# 第十章 分布式文件系统

Author： 中山大学 17数据科学与计算机学院 YSY

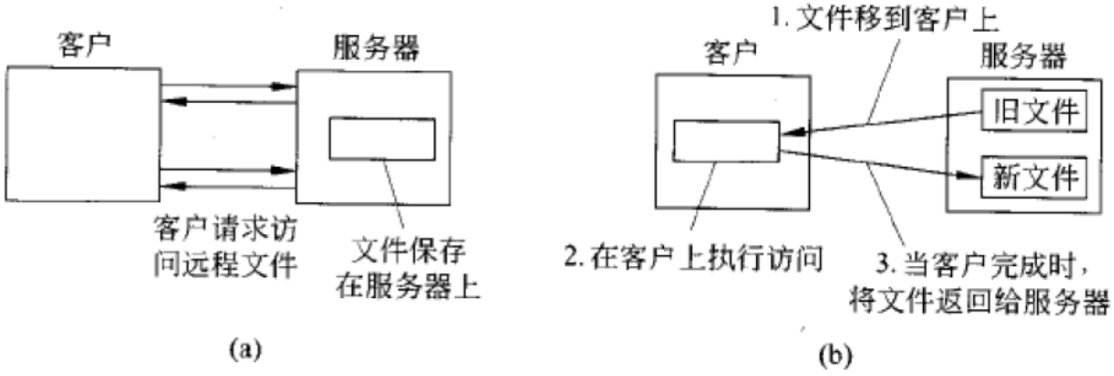
https://github.com/ysyisyourbrother

## 体系结构

客户服务器体系结构：典型代表是NFS（Network File System）

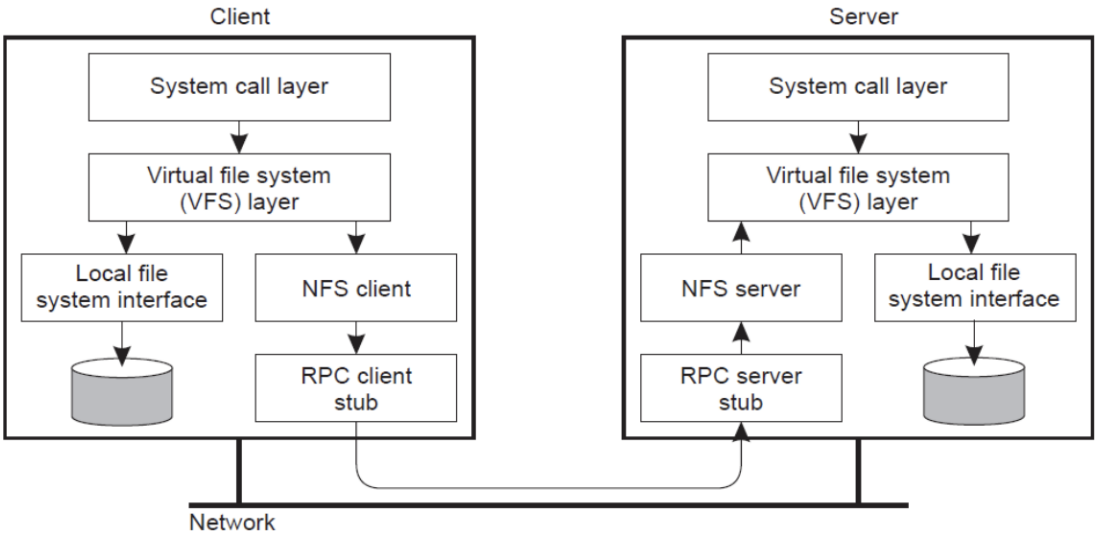
NFS：

1. 每个文件服务器都提供其本地文件系统的一个标准化视图；
2. 每个NFS服务器都支持相同的模型；
3. 底层模型是远程文件访问模型；




## NFS架构

NFS采用VFS (Virtual File System)实现，VFS是不同文件系统接口事 实上的标准， 现代OS都提供VFS。VFS提供了系统结构，屏蔽了访 问本地和远程文件的差异性。




