NFS、WAFL和GFS

中国科学院大学计算机与控制学院
中国科学院计算技术研究所
2019-12-25

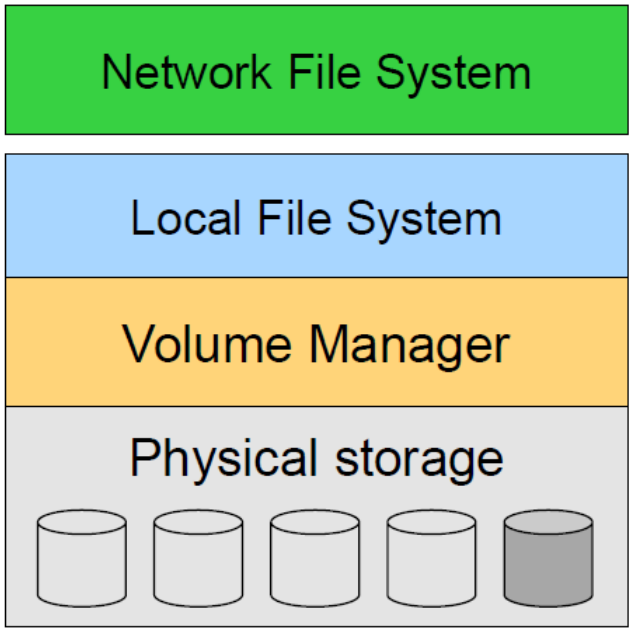
内容提要

- NFS
- WAFL
- GFS
- 安全保护

文件系统抽象

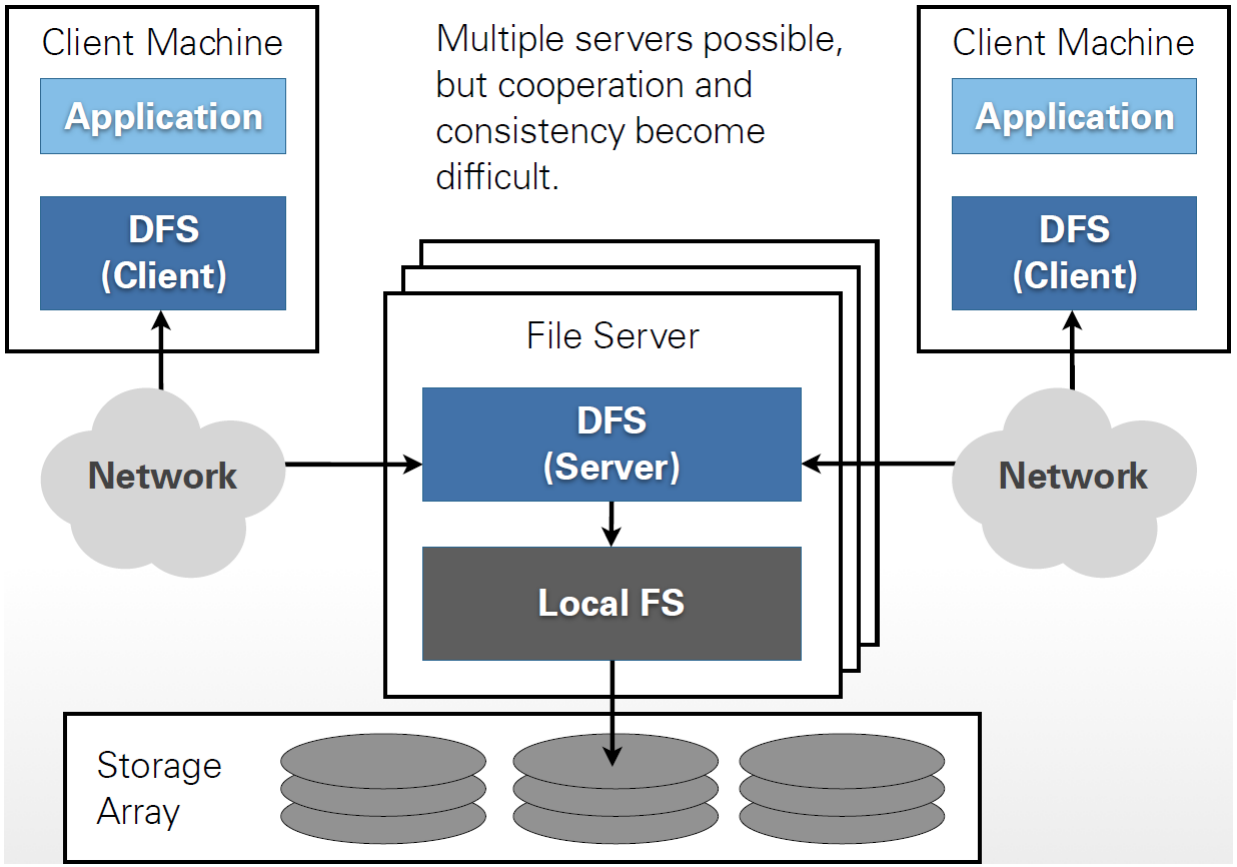
自下而上：

- 块存储设备（物理的）
  - 磁盘/RAID/SSD
- 卷管理
  - 块存储之上构建逻辑卷
  - 映射到物理存储
  - RAID和重构
- 本地文件系统
  - 块设备之上实现文件系统抽象
  - FFS，XFS，JFS，ZFS，…
  - Ext2/3/4, Reiserfs，BtrFS, …
- 网络文件系统
  - 通过网络访问远端机器上的一个本地文件系统
  - NFS，CIFS，等



远程文件访问

- 如何访问通过网络连接的另一台计算机上的文件？
  - FTP
  - DFS



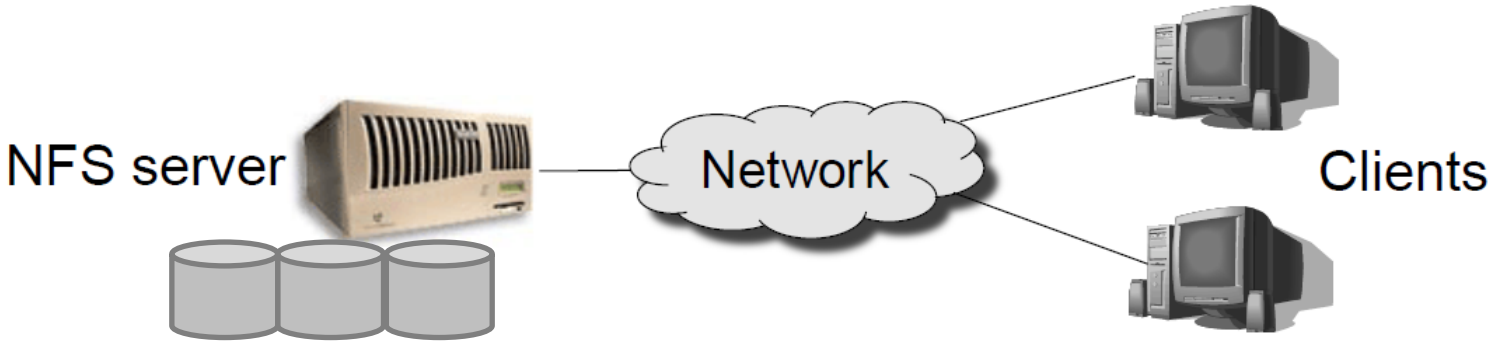
DFS先驱（1980s）

DFS	Inventor	Publication year	Layer	Communication	Client Caching	Semantics
UNIX United	U. of Newcastle, England	1982, 1983	user level	RPC	UNIX buffering	
Locus	UCLA	1983, 1985	kernel	Special Protocol	Similar to UNIX buffering	Complete UNIX
NFS	**SUN**	1986, 1988	kernel	RPC/XDR	Similar to UNIX buffering	Timing-dependent
AFS	**CMU**	1985, 1988	Kernel	RPC	Write-on-close	Session
RFS	AT&T	1986	Kernel	RPC	Write-through	Complete UNIX
Sprite	**UC, Berkeley**	1988	kernel	RPC	Similar to UNIX buffering	UNIX

