北京邮电大学计算机学院

网络存储技术

大作业报告

班级：2020211310

学号：2020211616

姓名：付容天

《网络存储技术》期末大作业
五种常见网络存储技术架构的陈述

付容天 学号 2020211616

班级 2020211310

计算机科学与技术系

计算机学院（国家示范性软件学院）

2022 年 12 月 20 日

目录

1 引言

本学期我学习了若干常见的网络存储技术，现在我将介绍NAS、SAN、HDFS、Ceph 和 NDB 五种技术，并在陈述其基本概念的同时，分析其优缺点，并分别给出各自的应用案例介绍与分析。

2 NAS 技术

NAS（Network Attached Storage，网络附加存储）技术是一种将分布、独立的数据整合为大型、集中化管理的数据中心、从而便于不同主机和应用对服务器进行访问的文件级技术。简单来讲，NSA 就是一种“将文件数据存储在网络上供主机进行存取和处理”的技术。

NAS 可以被定义为一种特殊的专用数据存储服务器，其最重要的功能就是跨平台文件共享功能。NAS 通常在一个 LAN 上占有自己的节点，无需其他应用服务器的干预，并允许用户从该网络上进行数据的存取、更新、删除等操作。在这种配置中，NAS 集中管理和处理所在局域网上的所有数据，有效降低数据管理和共享的成本。

分析可知，NAS 的主要特点为以下两点：

- 专用目的：NAS 通常占用所在 LAN 上的一个专用节点，从而对其他其他主机提供数据服务。从此角度来看，我们可以认为 NAS 服务器是一种专用的数据服务器，它可以授权网络用户和异质客户端从集中位置存储和检索数据。很容易看出，NAS 具有灵活和易于横向扩展的特点。并且，NAS 的硬件和软件基础也通常是专用的，其硬件、软件和配置共同构成其颇具特色的文件服务。NAS 的实现基础通常包含一个或多个通常排列成逻辑存储器、冗余存储器或 RAID 存储驱动器的网络设备
- 文件级存储：我们知道，NAS 技术是一种“文件级存储”的技术，也就是说文件存储的用户是自然人，因此数据需要按照不同应用要求的结构组成不同类型的文件（可以用不同后缀指代不同类型的文件），并且所有的文件和目录共同构成一个树状结构。文件级存储方式的一大优点就是便于共享，相反地，如果采用块级存储，那么由于不同主机上的文件系统往往具有不同的数据格式化方式，因此不同文件系统间的文件是难以进行共享的。文件级存储方式本身具有文件的管理功能，因此不需要主机系统再对文件进行格式化，相当于引入了一种“中间件”来进行文件的统一管理并提供对外的统一接口

下面来看 NAS 的常见应用。

2.1 AFS 技术

Andrew File System（AFS）是一种分布式的文件系统，它使用一组服务器向所有客户端提供概念上完全相同的存储空间，AFS 技术在分布式计算领域具有广泛的应用。

AFS 的最大特点是安全性和可扩展性：

- 安全性：AFS 使用 Kerberos 协议对用户身份进行验证，该协议允许通过非安全网络进行通信的节点以安全的方式验证彼此的身份（相互验证），并且要求一个受信任的第三方来辅助处理。也就是说，AFS 中被选择的那一组服务器必须是受信任的，从而使 Kerberos 协议可以正常工作
- 可扩展性：AFS 能够很容易地支持数百个节点，甚至数千个节点的分布式环境。同时，在大规模的分布式文件系统中，AFS 利用本地存储作为分布式文件的缓存，在远程文件无法访问时，依然可以部分工作，这无疑提高了系统可用性

在 AFS 技术中，卷是由管理员创建并且链接到 AFS 单元中的特定命名路径，卷的物理位置对文件系统的普通用户是不重要的（不可见的），用户可以正常创建目录和文件。一个卷可能有一个配额，以限制其空间消耗量，管理员可以改变这个配额、甚至可以改变卷的存储位置。

对于存储空间而言，AFS 系统将其划分为共享名称空间和本地名称空间。共享名称空间在所有工作站上都是相同的，而本地名称空间则对于每个工作站而言是唯一的。

在修改文件时，AFS 则采用了弱一致模型：用户在打开文件后，将文件缓存在本地文件系统上，对打开文件的更新操作仅对本地副本进行修改，当修改后的文件关闭时，修改的部分才会被复制回文件服务器。

AFS 可以对不同用户进行以下授权：

- Lookup(l)：允许用户列出 AFS 系统的文件目录、访问子目录和进行检索
- Insert(i)：允许用户向存储系统中添加新文件或创建新的子目录
- Delete(d)：允许用户删除文件或子目录
- Administrator(a)：给一个用户在对应目录下授予管理员权限
- Read(r)：允许用户查看目录中文件的内容并列出子目录中的文件
- Write(w)：允许用户修改目录中的文件

2.2 AFP 技术

Apple Filing Protocol（AFP）是一种专有网络协议，是 Apple 文件服务的一部分，主要用于控制访问远程文件系统，典型应用程序是 AppleShare。

AFP 主要由三个系统组件构成：

- 文件系统结构：由可通过网络寻址的资源（如文件服务器、卷、目录、文件和分支等）组成，这些资源称为 AFP 文件系统的可见实体，AFP 可以规定这些实体之间的关系，例如，一个目录可以是另一个目录的父目录
- AFP 命令：这是本地计算机用于操作 AFP 文件系统结构的命令，支持以文件服务器的形式发送文件系统命令，或者在本地计算机上运行的应用程序上直接执行 AFP 命令
- 与命令相关联的算法：用于指定 AFP 命令执行的操作

在访问文件时，用户通过本地计算机上运行的程序向本地文件系统发送命令。然后，由于本地存储器中的数据结构可以指示某些卷是由本机文件系统还是由外部文件系统管理，因此本地文件系统可以通过查看此数据结构来发现请求的文件是在本地还是远程访问。如果数据结构指示外部文件系统，则本地文件系统将该命令路由到 AFP 转换器（在 AFP 3.0 及更高版本中，使用 TCP/IP 协议经由端口 548 建立通信）。

2.3 SMB 技术

Server Message Block（SMB）是一种采用 TCP/IP 协议作为底层传输协议的通信协议，旨在提供经管身份验证的 IPC 机制并支持计算机网络上的文件共享、打印机共享、网络浏览和进程间管道通信等功能。SMB 是 Microsoft 分布式文件系统实现的基础。

2.3.1 发展历史

Barry Feigenbaum 最初于 1983 年初在 IBM 设计了 SMB，旨在将本地文件访问转变为网络文件系统，微软随后在 LAN Manager 操作系统中实现了 SMB 协议，这就是 SMB 1.0 协议。SMB 1.0 协议最初设计为在基于 IEEE 802.2 的 NetBIOS 帧上运行。从 Windows 2000 开始，SMB 使用 TCP 端口 445 在 TCP 上运行，该功能称为“直接主机 SMB”。

微软在 2006 年推出了 SMB 2.0 协议，该协议用于 Windows Vista 和 Windows Server 2008。SMB 2.0 通过将命令和子命令的数量从一百多个减少到十九个，从而大幅优化了性能。并且，它支持流水线机制，从而提高了高延迟链路的性能。SMB 2.0 还包括对符号链接的支持。其他改进包括文件属性的缓存、使用 HMAC SHA-256 散列算法改进的消息签名以及通过增加每台服务器的用户、共享和打

开文件的数量等来实现更好的可扩展性。SMB 2.0 使用 32 位或 64 位宽的存储字段，并在文件句柄的情况下允许使用 128 位，从而消除了 SMB 1.0 对块大小的限制，提高了通过快速网络传输大文件的性能。

SMB 2.1 随 Windows 7 和 Windows Server 2008 R2 发布，主要优化在于支持弹性句柄和多协议版本的协商机制，这些优化带来了性能增强。

SMB 3.0（以前称为 SMB 2.2）是随 Windows 8 和 Windows Server 2012 发布的，它带来了一些非常重要的新特性，主要包括：

- SMB 透明故障转移：让管理员可执行群集文件服务器中节点的硬件或软件维护，且不会中断将数据存储在这些文件共享上的服务器应用程序。此外，如果群集节点出现硬件或软件故障，SMB 客户端将以透明方式重新连接到其他群集节点，且不会中断将数据存储在这些文件共享上的服务器应用程序。即客户端能够持续、稳定的对远程文件服务器进行通讯，用户不会感受到单点服务器故障所带来的性能影响
- SMB 横向扩展：可构建横向扩展文件服务器（Scale-Out File Server），在使用群集共享卷（CSV）时，管理员可以通过文件服务器群集中所有节点，创建可供同时访问含直接 I/O 的数据文件的文件共享，这可以更好地利用文件服务器客户端的网络带宽和负载平衡，以及优化服务器应用程序的性能
- SMB 多通道：如果在 SMB 3.0 客户端及服务器之间提供多条路径，则支持网络带宽和网络容错的聚合，提升了网络可用性及文件服务器的稳定性，并让服务器应用程序可以充分利用可用网络带宽，以及在发生网络故障时快速恢复
- SMB 直接访问（SMB over Remote Direct Memory Access，即 RDMA）：支持使用具有 RDMA 功能且可全速运行的网络适配器，其中延迟非常低且 CPU 利用率极少，对于 Hyper-V 或 Microsoft SQL Server 等实现工作负载，这让远程服务器如同本地存储一般
- 用于服务器应用程序的性能计数器：全新 SMB 性能计数器提供有关吞吐量、延迟和 IOPS 的按共享列出的详细信息，从而让管理员可以分析用于存储数据的 SMB 3.0 文件共享的性能，这些计数器是为了将文件存储在远程文件共享上的服务器应用程序而设计的，如 Hyper-V 和 SQL Server
- 性能优化：SMB 3.0 客户端和 SMB 3.0 服务器均已针对小型随机读、写和 I/O 操作进行了优化，此外，默认情况下支持大型最大传输单元（MTU），这将大幅提高大型连续传输性能，如 SQL Server 数据仓库、数据库备份或还原、部署或复制虚拟硬盘
- SMB 加密：提供 SMB 数据的端对端加密并防止数据在未受信任网络中遭受窃听。无需新部署成本，且无需 Internet 协议安全性（IPsec）、专用硬件或 WAN 加速器，使用 AES CCM 128 位加密算法，它可按共享配置，也可针对整个文件服务器配置，并且可针对数据遍历未受信任网络的各种方案启动

SMB 3.0.2 是随 Windows 8.1 和 Windows Server 2012 R2 发布的，在该版本及更高版本中，可以选择禁用早期的 SMB 1.0 以提高安全性。并且，该版本的 SMB 支持客户端直读直写请求，且对 RDMA 进行了性能改进。

SMB 3.1.1 是随 Windows 10 和 Windows Server 2016 发布的。该版本除了支持 SMB 3.0 中添加的 AES CCM 128 加密外，还支持 AES GCM 128 加密，并使用 SHA-512 哈希实现预认证完整性检查。当连接到使用 SMB 2.0 及更高版本的客户端时，SMB 3.1.1 还强制进行安全协商。

SMB 具有一些第三方的实现，包括 Samba、Netsmb、NQ、MoSMB、Fusion File Share by Tuxera、Likewise 和 CIFSD 等。

2.3.2 实现方案

在具体的传输实现上，SMB 常有两种实现方式：

- 直接运行在 TCP 端口 445 上
- 最初运行在支持 IEEE 802.2 和 IPX/SPX 协议的 NetBIOS 上，后来运行在支持 TCP/IP 协议的 NetBIOS 上（NetBT），TCP/IP 协议的引入使复杂互联网络（尤其是 Internet）上的数据传输成为可能。NetBT 上 SMB 的服务器组件使用三个 TCP 或 UDP 端口：
  - 137：NetBIOS 名称服务
  - 138：NetBIOS 数据报服务
  - 139：NetBIOS 段服务

在具体的系统实现上，例如在 Windows 中，ID 为 LanmanServer 的服务和 ID 为 LanmanWorkstation 的服务共同实现了 SMB，其中，前者负责服务共享资源，后者负责维护计算机名称并帮助访问其他计算机上的共享资源。在安全方面，SMB 通常采用 Kerberos 协议来对用户身份进行核对（但在简单对等网络上往往使用 NTLM 协议进行身份验证）。

2.3.3 Opportunistic Locking

SMB 的一大特点就是支持 Opportunistic Locking。Opportunistic Locking 旨在通过控制客户端对网络文件的缓存来提高性能。为了实现对文件的控制，常见的思路是使用文件锁，但 OpLock 并不是严格意义上的文件锁，因为它并不是被用来提供文件访问的严格互斥特性的，具体来讲，OpLock 常有下面四种类型：

- Batch Lock：最初是为了支持 DOS 批处理文件执行操作的特定行为而创建的，当某文件在短时间内打开和关闭多次时，性能消耗过大。为了解决这个问题，可以创建一个 Batch Lock，从而使客户端延迟发送文件关闭请求，

如果后续存在文件打开请求，则两个请求相互取消

- Level-1 OpLock / Exclusive Lock：当应用程序以“共享模式”打开存储在 SMB 服务器、但未被任何其他进程（或其他客户端）打开的文件时，客户端会从服务器接收一个 Exclusive Lock。这意味着客户端现在可以假设它是唯一可以访问此特定文件的进程，并且客户端可以缓存对文件的所有更新，并在提交文件到服务器时再进行更新。通过 Exclusive Lock，文件读取和写入文件所需的往返次数更少。如果另一个客户端或进程试图打开同一个文件，服务器便会向客户端发送一条消息（称为 break 或 revocation），这会使先前提供给客户端的 Exclusive Lock 无效，然后客户端将刷新对文件的所有更改
- Level-2 OpLock：对于共享文件而言，如果客户端持有 Exclusive Lock，但是第三方打开该文件，则客户端必须放弃其 Exclusive Lock 以允许其他客户端的读写访问。这时，客户端可能会从服务器接收一个 Level-2 OpLock，该锁允许缓存读取请求，但不允许写入缓存
- Filter OpLock：该锁是在 Windows NT 4.0 中新加入的锁，类似于 Level-2 OpLock，但可以有效防止文件打开和锁定接收之间的共享模式冲突

2.3.4 性能表现

SMB 协议的实际性能表现通常与网络上广播流量有强联系，广播流量的增加通常会导致问题。注意，虽然 SMB 协议本身并不使用广播来实现服务，但是 Windows NT 4.0 服务器常常使用 NetBIOS 协议来定期广播特定主机上可用的服务来发挥作用。在主机数量较少的网络中这不会导致大的问题，但随着网络上主机数量的不断增加，一些由过多的广播流量导致的问题便会出现。

此外，研究发现延迟对 SMB 1.0 协议的性能有重大影响，使得它的性能比 FTP 等其他协议更差。例如，Internet 上的 VPN 连接通常会引入网络延迟，这是因为 SMB 1.0 是块协议而非流协议，最初仅是为小型 LAN 设计的，它的块大小限制为 64K，SMB 签名会产生额外的开销，并且 TCP 窗口大小未针对 WAN 链接进行优化。SMB 2.0 协议和 TCP 窗口缩放等技术就是为了解决这个问题而被提出的。

2.4 FTP 和 SFTP 技术

2.4.1 FTP

File Transfer Protocol（FTP）是一种标准通信协议，它工作在 OSI 模型的第七层，TCP 模型的第四层，用于将计算机文件从服务器传输到计算机网络上的客户端。FTP 建立在客户端-服务器（C/S）模型架构之上，在客户端和服务器之间使用单独的控制和数据连接，且在建立连接前要经过一个“三次握手”的过程，

保证客户与服务器之间的连接是可靠的。FTP 是面向连接的，这是为了保证数据传输的可靠性。

通常而言，FTP 用户可以使用明文登录协议对自己进行身份验证，例如采用用户名和密码的形式。但如果服务器配置允许，也可以匿名连接。为了保护用户名和密码并加密内容的安全传输，FTP 通常使用 SSL/TLS（FTPS）或替换为 SSH 文件传输协议（即 SFTP）。

FTP 允许用户以文件操作的方式（如文件的增、删、改、查、传送等）与另一主机相互通信。然而，用户并未真正登录到目标计算机上面而成为完全用户，而是使用 FTP 程序访问远程资源，从而实现用户往返传输文件、目录管理以及访问电子邮件等功能。

特别地，使用 FTP 方式进行远程访问可以屏蔽双方计算机在操作系统和文件存储方式方面的差异。

FTP 采用 Internet 标准文件传输协议，并向用户提供了一组用来管理计算机之间文件传输的应用程序。开发任何基于 FTP 的客户端软件都必须遵循 FTP 的工作原理，FTP 是基于 C/S 模型而设计的，在客户端与 FTP 服务器之间需要建立两个连接，这正是 FTP 的独特的优势、同时也是与其它客户服务器程序最大的不同点：它在两台通信的主机之间使用了两条 TCP 连接：

- 一条是数据连接，用于数据传送
- 另一条是控制连接，用于传送控制信息（命令和响应）

这种将命令和数据分开传送的思想大大提高了 FTP 的效率，而其它客户服务器应用程序一般只有一条 TCP 连接。

在 FTP 中，客户有三个构件：用户接口、客户控制进程和客户数据传送进程。服务器有两个构件：服务器控制进程和服务器数据传送进程。在整个交互的 FTP 会话中，控制连接始终是处于连接状态的，数据连接则在每一次文件传送时先打开后关闭。

在具体实现时，FTP 有两种模式：

- 主动模式（又称 Standard 方式、PORT 方式）：要求客户端和服务器端同时打开并且监听一个端口以创建连接，在这种情况下，安装了防火墙的客户端可能会有一些问题
- 被动模式（又称 Passive 方式、PASV 方式）：只要求服务器端产生一个监听相应端口的进程，这样就可以绕过客户端安装了防火墙的问题

主动模式下，FTP 连接创建需要遵循以下步骤：

1. 客户端打开一个随机的端口（端口号大于 1024，比如 x），同时一个 FTP 进程连接至服务器的 21 号命令端口。此时，该 TCP 连接的来源地端口为客户端指定的随机端口 x，目的地端口（远程端口）为服务器上的 21 号端口

2. 客户端开始监听端口 $x+1$，同时通过服务器的 21 号命令端口向服务器发送一个端口命令，此命令告诉服务器客户端正在监听的端口号并且已准备好从此端口接收数据，该端口就是我们所知的数据端口

3. 服务器打开 20 号源端口并且创建和客户端数据端口的连接。此时，来源地的端口为 20，远程数据（目的地）端口为 $x+1$

4. 客户端通过本地的数据端口创建一个和服务器 20 号端口的连接，然后向服务器发送一个应答，告诉服务器它已经创建好了一个连接

进行数据传输时，FTP 有以下两种方式：

- ASCII 传输方式：假定用户正在拷贝的文件包含的简单 ASCII 码文本，如果在远程机器上运行的不是 UNIX，则在文件传输时，FTP 通常会自动地调整文件的内容以便于把文件解释成另外那台计算机存储文本文件的格式。但是如果用户正在传输的文件包含的不是文本文件，而是程序，数据库，字处理文件或压缩文件等文件的时候，用户需要在拷贝任何非文本文件之前，用 binary 命令告诉 FTP 逐字拷贝
- 二进制传输模式：在二进制传输中，保存文件的位序，以便原始和拷贝的是逐位一一对应的，即使目的地机器上包含位序列的文件是没意义的

FTP 支持许多命令，例如：

命令	**RFC**	**描述**
ABOR		Abort an active file transfer
ACCT		Account information
ADAT	RFC 2228	Authentication/Security Data
ALLO		Allocate sufficient disk space to receive a file
APPE		Append (with create)
AUTH	RFC 2228	Authentication/Security Mechanism
AVBL		Streamlined FTP Command Extensions Get the available space
CCC	RFC 2228	Clear Command Channel
CDUP		Change to Parent Directory
CONF	RFC 2228	Confidentiality Protection Command
CSID		Streamlined FTP Command Extensions C/S Identification
CWD	RFC 697	Change working directory
DELE		Delete file
DSIZ		Streamlined FTP Command Extensions Get the directory size
ENC	RFC 2228	Privacy Protected Channel

命令	RFC	描述
EPRT	RFC 2428	Specifies an extended address and port to which the server should connect
EPSV	RFC 2428	Enter extended passive mode
FEAT	RFC 2389	Get the feature list implemented by the server
HELP		Returns usage documentation on a command if specified, else a general help document is returned
HOST	RFC 7151	Identify desired virtual host on server, by name
LANG	RFC 2640	Language Negotiation
LIST		Returns information of a file or directory if specified, else information of the current working directory is returned
LPRT	RFC 1639	Specifies a long address and port to which the server should connect
LPSV	RFC 1639	Enter long passive mode
MDTM	RFC 3659	Return the last-modified time of a specified file
MFCT	The 'MFMT', 'MFCT', and 'MFF' Command Extensions for FTP	Modify the creation time of a file
MFF	The 'MFMT', 'MFCT', and 'MFF' Command Extensions for FTP	Modify fact (the last modification time, creation time, UNIX group/owner/mode of a file)
MFMT	The 'MFMT', 'MFCT', and 'MFF' Command Extensions for FTP	Modify the last modification time of a file
MIC	RFC 2228	Integrity Protected Command
MKD		Make directory
MLSD	RFC 3659	Lists the contents of a directory in a standardized machine-readable format
MLST	RFC 3659	Provides data about exactly the object named on its command line in a standardized machine-readable format
MODE		Sets the transfer mode (Stream, Block, or Compressed)
NLST		Returns a list of file names in a specified directory
NOOP		No operation (dummy packet; used mostly on keepalives)
OPTS	RFC 2389	Select options for a feature (for example OPTS UTF8 ON)
PASS		Authentication password
PASV		Enter passive mode

命令	RFC	描述
PBSZ	RFC 2228	Protection Buffer Size
PORT		Specifies an address and port to which the server should connect
PROT	RFC 2228	Data Channel Protection Level
PWD		Print working directory. Returns the current directory of the host
QUIT		Disconnect
REIN		Re initializes the connection
REST	RFC 3659	Restart transfer from the specified point
RETR		Retrieve a copy of the file
RMD		Remove a directory
RMDA	Streamlined FTP Command Extensions	Remove a directory tree
RNFR		Rename from
RNTO		Rename to
SITE		Sends site specific commands to remote server (like SITE IDLE 60 or SITE UMASK 002). Inspect SITE HELP output for complete list of supported commands
SIZE	RFC 3659	Return the size of a file
SMNT		Mount file structure
SPSV	FTP Extension Allowing IP Forwarding (NATs)	Use single port passive mode (only one TCP port number for both control connections and passive-mode data connections)
STAT		Returns information on the server status, including the status of the current connection
STOR		Accept the data and to store the data as a file at the server site
STOU		Store file uniquely
STRU		Set file transfer structure
SYST		Return system type
THMB	Streamlined FTP Command Extensions	Get a thumbnail of a remote image file
TYPE		Sets the transfer mode (ASCII/Binary)
USER		Authentication username
XCUP	RFC 775	Change to the parent of the current working directory
XMKD	RFC 775	Make a directory
XPWD	RFC 775	Print the current working directory
XRCP	RFC 743	

命令	RFC	描述
XRMD	RFC 775	Remove the directory
XRSQ	RFC 743	
XSEM	RFC 737	Send, mail if cannot
XSEN	RFC 737	Send to terminal

2.4.2 SFTP

SSH File Transfer Protocol（SFTP）是一种网络协议，该协议可通过任何可靠的数据流提供文件访问、文件传输和文件管理。该协议假定在安全通道（例如SSH）上运行，服务器已经对客户端进行了身份验证，并且客户端用户的身份可用于协议。

与早期只允许文件传输的 SCP 协议相比，SFTP 协议允许对远程文件进行一系列操作，所以 SFTP 更像远程文件系统协议。SFTP 客户端的额外功能包括恢复中断的传输、目录列表和远程文件删除。通常而言，SFTP 比 SCP 更加独立于具体平台，例如，SCP 客户端指定的通配符扩展需要由服务器决定，而 SFTP 的设计则避免了这个问题。并且，SCP 通常在 UNIX 平台上实现，而 SFTP 服务器则在大多数平台上都可用。与 FTP 相比，通过 SFTP 上传的文件可能与其基本属性相关联，例如时间戳，这是 SFTP 优于普通 FTP 协议的优势。

最后来简要介绍 SFTP 技术的发展历程。Tatu 在 1995 年设计了第一个版本的协议（现称为 SSH-1）。SSH 的目标是取代早期既不提供强认证、也不保证机密性的 rlogin、TELNET、FTP 和 RSH 协议。1995 年 7 月时 Tatu 免费发布了该版本的协议，受到了广泛欢迎，到 1995 年底，SSH 用户群已在 50 余个国家普及，并增加到约 20000 个用户。