## NFS文件系统模型

NFS提供了与UNIX系统完全一样的文件系统模型；

- 分层组织成命名图；
- NFS像任何UNIX文件系统一样支持硬链接和符号链接；

| 操作 | 版本 3 | 版本 4 | 描述 |
| --- | --- | --- | --- |
| create | 有 | 无 | 创建一个常规文件 |
| create | 无 | 有 | 创建一个非常规文件 |
| link | 有 | 有 | 创建一个文件的硬链接 |
| symlink | 有 | 无 | 创建一个文件的符号链接 |
| mkdir | 有 | 无 | 在给定目录下创建一个子目录 |
| mknod | 有 | 无 | 创建一个特殊文件 |
| rename | 有 | 有 | 更改文件名 |
| remove | 有 | 有 | 从文件系统中删除一个文件 |
| rmdir | 有 | 无 | 从一个目录中删除一个空的子目录 |
| open | 无 | 有 | 打开一个文件 |
| close | 无 | 有 | 关闭一个文件 |

## 基于集群的分布式文件系统

- 当处理非常大的数据集合的时候，传统的客户端-服务器方法就不再适合了；
- 解决方案一: 加速文件访问速度，应用文件分片划分技术使文件可以并行访问；

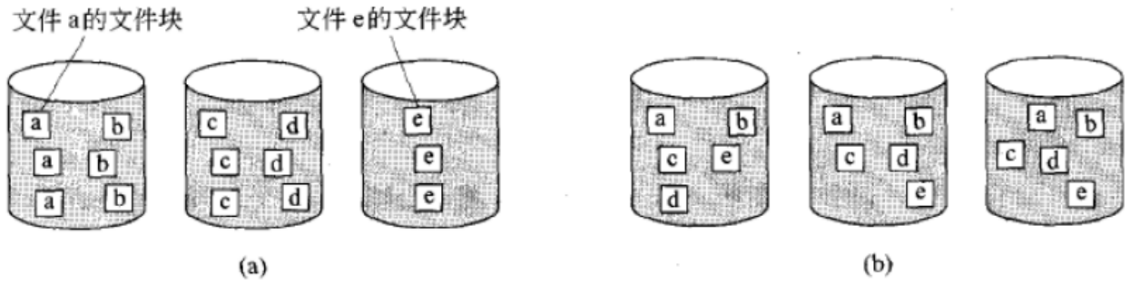
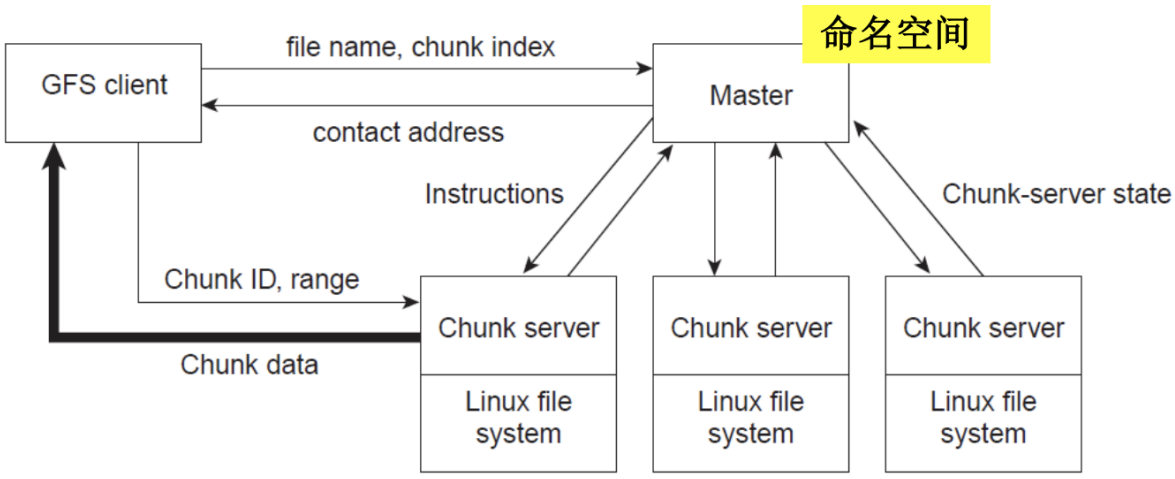



图 11.4　用于分布文件的两种方法之间的区别：
(a) 把整个文件分布在多个服务器上；(b) 对文件进行带区划分以进行并行访问

a是将文件分成多个小块，b相当于一个文件一个片

- 解决方案二: 将文件分成较大的数据块如64MB，然后将数据块分布、 复制到多个物理机器上（Amazon、Google）；




数据存在chunk上，master用来记录文件块对应的chunk位置

## GFS的特点

GFS是谷歌文件系统

- 主服务器只是在内存中维护了一张（文件名，块服务器）的**映射表** = > 最小化IO， **以日志的形式保存对数据的更新操作**，当日志过大时 将创建检查点，存储主存数据；
- 大量的实际工作是块服务器完成的。文件采用主备模式复制，主服务 器避开循环；

  chunk可能被某个客户端更改，其他的备份的chunk也要保持更新，这个过程不是master发起的，而是谁被更改了谁去发起更新。
- 上述两个特点决定了GFS主服务器master不会构成瓶颈，为系统带来重要的 可伸缩性，单个主服务器可以控制数百个块服务器；