E. Levy and A. Silberschatz, **Distributed File Systems: Concepts and Examples**, ACM Computing Surveys, 22(4), Dec.1990

NFS：Network File System

- 多个客户端（计算机）共享一台文件服务器



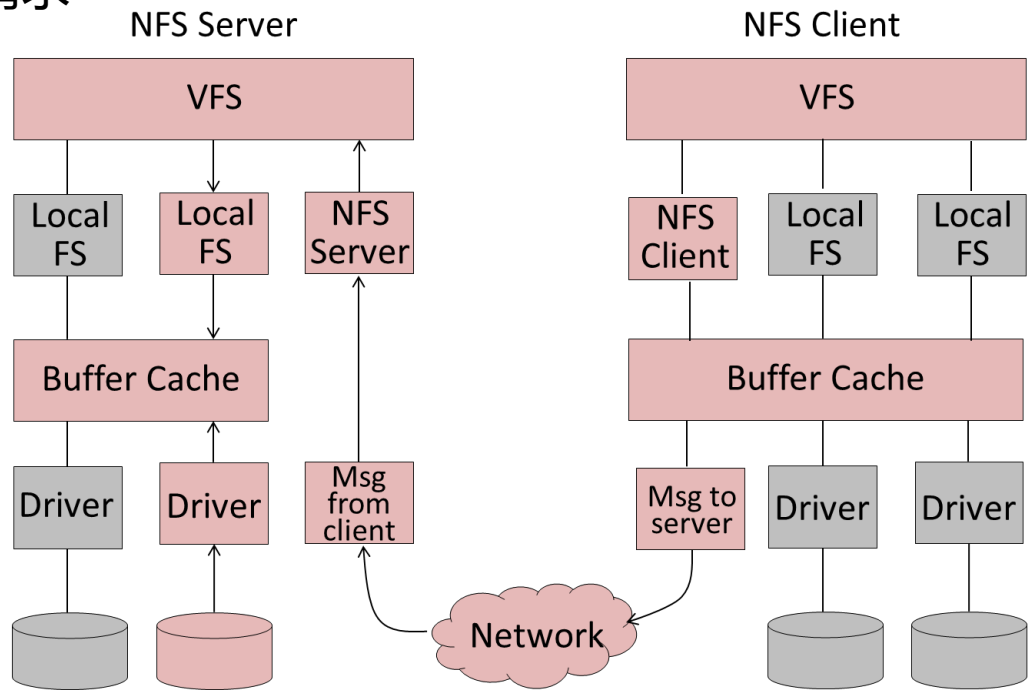
- History
  - NFS：SUN， 1985年
  - 开放协议：IETF标准
  - NFS v2：1989年，IETF RFC 1094
  - **NFS v3**[1]：1995年，RFC 1813[2]
  - NFS v4：2000年RFC 3010，03年RFC 3530，15年RFC 7530

[1] B. Pawlowski, C. Juszczak, P. Stauback, et al. NFS Version 3: Design and Implementation. USENIX Summer Conference, 1994

[2] B. Callaghan, B. Pawlowski, P. Staubach. **NFS Vesion 3 Protocol Specification**. RFC 1813, 1995 https://ietf.org/rfc/rfc1813.txt

NFS架构

- 多Client，单Server
  - NFS客户端: 实现FS功能和接口
    - 接口: syscall, 与本地FS相同接口 (透明性)
    - 把文件访问syscall转换成请求
    - 把请求发给服务器
    - 接收服务器发回的结果, 并返回给调用者
  - NFS服务器: 提供文件服务
    - 接收客户请求
    - 读写本地FS
    - 把结果发回给客户端
  - 缓存
    - 客户端缓存
    - 服务器端缓存

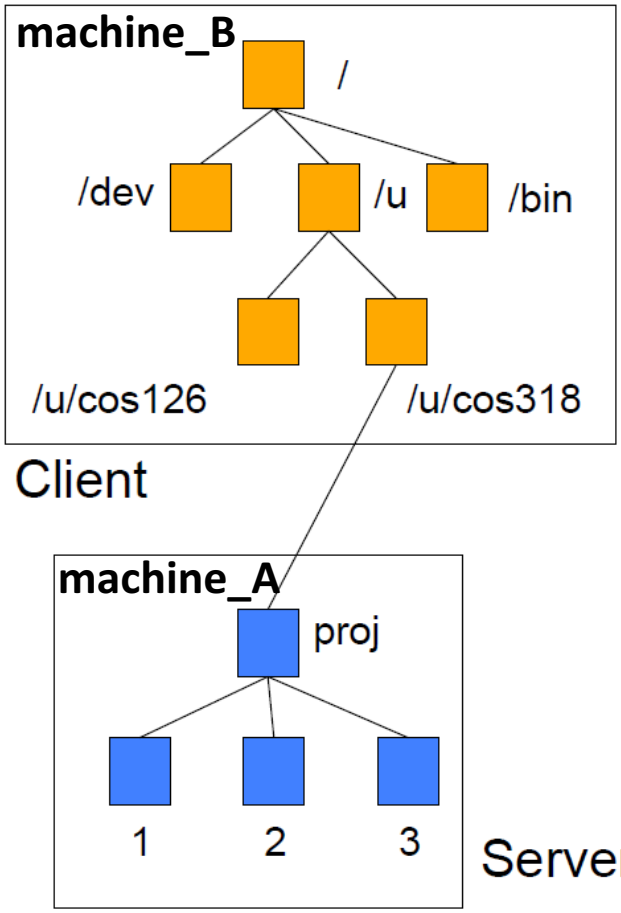


NFS设计

- 设计目标
  - 简单
  - 快速回复
- 核心思想: **无状态**服务器 (stateless)
  - 服务器不记录客户端打开的文件
  - 服务器不记录每个打开文件的当前偏移
  - 服务器不记录被客户端缓存的数据块
- 核心数据结构：File Handle (FH)
  - 唯一标识客户端要访问的文件或目录
  - Volume ID
  - ino
  - Generation number

NFS挂载（mount）

- NFS服务器 “export” 一个目录给客户端
  - 输出目录表：/etc/exports
  - 输出目录命令：exportfs



- NFS客户端挂载 (mount)
  - NFS服务器 (机器名或网络地址)
  - NFS服务器输出目录的路径名
  - 挂载点：本地目录的路径名

```
$ ls /usr/cos318
$ mount –t nfs  machine_A:/proj  /u/cos318/proj
$ ls /usr/cos318/proj
```

