性故障只发生一次，如果重复操作，故障可能不会再现。一只鸟在穿过微波波束时会引起无线网络上的信息丢失，这就是暂时性故障，如果下一步重传，它可能正常工作。

间歇性故障发生后消失，过一段时间后又出现。一个连接器中若结合不紧(如插座未插牢或虚焊等)，会引起间歇性故障。这种错误很难诊断，但一旦被找到故障点，系统很容易恢复。

永久性故障在故障修复之前一直存在。如烧坏的芯片、软件中的错误、磁头故障都会引起这种故障。

并不是所有的部件故障都会引起系统失败，但计算机系统的确会因某些部件故障而工作失败。设计和构造容错系统的目标是保证系统能够在部件故障出现时继续正常工作。这个目标与要求单个部件的高度可靠性是不同的。

## 二. 系统故障

在分布式系统中，我们最感兴趣的是系统在部件出错时能否继续工作。由于分布式系统中有大量部件，出错概率高，因此对系统的可靠性要求很高。

下面我们来看处理机故障．处理机故障分为两类：

(1) 悄然停(Fail-silent，或fail-stop)故障；

(2) 拜占庭(Byzantine)故障。

在悄然停故障中，出错的处理机停止运行，除了声明它停止运行外，不对后续输入产生反应也不产生输出.

而拜占庭故障则是，虽出故障，但仍继续运行，对于后续输入继续处理给出错误结果，给人一种仍在正常工作的假象。没有检测出的软件错误常常属于拜占庭故障。

通常处理拜占庭故障比处理悄然停故障更困难。

## 三、同步 / 异步系统

如果一个系统总能在有限时间内对消息做出反应，那么就称它为同步系统；反之，就称为异步系统。

异步系统比同步系统更复杂。如果一个处理机发出一条消息，并且知道在T秒内没有回答就意味着接收失败，它就可采取相应的正确行动；如果没有反应时间的限制，就很难判断是否发生故障。

## 四、冗余配置

常用的容错方法是冗余配置。它有信息冗余、时回冗余、物理冗余三种形式。

信息冗余就是，增加额外的信息位使错误信息可以得到纠正。例如海明码是存储器设计中的重要冗余配置技术，它可以用于检测和恢复传输错误。

时间冗余就是，执行一个操作，如果需要就再次执行。前面讲到的原子事务，就属于这种冗余．如果事务取消，它对系统无影响，可以再做。时间冗余对于解决暂时性故障和间歇性故障非常有效。

物理冗余就是增加额外的设备使系统可以承受某个部件的故障。例如给系统增加额外处理机，如果某台处理机出错，系统可以马上切换到正常的处理机上继续执行。

组织额外处理机有两种方法，一种是活动备份(ActiveReplicate)法，一种是主副(Primaryand Backup)结构法，比如对一个服务器，如果使用活动备份法，则所有的处理机都象服务器那样同时并行工作，来达到屏蔽故障。主副结构法则是使用一台处理机做工作，当它出现故障时，再用备份机来替代它.