“Secsh”是负责 SSH 协议版本 2 的 IETF 工作组的官方 Internet 工程任务组（IETF）的名称。2006 年，该协议的修订版本 SSH-2 被采纳为标准。此版本与 SSH-1 不兼容。SSH-2 具有 SSH-1 的安全性，并且有其他功能改进。例如，更好的安全性来自 Diffie-Hellman 密钥交换和通过消息认证码进行的强完整性检查。SSH-2 的新功能包括通过单个 SSH 连接运行任意数量的 shell 会话的功能。由于 SSH-2 在 SSH-1 上的优越性和普及性，一些实现如 libssh 等仅支持 SSH-2 协议。

之后的一段时间内，由于 Secsh 文件传输项目的范围不断扩大，一些专家开始将 SFTP 视为文件系统、而非单一的文件访问或文件传输协议，因此协议的开发工作暂时搁置了下来。2013 年，在中断七年后，SFTP 协议的修订工作再次开始，并陆续发布了版本 3、版本 4、版本 5 和版本 6 的协议。

2.5 HTTP 和 WebDAV 技术

2.5.1 HTTP

Hypertext Transfer Protocol（HTTP）是一个简单的请求-响应协议，它作为 Internet 协议套件模型中的一个应用层协议，主要用于分布式、协作、超媒体信息系统，且通常运行在 TCP 上。它指定了客户端可能发送给服务器什么样的消息以及得到什么样的响应。请求和响应消息的头以 ASCII 形式给出；而消息内容则具有一个类似 MIME 的格式。这个简单模型使开发和部署 Web 应用变得非常简单，极大促进了早期 Web 的发展。

HTTP 的开发由 CERN 于 1989 年开始，第一份文档描述了 HTTP 0.9 协议版本的内容。Berners Lee 随后倡议成立了 WWW 联盟，并组织了 IETF 小组进一步完善和发布 HTTP。HTTP 是应用层协议，是为了实现某一类具体应用的协议，并由某一运行在用户空间的应用程序来实现其功能。协议最新的进展是，HTTP/3 初稿已在 2020 年发布，主流网络浏览器和网络服务器也开始支持这个版本的协议。2022 年 6 月 6 日，IETF 将 HTTP/3 标准化为 RFC 9114。

HTTP 是基于 B/S 架构进行通信的，而 HTTP 的服务器端实现程序有 httpd、nginx 等，其客户端的实现程序主要是各种 Web 浏览器。由于 Web 服务是基于 TCP 的，因此为了能够随时响应客户端的请求，Web 服务器需要监听 TCP 端口 80，这样客户端浏览器和 Web 服务器之间就可以通过 HTTP 协议进行通信了。

HTTP 是基于客户/服务器模式，且面向连接的。典型的 HTTP 事务处理有如下的过程：

1. 客户与服务器建立连接：客户与服务器之间的 HTTP 连接是一种一次性连接，它限制每次连接只处理一个请求，当服务器返回本次请求的应答后便立即关闭连接，下次请求再重新建立连接。这种一次性连接主要考虑到 WWW 服务器面向的是 Internet 中成千上万个用户，且只能提供有限个连接，故服务器不会让一个连接处于等待状态，及时地释放连接可以大大提高服务器的执行效率
2. 客户向服务器提出请求：HTTP 规范定义了 9 种请求方法，每种请求方法规定了客户和服务器之间不同的信息交换方式，服务器将根据客户请求完成相应操作，并以应答块形式返回给客户，最后关闭连接
3. 1. 服务器接受请求，并根据请求返回相应的文件作为应答 2.
4. 客户与服务器关闭连接

HTTP 是一种无状态协议，即服务器不保留与客户交易时的任何状态。这大大减轻了服务器记忆负担，从而保持较快的响应速度。HTTP 又是一种面向对象的协议，允许传送任意类型的数据对象。它通过数据类型和长度来标识所传送的数据内容和大小，并允许对数据进行压缩传送。当用户在一个 HTML 文档中定义了一个超文本链后，浏览器将通过 TCP/IP 协议与指定的服务器建立连接。

HTTP 支持持久连接，在 HTTP/0.9 和 1.0 中，连接在单个请求/响应对之后关闭。在 HTTP/1.1 中，引入了保持活动机制，其中连接可以重用于多个请求。这样的持久性连接可以明显减少请求延迟，因为在发送第一个请求之后，客户端不需要重新协商 TCP 连接。

客户通常在一个特定的 TCP 端口（端口号一般为 80）上打开一个套接字。如果服务器一直在这个周知的端口上倾听连接，则该连接便会建立起来。然后客户通过该连接发送一个包含请求方法的请求块。接下来，服务器在收到请求之后便进行解析，之后将结果发回到请求该结果的客户机上。

HTTP 标头字段是客户端程序和服务器在每个 HTTP 请求和响应中发送和接收的字符串列表。这些标头通常对最终用户不可见，仅由服务器和客户端应用程序处理或记录。它们定义了如何对通过连接发送/接收的信息进行编码、客户端的会话验证和识别或其匿名性、服务器应该如何处理数据、驻留时间和正在下载的文档等。

一般来讲，服务器可以通过对表头字段进行解析从而获取到相关信息，进行处理，并按照协议定义构造出响应报文。同样地，响应报文中的标头字段将成为客户端处理应答的标准。

2.5.2 WebDAV

Web Distributed Authoring and Versioning（WebDAV）是 HTTP 的一组扩展，在 GET、POST、HEAD 等几个 HTTP 标准方法以外添加了一些新的方法，从而允许用户代理直接在 HTTP Web 服务器中协作创作内容，从而将 Web 视为可写的、支持协作的媒介，而不仅仅是只读媒介。WebDAV 在 RFC 4918 中定义。

WebDAV 协议为用户在服务器上创建、更改和移动文档提供了一个框架，支持写文件锁定及解锁，还可以支持文件的版本控制等功能。最重要的功能包括维护有关作者或修改日期的属性、命名空间管理、集合和覆盖保护。属性维护包括文件信息的创建、删除和查询等。命名空间管理处理在服务器命名空间内复制和移动网页的能力。集合处理各种资源的创建、删除和列表。最后，覆盖保护处理与文件锁定相关的方面。它利用了现有技术，例如传输层安全、摘要访问身份验证或 XML 来满足这些要求。许多现代操作系统为 WebDAV 提供内置客户端支持。

WebDAV 扩展了请求方法允许的标准 HTTP 动词和标头集，增加的动词包括：

- COPY：将资源从一个统一资源标识符（URI）复制到另一个
- LOCK：对资源加锁，WebDAV 支持共享锁和排他锁
- MKCOL：创建集合（也称为目录）
- MOVE：将资源从一个 URI 移动到另一个

- PROPFIND：从 Web 资源中检索存储为 XML 的属性。它也被重载以允许检索远程系统的集合结构（也称为目录层次结构）
- PROPPATCH：在单个原子动作中更改和删除资源上的多个属性
- UNLOCK：从资源中移除锁

常见的 WebDAV 客户端如下图所示：

Client	**Creator**	**Operating system support**	**License**	**Interface**
Cyderduck	David V. Kocher	Windows, OS X	GPL	GUI
davfs2	GNOME team	FUSE	GPL	VFS
davix	CERN	Windows, Linux, OS X	LGPL	CLI
GVfs	GNOME team	GNOME	GPL	VFS
KIO	KDE team	KDE	GPL	VFS
Konqueror	KDE team	KDE	GPL	GUI
GNOME Files	GNOME team	GNOME	GPL	GUI
SmartFTP	SmartSoft Ltd	Windows	Proprietary	GUI
WebDrive	South River Technologies	Windows, OS X, iOS, Droid	Proprietary	VFS
WinSCP	Martin Přikryl	Windows	GPL	VLI and GUI

2.6 NFS 技术

Network File System（NFS）是一种分布式文件系统协议，最初由 Sun 在 1984 年开发，允许客户端计算机上的用户通过计算机网络访问文件，就像访问本地存储一样。NFS 可以认为是文件系统上的一个网络抽象，来允许远程客户端以与本地文件系统类似的方式，来通过网络进行访问。NFS 允许在多个用户之间共享公共文件系统，从而利用数据集中的优势来最小化所需的存储空间。

NFS 具有以下特点：

- 提供透明文件访问以及文件传输
- 容易扩充新的资源或软件，不需要改变现有的工作环境
- 高性能，可灵活配置

NFS 与许多其他协议一样，建立在开放网络计算远程过程调用（ONC RPC）系统上。其工作原理是使用 C/S 架构，由客户端程序和服务器程序组成。服务器程序向其他计算机提供对文件系统的访问，其过程称为输出。NFS 客户端程序对共享文件系统进行访问时，把它们从 NFS 服务器中“输出”出来。

具体来讲，一旦发现了为 NFS 所指定的需求，虚拟文件系统（VFS）会将其传递给内核中的 NFS 实例。NFS 解释 I/O 请求并将其翻译为 NFS 程序（OPEN、ACCESS、CREATE、READ、CLOSE、REMOVE 等）。一旦从 I/O 请求中选择了程序，它会在远程程序调用层中执行。RPC 提供了在系统间执行程序调用的方法，它将封装 NFS 请求（附有相关参数）并发送到合适的远程对等级，然后管理并追踪响应，提供给合适的请求者。在服务器端，NFS 以类似的思路运行。需求到达网络协议栈，通过 RPC/XDR 然后到达 NFS 服务器。NFS 服务器负责满足需求。需求向上提交给 NFS 守护进程，它为需求标示出目标文件系统树，并且 VFS 再次用于在本地存储中获取文件系统。NFS 传输协议用于服务器和客户机之间文件访问和共享的通信，从而使客户机远程地访问保存在存储设备上的数据。

下面简要介绍不同版本的 NFS 协议：

- - NFSv2：NFS 协议的第 2 版（在 1989 年 3 月的 RFC 1094 中定义）最初仅通过用户数据报协议 UDP 运行，它的设计者旨在保持服务器端无状态，并在核心协议之外实现锁定。虚拟文件系统接口允许模块化实现，在一个简单的协议中定义。由于 32 位限制，NFSv2 仅允许读取文件的前 2GB
- NFSv3：其主要动机是减轻 NFSv2 中同步写入操作的性能问题，NFSv3（在 1995 年 6 月的 RFC 1813 中定义）添加了如下新特性：
  - 支持 64 位文件大小和偏移量，以处理大于 2GB 的文件
  - 支持服务器异步写入，提高写入性能
  - 将 TCP 作为传输协议
  - 许多回复中的附加文件属性，以避免重新获取它们的需要
  - 支持 READDIRPLUS 操作，用于在扫描目录时获取文件句柄、属性和文件名

  显然，NFSv3 协议比以前的版本具有更好的可扩展性，上述特点为文件系统在更广泛的网络中使用奠定了基础
- - NFSv4：NFSv4（在 2000 年 12 月的 RFC 3010 中定义；在 2003 年 4 月的 RFC 3530 和 2015 年 3 月的 RFC 7530 中修订）受 AFS 和 SMB 的影响，进行了性能改进，并增加了对安全性的更好的支持，以及引入有状态的协议（NFS 之前的版本都是无状态的）
- NFSv4.1（在 2010 年 1 月的 RFC 5661 中定义；在 2020 年 8 月的 RFC 8881 中修订）旨在提供协议支持以利用集群服务器部署，包括提供对分布在多个服务器之间的文件的可扩展并行访问的能力（pNFS 扩展）。pNFS 将生态系统拆分为三个部分：客户端、服务器和存储。pNFS 将数据布局与数据本身拆分，允许双路径架构（一条路径用于数据，另一条路径用于控制）。当客户想要访问文件时，服务器以布局响应。布局描述了文件到存储设备的映射。当客户端具有布局时，它能够直接访问存储，而不必通过服务器（这实现了更大的灵活性和更优的性能)。当客户端完成文件操作时，它会提交数据（更新）和布局。如果需要，服务器能够请求从客户端返回布局。

pNFS 实现了 LayoutGet 协议和 LayoutReturn 协议，从而分别从服务器获取和发布布局，而 LayoutCommit 则将来自客户端的数据提交到存储库，以便于其他用户使用。服务器也采用 LayoutRecall 协议从客户端回调布局。布局跨多个存储设备展开，来支持并行访问和更高的性能

- NFSv4.2（再 2016 年 11 月的 RFC 7862 中定义）的新功能包括：服务器端克隆和复制、应用程序 I/O 建议、稀疏文件、空间预留、应用程序数据块（ADB）、以及 pNFS 的两个新操作（LAYOUTERROR 和 LAYOUTSTATS）

在 NFSv4 之前，存在一定数量的辅助协议用于加载、锁定、和文件管理中的其他元素。在 NFSv4 中，这一流程被简化为一个协议，并将对 UDP 协议的支持作为传输协议移除。此外，NFSv4 还集成支持 UNIX 和基于 Windows 的文件访问语义，将本地集成 NFS 扩展到其他操作系统中。并且，NFSv4 相对于旧版本的协议的一大优势是仅使用一个 UDP 或 TCP 端口 2049 来运行服务，这简化了跨防火墙协议的使用。

2.7 NAS 与 DAS 的比较

DAS（Direct Attached Storage，直连式存储）技术是一种将数据直接存储在本地（未接入网络）的存储设备上的存储方式，其通常由机械硬盘、固态硬盘、光盘等存储单元构成。一个典型的 DAS 设备通常通过 HBA（Host Bus Adapter，主机总线适配器）直接连接到计算机上，通常作为已有存储设备的拓展，这决定了 DAS 技术不具有易于在不同主机之间直接进行数据共享的特点，因为在不同主机之间没有类似于 hub、switch、router 等网络设备进行连接。

和 DAS 相比，NAS 具有的优点包括：

- 通过网络连接，便于进行不同主机之间的数据共享
- 当网络情况稳定时，NAS 提供的服务往往具有更好的性能，因为 NAS 设备可以针对文件服务进行精确调整

而 NAS 相比 DAS 的缺点包括：

- NAS 提供的文件服务不如 DAS 稳定，NAS 提供的服务往往和当前网络的速度和拥塞程度有极大的关系，而 DAS 服务的质量则仅取决于本地计算机的硬件配置，其服务质量是可以在一定程度上得到保证的
- DAS 可以提供更加具有特色的存储服务，往往支持对硬件和低级软件的定制；而 NAS 受限于网络连接要求，其硬件基础往往是不支持定制的

2.8 NAS 与 SAN 的比较

SAN（Storage Area Network，存储区域网络）技术提供对整合的块级数据存储的访问，主要用于从服务器访问数据存储设备，以便这些设备在操作系统看来

是直连式存储。和 NAS 不同，SAN 通常无法通过 LAN 直接访问的专用存储网络，SAN 仅提供块级访问，但构建在 SAN 之上的文件系统则提供文件级访问，这个文件系统被称为共享磁盘文件系统。SAN 是最简单的数据存储专用网络，它源于以数据为中心的大型机架构，互相连接的存储设备确保了存储设备之间的数据传输发生在服务器后面，并且对一般用户来讲是透明的。

和 SAN 相比，NAS 具有如下优点：

- 造价较低，在 NAS 架构中，数据通过成熟的 TCP/IP 协议进行传输，但在 SAN 中则必须使用专有协议例如 Fibre Channel、iSCSI 和 Infiniband 等，因此 SAN 通常有自己的网络和存储设备，必须单独购买、安装和配置。这使得 SAN 技术的安装和使用比 NAS 技术昂贵
- NAS 提供统一的文件系统，方便文件的存储和共享，而 SAN 仅提供基于块的存储，并将文件系统问题遗留给了用户端

而 NAS 相比 SAN 的缺点包括：

- NAS 架构大量使用以太网和 TCP/IP 协议进行内存传输，这样做不但增加了大量的 CPU 指令周期，而且使用了低速传输介质；但 SAN 中大部分操作都由适配卡上的硬件完成，CPU 开销的增加有限，因此 NAS 架构的速度一般无法超越 SAN 架构
- 目前 NAS 的根本瓶颈是底层链路的速度，在底层链路速度较低或不稳定时，NAS 架构提供服务的质量显著低于 SAN 架构

需要注意的是，尽管存在差异，但 SAN 和 NAS 并不相互排斥，并且可以组合为 SAN-NAS 混合体，从而提供来自同一系统的文件级协议（NAS）和块级协议（SAN）。

2.9 NAS 总结与典型应用场景

NAS 技术是一种连接到网络上从而提供数据存储功能的技术，其主要特点包括：文件级存储、以数据为中心、将存储设备与服务器分离、集中管理数据等。在 NAS 技术中，一个在实际应用中是非重要的特点是：存储系统不再通过 I/O 总线附属于某个特定的服务器或者客户端，而是直接通过网络接口与网络直接相连，并且用户可以通过网络对存储设备进行访问。NAS 技术有效地释放了带宽、提高了性能、降低了总拥有成本和后续投资，这使其成本远低于使用服务器存储，但效率却远远高于后者。

NAS 技术的概念是比较清晰的，其主要难点在于具体的实现过程。基于 NAS 的思想，人们设计了许多协议来进行实现，主要包括 AFS、AFP、SMB、FTP 与 SFTP、HTTP 与 WebDAV、NFS 等，并且，这些不同的协议往往在不同的场景下进行应用。

下面列举 NAS 技术的几个典型应用场景并进行分析。

典型应用一：跨平台部署 NAS。NAS 技术往往又被认为是一种“跨平台文件共享服务器”，其跨平台性往往是应用热点。需要注意的是，NAS 作为一种专用的网络文件服务器，与其他的服务器一样需要有操作系统的支持。目前市场上的网络附加存储操作系统不仅仅可以支持微软等主流的操作系统，而且还支持一些开源的操作系统，如 Linux。采用开源的操作系统，不仅成本低廉，而且在安全性与稳定性上可能还要优于微软的 Storage Server 操作系统。现在不少的科技企业和银行等金融企业，采用的大部分都是基于 Linux 核心的客户端系统，此时如果采用 NAS 网络附加存储方案，那么就可以获得比较好的兼容性。

典型应用二：远程备份实现容灾技术。如今的企业规模不断扩大，不同地区间的分公司与办事机构往往具有很频繁的数据交换需求，并且需要保证数据的稳定性和安全性，对于传统的数据存储方案，这无疑是一种挑战。为了保障企业数据的安全，集团信息化部门可能要求各地办事处的数据需要在集团公司进行集中的备份。当办事机构因为某些意外事故而导致数据损坏时，可以通过集团的备份文件来进行恢复。可见，此时企业已经不能够仅仅满足于企业内部的异地备份，而是需要实现远程的容灾备份。在 NAS 存储方案出现之前，企业为了要实现远程的容灾备份，主要思路是在文件服务器的控制器嵌入远程复制功能，然后将两台或者多台部署在异地的存储设备使用光纤网络等高速线路连接起来，由控制器自动将需要备份的数据复制到远程备份设备，以实现远程容灾备份。很明显，这种方案的部署成本比较高，如需要采用光纤网络、如需要额外的硬件设备等等。而随着 NAS 技术的出现和普及，远程容灾备份的实现成本大幅降低，只需要将两台 NAS 设备接入到网络，就可以实现实时的数据备份。采用这个方案作为远程容灾备份的优势主要在于：

- 在网络的两端、即办事处的备份服务器与集团的备份服务器之间，可以采用不同的设备。如办事处的服务器采用的是硬盘备份，而在集团服务器中则可以采用磁盘备份等等。由于支持不同设备之间的备份，可以降低企业的硬件投资成本，提高灵活性
- 此方案支持多种通信方案，光纤网络是最理想的选择，或者企业条件有限时，采用 VPN 等技术也可以有效实现数据的实时备份与恢复

典型应用三：多应用系统的统一支持。现如今，为了存储海量的数据信息，各种各样的信息化管理系统越来越多，例如企业的邮件服务器、文件服务器、数据库服务器等都成为了必不可少的应用系统。为了管理这些这些系统，一些关键问题必须得到有效解决。例如，为了提高数据的安全，需要对这些服务器进行异地备份，这样即使服务器本身硬盘等发生故障，也可以利用备份文件来进行备份。此时，为了将这些不同的应用服务器的异地备份设备进行统一管理，不妨将它们都备份到同一个设备上进行集中管理。如果每一个应用服务器设置独立的异地备份设备，不仅会增加设备的投资成本，而且在管理上也会增加不少的工作量。所以我们希望能够将不同的应用服务器备份到同一个设备中。但是由于这些系统的数据量非常的大，如果要让同一个设备接纳这么多的备份文件，那么就必须要支持海量存储与灵活性。NAS 技术恰好可以满足这两个需求。NAS 技术可以有效集成现有的网络环境和软件、硬件资源，而不需要额外的投资。例如，企

业现在需要部署一个文件服务器，采用的是 Linux 操作系统，因为 NAS 本身的跨平台特性，所以可以在不更换 NAS 操作系统的情况下就可以将文件服务器上的文件备份到 NAS 设备上。这对于后续增加的信息化应用的远程备份非常的重要。另外，在采用 NAS 方案时，在客户端与 NAS 存储设备之间有一个“控制器”，将实际的存储产品同用户隔离开来，也就是说：如果出于海量存储的需要而不得不增加硬盘或者磁带等物理存储设备，那么这个扩容行为对于客户端来说是透明的，不需要经过任何的调整。如此，即使用户需要实现海量数据的存储，也可以很方便的实现。

典型应用四：作为 Web 的后台存储系统。现在的网站不仅有大量的文字与图片信息，还要各种各样的多媒体资料。此时复杂的多媒体门户网站单纯的依靠服务器本身的存储已经远远不够了：一方面服务器本身的存储容量有限，无法满足存储的需求；另一方面服务器本身毕竟不是专业的存储设备，在数据备份、数据检索方面与专业的存储设备还有一定的差距，这个差距可能随着存储规模的扩大而不断增加；最后，现在不少网站都通过服务器负载均衡来提高用户数据检索的效率。所以这些特性都要求将 Web 服务器与数据的存储设备隔离开来。在这种情况下，NAS 技术就成为了门户网站后台最好的存储平台之一：

- 如果 Web 应用程序需要存储海量的数据，NAS 技术的大容量存储空间可以满足 Web 应用程序的存储与检索需求
- 如果 Web 应用程序的客户端非常多，并且不同访问者所采用的操作系统也是不同的，由于 NAS 技术支持多用户多平台的数据共享、能够集中管理数据、拥有完善的数据保护措施，故客户数量再多、操作系统种类再多，NAS 技术仍然可以应对自如，这大幅提高了服务质量

需要注意的是，虽然网络附加存储在跨平台性、在投资成本上等等有很大的优势，但是其在性能上却没有显著的优势。对于性能要求比较高的应用，还需要采用其他的技术来改善 NAS 技术的工作效率。例如，在 NAS 的存储设备上通过采用硬件 RAID 磁盘阵列等技术来提高后台存储设备的工作效率等，或者在前端应用服务器上采用服务器负载均衡等手段来对客户的访问量进行分流等。也就是说，大部分情况下，NAS 技术不是单独起作用的，而是作为关键组件与其他一些技术结合使用的，这样才能够发挥其最大价值。

3 SAN 技术

存储区域网络（Storage Area Network，SAN）是一种计算机网络，它提供对整合的块级数据存储的访问。SAN 采用一种叫做“网状通道（Fibre Channel，FC）”的技术，这种技术通过 FC 交换机连接存储阵列和服务器主机，从而建立专用于数据存储的区域网络。虽然各厂商的光纤交换技术并不完全相同，但是经过产业界十多年的发展，SAN 技术已经相当成熟，并在业界标准上达成了较高程度的一致，这极大推动了 SAN 技术的发展。

SAN 主要用于从服务器访问数据存储的设备，例如磁盘阵列和磁带库，从

而使这些设备在概念上具备了“直连存储”的特性。注意，本质上 SAN 技术仅提供块级访问，但是构建在 SAN 之上的文件系统则可以提供文件级的访问，这个文件系统被称为“共享文件系统”。当前企业级存储面临的两个最突出的问题是：数据与应用系统紧密结合所产生的结构性限制、以及小型计算机接口（SCSI）标准带来的限制。由于 SAN 技术拥有便于集成、能改善数据可用性和优秀的网络性能，因此 SAN 被广泛认为是未来企业级存储问题的有效解决方案。

SAN 技术的前身是集中式数据存储方案，现在，由于 SAN 往往作为一种计算机网络提供对整合的块数据的访问，因此其可被称为服务器背后的网络。除了存储数据之外，SAN 还允许数据的自动等操作。

一般来讲，SAN 是硬件和软件的组合，它源于以数据为中心的大型机架构，其中网络中的客户端可以连接到多个服务器并存储不同类型的数据。随着数据量的不断增加，DAS 技术作为解决方案而出现，该技术通过将磁盘阵列或其他成组存储设备连接到服务器上来增加容量，但是很快发现 DAS 技术的致命缺陷是“单点故障问题”，这意味着存储设备中一点失效会导致系统整体出现极大的问题。