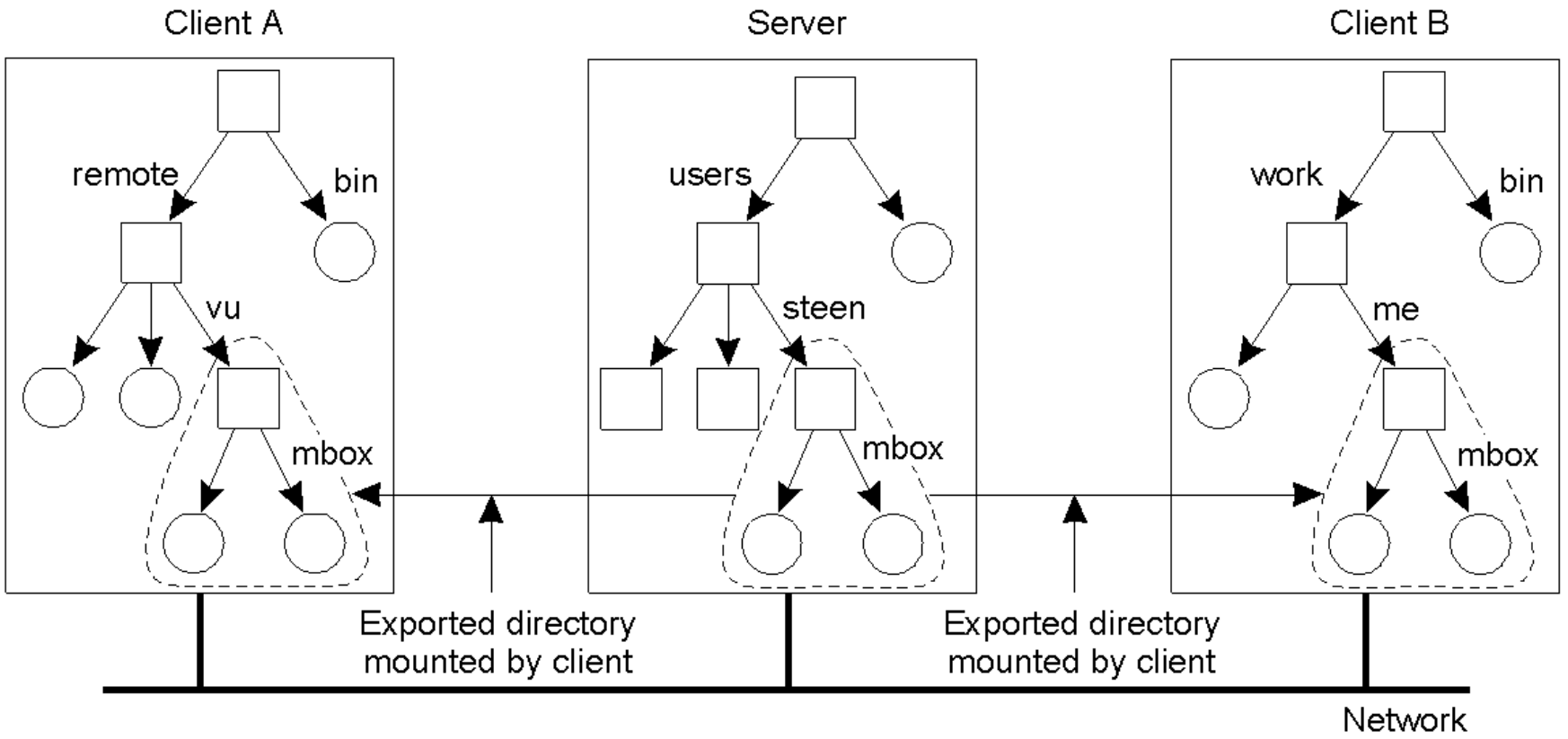
  - 服务器返回输出目录的File Handle

- 自动挂载（automount）

```
#  cat /etc/exports
/proj     machine_B(rw)
```

例子

- 两个客户端挂载同一个服务器输出的目录



- 三个机器共享一棵子树

NFS Protocol (v3)

1. NULL: Do nothing
2. GETATTR: Get file attributes
3. SETATTR: Set file attributes
4. LOOKUP: Lookup filename
5. ACCESS: Check Access Permission
6. READLINK: Read from symbolic link
7. READ: Read From file
8. WRITE: Write to file
9. CREATE: Create a file
10. MKDIR: Create a directory
11. SYMLINK: Create a symbolic link
12. MKNOD: Create a special device
13. REMOVE: Remove a File
14. RMDIR: Remove a Directory
15. RENAME: Rename a File or Directory
16. LINK: Create Link to an object
17. READDIR: Read From Directory
18. READDIRPLUS: Extended read from directory
19. FSSTAT: Get dynamic file system information
20. FSINFO: Get static file system Information
21. PATHCONF: Retrieve POSIX information
22. COMMIT: Commit cached data on a server to stable storage

- LOOKUP
  - 参数(客户端提供): 目录FH, name
  - (服务器)应答: name的FH
- READ
  - 参数: FH, **Off**, Count
  - 应答: 数据, 属性
- WRITE
  - 参数: FH, **Off**, Count, Data
  - 应答: 属性
- GETATTR
  - 参数: FH
  - 应答: **属性**

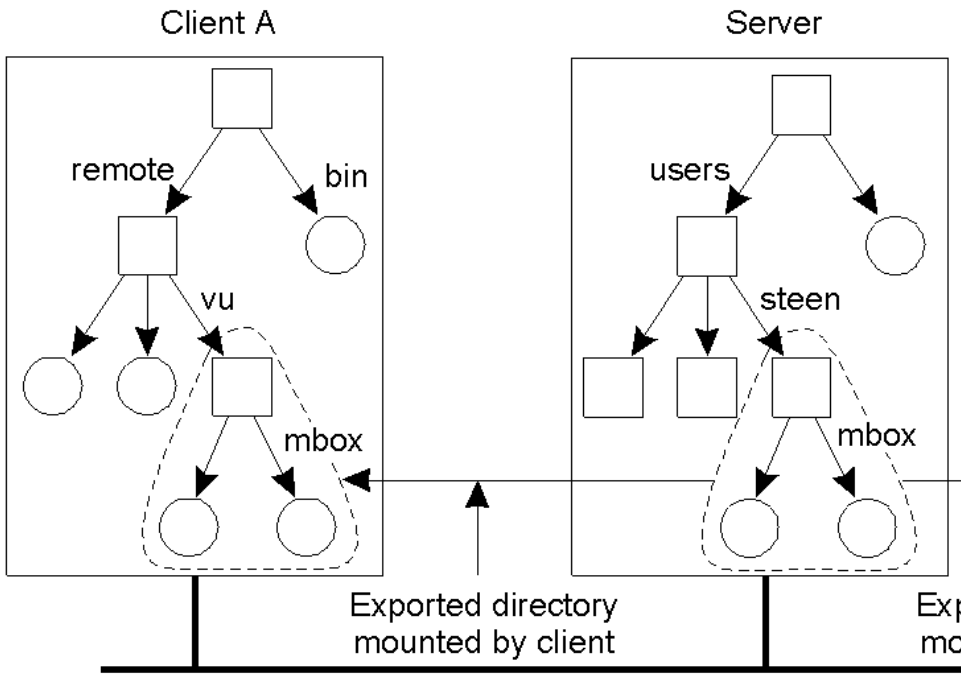
每个请求是自包含的

文件访问的实现原理

- 客户端open一个文件
  - open() syscall: 路径解析
  - 向服务器发LOOKUP请求
  - 接收服务器应答的FH
  - 将本地fd与FH关联
- 客户端read文件
  - read() syscall: fd, buf, count
  - 根据fd得到FH和偏移
  - 向服务器发READ请求
  - 参数为FH, 偏移, count
  - 接收服务器的应答数据
  - 把应答数据拷贝到buf
  - fd的偏移 += count
- 客户端close文件
  - 释放fd与打开文件结构
  - 无需与服务器交互

```
# mount –t nfs Server:/users/steen /remote/vu

fd = open ("/remote/vu/mbox", O_RDONLY);
n = read (fd, buf, 16384);
```

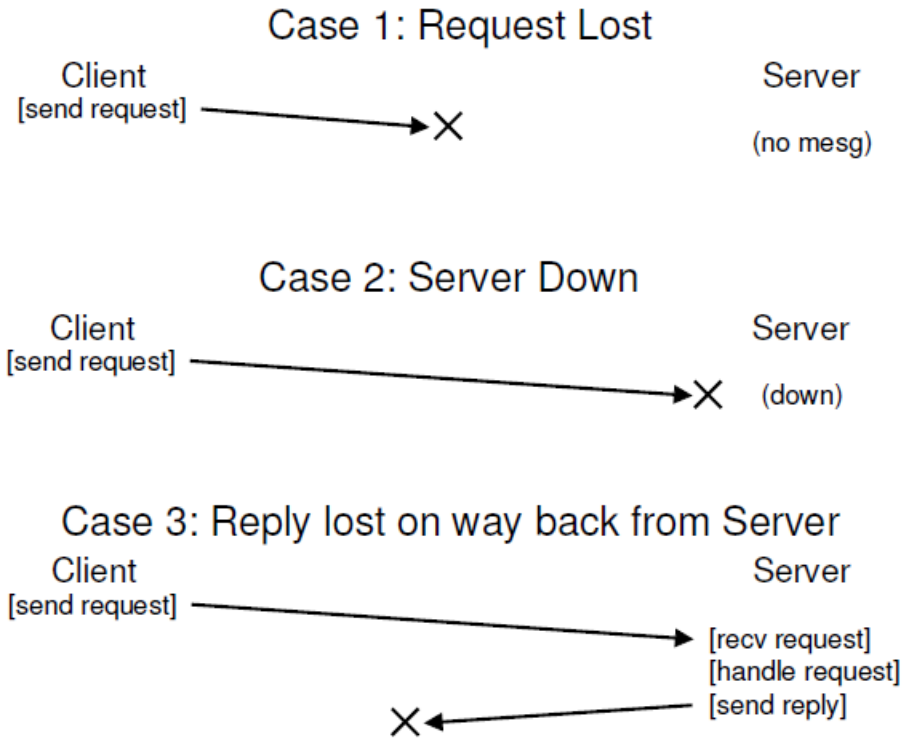


文件访问的实现原理

- 客户端open一个文件/目录
  - open() syscall: 路径解析
  - 向服务器发LOOKUP请求
  - 接收服务器应答的FH
  - 将本地fd与FH关联
- 客户端read文件
  - read() syscall: fd, buf, count
  - 根据fd得到FH和偏移
  - 向服务器发READ请求
  - 参数为FH, 偏移, count
  - 接收服务器的应答数据
  - 把应答数据拷贝到buf
  - fd的偏移 += count
- 客户端close文件
  - 释放fd与打开文件结构
  - 无需与服务器交互
- 服务器接收LOOKUP请求
  - 从目录FH中得到VID和目录ino
  - 读目录 i-node
  - 读目录块, 查找与name匹配的目录项 <name, ino>
  - 构造FH: VID, name的ino, gno
  - 发应答: name的FH
- 服务器接收READ请求
  - 从FH中得到VID和文件ino
  - 打开本地文件ino得到sfd
  - 设置本地文件的偏移 (lseek)
  - 读本地文件数据到sbuf中 read(sfd, sbuf, count)
  - 关闭本地文件close(sfd)
  - 发应答: sbuf的数据

