



# 第8章 分布式文件系统

---



## 第8章 分布式文件系统

---

- 简介
- 文件服务体系结构
- SUN网络文件系统(NFS)
- Andrew文件系统(AFS)
- DFS进展
- 小结



# 本地文件系统

---

- 文件系统提供文件的管理
  - 命名空间
  - 文件操作的API
    - create, delete, open, close, read, write, append ...
  - 物理的存储管理和分配
    - 块存储
  - 安全保护
    - 访问控制
- 层次化的命名空间
  - 目录和文件
- 文件系统已被安装
  - 不同的文件系统可以在同一个命名空间中



# 文件系统模块

---

目录模块:	将文件名与文件 <b>ID</b> 相关联
文件模块:	将文件 <b>ID</b> 与物理文件相关联
权限控制模块:	检查操作请求是否合法
文件存取模块:	读/写数据或者属性
块模块:	存取和分配磁盘块
设备模块:	磁盘 <b>I/O</b> 及缓冲





# 传统分布式文件系统

---

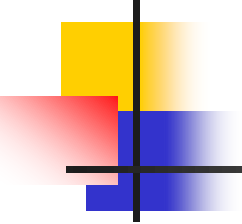
- 目的：模拟本地文件系统的行为
  - 文件没有被复制
  - 没有严格的性能保证
- 但是
  - 文件位于远程的服务器上
  - 多个客户可以访问服务器
- 为什么？
  - 用户有多台计算机
  - 数据被多个用户共享
  - 统一的数据管理（企业）



# 分布式文件系统的需求

---

- **透明性：** 分布式文件系统应该如同本地文件系统
  - **访问透明性：** 客户无需了解文件分布性，通过一组文件操作访问本地/远程文件
    - 需要支持相同的一组文件操作，程序要如同本地文件系统一样在分布式文件系统中工作
  - **位置透明性：** 客户使用单一的文件命名空间。路径不变，多个文件应该可重定位
    - 所有用户看到同样的命名空间
  - **移动透明性：** 当文件移动时，客户的系统管理表不必修改
    - 如果文件移动到另一个服务器，不应该让用户可见

- 
- 
- **性能透明性：** 负载在一个特定范围内变化时，性能可接受
    - 系统提供合理的一致性能
  - **伸缩透明性：** 文件服务可扩充，以满足负载和网络规模增长的需要
    - 系统可以通过增加服务器逐步扩展

# 分布式文件系统需求（续）

- 并发的文件更新

- 并发控制，客户改变文件操作不影响其他客户

- 文件复制

- 分担文件服务负载，更好的性能和系统容错能力

- 硬件和操作系统的异构性

- 接口定义明确，在不同操作系统和计算机上实现同样服务

- 容错

- 为了处理瞬时通信故障，设计可以基于最多一次的调用语义（5.3.1节）
  - 幂等操作：支持最少一次语义，重复执行的效果与执行一次的相同
  - 无状态服务器：在崩溃后无恢复重启





# 分布式文件系统需求（续）

---

- 一致性

- 单个拷贝更新语义，多个进程并发访问或修改文件时，它们只看到仅有一个文件拷贝存在

- 安全性

- 客户请求需要认证，访问控制基于正确的用户身份

- 效率

- 性能满足一定要求



# 案例研究

---

- SUN网络文件系统
  - 第一个被设计为产品的文件服务[1984]
  - 成为因特网标准
  - 被大多数平台所支持，如Windows NT，Unix
- Andrew文件系统
  - CMU的校园信息共享系统[1986]
  - 在CMU有800个工作站和40台服务器[1991]