尽管 DAS 作为第一个成熟的大规模存储架构得到了极为广泛的应用，但是对于数据存储要求很高的地方仍然具备一定的局限性，因此人们开发了网络附加存储（NAS）技术架构。在 NAS 技术中，一个或多个专用文件服务器或专用存储设备在 LAN 中激活并使用，提供文件级的服务。这种分布式的设计思路在一定程度上解决了 DAS 技术中的“单点故障问题”。但是，随着信息时代的不断发展，数据传输、尤其是数据备份的要求越来越高，面对更多的数据和更高的要求，NAS 技术逐渐显露出其局限性。

因此，经过仔细设计的 SAN 技术正式问世，它通过将专用存储网络连接到 LAN、以及专用高速和带宽网络传输 TB 级别的数据来提供更加强大和稳健的数据服务。在 SAN 中，互相连接的存储设备使彼此的数据传输（例如备份）的实现隐藏在了服务器后面（并且对于普通用户是透明的），比如，数据可以通过 TCP/IP 协议进行传输。SAN 配备了不同的支持协议，包括光纤通道（FC）、SCSI 与 iSCSI、Infiniband 等。但是，SAN 通常有自己的网络和存储设备，必须单独购买、安装和配置。这使得 SAN 架构比 NAS 架构和 DAS 架构都更昂贵。

3.1 SAN 组成成分

SAN 通常有自己的网络设备，例如 SAN 交换机。为了访问 SAN，引入了 SAN 服务器的概念，这些服务器又连接到对应的 SAN 主机适配器。SAN 支持互连一系列数据存储设备，例如为 SAN 技术定制的磁盘阵列、JBODS 和磁带库等。

3.1.1 Host Layer

允许访问 SAN 系统及其相应存储设备的服务器被称为 Host Layer，这类服务器通常具有主机适配器，它们通常是一张连接到服务器主板上的插槽（通常

是 PCI 插槽）的卡，这张卡与相应的固件和设备驱动程序一起运行，从而提供必不可少的访问服务。显然，服务器的操作系统可以通过主机适配器及硬件卡与 SAN 中的存储设备进行通信。

在光纤通道解决方案中，千兆接口转换器（GigaBit Interface Converter，GBIC）用于 SAN 内的交换机和存储设备，从而将数字位转换为光脉冲，然后将它们发送到光纤通道进行传输。在到达目的地时，GBIC 又可以将传入的光脉冲转换回数字位，从而完成信息解码的工作。

3.1.2 Fabric Layer

Fabric Layer 由 SAN 技术中提供网络服务的设备组成，包括 SAN 交换机、路由器、协议桥、网关设备和电缆等。SAN 网络设备在 SAN 内或在发送方（如服务器的 HBA 端口）与目标方（如存储设备的端口）之间移动数据。

最初构建 SAN 时，集线器是唯一具有光纤通道功能的设备，但随着光纤通道交换机被开发并不断完善，现在 SAN 技术已经将集线器淘汰了。与集线器相比，交换机可以提供一条专用链路来将其所有端口相互连接，因此交换机最大的优势就在于它们允许所有连接的设备同时进行通信，这是集线器所不能做到的。

为了保证系统的稳定性，SAN 的实际构建往往遵循冗余性原则。例如，单个 SAN 交换机可以有最少 8 个端口，最多可以有 32 个带有模块化扩展的端口；导向器级交换机则可以有多达 128 个端口。此外，SAN 交换机非阻塞地将服务器与存储设备连接起来，允许同时通过所有连接的线路传输数据。

支持交换技术的 SAN 实现中使用了光纤通道交换结构协议 FC-SW-6，在该协议下，SAN 中的每个设备在主机总线适配器（HBA）中都有一个硬编码的全球名称（WWN）地址。如果设备连接到 SAN，则它的 WWN 会在 SAN 交换机名称服务器中注册。SAN 光纤通道存储设备供应商还可以硬编码全球节点名称（WWNN）。存储设备的端口通常有一个以 5 开头的 WWN，而服务器的总线适配器则以 10 或 21 开头。

3.1.3 Storage Layer

SAN 中的各种存储设备构成了它的 Storage Layer。在 SAN 中，磁盘阵列通过 RAID 技术进行连接，这使得许多硬盘看起来和执行起来就像一个大型存储设备。

各种存储数据的硬盘和磁带设备都有一个逻辑号（LUN），这个编号在 SAN 中是唯一的。SAN 中的每个节点、无论是服务器还是其他存储设备，都可以通过引用 LUN 来访问存储设备。并且，LUN 允许对 SAN 的存储容量进行分段并实施访问控制。例如，一个特定的服务器或一组服务器可以仅被授予对某些 LUN 指定的 SAN 存储层的特定部分的访问权。当存储设备接收到读取或写入数据的请求时，它将检查其访问列表以确定由其 LUN 标识的节点是否被允许访问同样由 LUN 标识的存储区域。

LUN 屏蔽是一种技术，通过该技术，主机总线适配器和服务器的 SAN 软件限制接受命令的 LUN。这样做时，服务器永远不应访问的 LUN 将被绝对屏蔽。另一种限制服务器访问特定 SAN 存储设备的方法是基于结构的访问控制或分区，由 SAN 网络设备和服务器强制执行。在分区下，服务器访问仅限于特定 SAN 区域中的存储设备。

3.2 SCSI 技术

Small Computer System Interface（SCSI）是一组用在计算机和外围设备之间进行物理连接和传输数据的标准，该标准定义了命令、协议、电气以及光学和逻辑接口。理论上，SCSI 可以用于几乎任何设备的接口上，这是因为 SCSI 协议本质上和传输介质无关，SCSI 可以在多种介质上实现（甚至是虚拟介质）。最初的并行 SCSI 常用于硬盘驱动器和磁带驱动器，但它也可以广泛连接其他设备，包括扫描仪和 CD 驱动器。

最早的 SCSI 标准 X3.131-1986，一般简称为 SCSI-1，是由美国国家标准协会（American National Standards Institute，ANSI）的 X3T9 技术委员会于 1986 年发布。SCSI-2 于 1990 年 8 月发布为 X3.T9.2/86-109，在 1994 年进行了进一步修订，随后采用了多种接口。随着发展的不断推进，SCSI 性能得到了持续不断的改进，其对不断增加的数据存储容量的支持也越来越强大。

在发展历程中，SCSI 逐渐发展出了下面的类型：

- SCSI-1：这是最原始的版本，异步传输的频率为 3MB/S，同步传输的频率为 5MB/s。虽然该版本几乎被淘汰了，但还会使用在一些扫描仪和内部 ZIP 驱动器中，采用的是 25 针接口。也就是说，若是将 SCSI-1 设备联接到常见的 SCSI 卡时，必须要有一个内部的 25 针对 50 针的接口电缆；若使用外部设备，就不能采用内部接口中的任何一个（即此时的内部接口均不可以使用）
- SCSI-2：早期的 SCSI-2 也称为 FastSCSI，该版本的 SCSI 通过提高同步传输的频率使据传输速率从原有的 5MB/s 提高为 10MB/s，并支持 8 位并行数据传输，最多可连接 7 个外设。后来出现的 Wide SCSI，则最多支持 16 位并行数据传输，数据传输率也进一步提高到了 20MB/s，最多可连接 16 个外设。此版本的 SCSI 使用一个 50 针的接口，目前主要用于扫描仪、CD-ROM 驱动器及老式硬盘中
- SCSI-3：1995 年，更为高速的 SCSI-3 正式面世，称为 UltraSCSI，数据传输率达到了 20MB/s。若使用 16 位传输的 Wide 模式，数据传输率更可以提高至 40MB/s。此版本的 SCSI 使用一个 68 针的接口，主要应用在硬盘上。SCSI-3 的典型特点是总线频率大大提高，信号的干扰也大大降低，稳定性得到了极大的增强。SCSI-3 有很多型号，比如：
  - Ultra（fast-20）传输频率 20MHz，数据频宽 8 位，传输率 20MBps
  - Ultra wide 传输频率 20MHz，数据频宽 16 位，传输率 40MBps

– Ultra 2 传输频率 80MHz，数据频宽 16 位，传输率 80MBps
– Ultra 160 传输频率 80MHz，数据频宽 16 位，传输率 160MBps
– Ultra 320 传输频率 80MHz，数据频宽 16 位，传输率 320MBps
– Ultra 640 的传输频率 160MHz，数据频宽 16 位，传输率 640MBps

SCSI 总线技术拥有诸多优点，可以总结如下：

- SCSI 可支持多个设备，SCSI-2（FastSCSI）最多可接 7 个 SCSI 设备，Wide SCSI-2 以上版本的 SCSI 则可接 15 个 SCSI 设备。也就是说，所有的设备只需占用一个 IRQ，同时 SCSI 还支持相当广的设备，如 CD-ROM、DVD、CDR、硬盘、磁带机、扫描仪等
- SCSI 还允许在对一个设备传输数据的同时，另一个设备对其进行数据查找，这就可以在多任务操作系统（如 Linux 和 WindowsNT 等）中获得更高的性能
- SCSI 占用 CPU 极低，确实在多任务系统中占有着明显的优势。由于 SCSI 卡本身带有 CPU，可处理一切 SCSI 设备的事务，在工作时主机 CPU 只要向 SCSI 卡发出工作指令，SCSI 卡就会自己进行工作，工作结束后返回工作结果给 CPU，在整个过程中，CPU 均可以进行自身工作
- SCSI 设备还具有智能化，SCSI 卡自己可对 CPU 指令进行排队，这样就提高了工作效率。在多任务时硬盘会在当前磁头位置，将邻近的任务先完成，再逐一进行处理
- 最快的 SCSI 总线有 160MB/s 的带宽，这要求使用一个 64 位的 66MHz 的 PCI 插槽，因此在 PCI-X 总线标准中所能达到的最大速度为 80MB/s，若配合 10000rpm 或 15000rpm 转速的专用硬盘使用将带来明显的性能提升

3.2.1 SCSI 逻辑拓扑与工作流程

SCSI 协议在设计上是颇为巧妙的，通过将存储系统进行巧妙地 “分层设计”，从而提高了系统的稳定性和提供服务的质量。

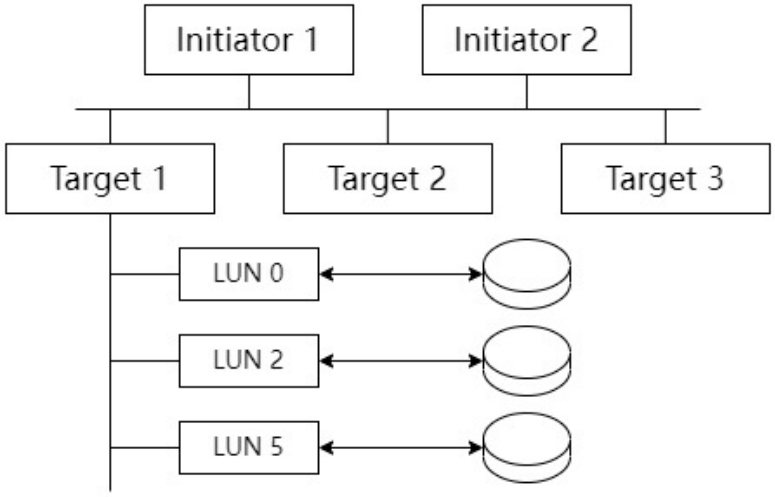

图 1: SCSI 逻辑拓扑图

上图展示了 SCSI 技术的逻辑拓扑图，其中：

- 逻辑单元（LUN）: LUN 是 SCSI 目标器中所描述的名字空间资源，一个目标器可以包括多个 LUN，而且每个 LUN 的属性可以有所区别，比如 LUN#0 可以是磁盘，LUN#1 可以是其他设备
- 启动器（Initiator）：SCSI 是一个 C/S 架构，其中客户端为 Initiator，负责向 SCSI 目标器发送请求指令，一般主机系统都充当了启动器的角色
- 目标器（Target）：处理 SCSI 指令的服务端称为目标器，它接收来自主机的指令并解析处理，比如磁盘阵列的角色就是目标器。SCSI 的 Initiator 与 Target 共同构成了一个典型的 C/S 模型，每个指令都是“请求/应答”这样的模型来实现

对于 Initiator 而言，从 SCSI 的体系结构来说，一共有架构层（中间层）、设备层、传输层三个层级，因此无论 Initiator 上运行的操作系统是什么，SCSI 都分为这三个层次。例如，Windows 下的 Initiator 分为三个逻辑层次，其中 ScsiPort 负责实现 SCSI 的基本框架性处理流程，比如设备发现，名字空间扫描等；Linux 下的启动器架构则将 SCSI 的 Initiator 分为三个逻辑层次，其中 scsi_mod 中间层复杂处理 SCSI 设备无关和适配器无关的流程处理，比如一些异常，名字空间维护等。

对于 Target 而言，往往也参考 SCSI 体系结构将 Target 分为三个层次，分别是链路端口层、中间层和目标设备层，它们之间的相互关系如下图所示。其中最重要的是中间层，在中间层中将以 SAM/SPC 为规范，对 LUN 命名空间，链路端口，目标设备，任务，任务集，会话等进行管理维护。端口层的驱动都以注册的形式动态载入，设备层的驱动也是动态载入。

在 SCSI 工作时，控制器首先向总线处理器发出请求使用总线的信号。该请求被接受之后，控制器高速缓存就开始执行发送操作。在这个过程中，控制器占用了总线，总线上所连接的其它设备都不能使用总线。当然，由于总线具备中断

功能，所以总线处理器可以随时中断这一传输过程并将总线控制权交给其它设备，以便执行更高优先级的操作。

SCSI 控制器相当于一个小型 CPU，有自己的命令集和缓存。SCSI 可以认为是一种特殊的总线结构，可以对计算机中的多个设备进行动态分工操作，对于系统同时要求的多个任务可以灵活机动的适当分配，动态完成。SCSI 总线支持通过 SCSI 卡接入计算机总线，并且在接入计算机总线之后，利用 SCSI 卡的管理，可以实现某种程度上的存储空间的扩展。

具体来讲，SCSI 卡一端接入主机的 PCI 总线，另一端用一个 SCSI 控制器接入 SCSI 总线。卡上有自己的 CPU（频率很低，一般为 RISC 架构），通过执行 ROM 中的代码来控制整个 SCSI 卡的工作。经过这样的架构，SCSI 卡将 SCSI 总线上的所有设备经过 PCI 总线传递给内存中运行着的 SCSI 卡的驱动程序，这样操作系统便会知道 SCSI 总线上的所有设备了。如果这块卡有不止一个 SCSI 控制器，则每个控制器都可以单独掌管一条 SCSI 总线，这就是多通道 SCSI 卡。通道越多，一张卡可接入的 SCSI 设备就越多。

注意，在 SCSI 链的最后一个 SCSI 设备要用终结器（中间设备是不需要终结器的，一旦中间设备使用了终结器，那么 SCSI 卡就无法找到以后的 SCSI 设备了）。如果最后一个设备没用终结器，SCSI 是无法正常工作的。终结器是由电阻组成的，位于 SCSI 总线的末端，用来减小相互影响的信号，维持 SCSI 链上的电压恒定。

3.2.2 SCSI 寻址过程

为了对连接在 SCSI 总线上的设备进行寻址，SCSI 协议引入了 SCSI 设备 ID 和逻辑单元号 LUN（Logical Unit Number）。在 SCSI 总线上的每个设备都必须有一个唯一的设备 ID，当然服务器中的主机总线适配器也拥有自己的设备 ID，固定为 7。每条总线，包括总线适配器，最多允许有 8 个或者 16 个设备 ID。设备 ID 一方面用以寻址，另一个作用是标识该设备在总线使用上的优先级。此外，在同一条总线上连接的不同的设备的设备 ID 必须不同，否则就会引起寻址和优先级的冲突。

每一个存储设备可能包括若干个子设备，如虚拟磁盘、磁带驱动器等。因此 SCSI 协议引入了逻辑单元号 LUN ID，以便于对存储设备中的子设备进行寻址。

传统的 SCSI 控制器连接单条总线，相应的只具有一个总线号。企业级的一个服务器则可能配置了多个 SCSI 控制器，从而就可能有多条 SCSI 总线。在引入存储网络之后，每个 FC HBA（Host Bus Adapter）或 iSCSI（Internet SCSI）网卡也都各连接着一条总线，因此必须对每一条总线分配一个总线号，在他们之间依靠不同的总线号加以区分。我们可以使用一个三元描述标识一个 SCSI 目标：总线号/目标设备 ID/逻辑单元号 LUN ID，它们之间的关系如下所示：

$$\text{总线号} \rightarrow \text{设备}ID \rightarrow \text{逻辑单元号}$$

3.2.3 SCSI 的读操作和写操作

下面以 SCSI 读操作和写操作为例说明 SCSI 具体的工作流程。

对于 SCSI 读操作而言，发起方在获得总线仲裁之后，会发送一个 SCSI Command 读命令帧。接收端接收后，立即将该命令中给出的 LUN 以及 LBA 地址段的所有扇区的数据读出，传送给发起端。所有数据传输结束后，目标端发送一个 RESPONSE 帧来表示这条 SCSI 命令执行完毕。SCSI 协议语言就是利用这种两端节点之间相互传送一些控制帧，来达到保障数据成功传输的目的。

而对于写操作，发起方在获得总线仲裁之后，会发送一个 SCSI Command 写命令帧，其中包含对应的 LUN 号以及 LBA 地址段。接收端接收后，就知道下一步对方就要传输数据了。接收方做好准备后，向发起方发送一个 XFER_RDY 帧，表示已经做好接收准备，可以随时发送数据。发起方收到 XFER_RDY 帧之后，会立即发送数据。每发送一帧数据，接收方就回送一个 XFER_RDY 帧，表示上一帧成功收到并且无错误，可以立即发送下一帧，直到数据发送结束。接收方发送一个 RESPONSE 帧来表示这条 SCSI 命令执行完毕。

3.3 iSCSI 技术

iSCSI（Internet Small Computer System Interface，Internet 小型计算机系统接口）是一种基于 Internet 协议和的 SCSI-3 协议的存储网络标准，用于链接数据存储设施，iSCSI 由 IBM 和 Cisco 于 1998 年首创，并于 2003 年 2 月 11 日成为正式的标准。iSCSI 也是 IP-SAN 中最重要的、最先获批实施的协议标准，除了 iSCSI 外，还有 iFCP、FCIP 等标准。

iSCSI 通过在 TCP/IP 网络上传输 SCSI 命令来提供对存储设备的块级访问。iSCSI 有助于通过 Intranet 传输数据并管理远距离存储。它可用于通过 LAN、WAN 或 Internet 来传输数据，并且可以实现与位置无关的数据存储和检索。同时，iSCSI 协议允许客户端（称为 Initiator，启动器）向远程服务器上的存储设备（称为 Target，目标）发送 SCSI 命令。

作为一种 SAN 协议，iSCSI 协议允许存储整合到存储阵列中，同时使客户端（例如数据库和 Web 服务器）就像访问本地连接数据存储一样来访问网络存储。

iSCSI 技术有以下技术优势：

- iSCSI 的基础是传统的以太网和 Internet，同时能大大减少总体拥有成本
- IP 网络的带宽发展相当迅速，1Gbps 以太网早已大量占据市场，10Gbps 以太网也已整装待发
- 在技术实施方面，iSCSI 以稳健、有效的 IP 及以太网架构为骨干，使鲁棒性大大增加
- 简单的管理和布署，不需要投入培训，就可以轻松拥有专业的 iSCSI 人才

- iSCSI 是基于 IP 协议的技术标准，它实现了 SCSI 和 TCP/IP 协议的连接，只需要不多的投资，就可以方便、快捷地对信息和数据进行交互式传输及管理
- 完全解决数据远程复制及灾难恢复的难题。安全性方面，iSCSI 已内建支持 IPSEC 的机制，并且在芯片层面执行有关指令，确保安全性

尤其是与传统的 SCSI 技术相比，iSCSI 技术有以下的三个革命性的变化：

- 把原来只用于本机的 SCSI 协议透过 TCP/IP 网络发送，使连接距离可作无限的地域延伸
- 连接的服务器数量无限（原来的 SCSI-3 的上限是 15）
- 由于是服务器架构，因此也可以实现在线扩容甚至动态部署

3.3.1 iSCSI 优势功能

iSCSI 技术允许两台主机进行协商，然后使用 IP 网络交换 SCSI 命令。这样，iSCSI 就可以采用流行的高性能本地存储总线并在广泛的网络上对其进行仿真，从而实现 SAN 技术。与某些其他 SAN 协议不同，iSCSI 不需要专用电缆，它可以在现有的 IP 基础设施上运行。因此，iSCSI 通常被视为 FCP 版本的 SAN 的低成本替代方案，但是，如果不在专用网络或子网（如 LAN 和 VLAN）上运行，那么 iSCSI SAN 部署的性能可能会因为固定带宽的竞争而严重下降。

尽管 iSCSI 可以与任意类型的 SCSI 设备通信，但它几乎总是被使用来允许服务器（例如数据库服务器）访问存储阵列上的磁盘卷。iSCSI SAN 通常具有以下两个目标之一（这两个目标也是 iSCSI 最具优势的功能）：

- Storage Consolidation：iSCSI 可以在数据中心内将分散的存储资源从其网络周围的服务器转移到中央位置，这可以提高存储分配的效率，因为存储本身不再与特定服务器绑定。在 SAN 环境中，可以为服务器分配一个新的磁盘卷，这仅需要对系统软件进行适当的修改，而无需对硬件进行任何更改
- Disaster Recovery：iSCSI 可以将存储资源从一个数据中心镜像到远程数据中心，在长时间中断的情况下，该数据中心可以作为备用数据总线，这对于提供鲁棒性的数据总线服务尤其重要。而且，iSCSI SAN 允许通过极少的配置更改来使用 WAN 迁移整个磁盘阵列，通过这种思路，存储变成了“可路由的”，就像普通的网络通信一样

3.3.2 iSCSI 工作原理

iSCSI 在物理实现上仍然基于传统的 SCSI 模式，即 Initiator-Taregt 模式（两者之间的 SCSI 线缆可以为普通电缆或光缆），如下所示：

$$OS \longleftrightarrow Initiator \longleftrightarrow TCP/IP\ Network \longleftrightarrow Target \longleftrightarrow LU$$

iSCSI 通过 TCP 面向连接的协议来保护数据块的可靠交付。由于 iSCSI 基于 IP 协议栈，因此可以在标准以太网设备上通过路由或交换机来传输，其数据块从外到内分别为：

- IP Header
- TCP Header
- iSCSI Header
- SCSI commands and data

并且，有 IPv4 和 IPv6 两种实现方式，只需要改变对应的 IP Header 即可实现。在收到数据块的时候，只需要按照协议格式进行解析即可。并且得益于计算机网络的分层结构，外部技术的改变并不会影响到内部的技术实现。

3.3.3 iSCSI 的实现

在 iSCSI 配置中，iSCSI 主机或服务器将请求发送到节点。主机包含一个或多个连接到 IP 网络的启动器，以发出请求，并接收来自 iSCSI 目标的响应。

每个启动器和目标都指定了一个唯一的 iSCSI 名称，如 iSCSI 限定名（IQN）或扩展的唯一标识（EUI）。IQN 是 223 字节的 ASCII 名称。EUI 是 64 位标识。iSCSI 名称代表全球唯一命名方案，该方案用于标识各启动器或目标，其方式与使用全球节点名（WWNN）来标识光纤通道光纤网中设备的方式相同。

iSCSI Target 是响应 iSCSI 命令的设备。iSCSI 设备可以是诸如存储设备的结束节点、或者可以是诸如 IP 与光纤通道设备之间的网桥的中间设备。每个 iSCSI Target 由唯一的 iSCSI 名称标识。如果要通过 IP 网络传输 SCSI 命令，iSCSI 驱动程序必须安装到 iSCSI Initiator 和 Target 中。驱动程序用于通过主机或目标硬件中的网络接口控制器（NIC）或 iSCSI HBA 来发送 iSCSI 命令和响应。注意，为实现最佳性能，需要使用传输速度为 1000 Mbps 的千兆以太网适配器在 iSCSI Initiator 和 iSCSI Target 间进行连接。

iSCSI 会话建立在一个 Initiator 与一个 Target 之间，一个会话允许多个 TCP 连接，并且支持跨连接的错误恢复。大多数通信还是建立在 SCSI 基础之上的，例如，使用 R2T 进行流量控制。iSCSI 添加于 SCSI 之上的有：立即和主动的数据传输以避免往返通信；连接建立阶段添加登录环节，这是基于文本的参数协商。建立一个 iSCSI 会话，包括如下阶段：

- 命名阶段：确定需要访问的存储，以及 Initiator，与 FC 不同，命名与位置无关
- 发现阶段：找到需要访问的存储
- 登录阶段：建立于存储的连接，读写之前首先进行参数协商，按照 TCP 连接登录