NFS的失效处理

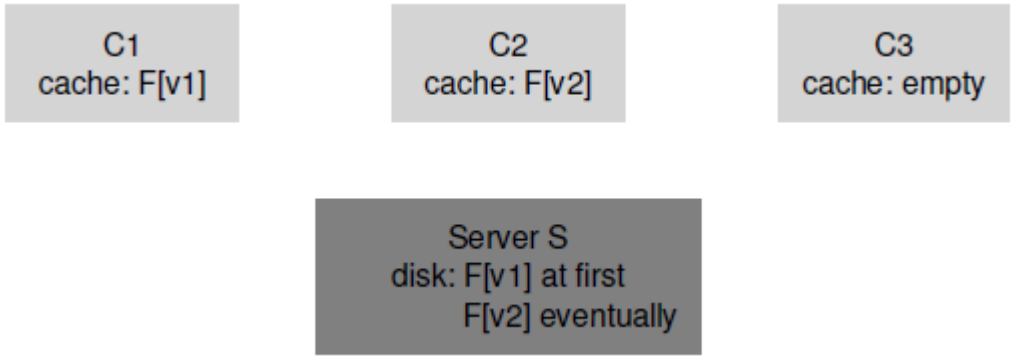
- 三种失效
  - 客户端请求丢失
  - 服务器宕机
  - 服务器应答丢失
- NFS的策略: retry
  - 客户端会超时重发请求
- 前提
  - 协议请求是幂等的



1. 由服务器维护当前偏移是幂等的吗?
2. 协议中所有请求都是幂等的吗?

客户端缓存

- 客户端用一部分kernel内存来缓存数据和元数据
- 好处
  - 提高文件读写性能：减少与服务器的交互（网络 & 磁盘I/O）
- 缓存一致性问题
  - 当多个客户端同时读写同一个文件：多读单写、多写



  - 当某个客户端写，导致

1. 修改不可见：客户端C3打开文件时读到旧版本 (服务器不是最新版本)
2. 陈旧数据：客户端C1缓存中是旧数据

客户端缓存一致性问题

- NFS的解决办法
  - Close-to-open consistency
    - Flush-on-close
    - open时用GETATTR来检查缓存中数据块的有效性
  - 数据块 (文件/目录) 60s过期，属性缓存3s过期
  - 脏数据：30秒之内写回NFS服务器
- 附加手段
  - 不共享缓存
    - 只能被一个客户端缓存
  - 网络锁管理 （ NLM v4)
    - 顺序一致性：one writer or N readers

服务器端缓存

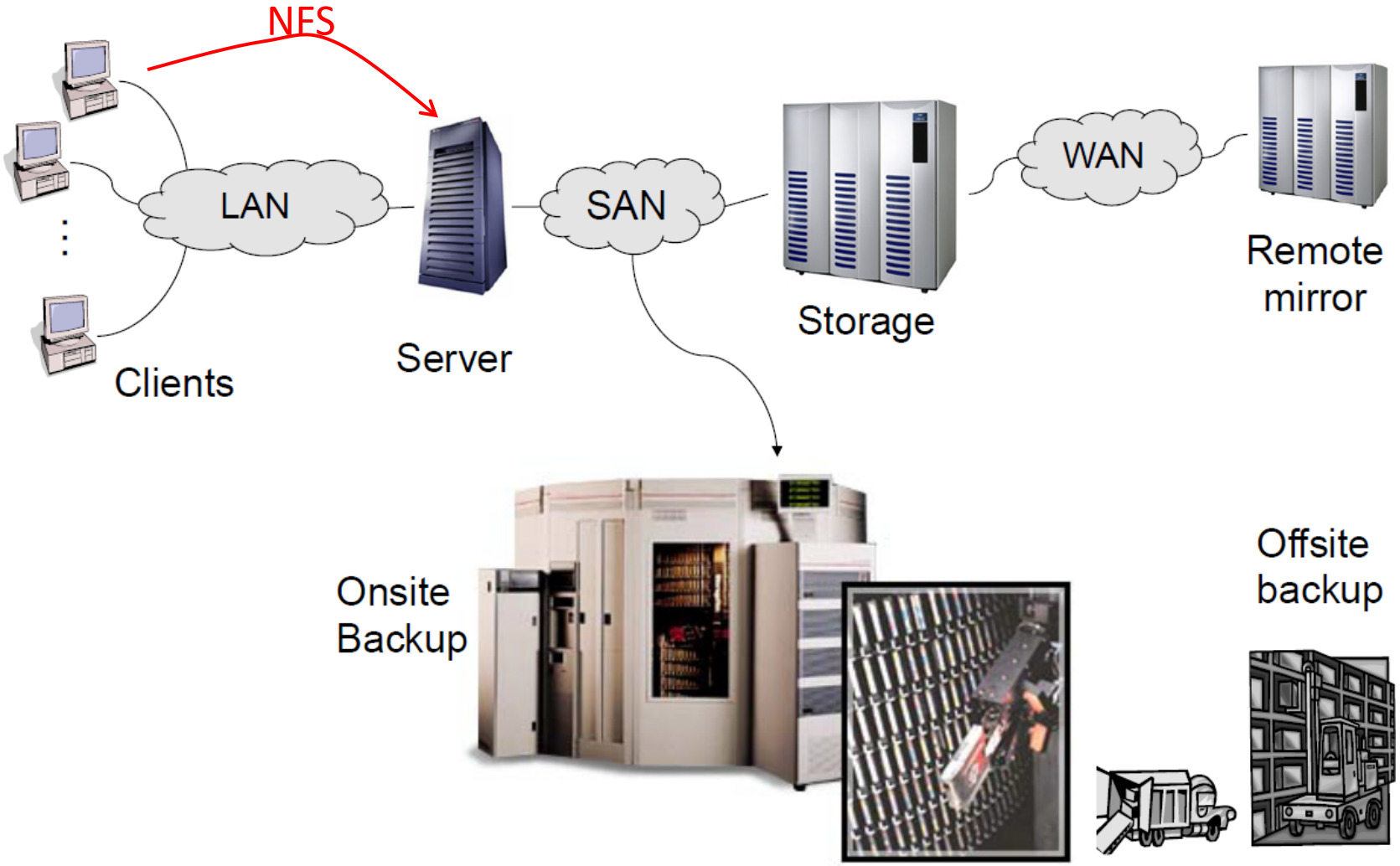
- 服务器用一部分kernel内存来缓存数据和元数据
- 好处
  - 提高文件读写性能：服务器端减少磁盘I/O
- 问题
  - 服务器宕机可能丢数据
- 解决办法 (NFS v3)
  - COMMIT
    - 服务器把之前WRITE写在缓存中的数据写到持久化存储
    - 参数：FH, 偏移, count
    - 如果COMMIT超时未收到应答
      - 之前的WRITE和COMMIT本身都要重发

NFS协议的发展

- v2的不足
  - 18个操作
  - 文件大小用32位表示（≤ 4GB）
  - 写性能差：NFS服务器采用同步写 (write through)
- v3的改进（大部分产品仍然在用这个版本）
  - 22个操作：增加COMMIT、READDIRPLUS、FSINFO、FSSTAT
  - 文件大小用64位表示
  - 写性能改进：WRITE 和 COMMIT
  - 仍然是无状态的
- v4的改变
  - 42个操作
  - 有状态
  - 解决了一致性问题
  - 安全性问题