采用哪种办法，主要取决于应用对于以下几点的基本需求：

(1)所需的备份(冗余)程度；

(2)有故障时的平均和最坏性能；

(3)无故障时的平均和最坏性能。

## 7.3.2 活动备份技术

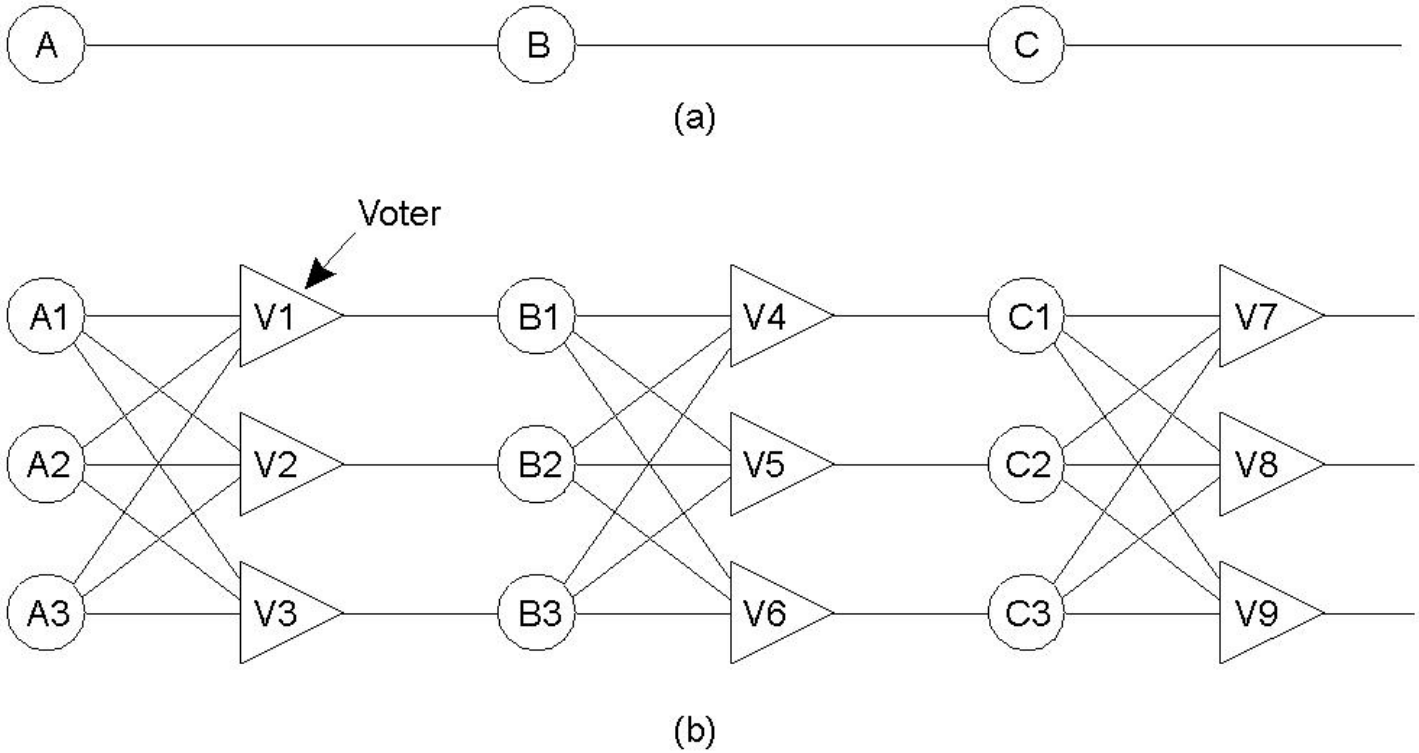
活动备份是一种典型的技术，它的特点就是使用物理冗余．人类生活和工程中许多都用到了这种技术，如哺乳动物有两只眼、两个耳朵、两个肺等；飞机一般要用多个发动机(波音747则有4台发动机)；体育比赛中要有多个裁判等等。

在电子线路中也经常采用活动备份技术。

典型设计称TMR(三模件冗余)技术。

冗余能够屏蔽故障．但需要多少个副本才合适呢?这主要依赖于应用要求的容错能力。如果一个系统被称为K级容错，那就意味着它可以容忍K个同样部件的错误。如果K个处理机出现的是悄然停故障。那么用K+1个处理机就可以实现K级容错。这样，在K个处理机停止工作后，还有一个处理机在运行。

# 模三冗余 TMR




(a) 原始电路

(b) 模三冗余电路：三是实施“少数服从多数” 原则的最少选举人数。

如果处理机出现拜占庭故障，那么就必须使用至少2K+1个相同的处理机来处理。这样，在最坏情况下，即K个处理机给出了错误结果，但还有K+1个处理机会给出正确答案，仍可屏蔽错误。

在实际应用中，由于环境的复杂性，很难保证只有K个处理机出问题，另外K+1个不出问题。因此，在容错系统的设计中，还需要依据统计分析结果来设置冗余备份的数量。

总之，活动备份方法并不是非常困难的问题。

## 7.3.3 主副容错技术

这个方法的基本思想就是，在任一时刻，有一台机器是主服务器，完成所有的工作。一旦这个主服务器出现故障，那么副服务器就接替工作。理想情况下，切换应当干脆利落，并且只能被客户机操作系统感知，对应用程序是透明的。这个方法同活动备份一样，也是源自日常工作和生活中的常用方法。如国家设副主席、政府设副总理、一般单位都设副职，在汽车上也都配有备用轮胎等。

主副容错方法比活动备份有两大优点。首先是简单，因为在正常操作中，消息仅仅送到主服务器而不是整个服务器组；第二，实际用的机器也较少，任意时刻只需一台副服务器备份即可。当一个副服务器成为主服务器时，马上要加入一台新的副服务器。它的缺点是一般很难处理拜占庭故障，另外在主服务器故障后需要很复杂并且时间较长的恢复过程，同时如何选择切换时间也是需解决的问题

## 7.3.4 容错系统的协调

在许多分布式系统中，经常需要让进程就某事件达成一致的协议。前面讨论的分布事务就涉及到许多这种协调活动，例如选举管理员，决定是否提交一个事务等等。当通信系统和处理机郡正常时；达成一致协议是很简单的。但若认为处理机和通信设施都可能出故障，要达成一致协议就麻烦多了。

分布式协调算法的总目标是让所有无错处理机在有限时间和操作步内，就某个问题达成一致协议。根据不同的系统情况会要求不同的处理办法。

(1) 消息是否可靠传输?