## 对称式体系结构

基于对等（P2P）技术的完全对称的体系结构；

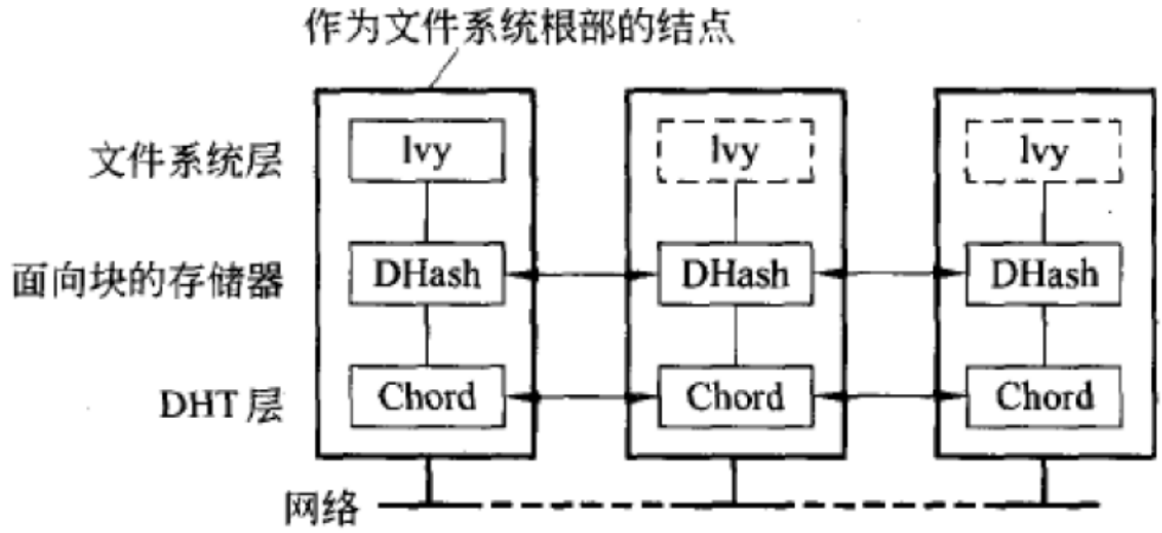



图 11.6　Ivy 分布式文件系统的组织结构

Ivy：主要是抽象读写等基础功能

DHT：计算每个文件具体需要存到哪个节点上 Chord

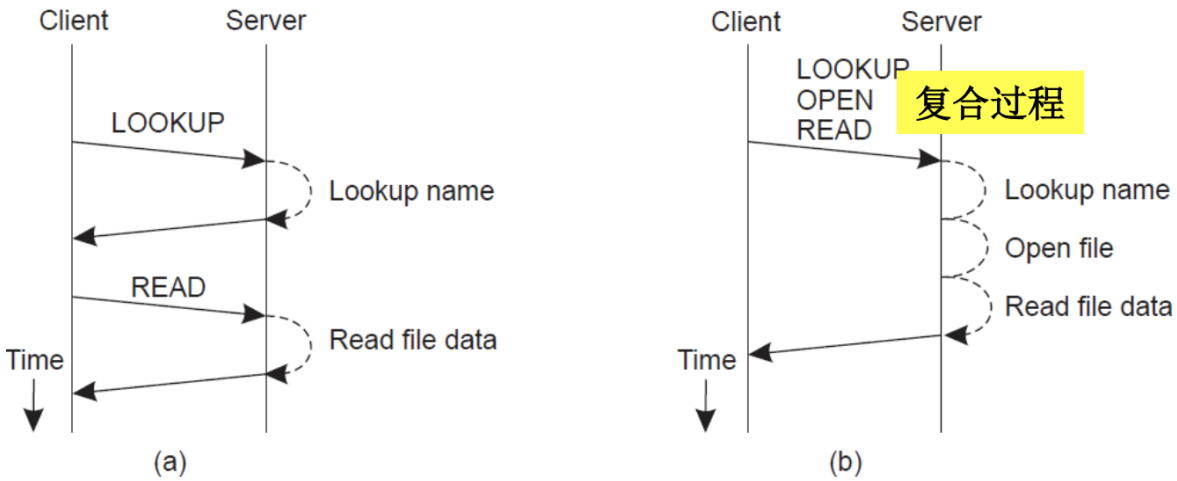
## 文件系统中的进程

**无状态的NFS**：不需要记录和client的连接记录， 实现简单， 服务器崩溃后，不需要恢复阶段；无法锁定访问文件；缺点是性能可能会较差。

**有状态的NFS**： 需要服务器维护关于客户端的大量信息；优点是性能较好，但缺点是需要保存大量的客户端信息。

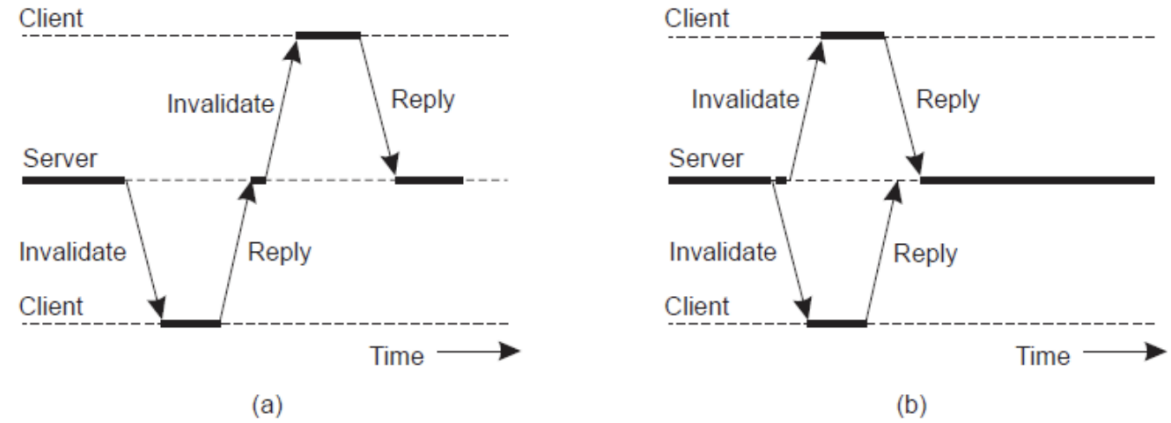
## 分布式文件系统中的RPC

选择RPC是为了使系统独立于底层操作系统、网络和传输协议； 但是当需要跨网络访问文件时，则需要另外的方案；




原始的方法是多次RPC，查询名字，读文件，每次都重新请求一次RPC连接。优化的方法是，可以一次性传递多个信息。但缺点是在并发情况下，一下进行多个操作，可能会影响到别的其他client访问服务器。

当处理文件副本特别是客户端的副本更新时，服务器顺序地发送 副本失效指令显然效率不高；




## 命名空间

## 自动挂载

## 构造全局名称服务

## 同步

当多进程同时操作文件的时候，出现一致性问题。