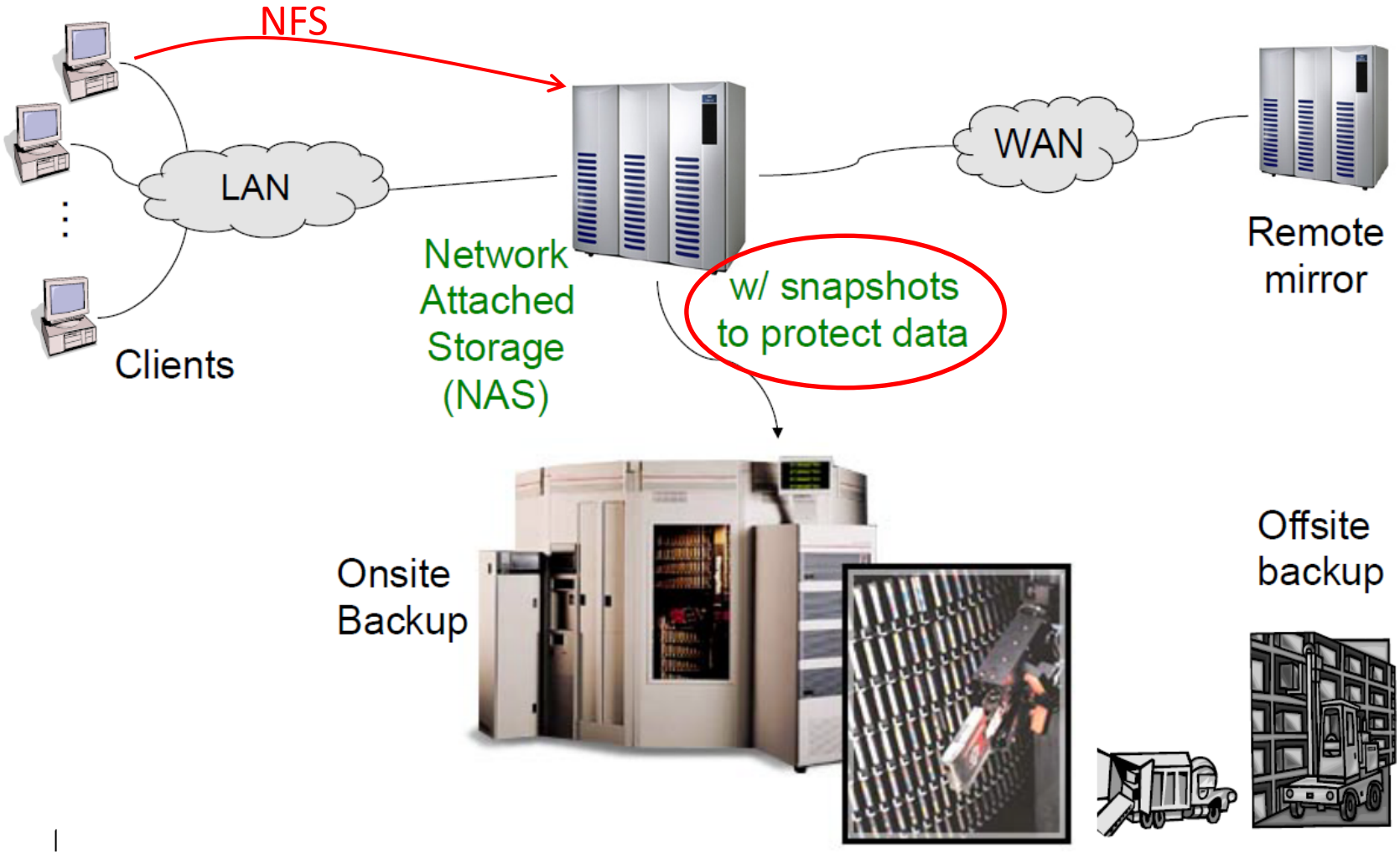
NFS的影响力：企业级存储

- 企业数据中心： SAN架构



NFS的影响力：企业级存储

- 企业级数据中心：NAS架构（快照）



内容提要

- NFS
- WAFL
- GFS
- 安全保护

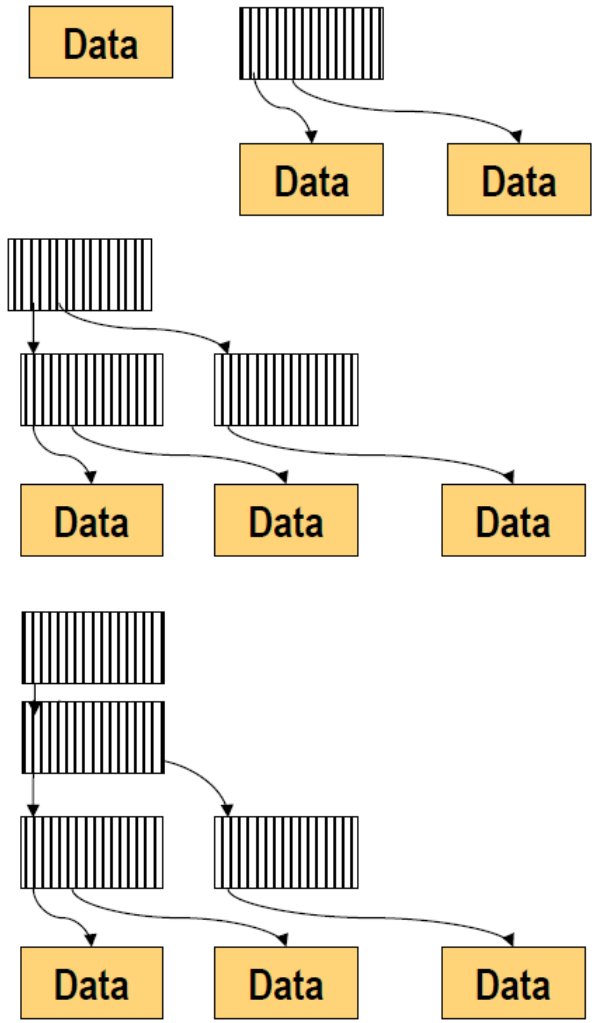
NetApp的NFS文件服务器

- WAFL：Write Anywhere File Layout [1]
  - NetApp为其NFS产品（FAS）设计的文件系统
- 设计目标
  - 请求服务速度快：吞吐率(op/s)更多，I/O带宽更高
  - 支持大文件系统，且文件系统不断增长
  - 高性能软件RAID
  - 宕机后快速恢复
- 独特之处
  - 磁盘布局受LFS启发
  - 引入快照
  - 使用NVRAM记录日志（写前日志）

[1] D. Hitz, J. Lau, M. Malcolm, **File System Design for an NFS File Server Appliance**, USENIX Winter Conference, 1994

i-node、间址块和数据块

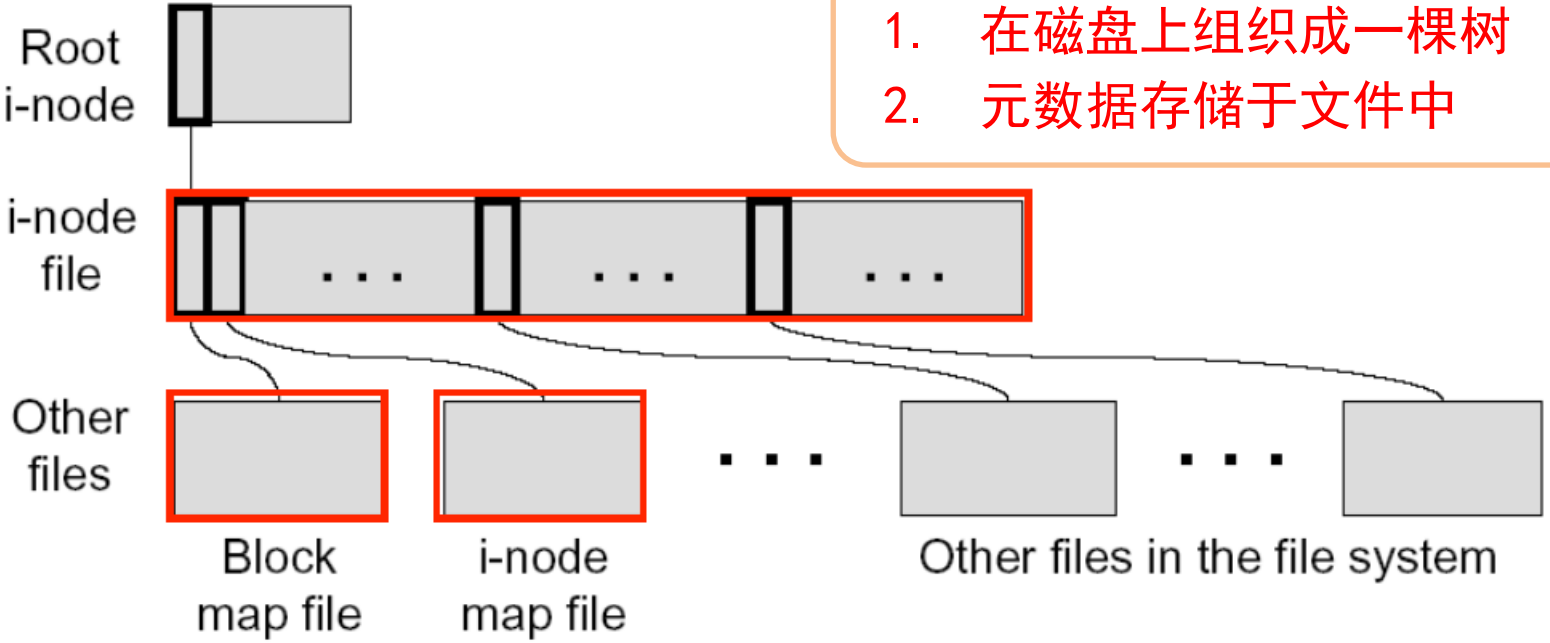
- WAFL使用4KB块
  - i-node: 借鉴UNIX FS
  - 16个指针（64B）用于文件块索引
- 文件大小 ≤ 64B
  - 文件数据直接存储在 i-node中
- 文件大小 ≤ 64KB
  - i-node存储16个指向数据块的指针
- 文件大小 ≤ 64MB
  - i-node存储16个指向间址块的指针
  - 每个间址块存储1024个指向数据块的指针
- 文件大小 > 64MB
  - i-node存储16个指向二级间址块的指针



二级间址能索引的最大文件有多大?