(2) 进程是否会发生故障?如果出现故障，是悄然停故障?还是拜占庭故障?

(3) 系统是同步的还是异步的?

算法证明：在一个有M个故障处理机的系统中，仅能在还有2M+1个正常工作的处理机时，才能达成一致协议。

而在一个异步的无传输延迟限制的分布式系统，即使只有一个处理出现故障也不能达成协议

# 7.4 实时分布式系统

## 7.4.1什么是实时系统?

实时系统根据时限要求的严格程度及漏掉一次处理所带来的后果分成软实时系统和硬实时系统。

所谓软实时系统就是指漏掉一个偶发事件处理不会影响系统正常工作。例如：电话交换机允许在超载情况下，丢失或接错一个电话。相反，硬实时系统不允许漏掉任何一个事件. 在实际当中，还有些系统处于两者之间，即如果有一事件没有在最后期限内得到处理，那就必须关闭当前活动，但却不产生致命的结果。另外，在某些实时系统中，有的子系统是硬实时的. 有些子系统是软实时的.

## 7.4.2 设计问题

实时系统有许多特有的问题，下面来讲一些重要的。

一、时钟同步(参见第三章)

二、事件触发和时间触发系统

事件触发是:如外界发生某事件,被传感器检测到,就发一个

中断信号，事件触发触发系统在系统负裁很重的情况下，效果不好。

在时间触发系统中就不存在上述问题，在时间触发的实时系统中，每隔△T发生一次时钟中断。在时间触发实时系统中，间隔时间△T的选择很重要。

三. 可预见性

行为预见性：即系统设计时应清楚所要满足的所有时间限制

因而实时系统是一个确定的系统，而不是随机系统。

四. 容错，可采用前面介绍的两种方法

五. 语言支持：通用的语言，也可用专用的语言

## 7.4.3 实时通信

实时系统中的通信与其它分布式系统的通信不同。由于对性能的要求很高，因此可预测性和确定性是其关键问题.

在分布式系统中实现可预测性就意味着处理机间的通信是可预测的。与以太网相反，令牌环网比较适合于实时系统。另一个是TDMA(分时多路复用)。通过时间槽的方法。

实时分布式系统的通信协议常是专用的.如TTP协议（时间触发协议）

它的特别之处：接收者能够发现信包的丢失、自动的成员关系协议、信包和全局状态的CRC效验码、及时钟同步的方法。

## 7.4.4 实时调度

实时调度算法需刻画的参数：

1）硬实时或软实时，硬实时要满足时间限制，软实时较松。

2）抢占式或非抢占式，如何让出CPU，

3）动态或静态，动态为执行中做出调度决策，静态事先计划。

典型动态算法有速度单调算法，即任务的优先级与它的执行频率有关。还有抢占式动态调度（最早期限优先算法）及最小松弛度算法

静态调度适合时间触发系统；动态调度适合事件触发系统

4）集中式或分散式，集中式一台决定，分散式处理机自己做出决定

## 7.4.5 实时系统的设计依据和主要措施

按照实时系统的要求，时限是基本的设计依据。我们必须从应用要求来分析和选择系统的硬件平台和软件平台，特别是前面已经说明，由于系统的日趋复杂化，再利用汇编语言来编写实时系统已经是不可能的。假设已经选择了一个硬件平台，我们要从以下几个方面分析系统对于实时性的支持。

## 一、实时响应

实时应用需要对外部事件有可预测的响应时间，如设备中断等。典型的实时应用包括三部分：中断产生设备、从设备上采集数据的中断服务程序和处理数据的用户级代码。

实时响应其实是反映应用系统与操作系统如何高速地共同工作来快速处理外部事件的一个指标。

在系统中一般有两类延迟：

(1) 中断服务程序(1SR) 延迟

该延迟是指从中断产生到中断服务程序第一条指令执行所经过的时间。

(2)进程调用延迟(PDL,即ProcessDispatchLatency)

进程调用延时是指从中断产生到等待进程开始执行的时间。包括以下几个时间段：

① 中断服务程序延迟；

② 中断服务程序执行时间；

③ 从中断服务程序返回的时间；

④ 等待中断进程的语景切换(ContextSwitch)时间

## 二、主要措施

(1) 多级流水(信息缓冲)

为了满足系统强实时要求，单靠串行程序的一次处理是很难完成任务的，可以采用多进程调度，进程之间采用信息缓冲的技术，使一次服务在总时间较长的情况下，能够满足及时处理高速实时输人信息的处理。

(2) 内存锁定，提供高速的数据访问

为了满足实时性要求，实时处理所需的数据一般不能访问磁盘，因此，可将常用数据全部锁定在内存中，这样保证所需要的数据访问均能在确定的时间(内存访问时间)内完成，进而保证系统的实时性。