建立了 iSCSI 会话之后，便可以开始使用 iSCSI 服务。

3.4 FC SAN 技术

FC SAN 是由网线、网络适配器和集线器或交换机等组件组成的 SAN，其中不同类型的 FC 架构有：

- 点到点架构（Point-to-point）：在该架构中，两个节点直接相互连接。此配置为节点之间的数据传输提供专用连接。但是，点对点架构限制了连接性和可扩展性
- 光纤通道仲裁环路架构（Fibre channel arbitrated loop）：在该架构中，设备连接到共享环路。每个设备都与其他设备竞争执行 I/O 操作。环路上的设备必须通过仲裁机制获得对环路的控制，在任何给定时间，只有一个设备可以在环路上执行 I/O 操作。由于每个设备必须等待轮到它处理 I/O 请求，所以 FC AL 环境中的整体性能较低。此外，添加或移除设备会导致环路重新初始化，这可能会导致环路流量暂时暂停。作为环路架构，FC AL 可以在没有任何互连设备的情况下通过电缆将一个设备直接连接到环中的另外两个设备来实现。FC AL 实现也可以使用 FC 集线器，仲裁环路通过该集线器以星形拓扑物理连接
- 光纤通道交换架构（Fibre channel switched fabric）：该架构通常涉及单个 FC 交换机或 FC 交换机网络来互连节点，所有节点在同一个逻辑空间相互通信。在该架构下，任意两台交换机之间的链路称为交换机间链路，这种链路使交换机能够连接在一起以形成单个更大的结构

下面我们将结合具体实现对上面的概念进行详细介绍。

3.5 FC 与 FCP 技术

Fibre Channel（FC）是一种高速数据传输协议，可提供有序、无损的原始块数据传输，主要用于将计算机数据存储连接到商业数据中心的存储区域网络（SAN）中的服务器。

FC 形成了一种交换结构，因为网络中的交换机作为一个大交换机统一运行。FC 通常在数据中心内部和之间的光纤电缆上运行，但也可以在铜缆上运行。支持的数据速率包括 1、2、4、8、16、32、64 和 128 Gbps，这是由于连续几代技术的改进而产生的。

FC 有多种上层协议，包括两种用于块存储的协议：Fibre Channel Protocol（FCP）是一种通过光纤通道网络传输 SCSI 命令的协议；Fibre Connection（FICON）是一种通过光纤通道传输 IBM 大型计算机使用的 ESCON 命令的协议。

Fibre Channel Protocol（FCP）是利用底层光纤通道连接的 SCSI 接口协议。光纤通道标准定义了一种高速数据传输机制，可用于连接工作站、大型机、超级

计算机、存储设备和显示器。FCP 解决了对大量信息的快速传输的需求，并且可以减轻系统制造商支持各种渠道和网络的负担，因为它为网络、存储和数据传输提供了一种标准。光纤通道的一些特性包括：

- 性能从 266Mbps 到 16Gbps
- 支持光纤和铜线介质，传输距离最长可达 10 公里
- 小型连接器（最常见的是 sfp+）
- 对距离不敏感的高带宽利用率
- 支持从小型系统到超级计算机的多种成本/性能级别
- 能够承载多个现有接口命令集，包括 Internet 协议（IP）、SCSI、IPI、HIPPI-FP 和音频/视频等协议

自从 1988 年出现以来，FCP 已经发展成为一项非常复杂、高速的网络技术。它最初并不是研究来作为一种存储网络技术的。最早版本的 FCP 是一种为了包括 IP 数据网在内的多种目的而推出的高速骨干网技术，它是作为惠普、Sun 和 IBM 等公司组成的 R&D 实验室中的一项研究项目开始出现的，当时的 FCP 开发者认为这项技术有一天会取代 100BaseT 以太网和 FDDI 网络，他们将 FCP 作为一种高速骨干网络技术，而将存储作为不重要的应用。

3.5.1 光纤通道介绍

光纤通道（FC）在国际信息技术标准委员会（INCITS）的 T11 技术委员会中进行了标准化，该委员会是 ANSI 认可的标准委员会。光纤通道始于 1988 年，并于 1994 年获得 ANSI 标准批准，主要优势是合并多个物理层的实施，包括 SCSI、HIPPI 和 ESCON。

光纤通道被设计为串行接口，从而克服 SCSI 和 HIPPI 物理层并行信号铜线接口的限制。此类接口面临的挑战之一是保持所有数据信号线（SCSI 为 8、16 和 32 条，HIPPI 为 50 条）之间的信号时序一致性，以便接收器可以确定所有电信号值何时稳定有效。随着数据信号频率的增加，这一挑战在大规模制造的技术中变得越来越困难，部分技术补偿是不断减少支持的连接铜平行电缆长度。光纤通道采用领先的多模光纤开发克服了 ESCON 协议的速度限制的技术。通过使用大量 SCSI 磁盘驱动器并利用大型机技术，光纤通道实现了规模经济效应，并且开始广泛而经济地进行部署。

光纤通道还增加了对任意数量的“上层”协议的支持，包括 ATM、IP（IPFC）和 FICON，其中 SCSI（FCP）是主要用途。需要注意的是，光纤通道并不遵循 OSI 模型分层，而是常分为五层：

- FC-4（协议映射层）：可以通过光纤通道执行的应用程序接口，将 NVMe、SCSI、IP、FICON 等上层协议封装到信息单元中并交付给 FC-2。当前的 FC-4 包括 FCP-4、FC-SB-5 和 FC-NVMe

- FC-3（公共服务层）：一个最终可以实现加密或 RAID 冗余算法等功能的薄层，可以实现条带化、寻线组和多播等高级功能所需的通用服务，遵循多端口连接原则
- FC-2（信令协议层）：由光纤通道成帧和 FC-FS-5 标准定义，由低级光纤通道网络协议组成，完成帧、序列和交换的传输（包括协议信息单元的传输），遵循端口到端口连接原则
- FC-1（传输协议层）：实现信号的线路编码，即对于物理媒体上传输的数据和带外物理链路控制信息的编码和解码
- FC-0（物理层），包括电缆、连接器等设备

3.5.2 FC 网络拓扑

FC Arbitrated Loop（FC AL）是一种光纤通道拓扑，其中设备以环形拓扑中的单向环路方式连接。它允许连接许多服务器和计算机存储设备，而无需使用较为昂贵的光纤通道交换机。现在，随着光纤通道交换机的成本不断下降，FC AL 的应用范围不断缩小，但目前在存储系统中仍然很常见。

FC AL 拓扑类似于以太网共享总线拓扑，但是连接方式不是总线，而是一条仲裁环路。每个 FC AL 设备首尾相接构成了一个环路。一个环路能接入的最多节点是 128 个，实际上是用了一个字节的寻址容量，但是只用到了这个字节经过 810b 编码之后奇偶平衡的值，也就是 256 个值中的 134 个值来寻址，这些被筛选出来的地址中又被广播地址、专用地址等占用了，最后只剩下 127 个实际可用的节点地址。

FC AL 可以以环形方式或使用集线器进行物理连接。如果链中的一个设备发生故障，物理环将停止工作。另一方面，集线器在保持逻辑环的同时，允许在电缆级别上采用星形拓扑。集线器上的每个接收端口都被简单地传递到下一个活动传输端口，而绕过任何非活动或故障端口。

因此，光纤通道集线器具有另一项功能：提供旁路电路，以防止在一个设备发生故障或被移除时环路中断。如果设备从环路中移除，集线器的旁路电路会检测到信号缺失并立即开始将传入数据直接路由到环路的下一个端口，从而完全绕过丢失的设备。这至少为循环提供了一定程度的鲁棒性，即循环中的一个设备发生故障不会导致整个循环无法运行。

综上所述，FC AL 具有如下的特点：

- 是一种串行架构，可用作 SCSI 网络中的传输层，最多可支持 127 个设备，环路可以通过其端口之一连接到光纤通道结构
- 环路上的带宽在所有端口之间共享
- 在环路上一次只能通信两个端口，一个端口赢得仲裁，并且可以在半双工或全双工模式下打开另一个端口

- 能够仲裁环路通信的光纤通道端口有节点环路端口（NL_port）和结构环路端口（FL_port），统称为 L_ports。端口可以通过集线器相互连接，电缆从集线器延伸到端口，但集线器上的物理连接器不是协议的端口，集线器不包含端口
- 没有结构端口（而只有 NL_ports）的仲裁环路是私有环路
- 通过 FL_port 连接到结构的仲裁环路是公共环路
- NL_Port 必须提供结构登录（FLOGI）和名称注册设施，以通过结构启动与其他节点的通信而成为发起者（Initiator）

3.5.3 FC 帧结构

FC 网络中数据传输的基本单位是 FC 帧，在 FC-2 中对帧的格式给出了统一的规定：一个 FC 帧是由 SOF、帧内容以及 EOF 3 部分组成，而帧内容又可以分为帧头、数据字段及 CRC 共三个部分。其中部分字段的含义如下：

- SOF 和 EOF：这 2 个有序集用于标识帧的开始和结束，且 SOF 和 EOF 在不同的帧、不同的使用环境中的具体数值是不一样的
- 帧头：在帧格式中，帧头是一个 24 B 的字段，按照字边界进行传输。用于控制链路操作、设备协议的传输以及帧丢失或乱序检测，帧头中每个字段的含义如下：
  - R_CTL：说明帧的类型，1 字节
  - D_ID：帧的目标地址，3 字节
  - CS_CTL：帧的分类控制信息，1 字节
  - S_ID：帧的源地址，3 字节
  - Type：帧内的协议类型，1 字节
  - F_CTL：帧控制字段，3 字节
  - SEQ_ID、DF_CTL、SEQ_CNT：用于帧的顺序控制，共 4 字节
  - OX_ID：发起者交换机的 ID，2 字节
  - RX_ID：响应者交换机的 ID，2 字节
  - PARM：参数字段，4 字节
- CRC 字段：该字段包含 4 B 的 CRC 校验码，用于验证 FC 帧中的帧头和数据字段的完整性，但是 SOF 和 EOF 没有包含在 CRC 校验中

3.5.4 FC 流量控制

在链路层上，FC 定义了两种流控策略：一种为端到端的流控，另一种为缓存到缓存的流控。端到端流控比缓存到缓存流控要上层和高级。在一条链路的两端，首先面对链路的一个部件就是缓存。接收电路将一帧成功接收后，就放入了缓存中。如果由于上位程序处理缓慢而造成缓存已经充满，FC 协议还有机制来通知发送方减缓发送。如果链路的一端是 FC 终端设备，另一端是 FC 交换机，则二者之间的缓存到缓存的流量控制只能控制这个 FC 终端到 FC 交换机之间的流量。而通信的最终目标是网络上的另一个 FC 终端，这之间可能经历了多个 FC 交换机和多条链路。而如果数据流在另外一个 FC 终端之上发生拥塞，则这个 FC 终端就必须通知发起端降低发送频率，这就是“端到端”的流量控制。

3.5.5 FC Fabric

Fabric 网络架构是近些年十分流行的数据中心网络架构，其中最具有代表性的就是 Facebook 在 2014 年公开的其数据中心架构。Fabric 网络架构主要包含客户端节点、CA 节点、Peer 节点、Orderer 节点等部分，其中，各个节点功能简述如下：

- CA 节点功能：Fabric 网络中的成员提供基于数字证书的身份信息，可选，可以用第三方生成的证书
- 客户端节点的功能：与 Fabric 区块链交互，实现对区块链的操作。常见 cli 容器及 SDK 客户端。客户端代表代表最终用户的实体。它必须连接到 peer 与区块链进行通信。客户端可以连接到其选择的任何 peer。客户端创建交易
- Peer 节点功能：Fabric 每个组织都包含一个或者多个 peer 节点，每个节点可以担任多种角色，常见的有：
  - Endorser Peer（背书节点）功能：对客户端发送交易提案时进行签名背书，充当背书节点的角色。背书（Endorsement）是指特定 Peer 节点执行交易并向生成交易提案（proposal）的客户端应用程序返回 YES/NO 响应的过程。背书节点是动态的角色在链码实例化的时设置背书策略（Endorsement policy），指定哪些节点对交易背书才有效。只有在背书时是背书节点，其他时刻是普通节点
  - Leader Peer（主节点）功能：主要负责与 orderer 排序节点通信，获取区块及在本组织进行同步。主节点的产生可以动态选举或者指定
  - Committer Peer（记账节点）功能：对区块及区块交易进行验证，验证通过后将区块写入账本中
  - Anchor Peer（锚节点）功能：主要负责与其他组织的锚节点进行通信
- Orderer 节点功能：排序服务节点接收包含背书签名的交易，对未打包的交易进行排序生成区块，广播给 Peer 节点。排序服务提供的是原子广播，保证同一个链上的节点为接收到相同的消息，并且有相同的逻辑顺序

并且，在 Fabric 中，引入了通道的概念。一般情况下，一条区块链网路的子链是按照“1 个通道 +1 个账本 +N 个成员”的基本组成。通道是两个或多个特定网络成员之间的通信的私有“子网”，用于进行需要数据保密的交易。在 Fabric 中，建立一个通道相当于建立了一个子链。创建通道是为了限制信息传播的范围，是和某一个账本关联的。每个交易都是和唯一的通道关联的。这会明确地定义哪些实体（组织及其成员）会关注这个交易。

3.5.6 FC Fabric 寻址

鉴于上述 Fabric 网络架构的特点，Fabric 网络架构中的寻址就是一个特别需要注意的问题，现在讨论该问题如下。

和以太网端口 MAC 地址类似，FC 网络中的每个设备自身都有一个 WWNN（World Wide Node Name），不管这个设备上有多少个 FC 端口，设备始终拥有一个固定的 WWNN 来代表它自身。然后，FC 设备的每个端口都有一个 WWPN（World Wide Port Name）地址，也就是说这个地址在世界范围内是唯一的，世界上没有两个接口地址是相同的。

FC Fabric 拓扑在寻址和编址方面与以太网又有所不同。具体体现在以太网交换设备上的端口不需要有 MAC 地址，而 FC 交换机上的端口都有自己的 WWPN 地址。这是因为 FC 交换机要做的工作比以太网交换机多，许多 FC 逻辑都被集成在 FC 交换机上，而以太网的逻辑相对就简单了许多，因为上层逻辑都被交给诸如 TCP/IP 这样的上层协议实现了。然而 FC 的 Fabric 网中，FC 交换机担当了很重要的角色，它需要处理到 FC 协议的最上层。每个 FC 终端设备除了和最终通信的目标有交互之外，还需要和 FC 交换机打好交道。

WWNN 每个 FC 设备都被赋予一个 WWNN，这个 WWNN 一般被写入设备的 ROM 中不能改变，但是在某些条件下也可以通过运行在设备上的程序动态的改变。注意，WWPN 和三个 IDWWPN 地址的长度是 64 位，比以太网的 MAC 地址还要长出 16 位。然而，如果 8B 长度的地址用于高效路由的话，无疑效率太低。所以 FC 协议决定在 WWPN 之上再映射一层寻址机制，就是像 MAC 和 IP 的映射一样，给每个连接到 FC 网络中的接口分配一个 Fabric ID，用这个 ID 而不是 WWPN 来嵌入链路帧中做路由。这个 ID 长 24 位，高 8 位被定义成 Domain 区分符、中 8 位被定义为 Area 区分符、低 8 位定义为 PORT 区分符。

也就是说，WWPN 被映射到 Fabric ID，一个 Fabric ID 所有 24 位又被分成 Domain ID、Area ID、Port ID 这三个亚寻址单元，它们各自的含义如下：

- Domain ID：用来区分一个由众多交换机组成的大的 FC 网络中每个 FC 交换机本身。一个交换机上所有接口的 Fabric ID 都具有相同的高 8 位，即 Domain ID。Domain ID 同时也用来区分这个交换机本身，一个 Fabric 中的所有交换机拥有不同的 Domain ID。一个多交换机组成的 Fabric 中，Domain ID 是自动被主交换机分配给各个交换机的。根据 WWNN 号和一系列的选举帧的传送，WWNN 最小者获胜成为主交换机，然后这个主交换机向所有其他交换机分配 Domain ID，这个过程其实就是一系列的特殊帧的传送、

解析和判断

- Area ID：用来区分同一台交换机上的不同端口组，比如 1、2、3、4 端口属于 Area 1，5、6、7、8 端口属于 Area 2 等。其实 Area ID 这一层亚寻址单元意义不是很大。我们知道，每个 FC 接口都会对应一块用来管理它的芯片，然而每个这样的芯片却可以管理多个 FC 端口。所以如果一片芯片可以管理 1、2、3、4 号 FC 端口，那么这个芯片就可以属于一个 Area，这也是 Area 的物理解释。同样，在主机端的 FC 适配卡上，一般也都是用一块芯片来管理多个 FC 接口的
- Port ID：用来区分一个同 Area 中的不同 Port

经过这样的三段式寻址体系，我们可以区分一个大 Fabric 中的每个交换机、交换机中的每个端口组及每个端口组中的端口，这就是 FC Fabric 的寻址过程。

3.5.7 FC Fabric 登录

下面以 FC 设备登录到 Fabric 网络为例介绍 FC Fabric 的应用过程。

首先，发起者向地址为 0xFFFFFE 的注册服务器发送 FLOGI 请求，注册服务器向发起者发送回端口地址。发送者获取许可并接收到端口地址之后，向名称服务器发送端口注册请求，是否允许注册则由名称服务器决定，如果发送者得到注册许可，那么将同时收到由名称服务器发来的可访问设备列表。

在这一过程中，有两套编址体系（Fabric 和 WWPN）起作用，那么就必须有地址映射法则来处理这个问题，FC 协议中地址映射步骤如下：

- 当一个接口连接到 FC 网络中时，如果是 Fabric 架构，那么这个接口会发起一个登录注册到 Fabric 网络的动作，也就是向目的 Fabric ID 地址 FFFFFE 发送一个登录帧，称为 FLOGIN
- 交换机收到地址为 FFFFFE 的帧之后，会动态地给这个接口分配一个 24b 的 Fabric ID，并记录这个接口对应的 WWPN，做好映射
- 此后这个接口发出的帧中不会携带其 WWPN，而是携带其被分配的 ID 作为源地址

这样，就完成了 FC 设备登录到 Fabric 网络的过程。

3.6 FC SAN 与 IP SAN 的比较

下表列出了 FC-SAN 技术和 IP-SAN 技术的比较：

	IP SAN	FC SAN	说明
网络速度	1Gb、10Gb	1Gb、2Gb、4Gb、8Gb	
网络架构	使用现有 IP 网络	单独建设光纤网络和 HBA 卡	SAN 技术中的 iSCSI 协议可以使用现有 IP 网络，但这会导致 SAN 性能受限
传输距离	理论上没有距离限制	受到光纤传输距离的限制	
技术成熟度	IP SAN 是在最近 2 年开始为业务所推行，并为用户说认识的新技术	FC SAN 是从九十年代末即开始发展的存储网络技术，其发展已经历至少三代：1Gb、2Gb 以及目前的 4Gb 乃至 8Gb。其应用已将近十年，是非常成熟和可靠的技术，其采用和认可的广泛程度可以说是遍布几乎各类机构，大中小型企业的数据中心里	在技术成熟度上，FC SAN 比 IP SAN 要高得多
协议效率	IP SAN 实质上就是将 SCSI 指令封装在 IP 包中，利用 IP 技术进行包的传输，利用的是 IP 技术的广泛性和普及性。但是 IP 封装有一个显著的弱点：就是 IP 封装的开销大，效率低——即任何一个 IP 包中要附加的包头和包尾，以及检验码过多因此其总体的效率不高	比较 FC SAN，将 SCSI 指令在 FC 包中进行封装，包头和包尾以及校验码所占比例非常低，因此其效率非常高	从效率上看，FC SAN 明显高于 IP SAN，因此 FC SAN 更加适合于对效率敏感的应用，例如对性能要求很高的数据库应用，而 IP SAN 则主要应用到对性能和效率要求不高的环境中，例如 OA，文档处理，多媒体环境等
性能	IP SAN 协议中的封装效率不高，因此 IP SAN 对环境的硬件速度要求更好，才能获得与 FC 差不多的性能，可惜的是目前 IP SAN 最好环境还只是在千兆网中，因此其性能目前还无法与 FC SAN 相比	FC SAN 协议本身效率高，同时目前 FC SAN 已经开始普遍部署 4Gbps 的环境，所以说 FC SAN 要比目前 IP SAN 性能块很多	乐观地讲，10Gb 即万兆网中 IP SAN 的性能可能会有显著改善，并能初步满足相关应用的要求，与目前 FC SAN 中的性能可以一比

	IP SAN	FC SAN	说明
稳定性和安全性	较低，容易丢包、截取	较高	IP SAN 是建立在普通 IP 网上，FC SAN 是建立在 FC 网络中。FC 网络的抗干扰性要强；同时 FC 网络的封闭性要高一些，不想 IP 网络非常开放，因此 FC SAN 协议上要相对安全和稳定
成本	与 FC SAN 相比，购买与维护成本都比较低，有更高的投资收益比例	购买（光纤交换机、HBA 卡、光纤磁盘阵列等）、维护（培训人员、系统设置与监测等）成本高	IP SAN 的成本低主要体现在：设备价位低和运行维护费用低
容灾能力	本身可以实现本地和异地容灾，且成本低	容灾的硬件、软件成本高	
兼容性	目前 IP SAN 主要完成的是 Windows，Linux 等较低端的服务器的兼容性测试；厂商支持度：服务器方面，主要是 PC Server 厂商和低端 Unix 服务器明确支持，部分高端服务器还不支持兼容性；存储方面：虽然大多数存储都能支持 IP SAN，但是在用户环境中应用的主要还是中低端存储	FC SAN 兼容性测试已非常充分，遍布所有高端、中端、以及低端的服务器均能支持，厂商支持度：不管服务器还是存储方面，几乎所有的服务器（不论档次）和独立存储系统都完全支持 FC	在兼容性和厂商支持度上看，同样建议对于重要和关键业务系统，目前还是要采用 FC 要更稳定和可靠

从上表可以看出，在非关键环境中、或在建设成本非常有限的条件下，可以考虑采用 IP SAN；而对于企业或者机构核心业务、关键业务中，其稳定性、性能、对技术的成熟度要求高，应当采用 FC SAN，这样企业和机构的主要业务系统将更有保障。

3.7 SAN 与 DAS 的比较

DAS（Direct Attached Storage，直连式存储）技术是一种将数据直接存储在本地（未接入网络）的存储设备上的存储方式，其通常由机械硬盘、固态硬盘、光盘等存储单元构成。一个典型的 DAS 设备通常通过 HBA（Host Bus Adapter，主机总线适配器）直接连接到计算机上，通常作为已有存储设备的拓展，这决定

了 DAS 技术不具有易于在不同主机之间直接进行数据共享的特点，因为在不同主机之间没有类似于 hub、switch、router 等网络设备进行连接。

和 DAS 相比，SAN 具有的优点包括：

- 通过网络连接，便于进行不同主机之间的数据共享，尤其是数据的备份，其可以隐藏在服务器后透明地完成，极大方便了用户的使用
- 当网络情况稳定时，SAN 提供的服务往往具有更好的性能，因为 SAN 设备可以针对文件服务进行精确调整

而 SAN 相比 DAS 的缺点包括：

- SAN 提供的文件服务不如 DAS 稳定，SAN 提供的服务往往和当前网络的速度和拥塞程度有极大的关系，而 DAS 服务的质量则仅取决于本地计算机的硬件配置，其服务质量是可以在一定程度上得到保证的
- DAS 可以提供更加具有特色的存储服务，往往支持对硬件和低级软件的定制；而 SAN 受限于网络连接要求，其硬件基础往往是不支持定制的
- 块级存储固有缺陷决定了文件访问时具有的局限性，所以 SAN 往往需要搭配“共享文件系统”进行使用

3.8 SAN 和 NAS 的比较

NAS（Network Attached Storage，网络附加存储）技术是一种将分布、独立的数据整合为大型、集中化管理的数据中心、从而便于不同主机和应用对服务器进行访问的文件级技术。简单来讲，NSA 就是一种“将文件数据存储在网络上供主机进行存取和处理”的技术。NAS 可以被定义为一种特殊的专用数据存储服务器，其最重要的功能就是跨平台文件共享功能。NAS 通常在一个 LAN 上占有自己的节点，无需其他应用服务器的干预，并允许用户从该网络上进行数据的存取、更新、删除等操作。在这种配置中，NAS 集中管理和处理所在局域网上的所有数据，有效降低数据管理和共享的成本。