WAFL的磁盘布局

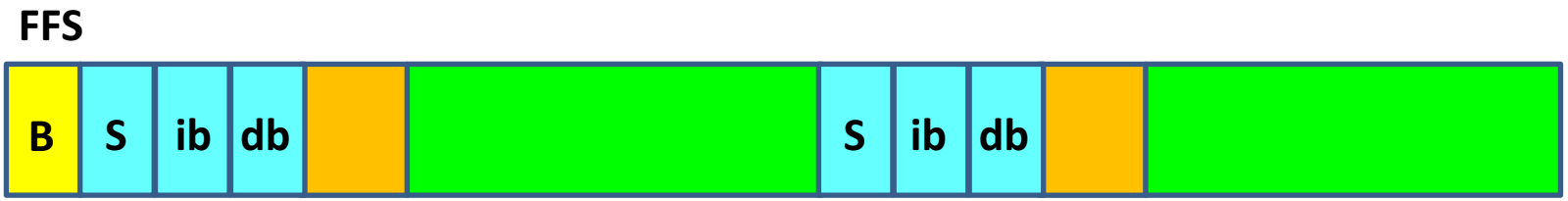
- 主要数据结构
  - 一个根i-node：整个FS的根
  - 一个i-node file：包含所有i-node
  - 一个block map file：指示所有空闲块
  - 一个i-node map file：指示所有空闲i-node



1. 在磁盘上组织成一棵树
2. 元数据存储于文件中

为什么将元数据存储于文件中

- 元数据块可以写在磁盘上任何位置
  - 这是“WAFL”名字的由来，Write Anywhere File Layout
  - 性能上有好处吗？
- 使得动态增加文件系统的大小变得容易
  - 增加一个磁盘引发i-node个数的增加
  - 将卷管理的功能集成到WAFL中
- 能够通过copy-on-write(COW)来创建快照
  - 新的数据和元数据都可以COW写到磁盘上的新位置
  - 固定元数据位置无法COW



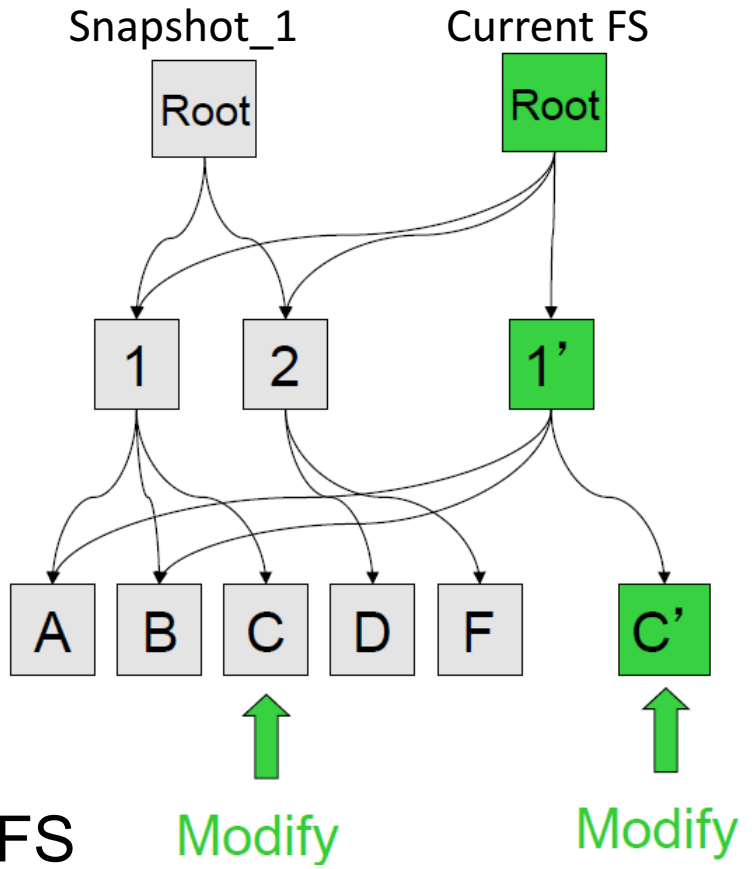
快照 (snapshot)

- 快照是文件系统的一个只读版本
  - 1993年提出
  - 成为文件服务器必备特性
- 快照用法
  - 系统管理员配置快照的个数和频率
  - 最初系统能支持20个快照
  - 用快照恢复其中任何一个文件
- 一个例子

```
% cd .snapshot
% ls
hourly.0  hourly.2  hourly.4  nightly.0  nightly.2  weekly.1
hourly.1  hourly.3  hourly.5  nightly.1  weekly.0
%
```

快照的实现

- WAFL：所有的块构成一棵树
- 创建快照
  - 复制根i-node
  - 新的根i-node指向活跃FS
  - 旧的根i-node指向快照
- 创建快照之后
  - 第一次写一个块: 把从它到根的数据块都复制(COW)
  - 活跃FS的根i-node指向新数据块
  - 写数据块
  - 以后对这些数据块的写不再触发COW
- 每个快照都是一个一致状态的只读FS



WAFL快照占用多少额外空间?

在传统文件系统(FFS)上能够实现文件系统快照功能吗?

文件系统一致性

- 定期创建一致点
  - 每10秒创建一个一致点
  - 特殊的内部快照，用户不可见
- 在一致点之间的多个请求
  - 第i个一致点
  - 若干写操作
  - 第i+1个一致点（自动增长）
  - 若干写操作
  - ……
- 宕机恢复
  - 将文件系统恢复到最后一个一致点
  - 最后一个一致点之后到宕机前的写操作：靠日志进行恢复

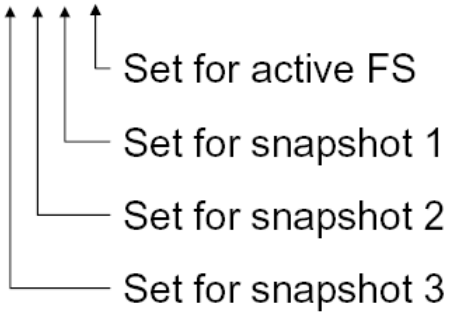
非易失RAM（Non-Volatile RAM）

- NVRAM
  - 闪存：写比较慢 vs. NVRAM
  - 带电池的DRAM：快
    - 电池容量有限，持续时间不长
    - DRAM容量有限
- 日志写入NVRAM
  - 记录自上一个一致点以来的所有写请求
  - 正常关机：先停止NFS服务，再创建一个快照，然后关闭NVRAM
  - 宕机恢复：用NVRAM中的日志来恢复从最后一个一致点以后的修改
- 使用两个日志
  - 一个日志写回磁盘时，另一个日志写入NVRAM中缓冲

快照数据结构

- Block map file
  - 每个4KB磁盘块
    对应一个32-位的表项
  - 表项值为0：
    该块为空闲块
  - 第0位=1：
    该块属于活动文件系统
  - 第1位=1：
    该块属于第一个快照
  - 第2位=1：
    该块属于第二个快照
  - ……

Time	Block map entry	Description
T1	0 0 0 0 0 0 0 0	Block is free
T2	0 0 0 0 0 0 0 1	Active FS uses it
T3	0 0 0 0 0 0 1 1	Create snapshot 1
T4	0 0 0 0 0 1 1 1	Create snapshot 2
T5	0 0 0 0 0 1 1 0	Active FS deletes it
T6	0 0 0 0 0 1 0 0	Delete snapshot 1
T7	0 0 0 0 0 0 0 0	Delete snapshot 2



