

分布式文件系统¹

黄华 杨德志 张建刚

摘要 本文简要回顾了分布式文件系统的发展历史和当前的最新进展,结合具体分布式文件系统分析了它们的体系结构以及每种结构的优缺点,然后对比分析了实现一个分布式文件系统所需要的关键技术,最后根据当前软、硬件的发展和应用的发

关键词 文件系统, 分布式文件系统, 机群文件系统, 体系结构, 缓存一致性, 安全, 扩展性

1 引言

文件系统是操作系统的一个重要组成部分,通过对操作系统所管理存储空间的抽象,向用户提供统一的、对象化的访问接口,屏蔽对物理设备的直接操作和资源管理。

根据计算环境和所提供功能的不同, M. Satyanarayanan[1]将文件系统划分为四个层次,从低到高依次是:(a)单处理器单用户的本地文件系统,如DOS的文件系统;(b)多处理器单用户的本地文件系统,如OS/2的文件系统;(c)多处理器多用户的文件系统,如UNIX的本地文件系统;(d)多处理器多用户的分布式文件系统。本地文件系统(Local File System)是指文件系统管理的物理存储资源直接连接在本地节点上,处理器通过系统总线可以直接访问。分布式文件系统(Distributed File System)是指文件系统管理的物理存储资源不一定直接连接在本地节点上,而是通过计算机网络与节点相连。上述按照层次的分类中,高层次的文件系统都是以低层次的文件系统为基础,实现了更高级的功能。比如(b)需要比(a)多考虑并发控制(Concurrency Control),因为可能存在多个处理器同时访问文件系统的情况;(c)需要比(b)多考虑数据安全访问方面的设计,因为多个用户存在于同一个系统中,保证数据的授权访问是一个关键;(d)需要比(c)多考虑分布式体系结构带来的诸多问题,比如同步访问(synchronous access)、缓冲一致性(cache coherency)等。随着层次的提高,文件系统在设计 and 实现方面的难度也会成倍地提高。但是现在的分布式文件系统一般还是保持与最基本的本地文件系统几乎相同的访问接口和对象模型,这主要是为了向用户提供向后的兼容性,也保持原来的简单的对象模型和访问接口。但这并不说明文件系统设计和实现的难度没有增加。恰恰由于对用户透明地改变了提供结构,满足用户的需求,以掩盖分布式文件操作的复杂性,大大增加了分布式文件系统的实现难度。

计算机技术在飞速的发展,摩尔(Moor)定律似乎在未来的若干年内仍将有效:计算机的处理器速度在不断地提高,计算机存储器的容量和传输速度在非常规地发展,计算机网络的吞吐量更是在跳跃式地增加。在计算机的性能不断提升的同时,计算机部件的平均价格却在不断下降。用户可以用更小的成本,购买更好更快更稳定的设备。存储系统,文件系统面临的新的挑战也随之而来:如何管理更多的设备,提供更好的性能,更加有效地降低管理成本等。各种新的存储技术和分布式文件技术都被不断地设计和实现出来,以满足用户日益增长的需求。因此有必要简要回顾分布式文件系统发展的历史,分析对比当前主流的分布式文件系统在体系结构、缓存一致性、安全等方面的长处和不足。

¹本文摘自中国科学院计算技术研究所内部刊物—信息技术快报 2004 年第 10 期

2 历史与现状

文件系统最初是用来管理本地磁盘，提供用户访问接口。某些数据的集合叫做一个“文件(file)”，并赋予每一个文件一定的属性以标识该数据集合的某些属性。文件按照树(tree)结构层次进行管理和检索。最初的文件系统只能管理本地磁盘空间。主机之间的文件共享与传输则通过文件传输协议[2] (FTP- file transfer protocol) 实现。但 FTP 没有提供与本地文件系统一致的访问接口和对象模型。

随着计算机应用范围的扩展，通过文件访问接口在不同主机之间共享文件的需求日益增强。在二十世纪 70 年代就出现了最初的分布式文件系统的尝试[3]。到了二十世纪八十年代中期，网络文件系统 (NFS-Network File System) [4][5] 的出现使得分布式文件系统逐渐发展并应用到各个领域。从早期的 NFS 到 StorageTank，分布式文件系统在体系结构、系统规模、性能、可扩展性、可用性等方面经历了较大的变化。下面分为几个阶段介绍分布式文件系统的发展过程。