#### 文件共享语义

自定义一些语义规则，相当于设定前面的单调读，单调写等

#### 同步解决方案

- 每次更新将缓存文件的所有改动传播回服务器，简单但是效率低；
- 对打开的文件所做的改动最初只对修改文件的进程可见，只有当文件关闭时，才把这次改动同步到文件服务器上，这些改动才对其他进程可见即会话语义；
- 所有文件都是不可改变的，文件上的操作只有 create和read 操作；
- 使用原子事务处理共享文件；

| 方法 | 注释 |
| --- | --- |
| UNIX 语义 | 一个文件上的每个操作对所有进程都是即时可见的 |
| 会话语义 | 在文件关闭之前，所有改动对其他进程都是不可见的 |
| 不可改变的文件 | 不允许更新文件；简化了共享和复制 |
| 事务 | 所有改动都以原子方式发生 |

图 11.17　4 种处理分布式系统中的共享文件的方法

### 文件锁定

实现方式：利用锁管理器

| 操作 | 描述 |
| --- | --- |
| lock | 为一定范围的字节创建锁 |
| lockt | 测试是否授予了相冲突的锁 |
| locku | 删除一定范围的字节上的锁 |
| renew | 继续租用指定的锁 |

### 共享预约

一种锁定文件的隐含方法，共享预约独立于锁定。主要包 含：客户打开文件时指定的访问类型，以及服务器应该拒 绝的其他客户的访问类型。（相当于自己决定允许别人进行什么操作）