快照创建

- 问题
  - 正在创建快照时，可能有很多NFS请求到来
  - 文件缓存可能需要写回
  - 不希望NFS被长时间被挂起不处理请求
- WAFL的解决方案
  - 在创建快照前，将块缓存中的脏块标记为“in-snapshot”
  - 所有对“in-snapshot”缓存块的修改请求被挂起
  - 没有标记为“in-snapshot”的缓存数据可以修改，但不能刷回磁盘

创建快照

- 步骤
  - 为所有“in-snapshot”的文件分配磁盘空间
    - 将i-node缓存中的脏 i-node写回至块缓存
    - 避免每一个写请求分配一次磁盘空间
  - 更新 block map file
    - 对每个表项，将活动FS位的值 拷贝到 新快照位
  - 刷回
    - 把所有“in-snapshot” 缓存块写到它们新的磁盘位置
    - 每写回一个块，restart它上面被挂起NFS请求
  - 复制根i-node
- 性能
  - 通常需要不到1秒钟

快照删除

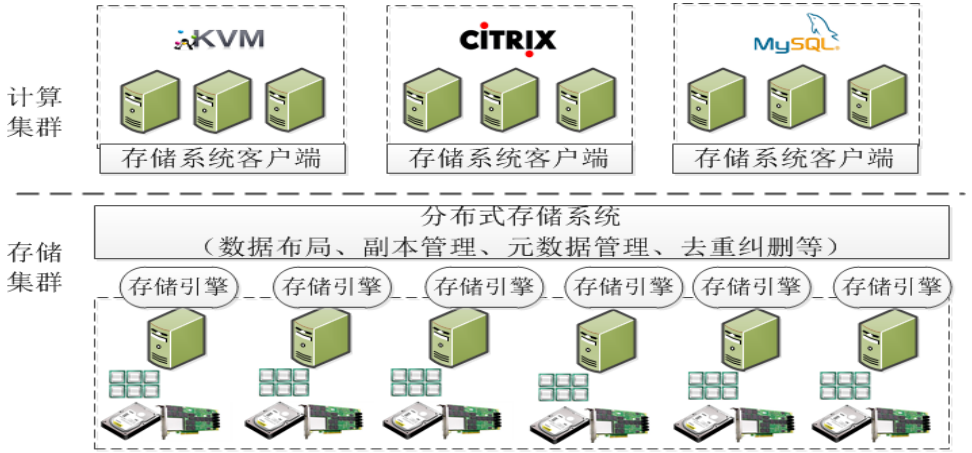
- 删除快照的根i-node
- 清除block map file中的位
  - 对于block map file的每一个表项，清除与该快照对应的位

内容提要

- NFS
- WAFL
- GFS
- 安全保护

Google File System

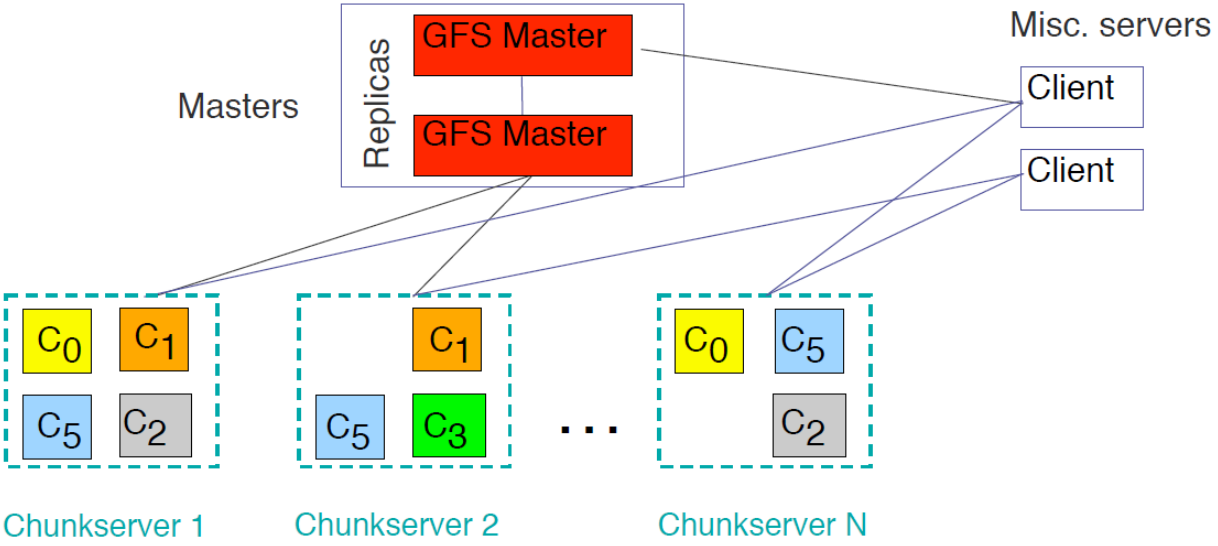
- 分布式文件系统



- 逻辑上，三类组件
  - 客户端提供库函数接口: 用户态
  - 大量Chunk Server (CS)：存储服务器，存储GFS文件
  - 单一Master：存储元数据和维护名字空间
- 物理上，两类机器
  - Master是集群中专门一台机器
  - 集群中所有其它机器，既为客户端 ，又为CS服务器

Google File System

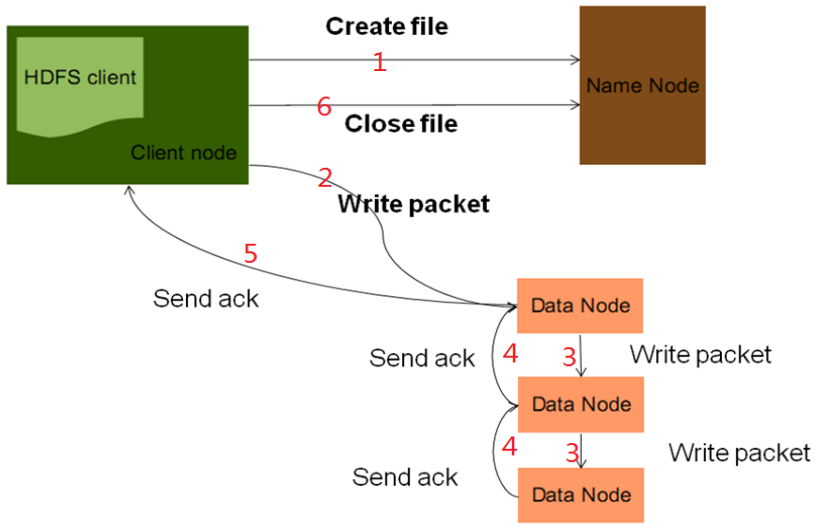
- 文件Chunk
  - 文件划分为固定长度的Chunk：默认为64MB
  - 以Chunk为粒度存储在多个CS服务器
    - CS服务器用一个本地文件来存储一个Chunk
  - 每个Chunk保存多个复本（默认为3）@不同CS
    - 容错：可用性 & 可靠性
    - 写文件：同时写多份，强一致性
    - 读文件：选择任意一份 → 更高的读带宽



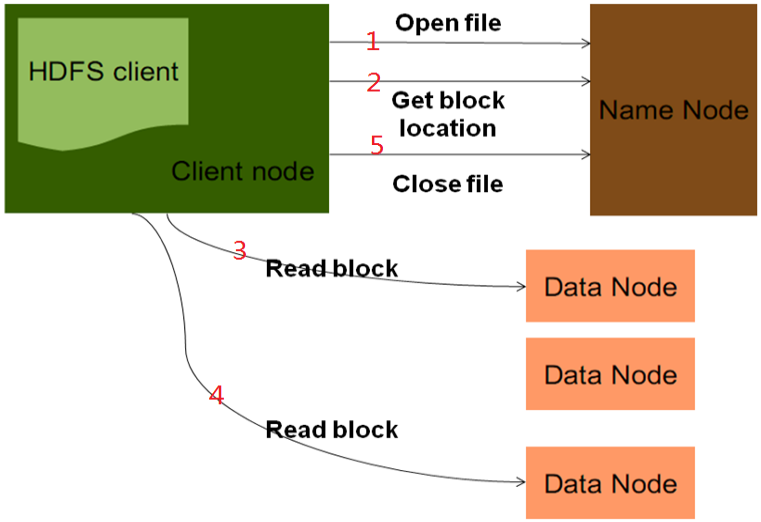
GFS：概述

- 文件读写

写文件



读文件



内容提要

- NFS
- WAFL
- GFS
- 安全保护

安全与保护（Security & Protect）

- 数据机密性：未经许可，不能看到数据
  - 任何用户不能读写其他用户的文件
- 数据完整性：未经许可，不能修改或删除数据
  - 数据在网络传输过程中被拦截和修改，可以采用加密
- 系统可用性：干扰系统使得它不可用
  - 给一个服务器发送大量的请求