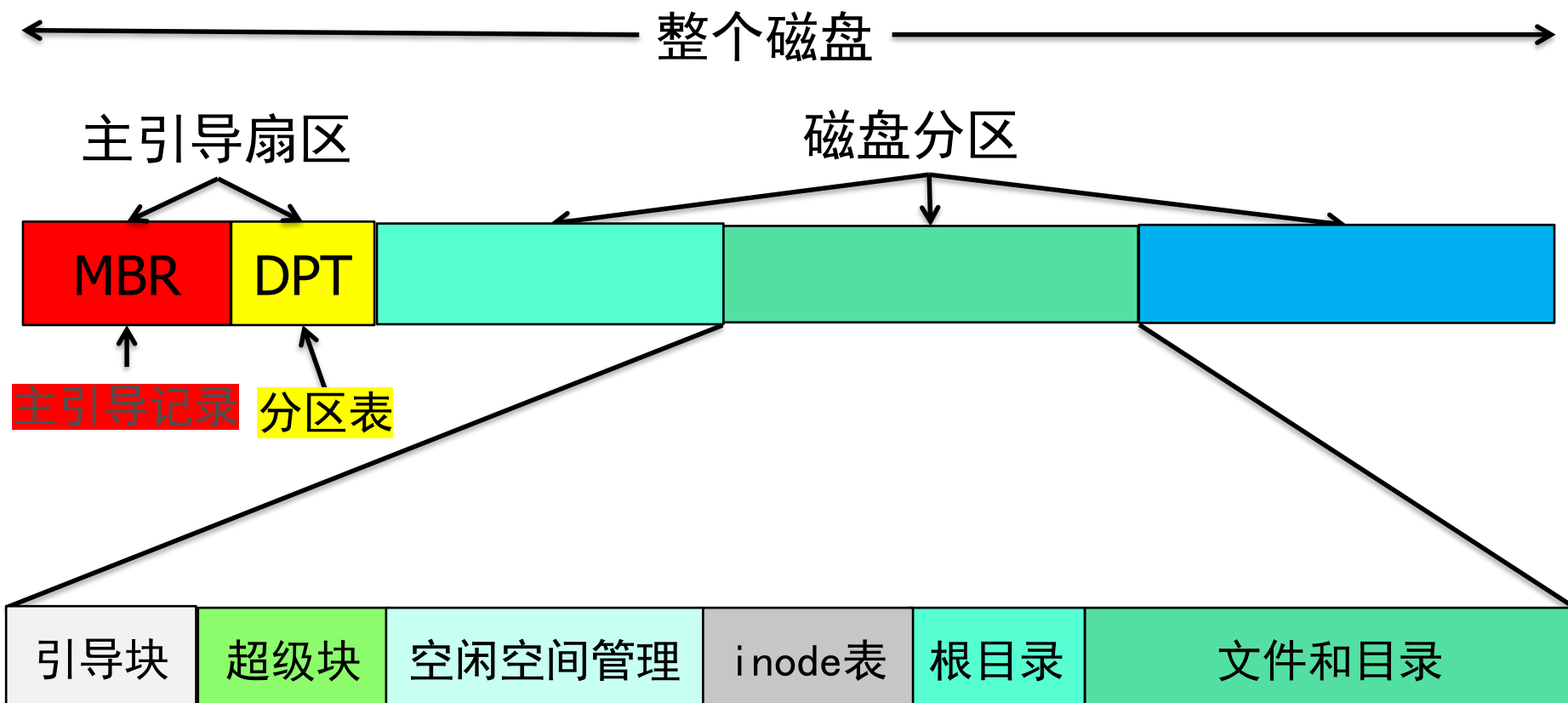
## 第8章 分布式文件系统

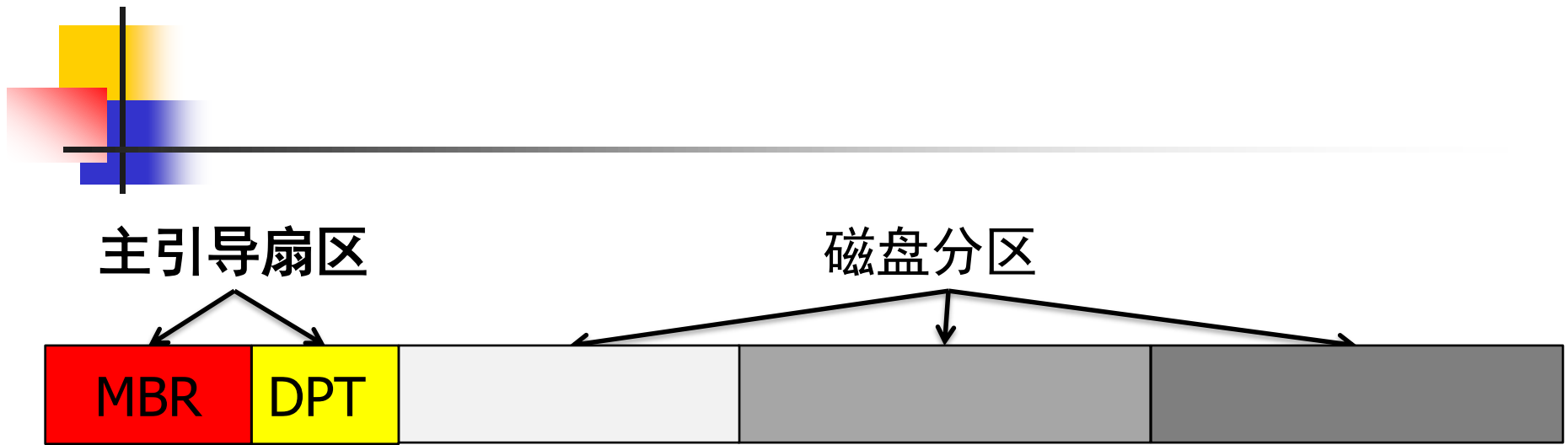
---

- 简介
- 文件服务体系结构
- SUN网络文件系统(NFS)
- Andrew文件系统(AFS)
- DFS进展
- 小结

# 第8章 分布式文件系统

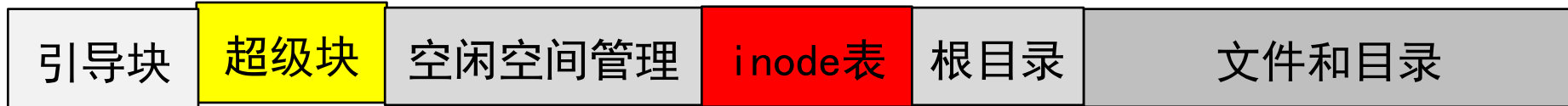
## ■ 文件系统布局





**主引导扇区**位于整个硬盘的0磁头0柱面1扇区，包括硬盘主引导记录**MBR** (Master Boot Record)、分区表**DPT**(Disk Partition Table)和签名两个字节55AA表示结束

主引导记录(MBR)负责检查分区表是否正确，并且引导操作系统调入内存执行。



**超级块**存储有关**文件系统**的  
大多数信息，其中包括：

- 文件系统的大小和状态
- 标号，包括文件系统名和卷名
- 文件系统逻辑块的大小
- 上次更新的日期和时间
- 柱面组的大小
- 柱面组中的数据块数
- 摘要数据块
- 最后一个安装点的路径名

**inode**包含**文件**的所有信息（文件  
名除外）1个inode为128个字节：

- 文件类型：
- 文件的模式（读-写-执行权限集）
- 指向文件的硬链接数
- 文件属主的用户 ID
- 文件所属的组 ID
- 文件中的字节数
- 包含 15 个 (0-14) 磁盘块号的数组  
(指向存储文件内容的数据块)
- 上次访问文件的日期和时间
- 上次修改文件的日期和时间
- 更改inode的日期和时间

0-11 **直接块**指向包含文件内容的block

- block大小为4KB，则仅可指向 $12 \times 4KB = 48KB$

12 **间接**指向一个block，该block包含的不是文件内容而是block号

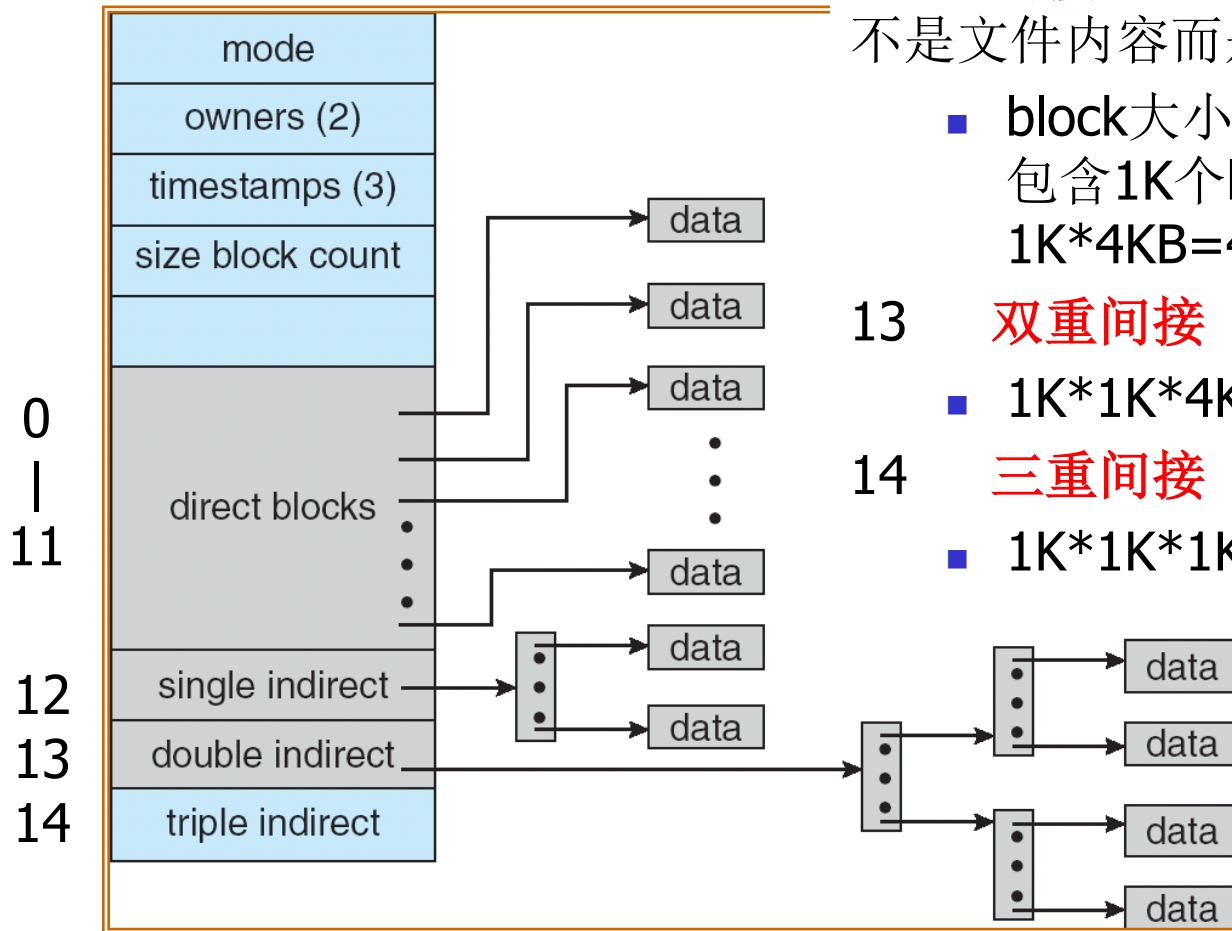
- block大小为4KB，block号大小为4B，包含1K个block号，可以指向 $1K \times 4KB = 4MB$