NAS 通常占用所在 LAN 上的一个专用节点，从而对其他其他主机提供数据服务。从此角度来看，不妨认为 NAS 服务器是一种专用的数据服务器，它可以授权网络用户和异质客户端从集中位置存储和检索数据。显然，NAS 具有灵活和易于横向扩展的特点。NAS 的专用目的决定了其硬件和软件基础也通常是专用的，其硬件、软件和配置共同构成其颇具特色的文件服务。NAS 的实现基础通常包含一个或多个通常排列成逻辑存储器、冗余存储器或 RAID 存储驱动器的网络设备。通常来讲，用于 NAS 的硬盘驱动器在功能上与其他驱动器相似，但在固件、振动耐受性或功耗上有不同之处，以使其更适合在 RAID 阵列中使用。

NAS 支持的协议包括 Andrew File System、Apple Filing Protocol、Server Message Block、File Transfer Protocol、SSH File Transfer Protocol、Hypertext Transfer

Protocol、Network File System 等协议，这些协议往往在不同领域具有各自的优势，在不同的使用场景中使用了网络附加存储（NAS）的概念。

和 NAS 相比，SAN 具有如下优点：

- SAN 架构的速度往往高于 NAS 架构，这是因为 NAS 架构大量使用以太网和 TCP/IP 协议进行内存传输，这样做不但增加了大量的 CPU 指令周期，而且使用了低速传输介质，但 SAN 中大部分操作都由适配卡上的硬件完成，CPU 开销的增加有限
- 由于目前 NAS 的根本瓶颈是底层链路的速度，因此当底层链路速度较低或不稳定时，SAN 架构所能提供服务的质量显著优于 NAS 架构

而 SAN 相比 NAS 的缺点包括：

- 造价较高，在 SAN 中必须使用专有协议例如 Fibre Channel、iSCSI 和 Infiniband 等，因此 SAN 通常有自己的网络和存储设备（尽管 iSCSI 可以运行在现有设备上，但这会导致 SAN 性能下降），必须单独购买、安装和配置，使得 SAN 技术的安装和使用比较昂贵。但在 NAS 架构中，数据通过成熟的 TCP/IP 协议进行传输，这可以利用已有的硬件基础
- NAS 提供统一的文件系统，方便文件的存储和共享，而 SAN 仅提供基于块的存储，这将文件系统问题遗留给了用户端，虽然 SAN 往往提供匹配的共享文件系统，但这种方案所能提供的服务及其稳定性仍与 NAS 有一定的差距

需要注意的是，尽管存在差异，但 SAN 和 NAS 并不相互排斥，并且可以组合为 SAN-NAS 混合体，从而提供来自同一系统的文件级协议（NAS）和块级协议（SAN）。

3.9 SAN 总结与典型应用场景

SAN 技术是一种连接网络从而提供数据存储功能的技术，其主要特点包括：块级存储、以数据为中心、将存储设备与服务器分离、集中管理数据等。SAN 技术有效地提高了数据服务的速度、提高了性能、增加了系统的鲁棒性，这使其实际性能表现远高于使用其他的一些存储方式。

通常来讲，SAN 技术由于使用了 FC 接口，因此可以提供比 NAS 技术更高性能的存储服务，现在如下列举几个典型应用场并进行相应的分析。

典型应用一：高性能高可靠服务。在金融、电信、电力、轨交等重点行业客户的关键业务部门的核心业务，希望采用“高性能、高可靠、低延迟、高造价”的服务来保证核心业务的稳定性和可靠性，这时采用 FC-SAN 架构就可以提高关键业务的高可用性。

典型应用二：异构混合存储架构的云服务。一方面，用户可能想要核心业务高性能、高可靠；另一方面，在多业务整合、弹性业务、备份归档等这些需要大容量存储但对性能要求不是很高的应用场景中，用户又希望采用分布式存储的方案提升业务的整合比与性价比，因此，越来越多的用户把目光转向兼具 FC-SAN 和 SDS 两者优势的异构混合存储架构，这种存储方案采用统一的存储资源池来统一管理基于 SAN 技术的异构混合存储架构。

典型应用三：数据库底层应用机制。当 Windows 或 Linux 系统采用 iSCSI、或 FC 协议进行网络连接时，文件系统在服务器中，并且，SAN 存储中没有文件系统，只有数据块。正是因为 SAN 中没有文件系统，存储设备的负载才会比较轻。因此，SAN 尤其适用于 I/O 请求次数多、数据访问频繁的场景，典型应用为数据库。实际上，在各种公有云、私有云中，基本底层的存储都是基于 SAN 的，如 FC-SAN、Server SAN 等。

4 HDFS 技术

Apache Hadoop 作为一个开源软件实用程序的集合，可以通过使用多台计算机的网络来解决涉及大量数据和计算的问题。它使用 MapReduce 为大数据的分布式存储（Clustered File System）及其处理提供了一个有效的软件框架。Hadoop 中的所有模块的设计都基于一个基本假设：硬件故障是常见的，并且应该由框架自动处理。

Apache Hadoop 的核心由称为 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS）的存储部分和作为 MapReduce 编程模型的处理部分共两个主要部分组成。Hadoop 将文件拆分成块并将它们分布在集群中的各个节点上，然后将打包的代码传输到节点以并行处理数据。这种方法很好地利用了数据局部性，节点在其中操纵它们有权访问的数据。这使得数据的处理速度甚至比在传统的超级计算机架构中更快、更有效。

4.1 HDFS 特点

HDFS 可以在多台机器上存储大文件，并通过跨多个主机复制数据来实现可靠性，因此理论上不需要独立磁盘冗余阵列（RAID）主机上的存储。默认情况下，数据存储在三个节点上：两个在同一机架（rack）上、一个在不同机架上。数据节点可以相互对话以重新平衡数据、移动副本并保持数据的高复制率。需要注意的是，HDFS 专为大多数不可变文件而设计，可能不适合需要并发写入操作的系统。

HDFS 文件系统包括一个 Secondary Name Node，该结构定期与 Name Node 连接并构建主要 Name Node 目录信息的快照，然后系统将其保存到本地或远程目录。这些带检查点的图像可用于重启失败的系统、而无需重播整个文件系统操作日志。在实际使用中，HDFS 存在一些问题，例如小文件问题、可扩展性问题、单点故障以及巨大元数据请求中的瓶颈。

使用 HDFS 的优势之一是 Job Tracker 和 Task Tracker 之间的数据感知。Job Tracker 在了解数据位置的情况下将作业映射或减少到 Task Tracker。例如：如果节点 A 包含数据 (a, b, c)，节点 X 包含数据 (x, y, z)，Job Tracker 调度节点 A 对 (a, b, c) 执行映射（map）或缩减（reduce）任务，同时调度节点 X 对 (x, y, z) 上执行映射或缩减任务。这样，HDFS 减少了通过网络的流量并避免了不必要的数据传输。

在使用时，可以通过 Java API、Thrift API（生成多种语言的客户端，例如 C++、Java、Python、PHP、Ruby、Erlang、Perl、Haskell、C#、Cocoa、Smalltalk 和 OCaml）实现文件访问，还可以使用命令行界面、基于 HTTP 的 HDFS-UI 的 Web 应用程序、第三方网络客户端库等实现访问。

HDFS 旨在实现跨各种硬件平台的可移植性，以及与各种底层操作系统的兼容性。但是，为了实现这一目标，HDFS 在设计时引入了一些可能导致性能瓶颈的可移植性限制。由于 HDFS 已经广泛集成到了企业级基础架构中，大规模监控 HDFS 性能已成为一个越来越重要的问题。监控端到端性能需要跟踪来自 Data Node、Name Node 和底层操作系统的指标。

4.2 HDFS 设计目标

我们前面提到，Hadoop 中的所有模块的设计都基于一个基本假设：硬件故障是常见的，并且应该由框架自动处理，这是因为 HDFS 通常由成百上千的服务器所构成，每个服务器上存储着文件系统的部分数据。我们面对的现实是构成系统的组件数目是巨大的，而且任一组件都有可能失效，这意味着总是有一部分 HDFS 的组件是不工作的。因此错误检测和快速、自动的恢复是 HDFS 最核心的目标之一。

其次，为了最高效地使用 HDFS，运行在 HDFS 上的应用应当有很大的数据集。HDFS 上的一个典型文件大小一般都在 G 字节至 T 字节。因此，HDFS 应当被调整至支持大型文件，从而提供较高的数据传输带宽，并能在一个集群里扩展到数百个节点。通常来说，一个单一的 HDFS 实例应该能支撑数以千万计的文件。

然后，流式数据访问在 HDFS 上是必要的，这是因为 HDFS 中存储的数据往往是规模很大的数据，因此最常见的操作方式就是一次写入、多次读取。数据源通常由源生成或从数据源直接复制而来，接着长时间在此数据集上进行各类分析，大数据不需要、也最好避免搬来搬去。因此运行在 HDFS 上的应用和普通的应用不同，需要流式访问它们的数据集。HDFS 的设计中更多的考虑到了数据批处理，而不是用户交互处理。

之后，由于 HDFS 需要一个“一次写入多次读取”的文件访问模型，那么简单的一致性模型是非常必要的：一个文件经过创建、写入和关闭之后就不需要改变。MapReduce 应用或者网络爬虫应用都非常适合这个模型。

而且，我们注意到一个应用请求的计算离它操作的数据越近就越高效，在数据达到海量级别的时候更是如此。因为这样就能降低网络阻塞的影响，提高系统

数据的吞吐量。显然，由于 HDFS 的数据的海量性，将计算移动到数据附近，显然比将数据移动到应用附近更好。因此，HDFS 需要为应用提供将它们自己移动到数据附近的功能接口。

最后，我们还需要考虑异构软硬件平台之间的可移植性，这种特性可以方便 HDFS 作为大规模数据应用平台的推广。

4.3 HDFS 架构

HDFS 是用 Java 为 Hadoop 框架编写的分布式、可扩展且可移植的文件系统。如前所述，Hadoop 实例分为 HDFS 和 MapReduce。HDFS 用于存储数据，MapReduce 则用于处理数据。

HDFS 采用主从架构。一个 HDFS 集群是由一个 Name Node 和一定数目的 Data Node 组成。Name Node 是一个中心服务器，负责管理文件系统的名字空间以及客户端对文件的访问。集群中的 Data Node 一般是一个节点一个，负责管理它所在节点上的存储。用户能够以文件的形式在 HDFS 管理的名称空间中存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组 Data Node 上。Name Node 执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体 Data Node 节点的映射。Data Node 负责处理文件系统客户端的读写请求。在 Name Node 的统一调度下进行数据块的创建、删除和复制。总的来说，HDFS 有以下五种节点：

- Name Node：HDFS 只包含一个 Name Node，称为主节点。主节点可以跟踪文件、管理文件系统并拥有其中所有存储数据的元数据。Name Node 还包含块数的详细信息、存储数据的 Data Node 的位置、存储复制的位置以及其他详细信息。Name Node 与客户端直接联系
- Secondary Name Node：这是为了处理 Name Node 中文件系统元数据而设置的检查点。这也称为检查点节点。它是 Name Node 的辅助节点。Secondary Name Node 指示 Name Node 创建并发送 fsimage 和 editlog 文件，Secondary Name Node 在该文件上创建压缩的 fsimage 文件
- Job Tracker：接收来自客户端的 MapReduce 执行请求。Job Tracker 与 Name Node 对话以了解将在处理中使用的数据的位置，Name Node 则以所需处理数据的元数据进行响应
- Data Node：Data Node 以块的形式存储数据。这也称为从节点，它将实际数据存储到 HDFS 中，HDFS 负责客户端读写。每个 Data Node 每 3 秒向 Name Node 发送一个 Heartbeat 消息，以通知它是活动的。这样，如果 Name Node 在 2 分钟内没有收到来自 Data Node 的 Heartbeat 消息时，它将认为这个 Data Node 已死亡，并在其他数据节点上启动块复制过程
- Task Tracker：它是 Job Tracker 的从属节点，它会从 Job Tracker 接受任务和代码。任务跟踪器将获取代码并应用于文件，在文件上应用该代码的过程称为映射器

前三者是主服务，后两名是从属服务。主服务可以相互通信，从属服务可以以相同的方式相互通信。例如，Name Node 是主节点，Data Node 是其对应的从属节点，可以相互通信。虽然 Hadoop 集群理论上只有一个 Name Node 和一个 Data Node 集群，但是 Name Node 也可以冗余构造，从而保证系统的正确且稳健地运行。每个 Data Node 使用特定于 HDFS 的块协议通过网络提供数据块。文件系统使用 TCP/IP 套接字进行通信，客户端则使用远程过程调用进行相互通信。

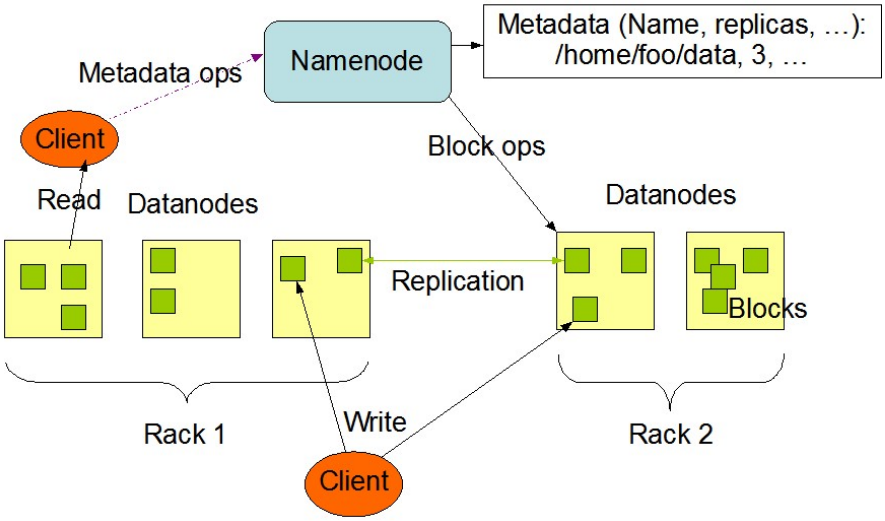

图 2: HDFS 架构图

此外，在 HDFS 系统中还有块（block）的概念。磁盘的物理块是磁盘操作最小的单元，读写操作均以块为最小单元，一般为 512 字节。文件系统块通常是物理磁盘块的整数倍，通常为数千字节。而 HDFS 的块比一般单机文件系统大得多，默认为 128M 字节。HDFS 的文件被拆分成基于块单元的 chunk，chunk 作为独立单元存储。比一个块小的文件不会占用整个块，而只会占据实际大小。HDFS 中设置这么大的块是为了最小化查找时间。假设定位到块所需的时间为 10ms，磁盘传输速度为 100M/s。如果要将定位到块所用时间占传输时间的比例控制在 1%，则块大小需要约 100M。但是如果块设置过大，则在 MapReduce 任务中，map 或者 reduce 任务的个数如果小于集群机器数量，会使得作业运行效率很低。

4.3.1 Name Node、HDFS 副本保存与 Secondary Name Node

Name Node 上保存着 HDFS 的名字空间。对于任何对文件系统元数据产生修改的操作，Name Node 都会使用一种称为 editlog 的事务日志记录下来。例如，在 HDFS 中创建一个文件，Name Node 就会在 editlog 中插入一条记录来表示；同样地，修改文件的副本系数也将往 editlog 插入一条记录。Name Node 在本地操作系统的文件系统中存储这个 editlog。整个文件系统的名字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为 fsimage 的文件中，这个文件也是放在 Name Node 所在的本地文件系统上。

Name Node 在内存中保存着整个文件系统的名字空间和文件数据块映射的映像。这个关键的元数据结构通常设计得很紧凑，因此一个 4G 内存的 Name Node 足够支撑大量的文件和目录。当 Name Node 启动时，它从硬盘中读取 editlog 和 fsimage，并将所有 editlog 中的事务作用在内存中的 fsimage 上，并将这个新版本的 fsimage 从内存中保存到本地磁盘上，然后删除旧的 editlog，因为这个旧的 editlog 的事务都已经作用在 fsimage 上了。这个过程称为一个检查点（checkpoint）。检查点可以在 Name Node 启动时起作用，也可以周期性起作用。

Data Node 将 HDFS 数据以文件的形式存储在本地的文件系统中，它并不知道有关 HDFS 文件的信息。它把每个 HDFS 数据块存储在本地文件系统的一个单独的文件中。Data Node 并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目录中支持大量的文件。当一个 Data Node 启动时，它会扫描本地文件系统，产生一个这些本地文件对应的所有 HDFS 数据块的列表，然后作为报告发送到 Name Node，这个报告就是块状态报告。

HDFS 副本的保存由 Name Node 决定。HDFS 的副本的存放策略是可靠性、写带宽、读带宽之间的权衡。默认策略如下：

- 第一个副本放在客户端相同的机器上，如果机器在集群之外，随机选择一个（但是会尽可能选择容量不是太慢或者当前操作太繁忙的）
- 第二个副本随机放在不同于第一个副本的机架上
- 第三个副本放在跟第二个副本同一机架上，但是不同的节点上，满足条件的节点中随机选择
- 更多的副本在整个集群上随机选择，虽然会尽量便面太多副本在同一机架上

副本的位置确定后，在建立写入管道时，会考虑网络拓扑结构。这样选择很好地平衡了可靠性、读写性能：

- 可靠性：块分布在两个机架上
- 写带宽：写入管道的过程只需要跨越一个交换机
- 读带宽：可以从两个机架中任选一个读取

鉴于 Name Node 的重要性，在 HDFS 中，Name Node 可能成为集群的单点故障：当 Name Node 不可用时，整个文件系统也是不可用的，这时往往采取下面两种方式进行处理：

1. 备份持久化数据：将文件系统的元数据同时写到多个文件系统，例如同时将元数据写到本地文件系统及 NFS。这些备份操作必须都是同步的、原子的

2. Secondary Name Node：该节点定期合并 Name Node 的 fsimage 以及 editlog 两个文件，同时为避免节点占用空间过大，可以通过创建检查点来合并。需要注意的是，Secondary Name Node 通常运行在另一台机器，因为合并操作需要耗费大量的 CPU 和内存。其数据落后于 Name Node，因此当 Name Node 完全崩溃时，会出现数据丢失。通常做法是拷贝 NFS 中的备份元数据到 Secondary Name Node，将其作为新的 Name Node

4.3.2 Data Node、缓存与 HA 方案

Data Node 负责存储和提取块，读写请求可能来自 Name Node，也可能直接来自客户端。数据节点周期性向 Name Node 汇报自己节点上所存储的块的相关信息。

Data Node 通常采用一种称为集中式缓存管理（Centralized Cache Management）的缓存方案，该方案可以防止那些被频繁使用的数据从内存中清除。Data Node 直接从磁盘读取数据，并且频繁使用的块可以在内存中缓存。默认情况下，一个块只有一个数据节点会缓存。但是可以针对每个文件可以个性化配置。Job Tracker 可以利用缓存提升性能，例如 MapReduce 可以把任务运行在有块缓存的节点上。用户或者应用可以向 Name Node 发送缓存指令（例如指定缓存哪个文件、缓存多久等），缓存池的概念用于管理一组缓存的权限和资源。

另外，由于 Name Node 的内存会制约文件数量，因此人们提出了 HDFS Federation 的概念来提供一种横向扩展 Name Node 的方式。在 Federation 模式中，每个 Name Node 管理命名空间的一部分，例如一个 Name Node 管理/user 目录下的文件，另一个 Name Node 管理/share 目录下的文件。每个 Name Node 同时维护一个块池（block pool）来保存块的节点映射等信息。各 Name Node 之间是独立的，一个节点的失败不会导致其他节点管理的文件不可用。为了与之匹配，客户端使用挂载表（mount table）将文件路径映射到 Name Node。挂载表其实就是在 Name Node 群组之上封装了一层。

显然，Data Node 的故障也是一种单点错误。虽然将元数据同时写到多个文件系统以及 Secondary Name Node 定期检查点有利于保护数据丢失，但是并不能提高可用性。这是因为 Name Node 是唯一一个对文件元数据和文件块进行映射的地方，当它 Data Node 出现问题之后，包括 MapReduce 在内的作业都无法进行读写。为了处理 Data Node 的故障，常见的做法是使用元数据备份重新启动一个 Name Node，元数据的备份来自以下两个地方：

1. 多文件系统中写入的备份
2. Secondary Name Node 的检查点文件

启动新的 Name Node 之后，需要重新配置客户端的 Data Node 和 Name Node 信息，耗时非常大，这是因为：

- 元数据镜像文件载入到内存耗时较长

- 需要重放 editlog 文件
- 需要收到来自 Data Node 的状态报告并且满足条件后才能离开安全模式提供写服务

因此，往往采取高可用（High Available，HA）方案。

采用 HA 的 HDFS 集群配置两个 Name Node，分别处于 Active 和 Standby 状态。当 Active Name Node 故障之后，Standby 接过责任继续提供服务，用户没有明显的中断感觉。这个操作一般耗时在几十秒到数分钟。HA 涉及到的主要实现逻辑有：

- Active 和 Standby 节点共享 editlog 存储：Active Name Node 和 Standby Name Node 共享同一份 editlog，在切换时通过回放 editlog 来同步数据。具体来讲，共享 editlog 存储往往有两种实现方式：
  - NFS：传统的网络文件系统
  - QJM：Quorum Journal Manager，QJM 是专门为 HDFS 的 HA 实现而设计的，用来提供高可用的 editlog。QJM 运行一组 Journal Node，editlog 必须写到大部分的 Journal Nodes。通常使用三个节点，因此允许一个节点失败，这类似于 ZooKeeper。虽然 QJM 没有使用 ZooKeeper，但 HA 方案的确使用了 ZooKeeper 来选举 Name Node。一般推荐使用 QJM 实现方式
- Data Node 需要同时向 Active 和 Standby 节点发送块报告（block report）：因为块映射数据存储在内存中，为了在 Active Name Node 出现故障之后，新的 Name Node 能够快速启动、而不需要等待来自 Data Node 的块报告，Data Node 需要同时向 Active 和 Standby 两个 Name Node 发送块报告
- 客户端需要 fallover 配置（对用户透明）：Name Node 的切换对客户端来说是无感知的，通过客户端库来实现。客户端在配置文件中使用的 HDFS URI 是逻辑路径，映射到一对 Name Node 地址。客户端会不断尝试每一个 Name Node 地址直到成功
- Standby 替代 Secondary Name Node：如果没有启用 HA，HDFS 独立运行一个守护进程作为 Secondary Name Node，定期检查点，合并 fsimage 和 editlog 两个文件

4.4 ZooKeeper 简介

现在简要介绍上面提到的 ZooKeeper。

在 Zookeeper 集群中，有领导者（Leader）、跟随者（Follower）和观察者（Observer）三种角色；而 Zookeeper 集群中的节点又有选举状态（Looking）、领导者状态（Leading leader）和跟随者状态（Following follower）三种状态。

Zookeeper Atomic Broadcast（ZAB）协议实现了主备模式下的系统架构，保持集群中各个副本之间的数据一致性。ZAB 协议定义了选举（election）、发现（discovery）、同步（sync）和广播（Broadcast）四个阶段：

- 选举阶段：选出哪台为主机
- 发现阶段和同步阶段：当主机选出后，要做的恢复数据的阶段
- 广播阶段：当主机和从选出并同步好数据后，正常的主写同步从写数据的阶段

每次写成功的消息，都有一个全局唯一的标识，叫 zxid，是 64bit 的正整数，高 32 为叫 epoch 表示选举纪元，低 32 位是自增的 id，每写一次加一。

Zookeeper 集群一般都是奇数（如 $n = 2n + 1$）个机器，且只有一个主机 leader，其余都是从机 follower。必须要有至少 $n + 1$ 台选举相同，才能执行选举的操作。在投票优先级方面：优先比较 zxid，如果相等，再比较机器的 id，都按从大到小的顺序。

当集群新建、或者主机死机、或者主机与一半或以上的从机失去联系后，都会触发选择新的主机操作。有两种算法 fast paxos 和 basic paxos 实现选举。

默认 ZAB 采用的算法是 fast paxos 算法。每次选举都要把选举轮数加一，类似于 zxid 里的 epoch 字段，防止不同轮次的选举互相干扰。每个进入 looking 状态的节点，最开始投票给自己，然后把投票消息发给其它机器。内容为 < 第几轮投票, 被投节点的 zxid, 被投节点的编号 >。其他 looking 状态的节点收到后，进行如下判断：

- 判断票是否有效，方法为看票的投票轮数和本地记载的投票轮数是否相等：
  - 如果比本地投票轮数的小，丢弃
  - 如果比本地投票轮数的大：
    - 第一，证明自己投票过期了，清空本地投票信息
    - 第二，更新投票轮数和结果为收到的内容
    - 第三，通知其他所有节点新的投票方案
  - 如果和本地投票轮数相等，按照投票的优先级比较收到的选票和自己投出去的选票：
    - 如果收到的优先级大，则更新自己的投票为对方发过来投票方案，把投票发出去
    - 如果收到的优先级小，则忽略该投票
    - 如果收到的优先级相等，则更新对应节点的投票
- 每收集到一个投票后，查看已经收到的投票结果记录列表，看是否有节点能够达到一半以上的投票数。如果有达到，则终止投票，宣布选举结束，更新自身状态。然后进行发现和同步阶段。否则继续收集投票