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service

保护：策略与机制

- 安全策略：定义目标，即要达到的效果
  - 通常是一组规则，定义可接受的行为和不可接受的行为
  - 例子
    - /etc/password文件只有root能写
    - 每个用户最多只能用50GB的磁盘空间
    - 任何用户都不允许读其他用户的mail文件
- 机制：用什么样的方法来达到目标

保护机制

- Authentication（身份认证）
  - 验明身份
    - UNIX: 密码/口令
    - 类比机场：身份证或护照
- Authorization（授权）
  - 决定“A是不是准许做某件事”
  - 通常使用角色（role）定义授予的操作权限，使用简单的数据库保存角色定义
- Admission control（访问控制）
  - 做出“访问是否准许”的决定
  - 有时和系统承载压力相关联，系统负载高时，进行访问控制

身份认证

- 通常是用密码来验证
  - 一串字符（字母+数字）
  - 用户必须记住密码
- 密码是以加密形式存储
  - 使用一种单向的“安全hash”算法
- 缺点
  - 每个用户都要记很多密码
  - 比较弱，“dictionary attack”

LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

使用Salt机制

Bobbie, 4238, e(Dog, 4238)
Tony, 2918, e(6%%TaeFF, 2918)
Laura, 6902, e(Shakespeare, 6902)
Mark, 1694, e(XaB#Bwcz, 1694)
Deborah, 1092, e(LordByron,1092)

Salt Password

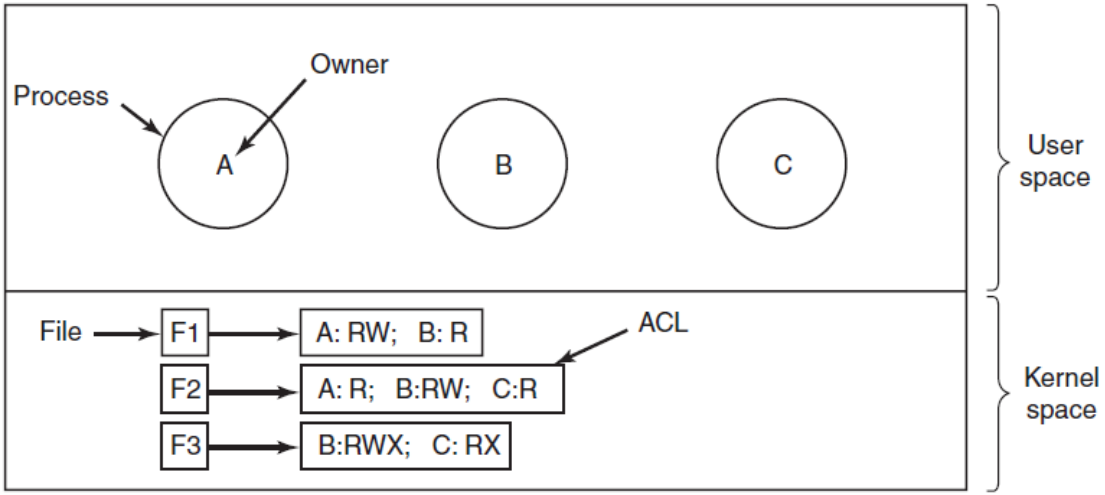
保护域

- 规则：每个身份准许做哪些事
  - Alice准许做什么
  - Bob准许做什么
  - …
- 保护矩阵：保护域 vs 保护源

	File A	Printer B	File C
Domain 1	R	W	RW
Domain 2	RW	W	…
Domain 3	R	…	RW

访问控制表（ACL）

- 每个对象有一个ACL表
  - 定义每个用户的权限
  - 每个表项为 <user, privilege>
- 简单，大多数系统都采用
  - UNIX的owner, group, other
- 实现
  - ACL实现在内核中
  - 在登录系统时进行身份验证
  - ACL存储在每个文件中或文件元数据中
  - 打开文件时检查ACL

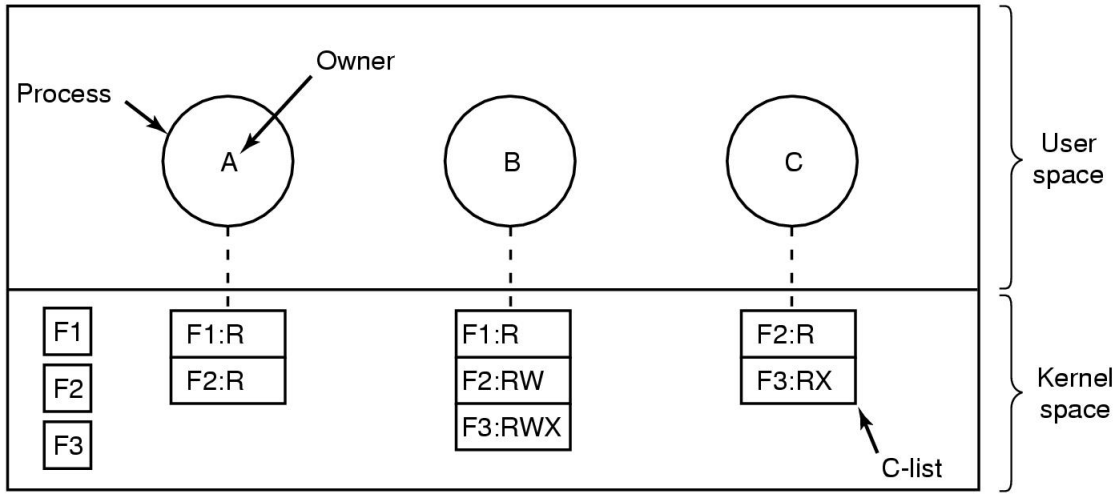


访问控制

- 需要一个可信权威
  - 进行访问控制
  - ACL或权能表都需要保护
- 内核是一个可信权威
  - 内核什么事可以做
  - 如果有bug，整个系统都可能被破坏
  - 它越小、越简单越好
- 安全的强度由保护系统链上最薄弱的环节决定

Capabilities

- 每个用户有一个权能表（capability list）
  - 定义有权访问的对象（例如文件、端口、内存范围等），及访问权限
  - 每个表项为 <object, privilege>
- 能保护对象的内容，也能保护对象的名字
  - 用户只能看到他有权限的对象
- 实现
  - 需要体系结构的支持
  - 权能表保存在内核
  - 权能表也可使用加密形式保存在用户空间



一些简单的攻击

- 滥用合法权利
  - UNIX：root能做任何事情
    - 例如：读你的mail文件, 以你的身份发送email, 把你的邮箱删除, ……
- 拒绝服务（DoS）
  - 耗尽系统所有资源
  - 例如
    - 运行一个shell脚本： “while (1) {mkdir foo; cd foo; }
    - 运行一个C程序： “while (1) { fork(); malloc (1000)[40]=1;}
- 偷听
  - 侦听网络上传输的包

总结

- NFS
  - 无状态协议
  - File Handle
  - 客户端和服务器都有缓存
  - 客户端缓存一致性问题
  - 服务器缓存丢数据问题
- WAFL
  - 支持在任意位置上写的磁盘布局（受LFS的影响）
  - 快照成为存储产品的必备特性
  - COW
  - 使用NVRAM来加速写日志

总结

- GFS
  - 单一Master管理整个名字空间和所有元数据：全部放内存
  - 多个Chunk Server存储文件：每个文件划分为固定粒度的Chunk
  - 每个Chunk存储多个复本，自动复制、自动容错
  - 复本一致性

- 数据保护：攻击和误操作
  - 数据机密性、数据完整性、系统可用性
  - 基于密码/口令的身份认证
  - ACL和Capability

计算机系统的发展与研究

- 操作系统
  - 提供抽象
  - 管理资源
- 研究方向
  - 极致性能
    - 用户态 vs. libos vs. 微内核
  - 系统规模
    - 云计算分布式系统 vs. IoT系统
  - 硬件异构与新型硬件
    - 超高速网络、持久化内存、众核
  - 新型应用
    - AI for system
    - System for AI