13 **双重间接**

- $1K \times 1K \times 4KB = 4GB$

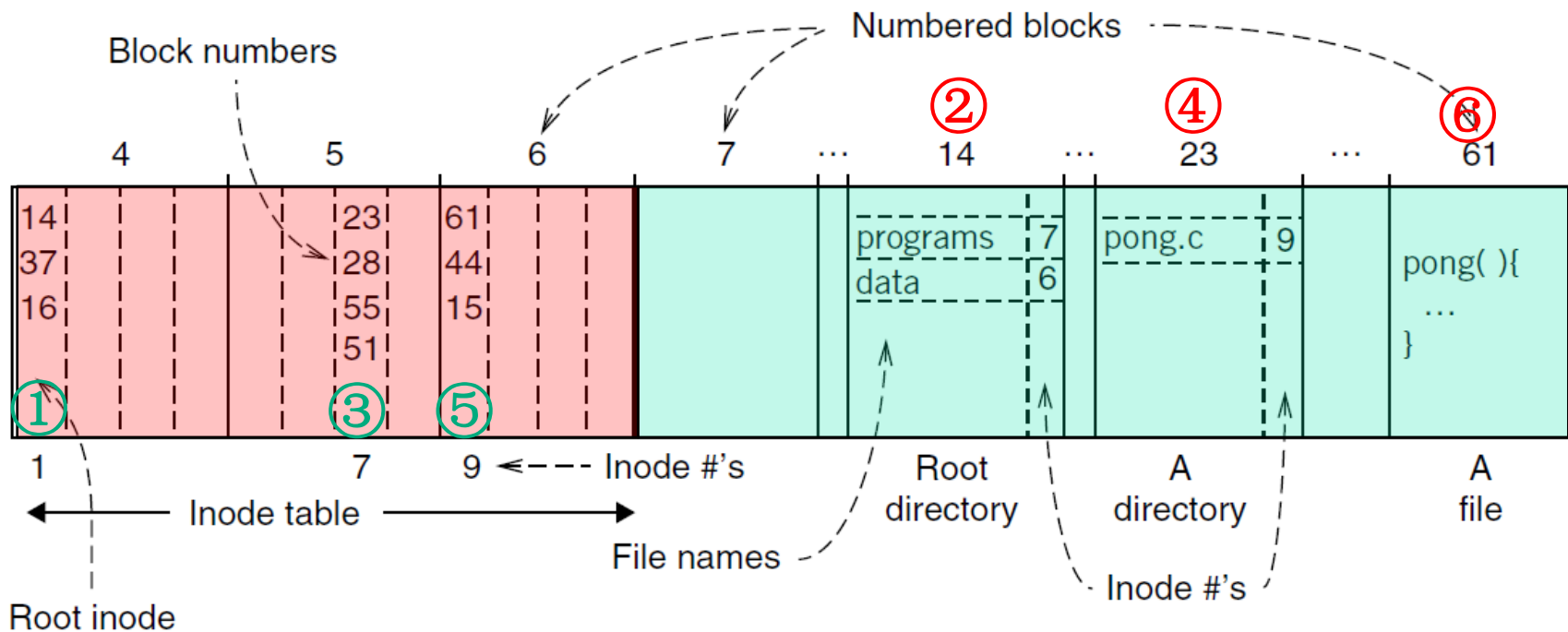
14 **三重间接**

- $1K \times 1K \times 1K \times 4KB = 4TB$



## 第8章 分布式文件系统

### ■ UNIX中的路径解析——/programs/pong.c







# 文件服务体系结构

---

## 文件服务的三个组件

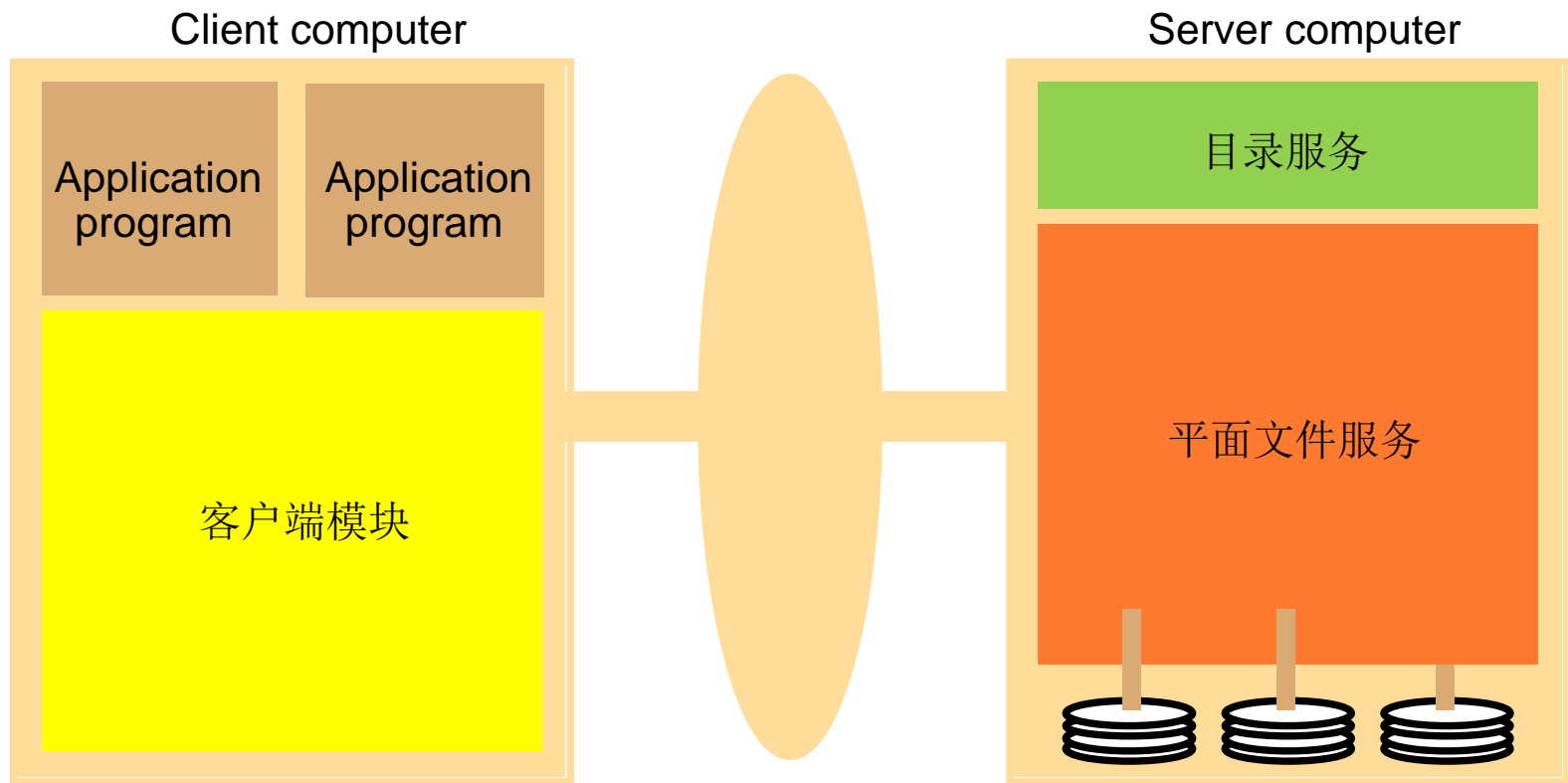
### 服务器

- 目录服务
- 平面文件服务

### 客户端

- 客户模块

# 文件服务体系结构





# 目录服务

---

- 元数据管理（元数据中包括修改、访问或创建文件时操作系统记录的时间）
- 创建或者更新目录（层次文件结构）
- 提供文件的文本名（Name）和平面文件结构中唯一文件标识（UFID）的映射



# 目录服务接口

## ■ 主要任务

### ■ Lookup操作执行Name→UFID的转换

*Lookup(Dir, Name) -> FileId*

— Throws *NotFound*

*AddName(Dir, Name, File)*

— Throws *NameDuplicate*

*UnName(Dir, Name)*

— Throws *NotFound*

*GetNames(Dir, Pattern) ->*  
*NameSeq*

Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.

If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file's attribute record.  
If *Name* is already in the directory: throws an exception.

If *Name* is in the directory, the entry containing *Name* is removed from the directory.  
If *Name* is not in the directory: throws an exception.

Returns all the text names in the directory that match the regular expression *Pattern*.



# 平面文件服务接口

---

- 平面文件服务操作和**UNIX**进行比较
  - 没有**open**和**close**操作，通过引用合适的**UFID**可以立即访问文件
  - **read**和**write**操作需要一个开始位置，**UNIX**操作中没有（读写指针指向当前位置开始操作）
- 与**UNIX**文件系统相比在容错方面的影响
  - 可重复性操作
    - 除了**create**，其它所有的操作都是**幂等的**
  - 无状态服务器
    - 如在文件上进行操作不需要读-写指针
    - 故障后重启无需客户或服务器恢复任何状态

read和write操作是最重要的文件操作

read和write操作都需要一个参数*i*来指定文件的读写位置

## 平面文件服务操作

<i>Read(FileId, i, n) -&gt; Data</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File})$ : Reads a sequence of up to $n$ items from a file starting at item $i$ and returns it in <i>Data</i> .
<i>Write(FileId, i, Data)</i> — throws <i>BadPosition</i>	If $1 \leq i \leq \text{Length}(\text{File})+1$ : Writes a sequence of <i>Data</i> to a file, starting at item $i$ , extending the file if necessary.
<i>Create() -&gt; FileId</i>	Creates a new file of length 0 and delivers a UFID for it.
<i>Delete(FileId)</i>	Removes the file from the file store.
<i>GetAttributes(FileId) -&gt; Attr</i>	Returns the file attributes for the file.
<i>SetAttributes(FileId, Attr)</i>	Sets the file attributes (only those attributes that are not shaded in ).