**当前文件拒绝状态**

| 请求访问 | NONE | READ | WRITE | BOTH |
| --- | --- | --- | --- | --- |
| READ | 成功 | 失败 | 成功 | 失败 |
| WRITE | 成功 | 成功 | 失败 | 失败 |
| BOTH | 成功 | 失败 | 失败 | 失败 |

(a)

**请求的文件拒绝状态**

| 当前访问状态 | NONE | READ | WRITE | BOTH |
| --- | --- | --- | --- | --- |
| READ | 成功 | 失败 | 成功 | 失败 |
| WRITE | 成功 | 成功 | 失败 | 失败 |
| BOTH | 成功 | 失败 | 失败 | 失败 |

(b)

图 11.19　使用 NFS 共享预约实现 open 操作的结果

(a) 在当前拒绝状态下，客户请求共享访问时的情况；(b) 在当前文件访问状态下，客户请求拒绝状态时的情况

## 一致性和复制

### 缓存

NFS缓存模型

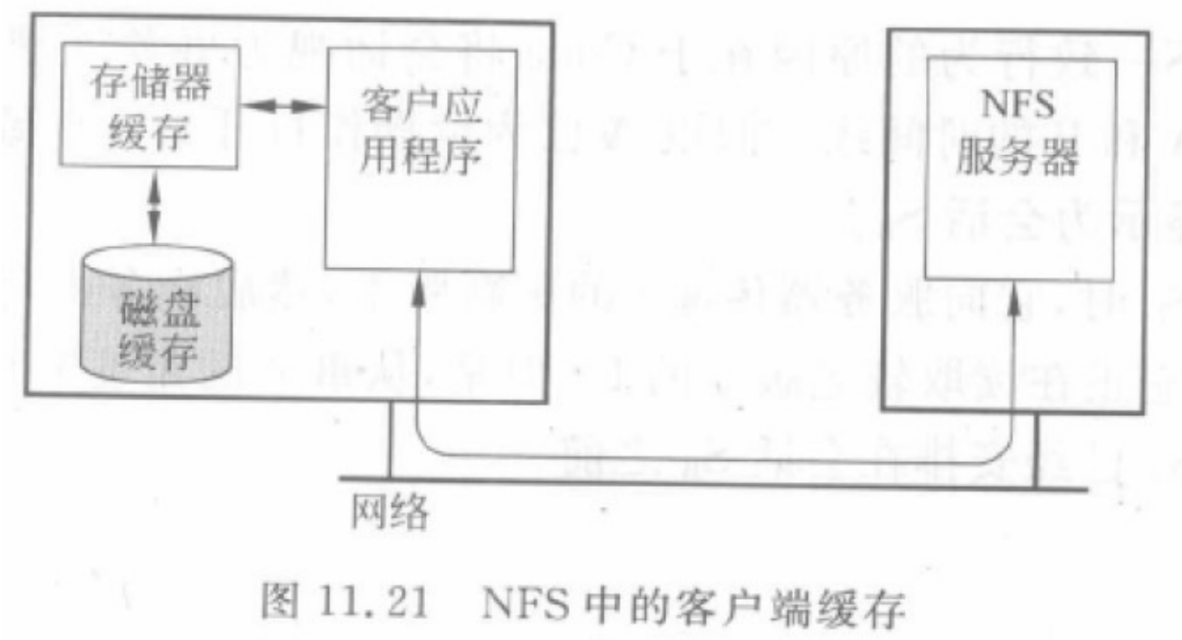



图 11.21 NFS 中的客户端缓存

**缓存方法1:**

- 当客户打开文件时，将它从服务器获得的数据缓存起 来，并把这些数据作为各种read操作的结果。
- 另外，write操作也可以在缓存中执行。 客户关闭，如 果文件被修改，必须回送到服务器。

**缓存方法2**

服务器可以将某些权限委托给客户，即**开放式委托。** 开放式委托发生在客户机被允许在本地处理来自同一 机器的其他客户的Open和close操作。

## 对等文件系统中的复制

复制的主要作用是加快搜索和查找请求的速度，还可以平 衡节点

**之间的负载非结构对等系统:**

- 基本原理:将查找数据归结为搜索网络中的数据
- 广播负载过高，如何寻找一种最优的复制策略

**结构化对等系统:**

用于平衡节点之间的负载;

## 容错性

**处理Byzantine故障:** 基本思想是通过构造有限状态机的集合来部署主动复制，并且这个集合中具有无故障的进程以相同的顺序执行操作。

**Byzantine故障解决方案**