而对于 basic paxos 算法来讲：

1. 每个 looking 节点先发出请求，询问其他节点的投票。其他节点返回自己的投票 < zk 的 id, zxid >，第一次都投自己
2. 收到结果后，如果收到的投票比自己投票的 zxid 大，更新自己的投票
3. 当收到所有节点返回后，统计投票，有一个节点的选举达到一半以上，则选举成功。否则继续开始下一轮询问，直到选择出 leader 结束

现在来比较 basic paxos 和 fast paxos 两种方法。fast paxos 方法是主动推送出，只要结果有更新，就马上同步给其他节点。其他节点可能还没把自己的票通知给所有节点，就发现自己投的票优先级低，要更新投票，然后更新再重新通知给所有节点。而 basic paxos 方法则要每一节点都询问完，才能知道新结果，然后再去问其他节点新的选举结果。fast 比 basic 快的地方是一个节点不用和每个节点都交换投票信息后才能知道自己的票是否要更新，这可以显著减少交互次数。

主从同步数据比较简单，当有写操作时，如果是从机接收，会转到主机。做一次转发，保证写都是在主机上进行。主先提议事务，收到过半回复后，再发提交。主收到写操作时，先本地生成事务为事务生成 zxid，然后发给所有 follower 节点。当 follower 收到事务时，先把提议事务的日志写到本地磁盘，成功后返回给 leader。leader 收到过半反馈后对事务提交。再通知所有的 follower 提交事务，follower 收到后也提交事务，提交后就可以对客户端进行分发了。

4.5 HDFS 读写流程

接下来简要介绍 HDFS 读写数据的流程。

4.5.1 读文件

执行读文件操作的执行步骤如下：

1. 客户端传递一个文件的路径给（分布式）文件系统的 open 方法
2. 分布式文件系统采用 RPC 远程获取文件最开始的几个块的 Data Node 地址，Name Node 会根据网络拓扑结构决定返回哪些节点（前提是节点有块副本），如果客户端本身是 Data Node 并且节点上刚好有块副本，则直接从本地读取
3. 客户端使用 open 方法返回的 FSDataInputStream 对象读取数据（调用 read 方法）
4. DFSInputStream（由 FSDataInputStream 修改得到的类）连接持有第一个块的最近的节点，反复调用 read 方法，从而流式地读取数据

5. 第一个块读取完毕之后，寻找下一个块的最佳 Data Node 来读取数据。如果有必要，DFSInputStream 会联系 Name Node 获取下一批块的节点信息。这些寻址过程对客户端都是不可见的

6. 数据读取完毕，客户端调用 close 方法关闭流对象

在读数据过程中，如果与 Data Node 的通信发生错误，DFSInputStream 对象会尝试从下一个最佳节点读取数据，并且记住该失败节点，后续块的读取不会再连接该节点。读取一个块之后，DFSInputStram 会进行检验和验证，如果块损坏，则尝试从其他节点读取数据，并且将损坏的块汇报给 Name Node。客户端连接哪个 Data Node 获取数据是由 Name Node 来指导的，这样可以支持大量并发的客户端请求，Name Node 尽可能将流量均匀分布到整个集群。

由于块的位置信息是存储在 Name Node 内存中的，因此相应位置请求非常高效，所以不会成为系统性能的瓶颈。

在读文件时，由于客户端和数据块在同一节点上，所以 Data Node 不需要出现在读取数据路径中。而客户端本身可以直接从本地磁盘读取数据，这样会使读取性能得到很大的提高。Data Node 将所有数据块路径的权限开放给客户端，客户端直接通过本地磁盘路径来读取数据。但是一个显而易见的问题是，这些权限更改引入了一个安全漏洞：具有读取 Data Node 上数据块文件权限的用户可以任意读取路径上所有数据块，而不仅仅是他们所需访问的数据块。为了处理这个问题，Data Node 不是将目录传递给客户端，而是打开块文件和元数据文件，并将它们的文件描述符通过 domain socket 传递给客户端。文件描述符是只读的，因此客户端无法修改传递描述符的文件。客户端无法访问数据块目录本身，所以也无法读取它不应该访问的任何其他数据块文件。

4.5.2 写文件

执行读文件操作的执行步骤如下：

1. 客户端调用 DistributedFileSystem 的 create 方法

2. DistributedFileSystem 通过 RPC 调用 Name Node 在文件系统的命名空间中创建一个新文件，此时该文件没有关联到任何块。这个过程中，Name Node 会做很多校验工作，例如是否已经存在同名文件，是否有权限，如果验证通过，返回一个 FSDataOutputStream 对象。如果验证不通过，抛出异常到客户端

3. 客户端写入数据的时候，DFSOutputStream 分解为 packets，并写入到一个数据队列中，该队列由 DataStreamer 消费

4. DateStreamer 负责请求 Name Node 分配新的块存放的数据节点。这些节点存放同一个块的副本，从而构成一个管道。DataStreamer 将 packet 写入到管道的第一个节点，第一个节点存放好 packet 之后，转发给下一个节点，下一个节点存放之后继续往下传递

5. DFSOutputStream 同时维护一个 ack queue 队列，等待来自 Data Node 确认消息。当管道上的所有 Data Node 都确认之后，packet 从 ack 队列中移除
6. 数据写入完毕，客户端 close 输出流，从而将所有的 packet 刷新到管道中，然后安心等待来自 Data Node 的确认消息。全部得到确认之后告知 Name Node 文件是完整的。Name Node 此时已经知道文件的所有块信息（因为 DataStreamer 是请求 Name Node 分配块的），只需等待达到最小副本数要求，然后返回成功信息给客户端即可

在写数据时，往 HDFS 上写入新的数据块时，Data Node 将会为这个块选择存储的地方。目前 Hadoop 支持两种 volume 选择策略：round-robin（循环策略）和 available space（可用空间策略）：

- 循环策略：将新块均匀分布在可用磁盘上
- 可用空间策略：它是优先将数据写入具有最大可用空间的磁盘（通过百分比计算的）

默认情况下，Data Node 使用基于 round-robin 策略来写入新的数据块。然而在一个长时间运行的集群中，由于 HDFS 中的大规模文件删除或者通过往 Data Node 中添加新的磁盘，仍然会导致同一个 Data Node 中的不同磁盘存储的数据很不均衡。即使使用的是基于可用空间的策略，卷（volume）不平衡仍可导致较低效率的磁盘 I/O。比如所有新增的数据块都会往新增的磁盘上写，在此期间，其他的磁盘会处于空闲状态，这样新的磁盘将会是整个系统的瓶颈。

4.6 HDFS 的其他内容

4.6.1 HDFS 网关

HDFS 的 NFS 网关允许客户端挂载 HDFS 并通过 NFS 与其进行交互，就像它是本地文件系统的一部分一样。网关支持 NFSv3。安装 HDFS 后，用户可以：

- 在 NFSv3 客户端兼容的操作系统上通过其本地文件系统浏览 HDFS 文件系统
- 在 HDFS 文件系统和本地文件系统之间上载和下载文件
- 通过挂载点将数据直接传输到 HDFS（支持文件追加，但不支持随机写入）

使用 NFS 网关具有如下两个前提条件：

1. NFS 网关机器必须运行运行 HDFS 客户端所需的所有组件，例如 Hadoop 核心 jar 文件和 HADOOP_CONF 目录

2. NFS 网关可以安装在任何 Data Node，Name Node 或 HDP 客户端计算机上，并在该计算机上启动 NFS 服务器

HDFS 允许管理员给私人目录设置其下面文件夹和文件的总数量配额（name quota）、或空间使用总量配额（space quota）。所以 HDFS 配额的对象是目录，而非用户。如果需要实现用户级别的配额，则需要采用第三方系统进行逻辑管理并映射到文件夹配额。

4.6.2 内存存储

HDFS 支持写入由 Data Node 管理的堆外内存，这称为内存存储（Memory Storage）思路。Data Node 将异步刷新内存中的数据到磁盘，从而从性能敏感的输入输出路径中去掉昂贵的磁盘 I/O 和校验计算，这种写入被称为 Lazy Persist 写入。HDFS 为 Lazy Persist Writes 提供尽力而为的持久性保证。在将副本持久保存到磁盘之前，如果节点重新启动，则可能会丢失数据。应用程序可以选择使用 Lazy Persist Writes 来换取一些持久性保证，以减少延迟。如果内存不足或未配置，使用 Lazy Persist Writes 的应用程序将继续工作，退回 DISK 存储。

4.6.3 基于路由器的 HDFS Federation 方案

在前面的内容中，我们曾提到：为了解决 HDFS 的水平扩展性问题，HDFS Federation 方案被提出并实现。虽然该方案可以解决单命名空间带来的一些问题，但是又引出了新的问题，比如现在集群中存在多个命名空间，每个命名空间管理一部分数据，那客户端如何知道要查询的数据在哪个命名空间上呢？为了解决这个问题，视图文件系统（View File System，ViewFs）被提出并实现。ViewFs 类似于某些 Unix/Linux 系统中的客户端挂载表。ViewFs 可用于创建个性化命名空间视图，也可用于创建每个群集的公共视图。ViewFs 通过在 core-site.xml 文件中引入路径映射配置来将文件映射到对应的命名空间上。

ViewFs 方案虽然可以很好的解决文件命名空间问题，但是它的实现有以下几个问题：

- ViewFS 是基于客户端实现的，需要用户在客户端进行相关的配置，那么后面对客户端升级就会比较困难，这个客户端相当于重客户端了
- 新增或者修改路径映射，需要多方配合完成，维护成本比较高

为了解决这个问题，一种基于路由的 Federation 方案（Router-Based Federation）被提出并实现。和之前的 ViewFS 不一样的是，基于路由的 Federation 方案是通过服务端实现的，其在服务端添加了一个 Federation 层，这个额外的层允许客户端透明地访问任何子集群，让子集群独立地管理他们自己的块池，并支持跨子集群的数据平衡。Federation 层必须将块访问路由到正确的子集群，维护命名空间的状态，并提供数据重新平衡的机制，同时这个层必须具有可扩展性，高可用性和容错性。

Federation 层包含了多个组件：路由器、State Store 以及 Rebalancing Mechanism。一个系统中可以包含多个路由器。路由器和 Name Node 具有相同的接口，并根据 State Store 里面的信息将客户端请求转发到正确的子集群。State Store 是远程挂载表（remote mount table，和 ViewFS 方案中的配置文件类似，但在客户端之间共享，State Store 的物理实现是分布式的），存储子集群相关的信息包括 load/capacity 等。

在基于路由器的 HDFS Federation 方案中，如果客户端需要进行读写操作，则它的步骤如下：

1. 客户端向集群中任意一个路由器发出某个文件的读写请求操作
2. 路由器从 State Store 里面的挂载表查询哪个子集群包含这个文件，并获取正确的 Name Node
3. 路由器获取到正确的 Name Node 后，会将客户端的请求转发到 Name Node 上，然后也会给客户端一个请求告诉它需要请求哪个子集群
4. 此后，客户端就可以直接访问对应子集群的 Data Node，并进行读写相关的操作

4.6.4 纠删码

传统技术中，HDFS 为了数据容错，在存储的时候每个数据块会被复制三次。但数据块复制技术的开销比较大，因此纠删码（Erasure Code，EC）作为 HDFS 的一种新的特性被提出，这种技术就是来代替数据块复制技术的。在比较两种存储模式的时候，重点考虑的是数据持久性和存储利用率两方面的问题。

在存储系统中，纠删码技术主要是通过利用纠删码算法将原始的数据进行编码得到校验，并将数据和校验一并存储起来，以达到容错的目的。其基本思想是将 k 块原始的数据元素通过一定的编码计算，得到 m 块校验元素。对于这 $k+m$ 块元素，当其中任意的 m 块元素出错（包括数据和校验出错），均可以通过对应的重构算法恢复出原来的 k 块数据。生成校验的过程被成为编码（encoding），恢复丢失数据块的过程被称为解码（decoding）。

里德所罗门（Reed-Solomon，RS）码是存储系统较为常用的一种纠删码，它有两个参数 k 和 m，记为 $RS(k□m)$。k 个数据块组成一个向量被乘上一个生成矩阵（Generator Matrix，GT）从而得到一个码字（codeword）向量，该向量由 k 个数据块和 m 个校验块构成。如果一个数据块丢失，可以用 $(GT)-1$ 乘以码字向量来恢复出丢失的数据块。$RS(k,m)$ 最多可容忍 m 个块（包括数据块和校验块）丢失。

对 HDFS 的一个普通文件来说，构成它的基本单位是块。但对于 EC 模式下的文件，构成它的基本单位为块组。块组由一定数目的数据块加上生成的校验块放一起构成。以 $RS(6,3)$ 为例，每一个块组包含 1 到 6 个数据块，以及 3 个校验块。进行 EC 编码的前提是每个块的长度一致。如果不一致，则应填充 0。

纠删码目前较为典型的应用就是用于分布式系统。在云计算中可以采用副本来防止数据的丢失，但翻倍的数据存储空间代价往往十分高昂，EC 技术的运用就可以有效解决这个问题。需要注意的是，EC 的使用也是需要一些代价的，一旦数据需要恢复，就会导致网络带宽和编码解码两大消耗。所以，将此技术用于线上服务可能会不够稳定，所以最好的选择是用于冷数据集群：

- 冷数据集群往往有大量的长期没有被访问的数据，体量确实很大，采用 EC 技术，可以大大减少副本数
- 冷数据集群基本稳定，耗资源量少，所以一旦进行数据恢复，将不会对集群造成大的影响

4.6.5 异构存储

异构存储就是将不同需求或者冷热的数据存储到不同的介质中去，实现既能兼顾性能又能兼顾成本。对于存储到 HDFS 的数据大致可以分为四个等级：

1. 实时数据
2. 热数据
3. 冷数据
4. 极冷数据

并且，实践表明大部分的数据都是冷数据或者极冷数据，对于这部分数据，读请求很少，写请求也非常少，对访问延迟不敏感。如果将这部分数据存储通过高压缩比，并且存储到普通的 SATA 大容量盘中去，能极大地节约成本。而对于热数据和实时数据，写请求比较高，读请求也很高，但是数据量很小。这个时候为了实现高并发低延迟，我们可以将这部分数据保存到 SSD 中。

HDFS 支持 RAM_DISK、SSD、DISK 以及 ARCHIVE 四种类型的异构存储。异构存储的原理可以总结如下：

- 在 HDFS 的配置文件 hdfs-site.xml 中配置对应的异构存储
- Data Node 启动的时候从配置文件中读取对应的存储类型，以及容量情况，并通过 Heartbeat 的形式不断的上报给 Name Node
- Name Node 收到 Data Node 发送的关于存储类型、容量等内容的 Heartbeat 包后，会进行处理，更新存储的相关内容
- 写请求发到 Name Node 后，Name Node 根据写请求具体的目录对应的存储策略选择对应的存储类型的 Data Node 进行写入操作

HDFS 异构存储的管理内容主要包括：

- 统计线上数据的访问频率，确认冷热数据所在目录，灰度进行调整
- 使用 HDFS StoragePolicies 相关命令进行策略的调整
- 修改存储策略以后，使用 Mover 工具进行数据的迁移

其中，Mover 是 HDFS 的一个数据迁移工具，类似 Balancer。区别在于，Mover 的目的是把数据块按照存储策略迁移，而 Balancer 是在不同 Data Node 之间直接进行平衡。如果 Data Node 挂载了多种存储类型，Mover 会优先尝试在本地迁移，从而避免开销代价大的网络 I/O 操作。

4.7 HDFS 总结与典型应用场景

从前面的分析中，我们可以得出 HDFS 的主要优点包括：高容错性、适合批处理、擅长大数据处理、流式文件访问、可在廉价机器上构建；而 HDFS 的缺点包括：对低延迟数据访问的支持有限、对小文件存储的支持有限、难以进行并发写入和文件随机修改。总的来说，HDFS 技术是一个高度容错性的系统，适合部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS 放宽了一部分 POSIX 约束，来实现流式读取文件系统数据的目的。HDFS 技术有效地提高了数据服务的速度、提高了性能、增加了系统的鲁棒性，这使其实际性能表现远高于使用其他的一些存储方式。

下面列举 HDFS 技术的几个典型应用场景并进行分析。

典型应用一：结构化数据库采集与分析。对于结构化数据库，我们通常使用 Sqoop 工具进行数据采集，可以实现结构化数据库中数据并行批量入库到 HDFS 存储，从而便于进行下一步的数据处理和分析。得益于 HDFS 技术的诸多优点，利用 HDFS 进行结构化数据采集和分析是非常方便的。我们还可以加强数据采集过程的调度和任务监控，从而在实际使用中得到更好的效果。

典型应用二：网页数据采集与分析。HBase 作为面向列的数据库运行在 HDFS 之上，其最初目的是为了解决 HDFS 缺乏随机读写操作的问题。HBase 以 Google BigTable 为蓝本，以键值对的形式存储。项目的目标就是快速在主机内数十亿行数据中定位所需的数据并访问它。可以将 HBase 认为是一个 NoSql 的数据库，像其他数据库一样提供随即读写功能。对于网页采集，前端可以使用 Nutch，全文检索则使用 lucense，而实际数据存储最好是入库到 Hbase 数据库，从而便于后续处理与分析。

典型应用三：日志分析与推荐系统。Hive 是一个基于 Hadoop 的开源数据仓库工具，用于存储和处理海量结构化数据。它是 Facebook 于 2008 年 8 月开源的一个数据仓库框架，提供了类似于 SQL 语法的 HQL 语句作为数据访问接口。淘宝搜索中的自定义筛选也使用了 Hive 技术。天猫的推荐系统使用了与 Hive 架构类似的技术。Pig 是一种操作 Hadoop 的轻量级脚本语言、也是一种数据流语言，用来快速轻松的处理巨大的数据。Pig 可以非常方便的处理 HDFS 和 HBase 的数据，和 Hive 一样，Pig 可以非常高效的处理需求，通过直接操作 Pig 查询可以节省大量的劳动和时间。利用 Pig 还可以做高级的数据处理，包括 Twitter、

LinkedIn 上用于发现您可能认识的人，可以实现类似 Amazon.com 的协同过滤的推荐效果。淘宝的商品推荐也使用了类似技术；雅虎公司 40% 的 Hadoop 作业是用 Pig 实现的，包括垃圾邮件的识别和过滤、用户特征建模等。

典型应用四：算法及机器学习。实现基于机器学习的自动的智能化数据价值挖掘是大数据和 Hadoop 领域十分具有前景的方向，也是很多企业对大数据平台的最终期望。随着可获得的数据越来越多，未来大数据平台的价值更多取决于其计算人工智能的程度。为了在 Hadoop 集群上支持深度学习，用户需要指定使用的 Spark executor 个数、每个 executor 分配的 GPU 个数、HDFS 上存放训练数据的路径以及模型在 HDFS 上的存储路径，用户使用标准 Caffe 配置文件来确定 Caffe 算法和深度网络的拓扑结构。相关研究表明，将 Hadoop 生态系统和深度学习集成在同一个异构集群的做法可以带来显著的性能提升。

5 Ceph 技术

Ceph 是一个开源软件定义存储平台，它在单个分布式计算机集群上实现对象存储，并为对象、块和文件提供三合一接口级存储。Ceph 的主要目标是完全分布式操作，主要特点包括：没有单点故障、可扩展到 EB 级别、并且可以免费使用。从版本 12 开始，Ceph 不再依赖其他文件系统，可以直接管理硬盘并且具有自己的存储后端。

Ceph 使用现有硬件设备和网络 IP，不需要特定的硬件支持。Ceph 的系统通过复制、擦除编码、快照和存储克隆等技术提供灾难恢复和数据冗余。Ceph 既能自我修复又能自我管理，是一个高度自治的系统，可以最大限度地减少管理时间和其他成本。这个统一的系统可以在一个通用的管理框架内收集存储。Ceph 整合了多个存储用例并提高了资源利用率。它还允许组织在需要的地方部署服务器。

目前一些投入使用的 Ceph 部署包括 CERN、OVH 和 DigitalOcean 等。

5.1 Ceph 特点

Ceph 主要具有以下四大特点：

1. 高性能：
    - 摒弃了传统的集中式存储元数据寻址的方案，采用 CRUSH 算法，数据分布均衡，并行度高
    - 考虑了容灾域的隔离，能够实现各类负载的副本放置规则，例如跨机房、机架感知等
    - 能够支持上千个存储节点的规模，支持 TB 到 PB 级的数据
2. 高可用性：

- 副本数可以灵活控制
- 支持故障域分隔，数据强一致性
- 多种故障场景自动进行修复自愈
- 没有单点故障，并可以自动管理

3. 高可扩展性：
    - 去中心化
    - 扩展灵活
    - 随着节点增加而线性增长
4. 特性丰富：
    - 支持三种存储接口：块存储、文件存储、对象存储
    - 支持自定义接口，支持多种语言驱动

现在来区分一下 Ceph 支持的三种存储类型：

- 块存储：典型设备是磁盘阵列与硬盘，主要是将裸磁盘空间映射给主机使用的
    - 优点：通过 Raid 与 LVM 等手段，对数据提供了保护；多块廉价的硬盘组合起来，提高容量；多块磁盘组合出来的逻辑盘，提升读写效率
    - 缺点：采用 SAN 架构组网时，光纤交换机，造价成本高；主机之间无法共享数据
    - 使用场景：docker 容器、虚拟机磁盘存储分配；日志存储；文件存储
- 文件存储：典型设备是 FTP、NFS 服务器。主要是为了克服块存储文件无法共享的问题，所以才有了文件存储。我们可以在服务器上架设 FTP 与 NFS 服务，就是文件存储
    - 优点：造价低；方便文件共享
    - 缺点：读写速率低；传输速率慢
    - 使用场景：日志存储，有目录结构的文件存储
- 对象存储：典型设备是内置大容量硬盘的分布式服务器（比如 swift，s3 等），通过对多台服务器内置大容量磁盘、安装对象存储管理软件，对外提供读写访问功能
    - 优点：具备块存储的读写高速；具备文件存储的共享等特性
    - 缺点：成本可能较高
    - 使用场景：图片存储；视频存储

5.2 Ceph 架构

在 Ceph 架构中，有以下的若干核心组件：

- 集群监视器（monitor）：一个 Ceph 集群需要多个 Monitor 组成的小集群，它们跟踪活动和失败的集群节点、集群配置以及有关数据放置和全局集群状态的信息，并可以通过 Paxos 同步数据，用来保存 OSD 的元数据
- 使用直接日志磁盘存储的对象存储设备（Object Storage Device，OSD）：负责响应客户端请求返回具体数据的进程，一个 Ceph 集群一般有多个 OSD。从版本 12 开始，独立于文件系统的 BlueStore 替代了原来的使用文件系统的 FileStore
- 元数据服务器（Metadata Server，MDS）：用于缓存和代理对 CephFS 文件系统内的 inode 和目录的访问，是 CephFS 服务依赖的元数据服务
- RADOS（Reliable Autonomic Distributed Object Store）：是 Ceph 集群的核心部分，用户据此实现数据分配、Failover 等集群操作
- Librados：是 RADOS 支持库，因为作为协议的 RADOS 很难直接访问，因此上层的 RBD、RGW 和 CephFS 都是通过 Librados 访问的，目前提供 PHP、Ruby、Java、Python、C 和 C++ 支持
- RADOS 网关（RADOS gateway，RGW）：将对象存储层与 Amazon S3 或 OpenStack Swift API 兼容的接口相接
- RADOS block device（RBD）：是 Ceph 对外提供的块设备服务
- 对象（Object）：Ceph 最底层的存储单元是 Object 对象，每个 Object 包含元数据和原始数据
- CRUSH：是 Ceph 使用的数据分布算法，类似一致性哈希，让数据分配到预期的地方
- 放置组（Placement Group，PG）：这是一个逻辑的概念，一个 PG 往往包含多个 OSD，引入 PG 这一层其实是为了更好的分配数据和定位数据
- CephFS（Ceph File System）：是 Ceph 对外提供的文件系统服务
- 管理器（Manager）：执行集群监控、簿记和维护任务，并与外部监控系统和管理进行接口

所有这些组件都是完全分布式的，并且可以在同一组服务器上运行。具有不同需求的客户可以直接与他们的不同子集进行交互。

Ceph 跨多个节点对单个文件进行条带化（striping）以实现更高的吞吐量，这类似于 RAID0 跨多个硬盘驱动器对分区进行条带化。Ceph 支持自适应负载平衡，从而将频繁访问的对象复制到更多节点上。