# 客户模块

---

- 支持应用程序对远程文件服务的透明存取
- 缓存最近使用的文件块提高性能



# 访问控制

---

- UNIX文件系统
  - 当用户进行open调用的时候系统将会核对其访问权限
- 无状态的分布式文件系统（DFS）
  - DFS的接口是对大众公开的，存在安全隐患；
  - 文件服务器不能保留用户标识（UID），否则服务就变成有状态了；
  - 实施访问控制的两个方法：
    - 基于权能的认证（访问能力列表，每个主体都附加一个该主体可访问的客体的明细表）
    - 将每个请求与UID关联起来，每次请求都进行访问检查
  - NFS和AFS使用Kerberos认证解决伪造用户标识的安全问题





# 层次文件系统

---

- 目录树

- 目录是一种特殊的文件
  - 包含有能通过它访问的文件名和目录名
- 路径名
  - 表示一个文件或者是目录
  - 多部分命名，如“/etc/rc.d/init.d/nfsd”

- 遍历目录

- 通过多次查找操作来解析路径名
- 客户端缓存目录

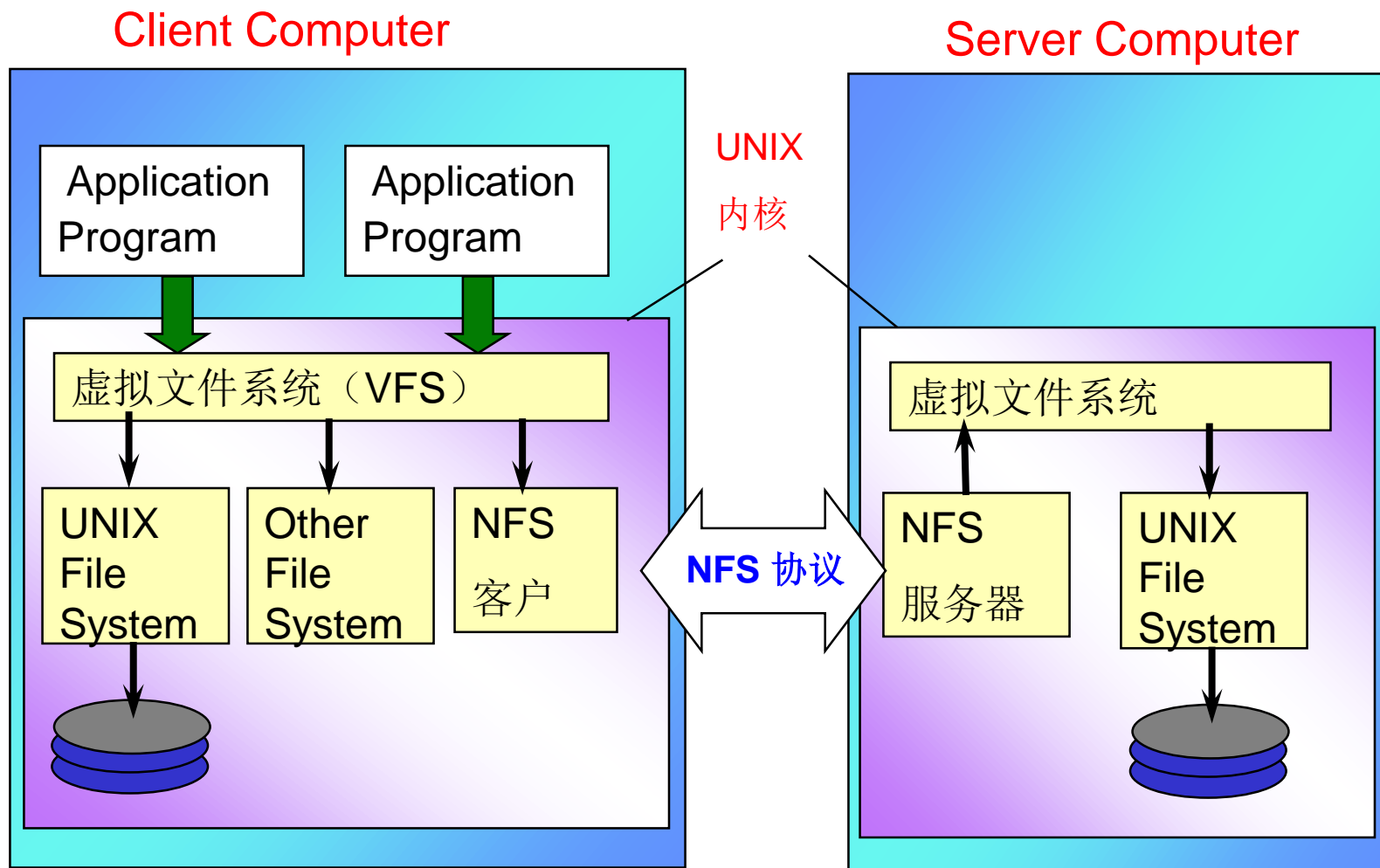


# 分布式文件系统

---

- 简介
- 文件服务体系结构
- **SUN网络文件系统(NFS)**
- Andrew文件系统(AFS)
- DFS进展
- 小结

# NFS体系结构





# 虚拟文件系统（VFS）

---

- 在**UNIX**内核的一个转换层
  - 使文件系统可以插入和共存
- 跟踪本地和远程可用的文件系统
- 将请求传递到适当的本地或远程文件系统
- 区分本地和远程文件

# 虚拟文件系统（续）

## ■ vnode

- 本地文件：引用一个inode
- 远程文件：引用一个文件句柄

## ■ 文件句柄

文件系统标识

文件的inode号

inode产生数

- 文件系统标识符
  - 服务器可能服务多个文件系统
- inode号
  - 在一个特定的文件系统内是唯一的
  - inode号是用于标识和定位文件的数值
- inode产生数
  - inode号在文件被删除后可被重用，每次inode被重用时inode产生数加1
  - 当inode被回收时（inode产生数变化），但文件仍然打开，抛出异常
- 客户与服务器之间传递文件句柄
  - 句柄对客户不透明



# 客户集成

---

- 将客户端集成到内核
  - 用户程序可以通过**UNIX**系统调用访问文件，不需要重新编译或者加载库
  - 一个客户模块通过使用一个共享缓存存储最近使用的文件块，为所有的用户级进程服务
  - 用于认证用户**ID**的密钥可以由内核保存，防止用户级客户冒用客户



# 访问控制和认证

---

- 与传统**UNIX**文件系统不同，**NFS**服务器是无状态的
- 在用户发出每一个新的文件请求时，服务器必须重新对比用户**ID**和文件访问许可属性，判断是否允许用户进行访问
  - 将用户**ID**绑定到每个请求
  - 在**NFS**中嵌入**Kerberos**认证
    - 在加载的同时认证客户
    - 凭据，认证和安全通道

# NFS服务器接口

<i>lookup(dirfh, name) -&gt; fh, attr</i>	Returns file handle and attributes for the file <i>name</i> in the directory <i>dirfh</i> .
<i>create(dirfh, name, attr) -&gt; newfh, attr</i>	Creates a new file <i>name</i> in directory <i>dirfh</i> with attributes <i>attr</i> and returns the new file handle and attributes.
<i>remove(dirfh, name) status</i>	Removes file <i>name</i> from directory <i>dirfh</i> .
<i>getattr(fh) -&gt; attr</i>	Returns file attributes of file <i>fh</i> . (Similar to the UNIX <i>stat</i> system call.)
<i>setattr(fh, attr) -&gt; attr</i>	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
<i>read(fh, offset, count) -&gt; attr, data</i>	Returns up to <i>count</i> bytes of data from a file starting at <i>offset</i> . Also returns the latest attributes of the file.
<i>write(fh, offset, count, data) -&gt; attr</i>	Writes <i>count</i> bytes of data to a file starting at <i>offset</i> . Returns the attributes of the file after the write has taken place.
<i>rename(dirfh, name, todirfh, toname) -&gt; status</i>	Changes the name of file <i>name</i> in directory <i>dirfh</i> to <i>toname</i> in directory <i>todirfh</i> .
<i>link(newdirfh, newname, dirfh, name) -&gt; status</i>	Creates an entry <i>newname</i> in the directory <i>newdirfh</i> which refers to file <i>name</i> in the directory <i>dirfh</i> .