(3)优先级动态可调、抢占式调度策略

在实时系统的设计中，应当将系统中的进程根据实时性要求，确定其静态优先级，保证具有高优先级的进程具有可抢占特性。同时，为保证优先级较低的进程能够运行，当其请求达到一定时限时，可以动态提高其优先级，以便及时完成这类服务。

(4)CPU的特定调度

为了保证实时性要求，特别是强实时的请求，可以在分布式系统中利用操作系统提供的支持，将某个确定进程分配在特定处理机上运行，从而保证在时限内完成相应服务。

# 7.5 分布式多媒体系统

## 7.5.1 简介

现代计算机可以处理像数字音频和数字视频数据这样连续的、基于时间的数据流。其处理能力导致了分布式多媒体应用程序的发展，如网络视频库、因特网电话和视频会议；这些应用程序能在当前网络和系统上运行，但它们的音频和视频质量常难以令人满足。许多像大范围的视频会议、数字电视产品、交耳式的电视以及视频监视系统，这样对实时数据要求很高的应用程序需要分布式系统技术所实现。

多媒体应用程序需要在有限时间内将多媒体数据流传输到客户端。音频和视频数据流被实时地生成和消耗，同时应用程序完整性的实质是实时地传输数据元素(音频采样，视频帧)，简单说，多媒体系统是实时系统

分布式多媒体系统它必须按照外部决定的调度方案执行任务和传输结果。底层系统达到这些要求的程度便是应用程序拥有的服务质量(QoS)。

实时系统所执行任务的特征和多媒体应用程序的特征不同。前者通常处理相对小的数据量相对少的硬时间限制，但是如果超过了时间限制，就会导致严重的甚至是灾难性的结果。 这种情况下，解决办法是充分估计所需要的资源并为其指定固定的调度计划，这样可以保证在最坏的情况下满足其要。

为了满足多媒体和其他应用程序的需要而进行的有计划的资源分配和资源调度，这被称为服务质量管理。

大多数当前的操作系统和网络并没有包含支持多媒体应用程序所需要的 QoS管理设施。

## 7.5.2 服务质量管理

当多媒体应用程序运行在个人计算机网络上时，它与运行着应用程序的工作站(处理器周期、主线周期、缓冲区容量)和网络(物理传输连接、开关、网关)竞争资源。工作站和网络可能必须同时支持多个多媒体程序和传统应用程序。

在多媒体和传统应用程序间就有竞争，在不同的多媒体应用程序之间甚至在单个应用程序的数据流之间都可能有竞争。

在多任务操作系统和共享网络中，物理资源都是可以被并发使用的。在多任务的操作系统中，中央处理器在每一时刻只处理一个任务(或进程)，一个轮转或其他调度方法的调度程序负责在当前竞争处理器资源的任务中选出一个，并调度它到处理器上运行。

网络是被设计用来使不同来源的信息进行交流的，它允许多个虚拟通道存在于同一个物理通道上。以太网这一主要的局域网技术以最优的方式来管理共享的传输介质。当通道上是平静时，任何结点都可以使用这一通道。但是这样可能会发生信息包冲突，当发生冲突时，结点会等待随机的一段时间，然而重发包，这样可以防止冲突。当网络负载很重时，很容易发生包冲突，但是这一发送方案在这种情况发生时，不能提供关于带宽和延迟的任何保证。

其资源分配方案特点：当对资源的需求增加时，它们将资源更稀疏地分配给每个竞争资源的任务。共享处理器周期和网络带宽的轮转和其他方法都不能满足多媒体应用程序的需要。

显而易见，它们不能实时地处理和传输多媒体数据流。迟到的传输数据是没有价值的。为了实现实时传输，应用程序需要保证在需要的时候能得到必要的资源，为了提供这一保障而进行的资源管理和分配被称为服务质量管理。

1）服务质量协商

为了在应用程序和它底层的系统之间进行QoS协商，应用程序必须向QoS管理指定自己的QoS需求，它是通过传递一个参数集实现的。当处理和传输多媒体数据时，有3个参数非常重要，它们是：

带宽、延迟和丢失率

为数据流设定QoS：

如摄像输出流需要带宽50Mbps,延时150ms,丢失率在帧10中少于1帧

流量调整：流量调整是用来描述使用输出缓冲来使数据元素流平滑这一方法的术语。

多媒体数据流的带宽参数通常给出发生在数据流传输时对实际传输模式的理想化近似。实际的传输模式越接近这一描述，系统就能越好地处理传输流量，特别是在系统使用为周期性请求设计的调度方法时，这一特点就会越显著.