2.1 1980-1990

早期的文件系统以 NFS 和 AFS (Andrew File System) 最具代表性，对以后的文件系统的设计也最有影响。

NFS 从 1985 年出现至今，已经经历了四个版本的更新，被移植到了几乎所有主流的操作系统，成为分布式文件系统事实上的标准。NFS 利用 Unix 系统中的虚拟文件系统 (VFS-Virtual File System) 机制，将客户机对文件系统的请求，通过规范的文件访问协议和远程过程调用，转发到服务器端进行处理；服务器端在 VFS 之上，通过本地文件系统完成文件的处理，实现了全局的分布式文件系统。Sun 公司公开了 NFS 的实施规范，互联网工程任务组 (The Internet Engineering Task Force, IETF) 将其列为征求意见稿 (RFC-Request for Comments)，这很大程度上促使 NFS 的很多设计实现方法成为标准，也促进了 NFS 的流行。NFS 也在不断地发展，在第四版中，提供了基于租赁 (lease) 的同步锁和基于会话 (session) 语义的一致性等。

Carnegie Mellon University 在 1983 设计开发的 AFS ^{[6][29]} 将分布式文件系统的可扩展性放在了设计和实现的首要位置，并且着重考虑了在不安全的网路中实现安全访问的需求 ^[31]，因此它在位置透明、用户迁移、与已有系统的兼容性等方面进行了特别设计。AFS 具有很好的扩展性，它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 ^[30]。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，提高了系统可用性 ^[32]。后来的 Coda File System ^{[7][33]}、Inter-mezzo File System ^[8] 都受到 AFS 的影响，更加注重文件系统高可用性 (High Availability) 和安全性，特别 Coda 在支持移动计算方面做了很多的研究工作。

此外，Sprite File System 也是早期比较有特色的分布式文件系统，它是 Sprite Network Operation System 的组成部分 ^[40]，为分布式计算环境提供 ~~(全局?)~~ 全局文件访问。与 NFS 相比，Sprite File System 在服务器端和客户端都设置缓存，大大提高了系统性能 ^[38, 41]；它通过简单的读写锁来保证整个系统缓存一致性，并且通过和虚拟存储部分交互，尽量多地缓存数据 ^[39]。它允许应用程序通过文件系统接口访问远程主机上的 I/O 设备，突破了 NFS 和 AFS 在这些方面的限制，真正实现了位置透明。

早期的分布式文件系统一般以提供标准接口的远程文件访问为目的，在受网络环境、本地磁盘、处理器速度等方面限制的情况下，更多地关注访问的性能和数据的可靠性。AFS 在系统结构方面进行了有意义的探索。它们所采用的协议和相关技术，为后来的分布式文件

系统设计提供了很多借鉴。

2.2 1990—1995

二十世纪九十年代初,随着磁盘技术的进步,单位存储的成本在不断下降。随着 Windows 的出现,极大地推动了处理器的发展和微机的普及;互联网的出现和逐渐普及,使得在网络中传输实时多媒体数据的需求和应用逐渐流行起来。大规模的并行计算和数据挖掘等应用也需要大容量高速度的分布式存储环境。

面对广域网和大容量存储应用的需求,借鉴当时先进的高性能对称多处理器的设计思想,University of California at Berkeley 设计开发的 xFS[9]克服了以前的分布式文件系统一般都是运行在局域网(LAN)上的弱点,同时还很好地解决了在广域网上进行缓存以减少网络流量的难题。它所采用的多层次结构很好地利用了文件系统的局部访问的特性,无效写回(invalidation-based write back)缓存一致性协议,减少了网络负载。对本地主机和本地存储空间的有效利用,使它具有较好的性能。

Tiger Shark[34, 35, 36]并行文件系统是针对大规模实时多媒体应用设计的。它采用了多种技术策略保证多媒体传输的实时性和稳定性:采用资源预留和优化的调度手段,保证数据实时访问性能;通过加大文件系统数据块的大小,最大限度地发挥磁盘的传输效率;通过将大文件分片存储在多个存储设备中,取得尽量大的并行吞吐率;通过复制文件系统元数据和文件数据克服单点故障,提高系统可用性。

基于虚拟共享磁盘 Petal[10]的 Frangipani[11]分布式文件系统采用了一种新颖的系统结构——分层次的存储系统。Petal 提供一个可以全局统一访问的磁盘空间。Frangipani 基于 Petal 的特性提供文件系统的服务。这种分层结构使两者的设计实现都得到了简化。在 Frangipani 中,每一个客户端也是文件系统服务器,参与文件系统的管理,可以平等地访问 Petal 提供的虚拟磁盘系统,通过分布式锁实现同步访问控制。分层结构使系统具有很好的扩展性,可以在线动态地添加存储设备、增加新用户、备份等,同时系统具有很好的机制来处理节点失效、网络失效等故障,提高了系统的可用性。

Slice File system (SFS) [44]考虑标准的 NFS 在容量、性能方面存在的限制,采用在客户机和服务器之间架设一个 μ proxy 中间转发器,以提高性能和可扩展性。它将客户端的访问分为小文件、元数据服务、大文件数据三类请求。通过 μ proxy 将前两种请求转发到不同的文件服务器上,将后者直接发送到存储服务器上。这样 SFS 系统就可以支持多个存储服务器,提高整个系统的容量和性能。 μ proxy 根据请求内容的转发是静态的,对于整个系统中负载的变化难以做出及时的反应。