# 安装服务

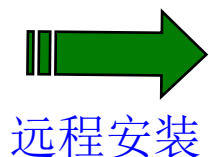
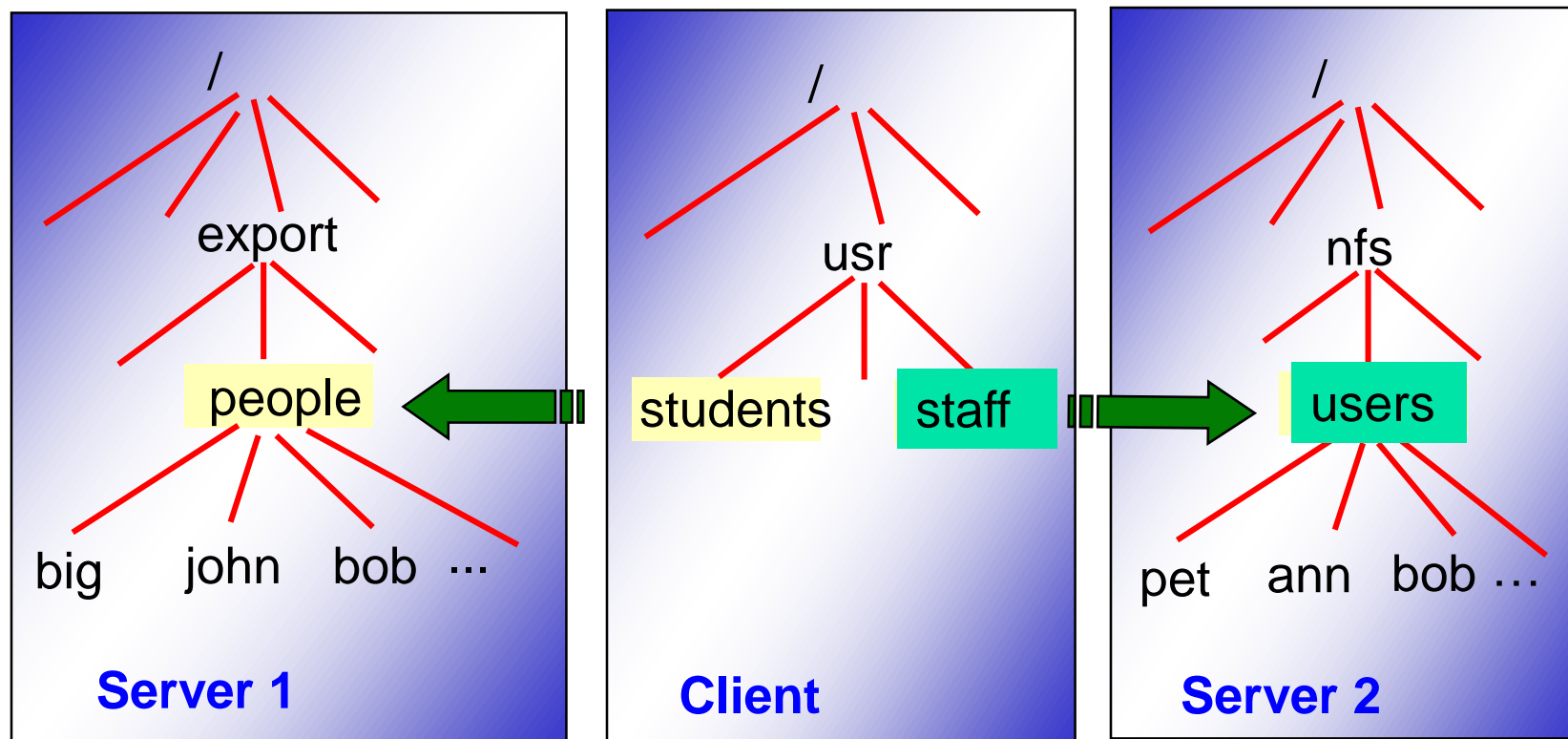
---

## ■ 服务器

- 每一个服务器上都有一个具有已知名字的文件（`/etc/exports`）——包含了本地文件系统中可被远程加载的文件名

## ■ 客户

- 使用**mount**命令请求安装远程文件集系统，该命令指定：
  - 远程主机名字
  - 远程文件集系统的目录路径名
  - 将要安装的本地名字

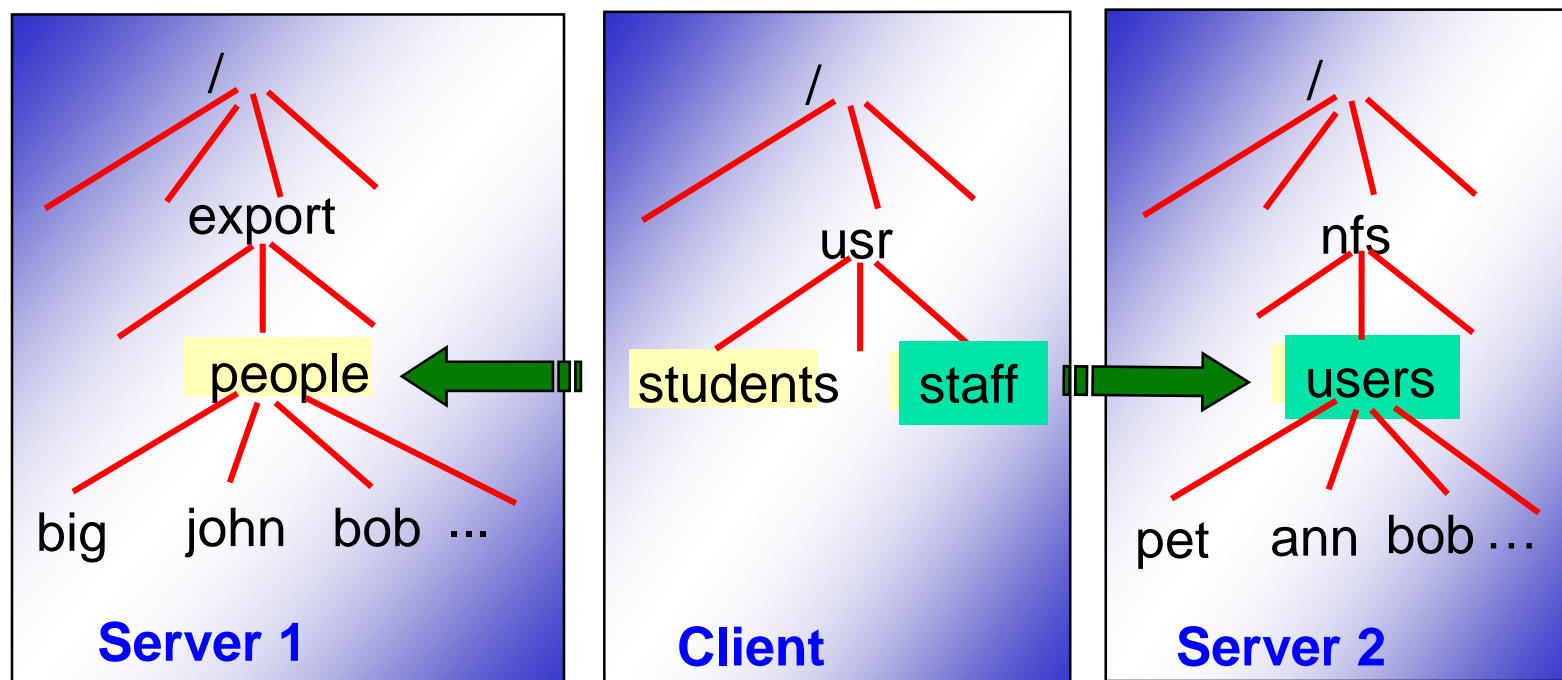


每台服务器都会记录可用于远程安装的本地文件

客户使用 `mount` 命令进行远程安装，提供名称映射

- 服务器1和2的`people`和`users`被安装到客户本地文件 `students`和`staff`结点上

# 在NFS客户端可访问的本地和远程文件系统



安装在客户 `/usr/students` 上的文件系统实际上是位于服务器1上的 `/export/people` 下的一个子树，例如，用户可以使用 `/usr/students/john` 访问服务器1上的文件

安装在客户 `/usr/staff` 上的文件系统实际上是位于服务器2上的 `/nfs/users` 下的一个子树，例如，用户可以使用 `/usr/staff/ann` 访问服务器2上的文件



# 路径名翻译

---

- UNIX文件系统（**本地**）每次使用open、create或stat系统调用时，一步步地将多部分文件路径名转为inode引用
- NFS服务器（**远程**）不进行路径名转换，需要由客户以交互方式完成路径名的翻译
- 客户向远程服务器提交多个lookup请求，将指向远程安装目录的名字的每一部分转换为文件句柄

# 例子

- `fd = open("./notes", O_RDONLY);`
- `read(fd, buf, n);`
- Client process has a reference to current directory's vnode.
- **Sends** `LOOKUP(dir-handle, "notes")` to server.
- Server **extracts i-number** from file handle.
- Asks local file system to turn that into a local vnode.
- Every local file system must support file handles...
- **Calls** the local vnode's lookup method.
- `dir->lookup("notes")` returns "notes" vnode.
- NFS server code **extracts i-number** from vnode, **creates** new file handle.
- Server **returns** new file handle to client.
- Client **creates** new vnode, sets its file handle.
- Client **creates** new file descriptor pointing to new vnode.
- Client app issues `read(fd, ...)`.
- Results in `READ(file-handle, ...)` being sent to server.