BlueStore 是 Ceph 自己特有的存储实现，现在已经成为实际使用中默认且广受欢迎的存储类型，它可以提供比文件存储后端更好的延迟和可配置性，并避免了基于文件系统的存储方案（如 FileStore）需要额外处理和缓存层的缺点。

5.2.1 RADOS

RADOS 是 Ceph 存储系统的核心，也称为 Ceph 存储集群。Ceph 的数据访问方法（如 RBD、CephFS、RADOSGW、Librados）的所有操作都是在 RADOS 层之上构建的。当 Ceph 集群接收到来自客户端的写请求时，CRUSH 算法首先计算出存储位置，然后这些信息传递到 RADOS 层进行进一步处理。RADOS 以小对象的形式将数据分发到集群内的所有节点，最后将这些对象存储在 OSD 中。当配置的复制数大于 1 时，RADOS 负责数据的可靠性，它复制对象，创建副本并将它们存储在不同的故障区域中。

RADOS 包含两个核心组件：OSD 和 MON。

OSD 是 Ceph 存储集群中最重要的一个基础组件，由一个已经存在 Linux 文件系统的物理磁盘驱动器和 OSD 服务组成。OSD 负责将实际的数据以对象的形式存储在每一个集群节点的物理磁盘中。对于任何读写操作，客户端首先向 MON 请求集群的映射地址，然后客户端就可以直接和 OSD 进行 I/O 操作。

一个 Ceph 集群包含多个 OSD，通常的做法是，一个 Ceph 集群部署方案会为集群节点上的每个物理磁盘创建一个 OSD 守护进程。OSD 上的每个对象都有一个主副本和几个辅副本，辅副本分散在其他 OSD。一个 OSD 对于一些对象是主副本，同时对于其他对象可能是辅副本，存放辅副本的 OSD 手主副本 OSD 控制，如果主副本 OSD 异常（或者对应的磁盘故障），辅副本 OSD 可以成为主副本 OSD。

对于任意一个 OSD，共有四种可能的状态：

- up 且 in：说明该 OSD 正常运行，且已经承载至少一个 PG 的数据。这是一个 OSD 的标准工作状态
- up 且 out：说明该 OSD 正常运行，但并未承载任何 PG，其中也没有数据。一个新的 OSD 刚刚被加入 Ceph 集群后，便会处于这一状态。而一个出现故障的 OSD 被修复后，重新加入 Ceph 集群时，也是处于这一状态
- down 且 in：说明该 OSD 发生异常，但仍然承载着至少一个 PG，其中仍然存储着数据。这种状态下的 OSD 刚刚被发现存在异常，可能仍能恢复正常，也可能会彻底无法工作
- down 且 out：说明该 OSD 已经彻底发生故障，且已经不再承载任何 PG

MON 则负责监控整个集群的健康状况。它以守护进程的形式存在，一个 MON 为每一个组件维护一个独立的 map，如 OSD、MON、PG、CRUSH 和 MDS map。这些 map 统称为集群 map。MON 不为客户端存储和提供数据，它只为客

户端以及集群内其他节点提供更新集群 map 的服务。客户端和集群内其他节点定期与 MON 确认自己持有的是否是集群最新的 map。一个 Ceph 集群通常包含多个 MON 节点，但是同一时间只有一个 MON 起决定作用。

5.2.2 对象

一个对象通常包含绑定在一起的数据和元数据，并且用一个全局唯一的标识符标识。这个唯一的标识符确保在整个存储集群中没有其他对象使用相同的对象 ID，保证对象唯一性。基于文件的存储中，文件大小是有限制的，与此不同的是，对象的大小是可以随着大小可变的元数据而变得很大。对象不使用一个目录层次结构或树结构来存储，相反，它存储在一个包含数十亿对象且没有任何复杂性的线性地址空间中。对象可以存储在本地，也可以存放在地理上分开的线性地址空间中，也就是说，在一个连续的存储空间中。任何应用程序都可以基于对象 ID 通过调用 restful API 从对象中获取数据。这个 URL 可以以同样的方式工作在因特网上，一个对象 ID 作为一个唯一的指针指向对象。这些对象都以复制的方式存储在 OSD 中，因此能提供高可用性。

5.2.3 Pool 与 PG

Pool 是一个命名空间，客户端向 RADOS 上存储对象时需要指定一个 Pool，Pool 通过配置文件定义并可以指定 Pool 的存储 OSD 节点范围和 PG 数量。

PG 则是 Pool 内部的概念，是对象和 OSD 的中间逻辑分层，对象首先会通过简单的 Hash 算法来得到其存储的 PG，这个 PG 和对象是确定的。然后每一个 PG 都有一个 primary OSD 和几个 Secondary OSD，对象会被分发都这些 OSD 上存储，而这个分发策略称为 CRUSH-Ceph 的数据均匀分布的核心。

总的来说，Pool 和 PG 的关系具有以下特点：

- Pool 是 Ceph 存储数据时的逻辑分区，它起到命名空间的作用
- 每个 Pool 包含一定数量（可配置）的 PG
- PG 里的对象被映射到不同的 Object 上
- Pool 是分布到整个集群的
- Pool 可以做故障隔离域，根据不同的用户场景不一进行隔离

需要注意的是，因为整个 CRUSH 以上的流程实现都是在客户端计算，所以客户端本身需要保存一份 Cluster Map，而这是从 Monitor 处获得。从这里我们也可以了解到 Monitor 主要职责就是负责维护这份 Cluster Map 并保证强一致性。

具体来讲，PG 通过复制它到不同的 OSD 上来提供存储系统的可靠性。根据 Pool 的复制级别，每个 PG 的数据会被复制并分发到 Ceph 集群的多个 OSD 上。

可以将 PG 看成一个逻辑容器，这个容器包含多个对象，同时这个逻辑容器被映射到多个 OSD。

计算正确的 PG 数对一个 Ceph 存储集群来说是至关重要的一步。PG 数计算公式如下：

$$\text{PG 总数} = (\text{OSD 总数} \times 100)/\text{最大副本数}$$

此外，在实际使用中，还支持数据扩容 PG 分布，每个 PG 会自动散落在不同的 OSD 上。如果扩容那么相应的 PG 会进行迁移到新的 OSD 上，从而保证 PG 数量的均衡。

5.2.4 CRUSH

CRUSH 全称 Controlled Replication Under Scalable Hashing，是一种数据分发算法，类似于哈希和一致性哈希。哈希的问题在于数据增长时不能动态增加 Bucket，一致性哈希的问题在于加 Bucket 时数据迁移量比较大，其他数据分发算法依赖中心的 Metadata 服务器来存储元数据效率较低，CRUSH 则是通过计算、接受多维参数的来解决动态数据分发的场景。

CRUSH 算法接受的参数包括 cluster map，也就是硬盘分布的逻辑位置，例如这有多少个机房、多少个机柜、硬盘是如何分布的等等。我们可以将 cluster map 进行层次化处理，得到类似树的多层结果（例如，根节点-数据中心-机房-机架-主机-OSD），从而更好地反映存储层级的物理拓扑结构。这个类似树的结构中的子节点是真正存储数据的 device，每个 device 都有 id 和权重，中间节点是 bucket，bucket 有多种类型用于不同的查询算法，例如一个机柜一个机架一个机房就是 bucket。另一个参数是 placement rules，它指定了一份数据有多少备份，数据的分布有什么限制条件，例如同一份数据不能放在同一个机柜里等的功能。placement rules 的主要特点包括：

- 从 CRUSH Map 中的哪个节点开始查找
- 使用哪个节点作为故障隔离域
- 定位副本的搜索模式（广度优先 or 深度优先）

事实上，每个 rule 就是一系列操作。例如，take 操作就是就是选一个 bucket；select 操作就是选择 n 个类型是 t 的项；emit 操作就是提交最后的返回结果。select 主要的考虑方面主要包括是否冲突、是否有失败和负载问题。CRUSH 算法的还有一个输入是整数 x，输出则是一个包含 n 个目标的列表 R，例如三备份的话输出可能是 $[1, 3, 5]$。

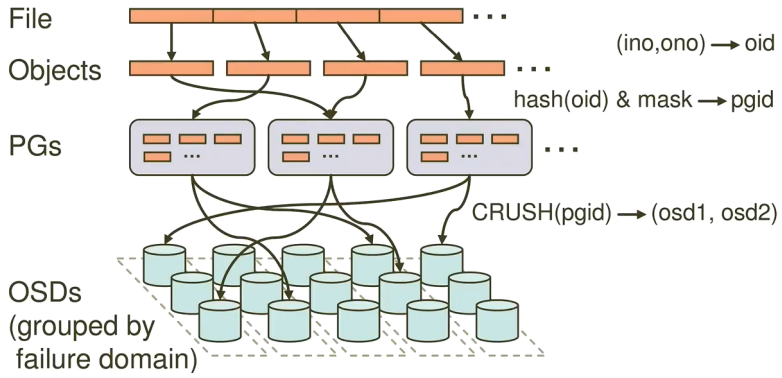

图 3: CRUSH 架构图

上图显示了使用 CRUSH 的一般结构，其中：

- File：是用户需要存储或者访问的文件。对于一个基于 Ceph 开发的对象存储应用而言，这个 File 也就对应于应用中的对象，也就是用户直接操作的对象
- Object：是 RADOS 所看到的对象。Object 与 File 的区别是，Object 的最大大小由 RADOS 限定（通常为 2MB 或 4MB），以便实现底层存储的组织管理。因此，当上层应用向 RADOS 存入大小很大的 File 时，需要将 File 切分成统一大小的一系列 Object（最后一个的大小可以不同）进行存储
- OSD：OSD 的数量关系到系统的数据分布均匀性，因此其数量不应太少。在实践当中，至少也应该是数十上百个的量级才有助于 Ceph 系统的设计发挥其应有的优势

Ceph 的客户端和 Ceph 的 OSD 守护进程都使用 CRUSH 算法来高效地对数据容器的信息需求进行计算，而不必依赖于一个查找表。CRUSH 与旧方法相比，提供了更好的数据管理机制，使大规模清晰地将工作分配到集群中的所有客户端和 OSD 守护。CRUSH 使用智能的数据复制，以确保弹性，这样可以更好的适合于超大规模存储。

CRUSH 通过伪随机算法来确保均匀的数据分布，它的输入是 PG、Cluster State 和 Policy，并且保证 CRUSH 的 Object Name 一样的情况下，即使后两者参数发生改变也会得到一致的读取。并且这个 CRUSH 算法是可配置的，通过 PG Number 和 Weight 的指定可以得到不同的分布策略。这个可配置的分布策略让 Ceph 的能力得到极大加强。

CRUSH 的具体使用流程可以说明如下：

1. 完成 File 到 Object 的映射：这次映射的目的是，将用户要操作的 File，映射为 RADOS 能够处理的 Object。其映射十分简单，本质上就是按照 Object 的最大大小对 File 进行切分，相当于 RAID 中的条带化过程。这种切分的好处有二：一是让大小不限的 File 变成最大大小一致、可以被 RADOS 高效

管理的 Object；二是让对单一 File 实施的串行处理变为对多个 Object 实施的并行化处理。每一个切分后产生的 Object 将获得唯一的 Object Id（oid）。其产生方式也是线性映射。在图 x 中，ino 是待操作 File 的元数据，可以简单理解为该 File 的唯一 id。ono 则是由该 File 切分产生的某个 Object 的序号。而 oid 就是将这个序号简单连缀在该 File id 之后得到的。举例而言，如果一个 id 为 filename 的 File 被切分成了三个 Object，则其 Object 序号依次为 0、1 和 2，而最终得到的 oid 就依次为 filename0、filename1 和 filename2。需要注意的是，ino 的唯一性必须得到保证，否则后续映射无法正确进行

2. 然后完成从 Object 到 PG 的映射：Ceph 指定一个静态哈希函数计算 oid 的值，将 oid 映射成一个近似均匀分布的伪随机值，然后和 mask 按位相与，得到 pgid。首先是使用 Ceph 系统指定的一个静态哈希函数计算 oid 的哈希值，将 oid 映射成为一个近似均匀分布的伪随机值。然后，将这个伪随机值和 mask 按位相与，得到最终的 PG 序号（pgid）。根据 RADOS 的设计，给定 PG 的总数为 m（m 应该为 2 的整数幂），则 mask 的值为 $m-1$。因此，哈希值计算和按位与操作的整体结果事实上是从所有 m 个 PG 中近似均匀地随机选择一个。基于这一机制，当有大量 Object 和大量 PG 时，RADOS 能够保证 Object 和 PG 之间的近似均匀映射。又因为 Object 是由 File 切分而来，大部分 Object 的大小相同，因而，这一映射最终保证了，各个 PG 中存储的 Object 的总数据量近似均匀。显然，只有当 Object 和 PG 的数量较多时，这种伪随机关系的近似均匀性才能成立，Ceph 的数据存储均匀性才有保证。为保证 “大量” 的成立，一方面，Object 的最大大小应该被合理配置，以使得同样数量的 File 能够被切分成更多的 Object；另一方面，Ceph 也推荐 PG 总数应该为 OSD 总数的数百倍，以保证有足够数量的 PG 可供映射

3. 最后实现从 PG 到 OSD 的映射：将作为 Object 的逻辑组织单元的 PG 映射到数据的实际存储单元 OSD。RADOS 采用一个名为 CRUSH 的算法，将 pgid 代入其中，然后得到一组共 n 个 OSD。这 n 个 OSD 即共同负责存储和维护一个 PG 中的所有 Object。前已述及，n 的数值可以根据实际应用中对于可靠性的需求而配置，在生产环境下通常为 3。具体到每个 OSD，则由其上运行的 OSD deamon 负责执行映射到本地的 Object 在本地文件系统中的存储、上一步中采用的哈希算法不同，这个 CRUSH 算法的结果不是绝对不变的，而是受到其他因素的影响。其影响因素主要有两点：

   (a) 当前系统状态，也就是上文逻辑结构中曾经提及的 cluster map。当系统中的 OSD 状态、数量发生变化时，cluster map 可能发生变化，而这种变化将会影响到 PG 与 OSD 之间的映射

   (b) 存储策略配置。这里的策略主要与安全相关。利用策略配置，系统管理员可以指定承载同一个 PG 的 3 个 OSD 分别位于数据中心的不同服务器乃至机架上，从而进一步改善存储的可靠性

在第三次映射中，只有在系统状态（cluster map）和存储策略都不发生变化的时候，PG 和 OSD 之间的映射关系才是固定不变的。在实际使用当中，策略一经配置通常不会改变。而系统状态的改变或者是由于设备损坏，或者是因为存储集群

规模扩大。Ceph 本身提供了对于这种变化的自动化支持，因此即便 PG 与 OSD 之间的映射关系发生了变化，也并不会对应用造成困扰。事实上，Ceph 正是需要有目的的利用这种动态映射关系。正是利用了 CRUSH 的动态特性，Ceph 可以将一个 PG 根据需要动态迁移到不同的 OSD 组合上，从而自动化地实现高可靠性、数据分布 re-blancing 等特性。

需要注意的是，之所以在此次映射中使用 CRUSH 算法、而不是其他哈希算法，原因之一正是 CRUSH 具有上述可配置特性，可以根据管理员的配置参数决定 OSD 的物理位置映射策略；另一方面是因为 CRUSH 具有特殊的“稳定性”，也即，当系统中加入新的 OSD，导致系统规模增大时，大部分 PG 与 OSD 之间的映射关系不会发生改变，只有少部分 PG 的映射关系会发生变化并引发数据迁移。这种可配置性和稳定性都不是普通哈希算法所能提供的。因此，CRUSH 算法的设计也是 Ceph 的核心内容之一。

总的来说，Ceph 通过三次映射，首先基于池 ID 将对象名和集群 PG 数应用散列函数得到一个 pgid，然后，针对这个 pgid 执行 CRUSH 查找得到主 OSD 和辅助 OSD，完成了从 File 到 Object、PG 和 OSD 整个映射过程，最后写入数据。

5.2.5 心跳机制

心跳（Heartbeat）用于节点间检测对方是否存在故障，从而及时发现故障节点并进入相应处理流程。但是，心跳机制往往存在以下问题：

- 故障检测时间和心跳报文带来的负载之间做权衡
- 心跳频率太高则过多的心跳报文会影响系统性能
- 心跳频率过低则会延长发现故障节点的时间，从而影响系统的可用性

同时，故障检测策略应该能够做到以下几点：

- 及时性：节点发生异常如宕机或网络中断时，集群可以在秒级时间内发现节点失效并汇报 Monitor，Monitor 将在分钟级时间内将失效 OSD 下线
- 压力分散：Ceph 实际上将故障检测过程中由中心节点承担的压力分散到所有的 OSD 上，以此提高中心节点 Monitor 的可靠性，进而提高整个集群的可扩展性通过调整对节点和网络的压力来尽可能调高利用率
- 稳定性：可以容忍一定程度的网络抖动与网络延迟
- 惰性扩散通知机制：节点存活状态改变导致的元信息变化需要通过某种机制扩散到整个集群，并且，作为中心节点的 Monitor 没有在更新 OSD map 后立刻广播通知所有的 OSD 和 Client，而是惰性的等待 OSD 和 Client 来减少 Monitor 压力并简化交互逻辑

心跳机制的实现思路为：

- 同一个 PG 内 OSD 互相心跳，他们互相发送 ping/pong 信息
- OSD 节点监听 public、cluster、front 和 back 四个端口，其中：
  - public 端口：监听来自 Monitor 和 Client 的连接
  - cluster 端口：监听来自 OSD Peer 的连接
  - front 端口：为客户端连接集群使用的网卡，用于临时监听集群内心跳
  - back 端口：为客户端集群内部使用的网卡，用于监听集群内部之间的心跳
- 每隔一段时间（典型值是六秒）检测一次，并且会随机小幅调整间隔时间避免峰值问题
- 如果 20 秒没有检测到心跳回复，则加入 failure 队列，进行后续的处理

正常情况下，OSD 周期性地向 Monitor 提交正常内容的报告；对于出现失效节点的情况，OSD 将按照下面的思路向 Monitor 提交指定内容的报告：

- OSD 检查 failure 队列中的伙伴 OSD 失败信息
- 向 Monitor 提交失效报告，并将失败信息加入 failure 等待队列，然后将其从 failure 队列中移除
- 如果收到来自 failure 队列或者 failure 等待队列中的 OSD 的心跳，则将其从两个队列中移除，并告知 Monitor 取消之前的失效报告
- 当发生与 Monitor 网络重连时，会将 failure 等待队列中的错误报告加回到 failure 队列中，并再次发送给 Monitor

5.2.6 Ceph 通信架构

在 Ceph 中，我们采用订阅发布模式（Subscribe/Publish，又称为观察者模式）来实现网络通信。通过定义对象之间的一对多关系，当一个对象的值发生变化时，所有依赖于它的对象都得到通知并被自动更新。结构如下：

- Dispatcher：组件包括 OSD、Monitor 和 MDS
- Messenger：可以进一步细分为 SimpleMessenger，其中 SimpleMessenger 中的组件包括 Accepter 和 Pipe

其中，Accepter 监听对象的请求，并调用 SImpleMessenger::add_accept_pipe() 方法创建新的 Pipe 到 SimpleMessenger::pipes 来处理该请求；Pipe 用于消息的读取和发送，主要组件包括 pipe::reader 和 pipe::write；Messenger 为消息的发布者，各个 Dispatcher 子类作为消息的订阅者，在收到消息之后通知该类进行处理；在 Dispatcher 中有一个 DispatchQueue 类来缓存收到的消息，然后唤醒 DispatchQueue::dispatch_thread 线程找到后端的 Dispatch 处理消息。

5.2.7 Ceph 后端存储引擎

Ceph 的后端存储引擎主要包括 FileStore、NewStore、BlueStore 等：

- FileStore：利用文件系统的 POSIX 接口实现 ObjectStore API。每个 Object 在 FileStore 层会被看成是一个文件，Object 的属性会利用文件的 xattr 属性存取，因为有些文件系统（如 Ext4）对 xattr 的长度有限制，因此超出长度的元数据会被存储在 DBObjectMap 或 omap 中。通常 DBObjectMap 的存储引擎时 LevelDB 或 RocksDB。在实际使用时，所有写事务的数据和元数据都会先写入 journal 文件，写入 journal 成功以后立即返回。journal 类似于数据库的 write-ahead-log。写入完成后，会使用 O_DIRECT 和 O_DSYNC 同步写入到 journal 文件，随后该事务会被转移到 FileStore 的写入队列中，数据和元数据被异步写到指定区域
- NewStore（又可称为 Key-File Store）：通过将元数据从传统的文件系统上分离出来，使用专门的 KV 数据库如 LevelDB 或 RocksDB 来进行管理，而对应的对象数据依旧保存在文件系统中，从而避免文件系统作为后端存储引擎时的元数据碎片化问题，从而提升 Ceph 的整体性能
- BlueStore：主要组件有三个：BlockDevice、BlueFS 和 RocksDB。BlockDevice 为最底层的块设备，通常为 HDD 或者 SSD，BlueStore 直接操作块设备，抛弃了 XFS 等本地文件系统；BlueFS 是一个小的文件系统，其文件系统的文件和目录的元数据都保存在全部缓存在内存中，持久化保存在文件系统的日志文件中，当文件系统重新 mount 时，重新 replay 该日志文件中保存的操作，就可以加载所有的元数据到内存中；RocksDB 是 Facebook 在 leveldb 上开发并优化的 KV 存储系统。本身是基于文件系统的，不是直接操作裸设备。它将系统相关的处理抽象成 Env，用户可实现相应的接。BlueFS 的主要的目的，就是支持 RocksDB Env 接口，保证 RocksDB 的正常运行

5.3 数据操作流程

当某个客户端需要向 Ceph 集群写入一个 File 时，首先需要在本地完成寻址流程，将 File 变为一个 Object，然后找出存储该 Object 的一组三个 OSD。这三个 OSD 具有各自不同的序号，序号最靠前的那个 OSD 就是这一组中的 Primary OSD，而后两个则依次是 Secondary OSD 和 Tertiary OSD。

找出三个 OSD 后，客户端将直接和 Primary OSD 通信，发起写入操作（步骤 1）。Primary OSD 收到请求后，分别向 Secondary OSD 和 Tertiary OSD 发起写入操作（步骤 2、3）。当 Secondary OSD 和 Tertiary OSD 各自完成写入操作后，将分别向 Primary OSD 发送确认信息（步骤 4、5）。当 Primary OSD 确信其他两个 OSD 的写入完成后，则自己也完成数据写入，并向客户端确认 Object 写入操作完成（步骤 6）。

之所以采用这样的写入流程，本质上是为了保证写入过程中的可靠性，尽可能避免造成数据丢失。同时，由于客户端只需要向 Primary OSD 发送数据，因此，

在 Internet 使用场景下的外网带宽和整体访问延迟又得到了一定程度的优化。

5.4 I/O 流程分析

5.4.1 正常 I/O 流程

正常的 I/O 流程需要客户端先创建 cluster handler，然后读取相关的配置文件，并在连接上 monitor 后获取到集群的 map 信息。此后 client 通过 I/O 接口根据 crashmap 请求得到对应主 osd 数据节点。主 osd 数据节点同时写入另外两个副本节点数据。客户端在等待主节点和两个副本节点写完数据之后，客户端收到相应消息，至此 I/O 写入完成。

5.4.2 新主 I/O 流程

现在的问题是：如果新加入的 osd1 取代了原有的 osd2 成为了主 osd，但是 osd1 上尚未创建 PG 故不存在数据，此时 PG 上的 I/O 理论上是无法运行的，这应该怎么办？解决方案是：客户端在连接 monitor 时获取到集群 map 信息，同时新主 osd1 由于没有 PG 数据会主动上报 monitor 告知让 osd2 临时接替为主 osd。临时主 osd2 会把全部数据同步给新主 osd1。这样，客户端的 I/O 读写将直接连接到主 osd2 进行读写，并且 osd2 在收到读写 I/O 后，一方面将数据同步给 osd1、另一方面写入另外两副本节点。在 osd2 及两副本节点写入成功后，osd2 会将三者的写入成功信息返回给客户端，客户端这时就知道读写 I/O 已经执行完毕。如果 osd1 数据同步完毕，临时主 osd2 则会交出主角色，从而 osd1 变为主节点、osd2 称为副本节点。

5.4.3 Ceph I/O 流程

Ceph I/O 流程中使用了 CURSH 方法，具体内容已在前面介绍，现简要回顾如下：首先完成从 File 到 Object 映射；然后 Ceph 指定一个静态哈希函数计算 oid 的值，将 oid 映射成一个近似均匀分布的伪随机值，然后和 mask 按位相与，得到 pgid；之后完成从 Object 到 PG 映射；最后完成从 PG 到 OSD 的映射。