**协商过程：** 对分布式多媒体应用程序，一个数据流的组件可能位于多个结点上。在每个结点上有一个QoS管理器。直接的QoS协商办法是从源端到目的端一直跟随着数据流。源端组件通过向本地QoS管理器发送一个流规范来启动协议过程。

这个QoS管理器可以检查数据库中记录的可用资源并决定所请求的QoS是否能满足。如果应用程序涉及到其他系统，流规范被传送到下一需要资源的结点。这一流规范传输过所有的结点，直到它最终到达目的端，然后系统可得出此QoS请求是否能满足的结论，并将该信息传输回源端。这种简单的协商方法可满足多种目的，但它没有考虑到在不同结点上的并发QoS协商之间可能会发生冲突。为了彻底解决问题，需要一个分布事务式的QoS协商过程。

### 2）许可控制

许可控制管理对资源的访问，以避免资源过载，并防止资源接收不可能实现的请求。它涉及关掉那些与当前的QoS保证冲突的资源清求。

一个许可控制方案是基于整个系统容量和每个应用程序产生的负载这两方面的知识的。一个应用程序的带宽需求规范可能是应用程序需要的最大带宽、保证其运行的最小带宽，或者是它们之间的平均值。相应地，许可控制方案可以基于这些值之一进行资源分配。

如果所有的资源只由一个分配器控制，那么许可控制是直接的。如果资源分布在各个结点上，例如许多局域网环境，其可以使用一个集中式的访问控制，也可以使用一个分布式的许可控制算法，由它避免并发许可控制的冲突。工作站的总线仲裁算法属于这一类；然而执行带宽分配的多媒体系统并不控制总线许可，因为总线带宽并不在内。

带宽预留 保证多媒体数据流某QoS级别的普通方法是预留一部分的资源带宽以便由它独占使用。为了在任一时刻实现数据流的需求，需要为它预留最大带宽。这是提供给应用程序有保障QoS惟一可能的方法 。

**统计的多路技术** 因为系统中可能存在潜在的未被利用的资源，这在超额预留资源的情况下常发生。而一些保证技术可提供使用这些资源的一些可能性，这些保证通常被称为统计保证或软保证，它与前面介绍的硬保证技术不同。

因为不考虑最坏的情况，统计性保证技术可以提供更高的资源利用率。但是如果仅仅只依据最小或平均需求来分配资源，那么短期的负载高峰可能会导致服务质量的下降；应用程序必须能应付这样的服务质量降低。

统计的多路技术是基于这样一个假设：对大量数据流来说，虽然单个的数据流可能会发生变化，但这些数据流需要的总带宽相对稳定。它假设当一个数据流发送大量的数据时，就有可能有另一个数据流发送的数据量较小，这样总带宽需求保存平衡。当然这些数据流之间应该是没有联系的。

## 7.5.3 资源管理

为了向应用程序提供一定等级的QoS服务，系统不仅需要充分的资源(执行)，还需要在应用程序需要时有能力将这些资源提供给程序使用(调度)。

**资源调度**

系统需要根据进程的优先级来为其分配资源。资源调度器根据特定的标准来决定进程的优先级。在传统的分时系统中，CPU调度进程基于程序的响应时间以及公平原则来指定优先级：I／O量大的进程会获得高优先级，这样可以保证对用户做出快速响应，与CPU联系紧密的任务获得低优先级，并且系统平等对待同一优先级的进程。

多媒体系统也可以使用这一标准，但是传输单个多媒体数据元素的时间限制改变了调度问题的特性。为解决这一问题，系统可以使用实时调度算法。因为多媒体系统必须处理离散的和连续的媒体，因此在不引起离散媒体访问和其他交互应用程序饥饿的情况下，可以为实时性的数据流提供充分的服务。

调度算法必须管理(或协同)影响多媒体应用程序的所有资源。在通常的情况下，系统从磁盘上读取多媒体数据流，并将其通过网络传输到目的机器，在目的机器上，该数据流和其他来源的数据流同步合成起来，并最终显示。在这个例子中，系统需要的资源包括磁盘、网络、CPU以及内存和总线。

1）公平调度 如果有多个数据流竞争同一资源，系统必须考虑到公平性，防止不正常的数据流占用过多的带宽。保证公平性的一个简单方法是对同一优先级的数据流使用轮转调度方法，称为公平排队。

2）实时调度 人们已经开发出来一些实时调度算法来满足应用程序如：航空工业过程控制的CPU调度需要。假设CPU资源并没有被过度分配(这是QoS管理器的任务)，调度算法将CPU时间片以某种方式分配给多个进程，而这种方式必须使进程能及时地完成任务。

传统的实时调度算法十分适合规则的连续多媒体数据流模型。最早时间限制优先(EDF)调度算法几乎是这些方法的同义词。一个EDF调度器根据每个工作项的时间限制来决定下一个要处理的工作项：具有最早时间限制的工作项优先处理。在多媒体应用程序中，EDF调度策略被证明在基于时序标准分配单个资源方面是最优的。