# UNIX和NFS缓存

---

- UNIX文件系统（**本地**）的缓存
  - 预先读
    - 将最近常用的页面装入内存
  - 延迟写
    - 该缓冲区将被其他页占用时才将该页的内容写入磁盘
    - 周期性同步写，如每隔**30**秒将改变的页面写到磁盘中（防止数据丢失）
- NFS（**远程**）的缓存
  - 服务器高速缓存（读写操作）
  - 客户高速缓存（请求结果）



# NFS服务器缓存

---

- NFS服务器的读缓存与本地文件系统相同
- NFS3服务器的写缓存，**写操作**提供两种选项：
  1. **写透**：在给客户发送应答前先将应答写入磁盘（写操作持久性），写透操作可能引起性能的瓶颈问题。
  2. **写透，但只在close()**：写操作的数据存储在内存缓存中，当系统接收相关文件的commit操作时（用于写而打开的文件关闭时），写入磁盘（提高性能）。

# NFS客户缓存

- 为了减少传输给服务器的请求数量，NFS客户模块将read, write, getattr, lookup和readdir操作的结果缓存起来。
- 保持一致性
  - 客户轮询服务器来检查他们所用的缓存数据是否是最新的。
- 基于时间戳的缓存块验证
  - 缓存中的每个数据块被标上两个时间戳
    - Tc: 缓存条目上一次被验证的时间
    - Tm: 服务器上一次修改文件块的时间
    - T: 当前时间
  - 有效性条件:  $(T - Tc < t)$  或者  $(Tm_{client} = Tm_{server})$
  - 选择t时对一致性和效率进行折衷，如3~30秒，对于目录t可以在30~60秒之间，目录更新风险更低



## 对服务器应用getattr操作（文件属性）获取 $T_{mserver}$

1. 有效性条件：  $(T - T_c < t)$  或者  $(T_{mclient} = T_{mserver})$
2. 如果第一个条件为真，不需要判断第二个
3. 如果为假，则需要从服务器获得获得 $T_{mserver}$ 值（对服务器应用getattr操作），并与本地 $T_{mclient}$ 比较

## 减少对服务器进行getattr操作的几种方法：

1. 当客户收到一个新的 $T_{mserver}$ 值时，将该值应用于所有相关文件派生的缓存项。
2. 将每一个文件操作的结果与当前文件属性一起发送，如果 $T_{mserver}$ 值改变，客户便可用它来更新缓存中与文件相关的条目。
3. 采用自适应算法来设置更新间隔值 $t$ ，对于大多数文件而言，可以极大地减少调用数量。



## 第8章 分布式文件系统

---

- 简介
- 文件服务体系结构
- SUN网络文件系统(NFS)
- **Andrew文件系统(AFS)**
- DFS进展
- 小结



# AFS的动机

---

- AFS与NFS是兼容的

- 通过NFS远程访问文件，文件句柄而不是inode引用文件

- AFS与NFS区别是可扩展性，5000-10000节点

- 海量用户
  - 海量文件
  - 海量用户存取热点文件

- 为实现可扩展性，AFS采用的设计特点：

- 整体文件服务：AFS服务器将整个文件和目录的内容都传到客户计算机上
  - 整体文件缓存：当一个文件或文件块被传输到客户计算机上时，它被存储到本地磁盘缓存中



# 使用AFS的典型场景

---

- 客户打开一个远程文件
  - 这个文件不在本地缓存时，AFS查找文件所在服务器，并请求传输此文件一个副本
- 在客户机上存储文件副本
- 客户在本地副本上进行读/写
- 客户关闭文件
  - 如果文件被更新，将它刷新至服务器，客户本地磁盘上的拷贝一直被保留，以供同一工作站的用户级进程下一次使用



# AFS设计时的考虑

---

- 大多数文件，更新频率小，始终被同一用户存取。
- 本地缓存的磁盘空间大，例如：**100MB**。
- 设计策略基于以下假设：
  - 文件比较小，大多数文件小于10KB。
  - 读操作是写操作的**6**倍
  - 通常都是顺序存取，随机存取比较少见
  - 大多数文件是被某一个特定的用户访问，共享文件通常是被某一个特定的用户修改
  - 最近使用的文件很可能再次被使用
- 不支持数据库文件

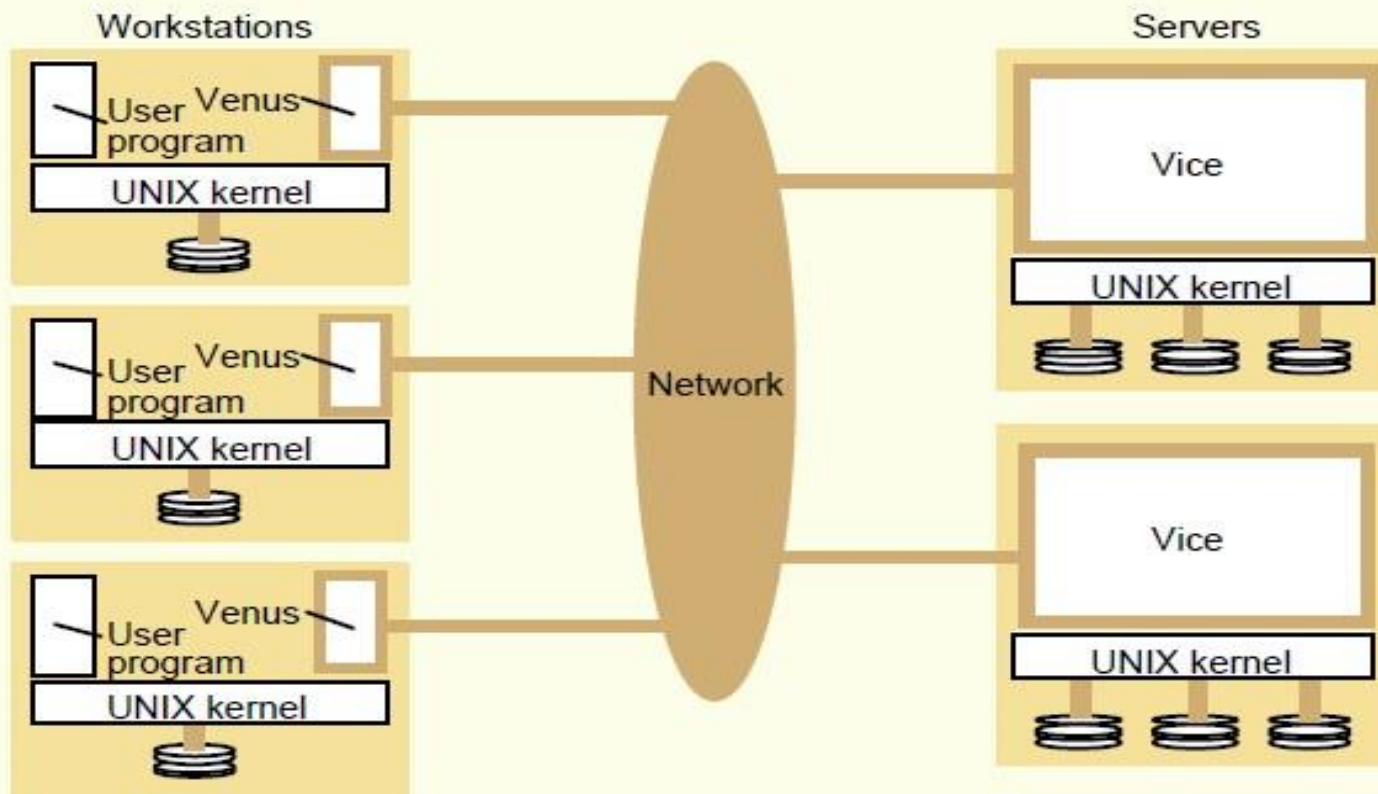


# AFS的组件

---

- AFS由两个软件组件实现，分别以UNIX进程Venus和Vice存在。
  - Venus是运行在客户计算机上的用户进程，相当于抽象模型中的客户模块；
  - Vice是服务器软件的名字，是运行在每个服务器计算机上的用户级UNIX进程。

# AFS中的进程



32位

卷号

32位

文件句柄

32位

唯一标识

## ■ Venus

- 通过文件标识符(**fid**)进行存取，类似平面文件服务的**UFID**
  - 卷号、文件句柄和唯一标识
- 一步一步地进行查找
  - 把路径名翻译成**fid**
- 文件缓存
  - 一个文件分区用做文件缓存：通常可以容纳数百个一般大小的文件
- 维护缓存一致性：回调机制

## ■ Vice

- 接收用**fid**表示的文件请求
- 平面文件服务





# Vice服务接口的主要组件

---

<i>Fetch(fid) -&gt; attr, data</i>	Returns the attributes (status) and, optionally, the contents of file identified by the <i>fid</i> and records a callback promise on it.
<i>Store(fid, attr, data)</i>	Updates the attributes and (optionally) the contents of a specified file.
<i>Create() -&gt; fid</i>	Creates a new file and records a callback promise on it.
<i>Remove(fid)</i>	Deletes the specified file.
<i>SetLock(fid, mode)</i>	Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.
<i>ReleaseLock(fid)</i>	Unlocks the specified file or directory.
<i>RemoveCallback(fid)</i>	Informs server that a Venus process has flushed a file from its cache.
<i>BreakCallback(fid)</i>	This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.