5.4.4 Ceph RBD I/O 流程

前面提到，RADOS block device（RBD）是 Ceph 对外提供的块设备服务。实际上，Ceph 兼具对象、块、文件三种存储形态，三种存储形态中，块设备方式可以完全兼容已有应用，因此使用范围最为广泛。Ceph RBD I/O 流程可以概括为：

1. 客户端创建一个 pool，需要为这个 pool 指定 pg 的数量
2. 创建 pool/image rbd 设备进行挂载

3. 用户写入的数据进行切块，每个块的大小默认为 4M，并且每个块都有一个名字，名字就是 object+ 序号
4. 将每个 object 通过 pg 进行副本位置的分配
5. pg 根据 cursh 算法会寻找 3 个 osd，把这个 object 分别保存在这三个 osd 上
6. osd 上实际是把底层的 disk 进行了格式化操作，一般部署工具会将它格式化为 xfs 文件系统
7. object 的存储就变成了存储一个文件 rbd0.object1.file

在这个思路下，客户端写数据 osd 过程即为：

1. 采用的是 librbd 的形式，使用 librbd 创建一个块设备，向这个块设备中写入数据
2. 在客户端本地同过调用 librados 接口，然后经过 Pool，RBD，Object、PG 进行层层映射，在 PG 这一层中，可以知道数据保存在哪三个 osd 上，这三个 osd 分为主从的关系
3. 客户端与主 osd 建立 socket 通信，将要写入的数据传给主 osd，由主 osd 再将数据发送给其他 replica osd 数据节点

5.5 Ceph 总结与典型应用场景

Ceph 技术是一个高性能、高可用性、高可扩展性和特性丰富的系统，能提供对象存储、块存储和文件存储三种服务，并支持数据中心的扩展，非常适合在规模灵活变化的集群中使用。Ceph 作为一个分布式的文件系统，能够在维护 POSIX 兼容性的同时加入了强大的复制和容错功能，有效地提高了数据服务的速度、提高了性能、增加了系统的鲁棒性，这使其实际性能表现远高于使用其他的一些存储方式。

下面列举 Ceph 技术的几个典型应用场景并进行分析。

典型应用一：海量小文件存储。海量小文件存储（Lots of Small Files，LOSF）出现后，就一直是业界的难题，与其它分布式系统相比，海量小文件存储更侧重于解决两个问题：

1. 海量小文件的元数据信息组织与管理：对于百亿量级的数据，元信息总数据量远超过目前单机服务器内存大小，若使用本地持久化设备存储，则必须高效满足每次文件存取请求的元数据查询寻址，同时为了避免单点还要有备用元数据节点。并且，单组元数据服务器也成为整个集群规模扩展的瓶颈。如果使用独立的存储集群存储管理元数据信息，则当数据存储节点的状态发生变更时，应及时通知相应元数据信息进行变更

2. 本地磁盘文件的存储与管理（本地存储引擎）：对于常见的 Linux 文件系统，读取一个文件通常需要三次磁盘 I/O。按目前主流 2TB-4TB 的 sata 盘，可存储 2kw 至 4kw 个 100KB 大小的文件，由于文件数太多，无法将所有目录及文件的 inode 信息缓存到内存，很难实现每个文件读取只需要一次磁盘 I/O 的理想状态，而长尾现象使得热点缓存无明显效果。当请求寻址到具体的一块磁盘，如何减少文件存取的 I/O 次数、从而高效地响应请求（尤其是读请求）已成为必须解决的另一问题

Ceph 是近年越来越被广泛使用的分布式存储系统，其重要的创新之处是基于 CRUSH 算法的计算寻址，这是真正的分布式架构、无中心查询节点，理论上无扩展上限。如果将 Ceph 用于海量小文件存储，则 CRUSH 算法直接解决了上面提到的第一个问题。对于上述第二个问题，Ceph 社区曾提出了一种基于 RGW 的解决方案。

典型应用二：软件定义的云存储技术。雅虎存储用户所提交的照片，视频，电子邮件和博客文章的数据量十分庞大，对象存储超过 2500 亿，并且对象存储每年以 20%-25% 的速度增长，增长原因主要有移动、图像、视频、用户量的增长这几方面。对此，雅虎选择了软件定义存储，在保证耐用性和延迟的基础上发挥存储成本效益。因为 Ceph 可以通过一个固有的架构把对象存储、块存储和文件存储整合到一个存储层，所以它提供的灵活性很好地满足了雅虎的需求。COS 部署由模块化的 Ceph 集群组成，多个这样的 Ceph 集群同时部署就形成了一个 COS “超团（supercluster）”。对象在超团里所有集群中均匀分布，并使用专有的散列机制来分配对象。

典型应用三：机器学习平台统一化分布式存储。自网易云音乐机器学习平台上线以来，就承担了音乐内部推荐、搜索、直播、社交、算法工程等各个业务团队机器学习场景的需求，这其中也遇到了很大的挑战，尤其是在分布式存储方面，相关团队花费大量时间、精力，解决其中的核心问题。网页云音乐相关团队引入 CephFS 提供共享分布式存储，为各部门的业务工程师们提供统一的机器学习架构。其 CephFS 实现核心功能如下：

- 为开发环境提供弹性存储：算法、工程人员在计算平台上申请存储资源存储代码、模型、训练样本，同时可以将相同的卷挂载到调试环境进行调试，而无需数据拷贝。在引入 CephFS 之前，模型调试，特别是分布式训练的调试是十分痛苦的，需要将模型、样本数据分发到不同机器，训练结束还需要从各个开发机上将日志，模型等收集到开发机进行分析；引入 CephFS 之后，只需要为分布式训练任务挂载相同的数据卷，就可以直接进行调试，而训练过程中产生的日志、模型在开发机上都是即时可见的，极大节省了调试时间
- 简化数据治理：在模型开发流程中，有很大一部分时间在进行数据治理，比如大量视频进行抽帧。引入 CephFS 后，可以将这些治理好的数据直接作为数据卷挂载到训练任务，避免了繁琐的数据分发流程，并可以实现数据治理和训练分离，从而并行处理
- 极大简化了调度系统：在机器学习流水线中，算法人员在模型开发调试结

束后，可以直接将模型所在数据卷配置到调度系统进行调度，极大简化模型上线流程

CephFS 为机器学习平台提供了弹性的、可共享的、支持多读多写的存储系统，但开源 CephFS 在性能和安全性上还不能完全满足真实场景需求，因此相关团队进行了相应的改进，在防误删系统、混合存储系统等方面进行了大量的优化。目前，网易云音乐 Goblinlab 机器学习平台，平均每天处理 2000+ 个线上调度任务，存储池容量 1PB+，数百个存储卷；每日平均 IOPS 数万，带宽 GB+，累积节省模型训练时间上千小时。此外，CephFS 混合盘场景的使用在降本增效方面具有重要价值，按照当前 ML 集群 PB 级别的存储量，可以为业务节省数十万的成本。

6 NDB 技术

NDB 集群是 MySQL 集群底层的分布式数据库系统，它可以独立于 MySQL 服务器使用，用户可以通过 NDB API 访问集群。从 MySQL 服务器的角度来看，NDB 是一个存储行表的存储引擎。而从 NDB 集群的角度来看，一个 MySQL 服务器实例则是一个连接到集群的进程。NDB 可以同时支持来自多种类型的 API 的进程的访问，包括 Memcached、JavaScript/Node.JS、Java、JPA 和 HTTP/REST 等。所有 API 进程都可以对存储在 NDB 中的相同表和数据进行操作。

MySQL 集群使用 MySQL 服务器在 NDB 集群上提供以下功能：

- SQL 解析、优化和执行能力
- 跨表连接机制
- 用户认证和授权
- 异步复制数据到其他系统

NDB 集群具有以下特征：

- 高可扩展性：NDB 集群可以在内部自动进行数据分片，随着数据节点的增加，可以做到非常高的读写扩展
- 高可用性：最新版本高达 99.9999% 的高可用性，每年停机运维时间不超过一分钟。因为采用了非共享的架构，不会出现单一故障，发生故障时的故障转移时间非常短，并且各种维护工作可以在线进行，因此极大地提高系统可用性
- 实时性：数据大部分情况下保存在内存中，可以快速执行事务，满足实时性高要求的应用需求
- 异地容灾：可以利用 NDB 集群的复制功能，对 NDB 集群进行异地容灾，与 MySQL 的复制功能不同，NDB 集群可以进行双向复制，并且能够对数据冲突进行校验

- 支持 SQL 与 NoSQL：数据节点上保存的数据除了可以通过 SQL 节点访问，还可以通过 NoSQL 访问。对比其它的 KV 型数据库，NDB 集群具有如下优点：
  - 支持 ACID 的完整事务
  - 数据具有持久性和冗余性
  - 自动故障转移
  - 在线备份
- 低成本：由于 NDB 集群不适用共享存储，一般的服务器即可运行，因此可以为用户节省大量的硬件成本

6.1 NDB 集群架构

NDB 是围绕分布式、多主机 ACID 兼容架构设计的，没有单点故障，使用自动分片（分区）扩展商品硬件上的读写操作，并且可以通过 SQL 和非 SQL 的 API 进行访问。其集群架构如下图所示：

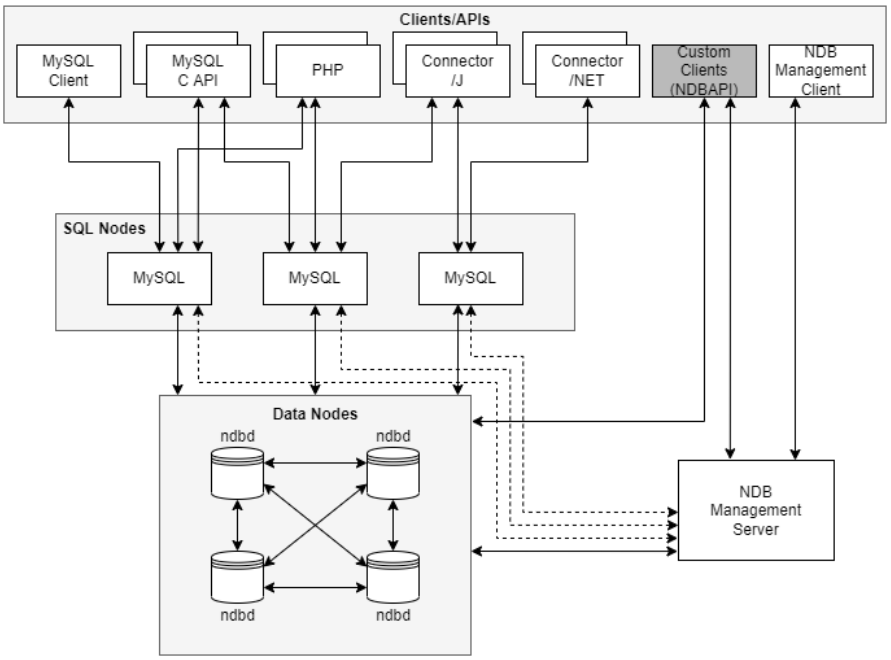

图 4: NDB 集群架构

其中主要涉及到下面三种节点：

- 管理节点：作用是管理 MySQL 集群内的其他节点，如提供配置数据、启动并停止节点、运行备份等。由于这类节点负责管理其他节点的配置，应在启动其他节点之前首先启动这类节点。理论上一般只启动一个，而且宕机也不影响集群的服务，这个进程只在集群启动以及节点加入集群时起作

用，所以这个节点不是很需要冗余，理论上通过一台服务器提供服务就可以了。另外，管理服务器负责管理集群配置文件和日志。集群中的每个节点从管理服务器检索配置数据，并请求确定管理服务器所在位置的方式。当数据节点内出现新的事件时，节点将关于这类事件的信息传输到管理服务器，然后，将这类信息写入日志

- 数据节点：用于保存集群的数据。数据节点的数目与副本的数目相关，是片段的倍数，分段的数目为节点总数除以副本的数量。例如，对于两个副本，每个副本有两个片段，那么就有四个数据节点，通常设置为两副本，两个以上时虽然能实现集群的高可用保证，但数据副本节点增加时，集群的处理速度会变慢。数据更新使用读已提交隔离级别来保证所有节点数据的一致性，使用两阶段提交机制，可以保证所有节点都有相同的数据。无共享的对等节点使得某台服务器上的更新操作在其他服务器上立即可见。传播更新使用一种复杂的通信机制，这一机制专用来提供跨网络的高吞吐量。MySQL 集群将所有的索引列都保存在主存中，其他非索引列可以存储在内存中或者通过建立表空间存储到磁盘上。如果数据发生改变，MySQL 集群将发生改变的记录写入重做日志，然后通过检查点定期将数据定入磁盘。由于重做日志是异步提交的，所以故障期间可能有少量事务丢失。为了减少事务丢失，MySQL 集群实现延迟写入（可配置延迟时间），这样就可以在故障发生时完成检查点写入，而不会丢失最后一个检查点。一般单个数据节点故障不会导致任何数据丢失，因为集群内部采用同步数据复制。由于同步复制一共需要四次消息传递，故 MySQL 集群的数据更新速度比单机 MySQL 要慢。所以 MySQL 集群要求运行在千兆以上的局域网内，节点可以采用双网卡，节点组之间采用直连方式。另一方面，对集群进行扩容增加数据节点组时不会导致数据更新速度降低。相反，数据更新速度会变快，这是因为数据是分别处理的，每个节点组所保存的数据是不一样的，可以减少锁定

- SQL 节点：用来访问集群数据的节点。对于 MySQL 集群，客户端节点是使用 NDB Cluster 存储引擎的传统 MySQL 服务器。集群中可以有多个 SQL 节点，通过每个 SQL 节点查询到的数据都是一致的。通常来说，SQL 节点越多，分配到每个 SQL 节点的负载就越小，系统的整体性能就越好

所有的这些节点构成一个完成的 MySQL 集群体系：数据保存在 NDB 存储服务器的存储引擎中，表（结构）则保存在 MySQL 服务器中。应用程序通过 MySQL 服务器访问这些数据表，集群管理服务器通过管理工具来管理 NDB 存储服务器。

6.2 NDB 存储引擎

MySQL 集群使用了一个专用的基于内存的存储引擎（NDB 存储引擎），这样做的好处是速度快，没有磁盘 I/O 的瓶颈，但是由于是基于内存的，所以数据库的规模受系统总内存的限制，如果运行 NDB，那么 MySQL 服务器一定要具有足够大的内存。NDB 引擎是分布式的，它可以配置在多台服务器上来实现数据的可靠性和扩展性，理论上通过配置两台 NDB 的存储节点就能实现整个数据库集群的冗余性和解决单点故障问题。

在 NDB 上可以建立两种类型的表：

- 内存表：所有数据（包括 index）都在内存中，同时会在磁盘上保存数据，因此不用担心数据会丢失，数据节点会在启动的时候把数据加载到内存
- 磁盘表：仅主键、索引字段保存在内存中，其他字段则保存在磁盘文件中

并且，NDB 具有下面的特点：

- SQL 节点可以直接访问集群中的数据，当应用程序更新数据后，所有 MySQL 服务器都可以立即看到此更改
- SQL 节点使用的 mysqld 服务器守护程序，在许多关键方面与 MySQL 8.0 发行版提供的 mysqld 二进制文件不同，两个版本的 mysqld 不可互换
- 集群可以处理单个数据节点的故障

- 单个节点可以停止和重新启动，然后可以重新加入集群
- NDB 集群表通常完全存储在内存中而不是磁盘上，但一些 NDB 集群数据可以存储在磁盘上
- 表和表数据存储在数据节点中，数据节点中存储的数据可以进行镜像
- 一个或多个节点构成节点组，节点存储数据分区，一个分区保存至少一个片段副本
- NDB 8.0 支持外键、支持 JSON 数据类型
- SQL 语法不兼容，例如：
  - 不支持临时表
  - 不支持全文索引
  - 不支持前缀索引，只能索引整个列
  - 如果外键不是（关联）表的主键，则作为外键引用的列都需要是显式的唯一键
  - 当外键引用是父表的主键时，不支持级联更新
- 数据节点的最大数量为 145
- 集群最大节点总数为 255，这个数字包括所有 SQL 节点（MySQL 服务器）、API 节点、数据节点和管理服务器
- NDB 存储引擎仅支持 READ COMMITTED 事务隔离级别
- NDB 集群不能很好地处理大型事务，与包含大量操作的单个大事务相比，执行多个小事务且每个事务包含少量操作要更好，大型事务需要非常大量的内存

- NDB 集群中所有 NDB 数据库对象的最大数量（包括数据库、表和索引）限制为 20320
- 每个表的属性（即列和索引）数最大为 512，每个键的最大属性数为 32
- 行数据大小最大为 30000 字节
- 不支持半同步复制
- SQL 节点没有分布式表锁

6.3 NDB 总结与典型应用场景

现在简要总结 NDB 集群的特点如下：

- 由于基于内存，数据库的规模受集群总内存的大小限制，并且在重启时数据节点将数据载入到内存耗时较长
- 多个节点通过网络实现通讯和数据同步、查询等操作，因此整体性受网络速度影响
- 多个节点之间可以分布在不同的地理位置，因此也是一个实现分布式数据库的方案
- 扩展性很好，增加节点即可实现数据库集群的扩展
- 冗余性很好，多个节点上都有完整的数据库数据，因此任何一个节点宕机都不会造成服务中断
- 实现高可用性的成本比较低，不象传统的高可用方案一样需要共享的存储设备和专用的软件才能实现，NDB 只要有足够的内存就能实现

下面列举 NDB 技术的几个典型应用场景并进行分析。

典型应用一：Beezz 使用 NDB。Beezz 由一群网络安全专家于 2013 年创立，他们在为以色列军方管理网络安全和情报方面拥有超过 60 年的综合经验。创始团队负责打造军队封闭式花园环境，保护军队最敏感的信息免受国际黑客攻击。认识到日益增长的网络安全威胁，他们与一级运营商的主要高管联手，致力于开发革命性的物联网安全解决方案。Beezz 需要获得实时处理来自物联网设备的大量交易的能力、确保物联网安全应用程序最终用户的数据完整性和最长正常运行时间、保证数据冗余和超高可用性，并通过签订具有约束力的服务级别协议来建立信任，以确保在不断发展的物联网领域中防止欺诈或数据泄露。因此，Beezz 选择了 MySQL 集群作为移动核心网络数据库，为用户提供稳健的服务，并采用 NDB 内存存储引擎，大幅提高了查询后端数据节点的能力。并且，Beezz 通过 MySQL Cluster Carrier Grade Edition 的混合功能扩大客户群，以部署大规模本地安装或通过共享私有云基础架构的方式服务中小型公司。得益于通过 MySQL 集群的独特功能，包括不停机的服务器升级、自动冲突解决、提高速度和 NDB 多

复制冗余，为安全应用程序提供 99.9999% 的可用性。最后，NDB 集群的备份和恢复能力，可以在半小时内恢复被删除的数据库。

典型应用二：Certigna 使用 NDB。Certigna 是欧洲信任服务提供商，可以向寻求网站身份验证、加密、电子签名和电子时间戳解决方案的实体颁发网络安全证书。该公司在法国拥有约 20000 名客户，其中包括税务机关和社会保障等主要政府机构。支持其多站点架构的 MySQL 分布式数据库具有自动复制功能，可确保最大限度地延长正常运行时间并增强故障排除能力。Certigna 部署了一个 MySQL NDB Cluster 解决方案，在两个位置的四台服务器上进行读写复制，提供 99.999% 的数据库可用性，并显着缩短数据库管理和监控时间。并且，跨服务器的自动复制提高了备份和恢复操作的速度和安全性，并使用 MySQL Cluster Manager 将每日数据库转储替换为 MySQL NDB 上的时间点快照，从而实现了数据库的集群配置和跨服务器自动复制备份和恢复，提高了数据安全性和数据库监控能力。

典型应用三：BitCash 使用 NDB。BitCash 是日本一家大型的预付费电子货币发行机构，主要业务是为网上游戏和各种形式的电子内容（例如音乐、视频和电子图书）提供安全便捷的支付解决方案。为了促进公司发展、将业务范围扩展到更多领域，该公司希望构建集群式数据库，以缩短系统响应时间，提高数据可用性。BitCash 通过将商家应用程序使用的传统数据库（占据业务流程的 30% 到 40%）升级到采用 NDB 技术的 MySQL 集群，提高了使用电子货币的网上交易的处理速度，系统整体响应时间加快了 30 倍。并且，完全在内存中运行集群式数据库，大大减少了应用程序级的不可预测工作量带来的风险。

7 总结

在本文中，我整理了几个典型的网络存储技术，现在可以将它们总结如下：

- NAS：将分布、独立的数据整合为大型、集中化管理，从而便于不同主机和应用对服务器进行访问的文件级技术。NAS 可以被定义为一种特殊的专用数据存储服务器，其最重要的功能就是跨平台文件共享功能。NAS 通常在一个 LAN 上占有自己的节点，无需其他应用服务器的干预，并允许用户从该网络上进行数据的存取、更新、删除等操作。在这种配置中，NAS 集中管理和处理所在局域网上的所有数据，有效降低数据管理和共享的成本。NAS 的典型应用包括：异地容灾、多应用系统的统一支持、Web 的后台存储系统等

- SAN：提供对整合的块级数据存储的访问，构建在 SAN 之上的“共享文件系统”可以提供文件级的访问。SAN 主要用于从服务器访问数据存储的设备，例如磁盘阵列和磁带库，从而使这些设备在概念上具备了“直连存储”的特性。一般来讲，SAN 是硬件和软件的组合，它源于以数据为中心的大型机架构，其中网络中的客户端可以连接到多个服务器并存储不同类型的数据。在 SAN 中，互相连接的存储设备使彼此的数据传输（例如备份）的实现隐藏在了服务器后面（并且对于普通用户是透明的），比如，数据可以

通过 TCP/IP 协议进行传输。SAN 配备了不同的支持协议，包括光纤通道（FC）、SCSI 与 iSCSI、Infiniband 等。但是，SAN 通常有自己的网络和存储设备，必须单独购买、安装和配置。这使得 SAN 架构比 NAS 架构和 DAS 架构都更昂贵。SAN 的典型应用包括：高性能高可靠服务、异构混合存储架构的云服务等

- HDFS：Apache Hadoop 作为一个开源软件实用程序的集合，其核心由称为 HDFS 的存储部分和作为 MapReduce 编程模型的处理部分共两个主要部分组成。Hadoop 将文件拆分成块并将它们分布在集群中的各个节点上，然后将打包的代码传输到节点以并行处理数据。这种方法很好地利用了数据局部性，节点在其中操纵它们有权访问的数据。这使得数据的处理速度甚至比在传统的超级计算机架构中更快、更有效。HDFS 可以在多台机器上存储大文件，并通过跨多个主机复制数据来实现可靠性，因此理论上不需要独立磁盘冗余阵列（RAID）主机上的存储。HDFS 旨在实现跨各种硬件平台的可移植性，以及与各种底层操作系统的兼容性。HDFS 的典型应用包括：结构化数据库采集与分析、网页数据采集与分析、日志分析与推荐系统、算法及机器学习等

- Ceph：是一个开源软件定义存储平台，它在单个分布式计算机集群上实现对象存储，并为对象、块和文件提供三合一接口级存储。Ceph 的主要目标是完全分布式操作，主要特点包括：高性能、高可用性、高可扩展性、没有单点故障等。Ceph 使用现有硬件设备和网络 IP，不需要特定的硬件支持。Ceph 的系统通过复制、擦除编码、快照和存储克隆等技术提供灾难恢复和数据冗余。Ceph 既能自我修复又能自我管理，是一个高度自治的系统，可以最大限度地减少管理时间和其他成本。这个统一的系统可以在一个通用的管理框架内收集存储。Ceph 整合了多个存储用例并提高了资源利用率。它还允许组织在需要的地方部署服务器。Ceph 的典型应用包括：海量小文件存储、软件定义的云存储技术、机器学习平台统一化分布式存储等

- NDB：NDB 集群是 MySQL 集群底层的分布式数据库系统，它可以独立于 MySQL 服务器使用，用户可以通过 NDB API 访问集群。MySQL 集群使用 MySQL 服务器在 NDB 集群上提供：SQL 解析优化和执行能力、跨表连接机制、用户认证和授权、异步复制数据到其他系统等功能。并且，NDB 集群具有高可扩展性、高可用性、实时性、异地容灾、支持 SQL 与 NoSQL、低成本等特点。NDB 的典型应用包括：Beezz、Certigna 和 BitCash 等公司使用 NDB 提升服务质量

随着互联网系统日益复杂，如何存储数据成为了非常重要的话题，本学期我学习了一些常见的网络存储技术，并在了解其概念的同时结合其实际应用分析了各项技术方案的优缺点，感觉收获颇丰，同时又惊叹于计算机工业界实践成果之富，这值得我们持续不断地学习与探索！