## 7.5.4 实例研究：Tiger视频文件服务器

提供多个并发实时视频数据流的视频存储系统被看作为支持面向消费者的多媒体应用程序的一个重要的系统组件。人们已经开发了多个这种类型的程序原型，并且其中的一些已经形成了产品，Tiger视频文件服务器（微软研究院）

系统的主要设计目标如下：

1. 适用于大量用户的视频点播　应用程序是向点播的用户提供电影的服务器。系统从大容量的数据电影库中选择电影。客户应在发送点播请求的数秒钟内就能获得电影图像的第一个帧，并且他还应该能随心所欲地执行暂停、回退和快进操作。尽管库中电影的数目很大，但是可能有一些电影是很受欢迎的，它们可能同时被多个客户不同步的访问，这就导致可能同时播放它们，但是播放的时间进度不同。

2.服务质量 视频数据流的 传输速率应保持稳定，其中客户端可用的缓冲区大小决定了系统能处理的最大的抖动，并且视频数据流还应保持低丢失率

3.可伸缩性和分布性 目的是以一种可伸缩的体系结构来设计系统，使它(通过增加计算机可以同时支持10000个客户。

4 低成本硬件 系统是由低价的硬件构

5. 容错性 在单个服务器计算机或者是磁盘驱动器发送故障时，系统可以继续运行并且执行

Tiger视频文件服务器设计核心是分布式调度计算机的工作负载。

# 第八章 计算机支持的协同工作

南京理工大学计算机学院

CSCW技术的应用领域非常广泛。例如，远程专家会诊，可以利用CSCW技术，求得远方专家的帮助；合作设计，可以利用CSCW技术在异地进行合作工程设计，而不必真正坐在一起，CIMS中的并发工程便属于这类应用；合作编著，多个作者可以在不同地点、不同时间共同编辑和编著同一著作，文件、图书和报刊等。远程会议更是目前最易接受的CSCW技术，节省时间。

CSCW技术在军事应用中，包括战时的协指挥和平时的协同训练等。

# 8.1概述

## 8.1.1 CSCW简介

CSCW(Computer Supported Cooperative Work)一词最初是在1984年提出,于1986年在美国召开的第一次国际CSCW会议上正式使用，它的含义是计算机支持的协同(合作)工作。从此，CSCW的研究发展非常之快，CSCW国际会议每两年举行一次，欧洲每两年也召开一次欧洲CSCW会议.

1989年，欧共体设立了COST工程计划．其中一项重要工程，专门用于支持合作系统的研究．该工程的目标分为三类：

整体科学目标;特定目标和政治目标

其科学目标为：

①为CT(合作技术，CooperationTechnology)的强化训练创建理论基础

②为科学团体参加CT的研究建立公共框架

其特定目标为：

①改进入机界面，改进总体工作条件，支持CT系统的终端用户；

②为决策人员和计划人员提供对于合作技术的基本理解；

③通过研究成果影响合作技术的标准化组织；

④通过适当的媒体，介绍科学和技术成果，使COST的努力为国际所注目。

其政治目标是使欧洲成为合作技术研究的领袖。

在美国，虽然没有相应的计划，但其研究和资助强度一点也不逊色，主要由各大商家和著名大学立项研究．由此，也可以看出美国和欧洲研究方式的不同，在美国是直接利益驱动，在欧洲则是技术驱动。

在我国，关于CSCW的研究是从多媒体领域展开的，最初大家讨论的热点是分布式多媒体技术，关注这一领域研究的文章最初发表在我国第一届多媒体技术研讨会(1992年，北京)上，到了第二(1993，杭州)，三(1994，上海)、四(1995，广州)届多媒体研讨会，文章便不断增多，表现出我国研究人员对于该领域研究的热情。主要研究单位有清华大学、国防科技大学、华中理工大学，南京大学等。其中，清华大学推出了会议系统和合著系统的原型，在会议和合著系统的研究中重点突出了合作机制的研究。

综合世界各国的相关研究情况，CSCW的主要需求有群体工作需求的分析、多用户界面、分布式结构、合作规则组织模型、群体协调理论租模型等方面。CSCW中的研究问题主要包括：合作认知模型的研究；合作控制机制的研究；合作通讯；合作软件支持工具和环境的研究。

自CSCW提出以来，人们在以下几个主要领域进行了广泛研究，并取得了成绩。

(1)消息系统(Messagesystem)，是电子邮件的后代，可以让用户通过中心机发送文字消息给其它用户。广域网的存在使电子邮件所完成的功能更为广泛。

电子邮件的发展导致了消息处理系统(MessageHandlingSystem)模型的出现，这一模型已经为CCITT的X．400系列标准所采用。每个消息系统都使用特定的消息格式来传输信息，结构化消息系统的主要原理是通过扩充现有消息格式来扩充计算机可处理的信息。