---



# 实现

---

- 问题：
  - 如何定位包含所需文件的服务器？
  - 当客户对共享文件空间内的文件发出open或close系统调用时，AFS怎样获得控制？
  - 在工作站上如何为缓存文件分配存储空间？
  - 当文件可能被多个客户更新时，AFS怎样保证缓存中的文件时最新的？

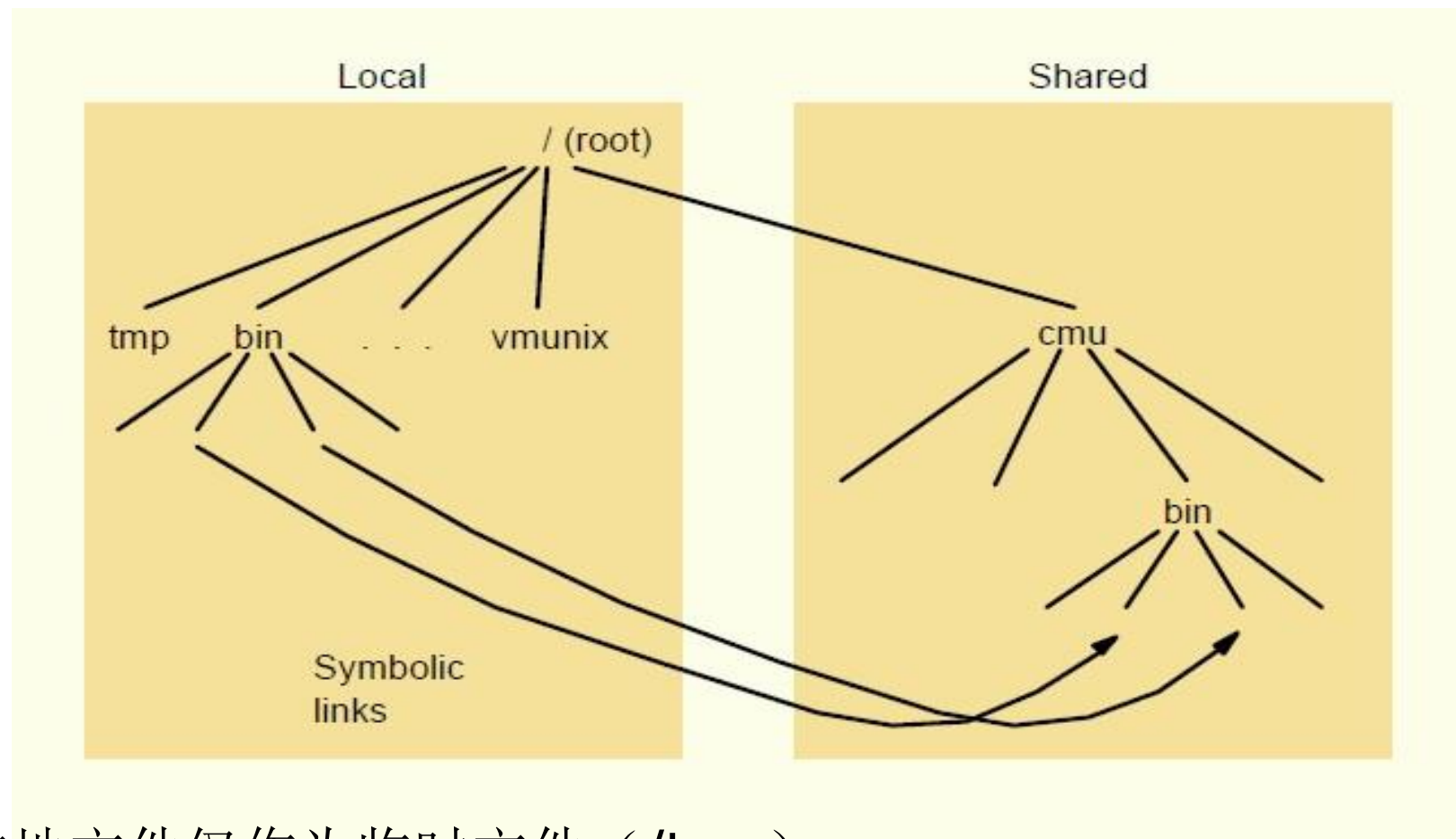


# AFS中的文件

---

- AFS中的文件分为本地的或共享的。
- 本地文件可作为普通的UNIX文件来处理，它们被存储在工作站磁盘上，只有本地用户可以访问它；
- 共享文件存储在服务器上，工作站在本地磁盘上缓存它们的拷贝。

# AFS中的文件



本地文件仅作为临时文件（`/tmp`）

其他标准UNIX文件（`/bin`）是通过将本地文件目录中的文件符号链接（类似Windows快捷方式）到共享文件空间

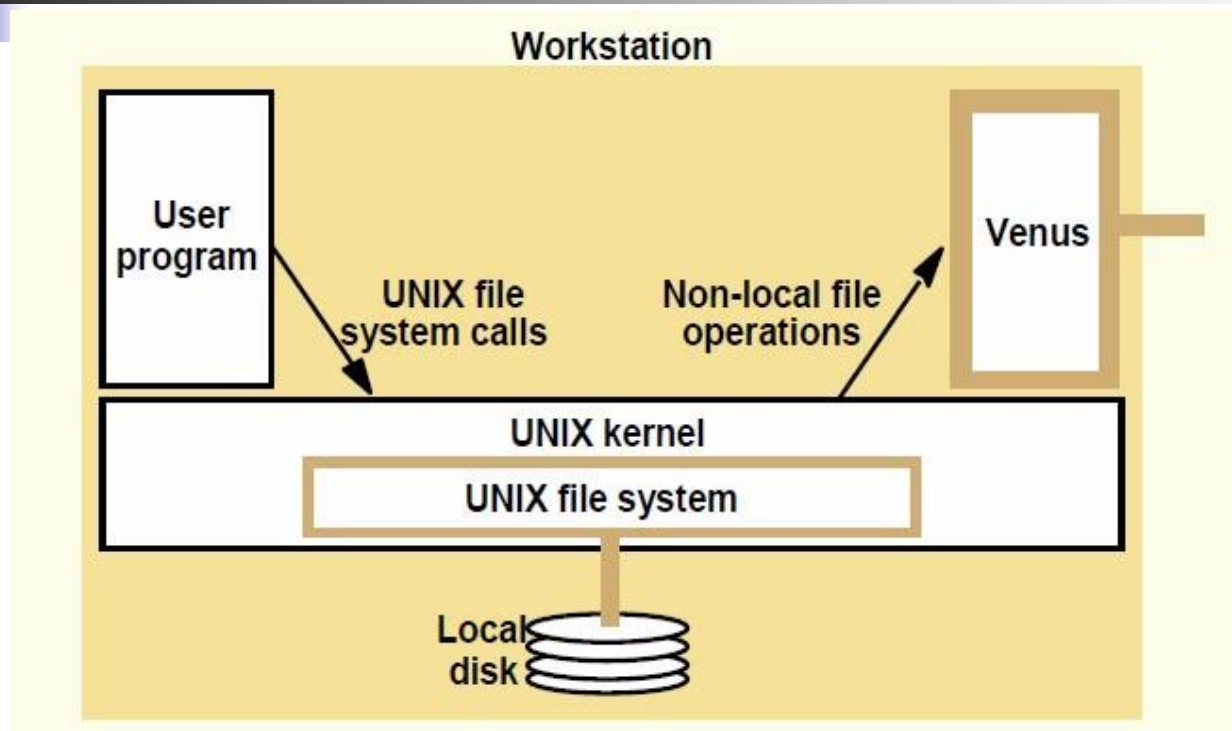


# 实现

---

- 问题：
  - 如何定位包含所需文件的服务器？
  - 当客户对共享文件空间内的文件发出open或close系统调用时，AFS怎样获得控制？
  - 在工作站上如何为缓存文件分配存储空间？
  - 当文件可能被多个客户更新时，AFS怎样保证缓存中的文件时最新的？

# AFS中系统调用拦截



UNIX修改版本截获那些指向共享名字空间文件的调用，如 open、close和其它一些系统调用，并将它们传递给Venus进程



# 实现

---

## ■ 问题：

- 如何定位包含所需文件的服务器？
- 当客户对共享文件空间内的文件发出open或close系统调用时，AFS怎样获得控制？
- 在工作站上如何为缓存文件分配存储空间？
- 当文件可能被多个客户更新时，AFS怎样保证缓存中的文件时最新的？



# 缓存的空间分配

---

- 每个工作站本地磁盘上都有一个文件分区被用作文件的缓存，保存共享空间中的文件拷贝，Venus进程管理这一缓存。
- 当文件分区已满，并且有新的文件需要从服务器拷贝过来时，它将最近最少使用的文件从缓存中删除
- 缓存都足够大，当客户缓存已经包含了当前用户文件和经常使用的系统文件时，工作站可以基本独立于Vice服务器工作





# 实现

---

- 问题：

- 如何定位包含所需文件的服务器？
- 当客户对共享文件空间内的文件发出open或close系统调用时，AFS怎样获得控制？
- 在工作站上如何为缓存文件分配存储空间？
- 当文件可能被多个客户更新时，AFS怎样保证缓存中的文件时最新的？



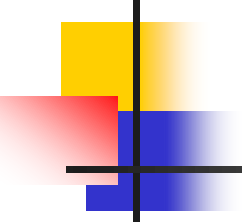
# 缓存一致性

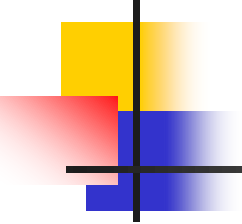
---

## ■ 回调承诺

- 由管理该文件的**Vice**服务器发送的一种标识，用于保证当其他客户修改此文件时通知**Venus**进程
- 两种状态：有效或取消
- 当**Vice**服务器**执行**一个**更新文件请求**时，它通知**所有Venus**进程将回调承诺标识设为**取消**状态。

- 当客户打开文件
  - 如果没有文件或是文件的标识值为**取消**，Venus从服务器取得文件。
  - 如果标识值为**有效**，Venus不需要引用Vice就可以使用缓存的文件拷贝
- 当客户关闭文件
  - 如果副本被修改，Venus向Vice发送此副本
  - Vice替换此文件内容
  - Vice通知所有的文件缓存设为取消状态
- 当客户重启或者在时间T内没有收到回调信息Venus将认为该文件已经无效
- 可扩展性
  - 由于大部分请求为读请求，与轮询相比，客户与服务器间的交互显著减少，提高了扩展性。

- 
- 缓存一致性目标：在不对性能产生严重影响的情况下，近似实现单个文件拷贝语义。
  - AFS-1更新语义(F—File, S—Server)
    - 在成功的open操作后：latest (F, S)：文件F在客户C的当前值和在服务端S上的值相同。
    - 成功的close操作后：updated (F, S)：客户C的文件F的值已经传播到服务端S上。
    - 在失败的open, close操作后：failure (S)：open和close并没有在S上执行。



AFS-2: 较弱的open保证, 客户可能会打开一个旧拷贝, 该文件已经被其他客户更新过了。

在成功的open操作后:

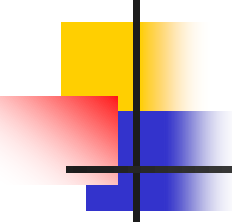
- **Latest(F,S,0):** 文件F在客户C的当前值和服务器S上的值相同 (F的拷贝是最新版本)
- 或者**lostCallback(S,T) and inCache(F) and latest(F,S,T)** (回调丢失 (通信故障) 不得不使用已缓存文件版本)
  - **lostCallback(S,T):** 最近T时间内从服务器S传递到客户C的回调信息已经丢失
  - **inCache(F):** 在open操作前客户C的缓存中就包含文件F
  - **latest(F,S,T):** 被缓存的文件F的拷贝过期时间不会超过T秒 (T通常设置为10分钟)



# 其他方面

---

- **内核修改：**修改UNIX内核，以支持Vice中使用文件句柄而不是UNIX文件描述符执行文件操作
- **线程：**Vice和Venus中使用非预先抢占性线程包，使客户和服务器并发处理请求
- **只读复制：**经常执行读操作，但很少修改的文件卷拷贝到多个服务器上
- **批量传输：**AFS以64KB的文件块进行传输以减小延迟。
- **部分文件缓存：**当应用程序只需要读文件的一小部分时仍需将整个文件传输到客户端，显然效率低。**AFSv3**解决这个问题，允许文件数据以**64KB**块的形式传输和缓存



## 其他方面（续）

---

- **位置数据库：**每个服务器包含一个位置数据库的拷贝，用于将卷名映射到服务器
- **性能：**AFS主要目标是可扩展性。通过缓存整个文件和回调机制减少服务器的负载
- **广域网支持：**AFSv3支持多个管理单元，每个单元有自己的服务器、客户、系统管理员和用户，是一个完全自治的环境，但这些协作的单元可以共同为用户提供提供一个统一的、无缝的文件名空间



# 第8章 分布式文件系统

---

- 简介
- 文件服务体系结构
- SUN网络文件系统（NFS）
- Andrew文件系统（AFS）
- DFS进展
- 小结





# NFS的改进

---

- **Spritley NFS**达到单个拷贝的更新语义
  - 客户对同一文件并发写的结果，与在一个**UNIX**系统中多个进程并发写本地文件结果相同
  - 使用回调通知客户文件发生改变，避免客户为检查文件是否改变而频繁调用**getattr**操作
- **NQNFS**：更精确的一致性，通过租借来保证缓存一致性
- **WebNFS**：通过**Web**直接访问**NFS**服务器
- **NFS第4版**：使**NFS**适用于广域网和互联网应用



# 存储组织的改进

---

- 廉价磁盘的冗余阵列（RAID）
  - 数据分解成固定大小的块
  - 存储在跨域多个磁盘的“条带”上
  - 冗余的错误更正代码，用于在磁盘故障时完全重建数据块，系统可以继续操作数据
- 日志结构的文件存储（LFS）
  - 内存积累若干写操作
  - 写到划分为大的、连续的、定长的段的磁盘上。



# 新的设计方法

---

- 以高伸缩性和高容错性的方式提供分布式文件数据的持久性存储系统；
- 把管理元数据和客户请求服务的职责与读写数据的职责相分离。



# 新的设计方法

---

- xFS（无服务文件系统）
  - 存储责任独立于管理和其它服务责任进行分布
  - 将文件数据分散存储到多个计算机上
  - 软件的**RAID**存储系统
  - 协同缓存
- Frangipani
  - 将持久存储责任和其他文件服务活动相分离
  - **Petal**为多个服务器磁盘提供了一个分布式的磁盘抽象
  - 日志结构的数据存储



# 分布式文件系统

---

- 简介
- 文件服务体系结构
- SUN网络文件系统
- Andrew文件系统
- DFS进展
- 小结



# 小结

---

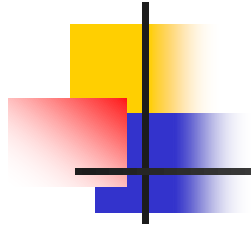
- NFS设计中的关键问题
  - 服务器缓存
  - 客户缓存
  - 客户或服务崩溃后的重启
  - 读/写的高度透明
  - 可扩展性
- NFS
  - 无状态，效率和可扩展性不高
- AFS
  - 可扩展性好



# 关键问题

---

- 透明性需要采取什么措施？
  - 虚拟文件系统，客户集成
- 保持性能需要采取什么措施？
  - 整体文件传送，整体文件缓存，回调
- 并发操作需要采取什么措施？
  - 无状态，没有open和close操作
- 系统容错需要采取什么措施？
  - 检查缓存的有效性，回调



The End