(2)计算机会议(ComputerConferencing)，本身也是由电子邮件：发展而来，但它主要关心的问题是如何组织消息，而不是如何传输消息。一个典型的计算机会议系统包括一组叫做会议的群体，每一个群体拥有——组成员和一组消息。会议的安排主要由成员各自选定一个论题展开讨论。申请参加会议的用户应当对会议中的论题感兴趣。通常情况下，系统存储的信息均为会议成员所共事和访问。这些信息通常存储在中心数据库的会议消息中，而不是在个人的邮箱中。现有的原型系统有Notepad、COM、潘多拉系统等。

高速可靠的通讯技术的发展使实时的计算机会议得以实现，如RCAL，可以允许与会人员之间实时通讯。另外，先进的工作站已经可以支持桌面会议系统，这类系统充分利用了工作站环境中的共享窗口来处理实时的会议需求。现已推出了一些多媒体会议系统，它们都集成了包括音频、趾频和文本等媒体信息的支持。

(3)合著系统，此类系统用于支持和表达参于群体工作成员之间的协商和讨论。合著系统便是具有明确合作成果的系统，即成果是反映所有参与人员意见的一个文档。

其中，合著系统在欧美都有专门的立项研究。在美国，由卡内基梅隆大学承担的合著系统研究项目PREP，三年期限，总投资达95万美元；在欧洲，则有专门的研究队伍，从社会学、心理学和计算机科学几个方面着手研究.

总的来看，CSCW系统及应用的研究目前正在努力解决下列问题：

第一，适合于人类自然合作方式的计算机支持；

第二，对实时性群体交互的支持；

第三，多媒体合作的支持；

第四，对于多媒体信息的高效传输提供支持。

## 8.1.2 CSCW系统的功能和特点

CSCW系统的主要功能有如下五点：

(1)通讯：是指信息的共享与传送；至少涉及两种不同的进程一千逍息发送进程和信息共享进程。

(2)任务调度:是指任务执行临时序列的确定。这取决于许多因素，如时限、任务的预计时间、对其它任务提出的要求、人员及资源的可用性等等.

(3)合作角色及责任的分配：包括指定合适的人员，确定他们所起的作用。通过定义人员，确定人员对于这一角色的适应性，保持责任的可跟踪性是支持这一功能的基本方法；

**(4)资源分配：**在任务执行过程中，使合适的资源可用。

**(5)进展跟踪：**包括监控任务的执存状态，以确认一项活动是否正在按计划进行；

CSCW系统具有如下特点：

(1)开放性：合作中没有获得希望结果的固定方法，不同人、不同组处理任务的方式是不同的。

(2)边界开放性：合作中没有确定的结束标记或结束点。

(3)异步性：合作中不同人具有自己不同的操作序列，即使同一工作组的不同个人也具有自己独立的行为。 ’

(4)信息共享性：这是合作的基础、合作的结果。

(5)自动化支持：这是CSCW系统不断追求的目标。

## 8.1.3 CSCW系统的基本需求

需解决的四个问题：

(1) 群体合作策略和规范；

(2) 计算机通信

(3) 多用户界面；

(4) 共享的多媒体信息服务。

其中每个问题都有许多尚待解决的技术问题如：

**1)存储与处理：**多媒体信息由不同类型的复合对象构成，如文字、图形／图像、声音、音频和视频等。每类信息都需要单独的获取、处理、传输和存储的工具和技术。再进一步，这些信息应当形成一定的语义和时态联系，并按照统一的表示方式进行管理(包括存储和检索)，即多媒体文档。因此，对于多媒体文档的有效支持是多媒体合作环境中的第一需求。

**2)功能表现：**不同的应用可能根据自己的需求有自己的功能要求，但在合作环境中我们可以发现，有许多功能属于任何系统都应具有的，这些功能是任何合作环境都应支持的

**3)合作工作的需求：**合作意味着用户间通讯和协调。因此，合作规则是合作中的第一要求，任何合作者都要遵守这一规则，否则合作将无法进行；第二，合作设施和合作协议也是必须的，否则合作者之间无法通讯和交流；第三，必须给所有合作者提供一个共同的视图；最后，合作者必须有一个共享的数据空间。

**4)通讯：**合作者之间的通讯实际上就是克服地理上的距离限制。因此，在合作系统中，网络通讯是必须的。要支持多媒体信息的全面且实时的通讯，必须支持高速传输，传输速率一般不应低于**100Mbps**。

## 8.1.4 CSCW研究中的几个问题

1）工作与利益的差异问题。

2）必需人数与“囚徒困境”问题。

3）破坏社会的正常秩序。

4）异常处理问题。

5）隐含访问问题。

6）评价的困难.

7）直觉的失败

8）适应性回题

## 8.1.5 CSCW与计算机体系结构的发展

就计算机领域的研究与发展来看，合作已经不是新名词。合作的概念早就已经与计算机系统结构结下了不解之缘。

(1)多个部件的合作．这是传统流水线计算机的一般结构，在这种结构中，采用多个部件的时间重叠(合作)，可以获得计算机系统的高性能已经得到了很好的验证。

(2)多个处理器的合作——多处理器(MP、MPP)结构。为了进一步获得计算机的高性能，除了开发细粒度的部件级并发(合作)以外，人们开始寻求更大粒度的合作，即处理机之间的合作。这种结构一般称为并行处理机或超并行处理机结构。

(3)多个计算机的合作——分布式系统结构。更大粒度的合作则表现为多种计算机之间的合作，这就是基于局域网的分布式系统，在这种体系结构中，分布、异构和合作是其主要特点。目前，市场上流行的客户／服务器计算机结构就是一种特殊的分布式计算机体系结构，它是将异构型计算机的功能、性能及责任进行必须的预划分，然后形成特定的分布计算机系统。

**(4)人的合作——CSCW系统。**当技术发展到一定程度，在较低一级粒度下的合作与并因此，CSCW的诞生和发展是与计算机体系结构的发展分不开的，同时，CSCW也是人类社会对于计算机提出更高要求所驱动形成的一个技术领域。

## 8.1.6 CSCW是一种环境仿真技术

人的合作就是指两个或两个以上的人协商、通讯，以完成同一种工作。合作本身就是一个复杂问题。

人与人的合作必须靠环境支持，而环境的主要部分是信息媒体的载体.

由此可见，人们的目标就是在逐步扩充的范围内，充分使人们利用人造的媒体载体，通过自然媒体进行通讯和交流，克服自然距离带来的不便。

为了不断支持人们跨越更大距离的合作与交流，从计算机领域来看，人们开发的先进技术主要表现在以下几个方面。

第一，网络的出现以及联网的工作站形成了新的信息处理和存储能力，提供了设计更广泛模拟人类合作环境的可能.

第二，分布式系统中的资源共享技术，特别是局域网和广域网技术的进一步发展使地理上分散的用户直接交互的可能性进一步增强；

第三，多媒体信息处理和管理技术的进一步成熟，更容易模拟人类的自然合作环境：

第四，高速网络技术的发展将进一步克服当前网络传输速率的限制，进一步提供各类媒体信息的实时传输，这是模拟人类自然合作环境必备的技术。

因此，CSCW技术便是为了追求更高层次、不受距离限制的、全方位利用各种信息媒体的合作而研究的技术。也就是说，CSCW研制的就是这样一个环境，无论人们的距离有多远，都能利用该种环境进行类似于自然环境支持的合作，它是一种模拟人类自然合作环境的虚拟环境。CSCW的研究意义在于，使世界变大(一个用户通过这些系统的支持可以涉猎到更加广泛的信息和知识)，又使世界变小(多个异地作者可以在同一虚拟桌面上讨论、合作处理同一信息等)。

## 8.1.7 CSCW与分布式系统的关系及异同

分布式系统通过网络来支持多个计算机系统合作完成同一项工作。但这里的合作是指不同计算机之间的合作，而非人之间的合作，CSCW要支持的是人之间的合作，这两种合作主之间存在着天然的联系

第一，完全自治的分布式系统：支持电子邮件，从而支持入之间异步的信息传输工作，能够部分完成对于犬之间合作的支持；

第二，资源共享系统：允许不同计算机的用户共享同样的系统资源。这也是CSCW系统中的必备技术；

第三，分布式操作系统：该种操作系统将分布的系统资源作为整个环境的资源来管理，提供用户透明的计算机资源管理，对于CSCW系统的支持更为直接。

总之，分布处理是CSCW系统的基础支持技术。

可以这样说，CSCW的实现基础是分布处理技术，CSCW的研究也将进一步促进分布处理技术的发展。

在分布处理技术还未提供足够的支持时，CSCW系统的研究人员还必须自行研制具有特殊要求的相关分布处理技术。

CSCW与分布式系统的主要区别：

第一，分布式系统追求透明性，即让用户在使用分布式系统时，感觉好像独占该系统的所有资源：而CSCW系统则追求非透明性，即系统尽可能地让所有用户相互察觉到大家所有的操作。

第二，分布式系统主要研究如何让系统中各个计算机充分发挥各自的优势，从而获得整个分布式计算机系统的高性能；而CSCW系统除了要求获得整个计算机系统的高性能外，更重要地是更好地支持计算机用户间的高效合作，从而获得人—机系统整体的高性能，所以，其研究分支中就有GDSS(群决策支持系统)，该项研究期望通过合作支持系统使人们通过计算机的合作效益超过自然方式下的合作效益。