由于应用对性能的需求,在这个阶段,分布式文件系统采用了多种方法提高系统性能,比如多级的缓存策略、资源管理的优化、采用多个存储设备、更好的调度算法等。同时,Frangipani 的分层设计思想、SFS 系统结构对后来的文件系统结构设计都有较大的影响。

2.3 1995—2000

在这个阶段,计算机技术和网络技术有了飞速的发展,单位存储的成本不断降低。这些为信息数字化提供了坚实的基础。而数据总线带宽、磁盘速度的增长无法满足应用对于数据带宽的需求,存储子系统成为计算机系统发展的瓶颈。

网络技术和普及应用极大地推动了网络存储技术的发展,基于光纤通道(Fiber Channel)的 SAN(Storage Area Storage,存储区域网),NAS(Network Attached Storage,网络附属储存)得到了广泛的应用。这也推动了分布式文件系统的研究。这个阶段,出现了多种体系结构,充分利用了网络技术。

Global File System (GFS) [12][13]吸取了对称多处理器 (SMP) 系统设计和实现的原理, 将系统中的每一个客户机类比于 SMP 中的一个处理器。客户机间没有任何区别, 可以平等地访问系统中的所有存储设备, 就像处理器可以机会均等地访问主存一样。这样的设计可以更好地利用系统中的资源, 消除单个服务器带来的性能瓶颈和单点失效问题。因为客户端之间无需通信, 可以很好地消除客户机失效带来的威胁。GFS 采用特殊设计的 DLOCK 锁机制来同步多个客户机对同一设备的访问, 具有很高的效率[12]。

General Parallel File System (GPFS) [14]是从 Tiger Shark 发展过来的, 是目前应用范围较广的一个系统。在系统设计中采用了多项先进技术。它是一个共享磁盘 (shared-disk) 的分布式并行文件系统, 客户端采用基于光纤通道或者 iSCSI 与存储设备相连, 也可以通过通用网络相连。GPFS 的磁盘数据结构可以支持大容量的文件系统和大文件, 通过采用分片存储、较大的文件系统块、数据预读等方法获得了较高的数据吞吐率; 采用扩展哈希 (extensible hashing) 技术来支持含有大量文件和子目录的大目录, 提高文件的查找和检索效率。GPFS 采用不同粒度的分布式锁解决系统中的并发访问和数据同步问题: 字节范围的锁用于用户数据的同步, 动态选择元数据节点 (metanode) 进行元数据的集中管理; 具有集中式线索的分布式锁管理整个系统中空间分配等。GPFS 采用日志技术对系统进行在线灾难恢复。每个节点都有各自独立的日志, 且单个节点失效时, 系统中的其他节点可以代替失效节点检查文件系统日志, 进行元数据恢复操作。GPFS 还有效地克服了系统中任意单个节点的失效、网络通信故障、磁盘失效等异常事件。此外, GPFS 支持在线动态添加、减少存储设备, 然后在线重新平衡系统中的数据。这些特性在需要连续作业的高端应用中尤为重要。

HP 公司的 DiFFS[15]和 SGI 公司的 CXFS[19]都是基于 SAN 的分布式文件系统。DiFFS 通过将存储系统划分成不同的区域, 把对资源的共享访问冲突限制在各个区域内部, 来解决机群文件系统的可扩展性问题。DiFFS 采用了动态分配策略、SoftUpdates[8]和文件级的负载平衡等多项技术。CXFS 是在 XFS[47]的基础上开发的。它实现了元数据服务器内置的失效接替和恢复功能; 采用快速元数据算法, 包括优秀的缓存机制, 精心组织的存储结构和优化的搜索算法, 来提高元数据访问性能。CXFS 采用令牌 (token) 来管理和控制元数据和数据的访问, 采用了多人读一个人写的策略保证数据和元数据的一致性。CXFS 客户机通过 SAN 直接访问文件数据, 具有很好的传输效率。

此外, 还有多种体系结构, 如 EMC 的 HighRoad[49], SUN 的 qFS[50], XNFS[51]等。数据容量、性能和共享的需求使得这一时期的分布式文件系统管理的系统规模更大、系统更复杂, 对物理设备的直接访问、磁盘布局和检索效率的优化、元数据的集中管理等都反映了对性能和容量的追求。规模的扩展使得系统的动态性如在线增减设备、缓存的一致性、系统的可靠性的需求逐渐增强, 更多的先进技术应用到系统实现中, 如分布式锁、缓存管理技术、SoftUpdates 技术、文件级的负载平衡等。

2.4 2000 以后

随着 SAN 和 NAS 两种体系结构逐渐成熟, 研究人员开始考虑如何将两种体系结构结合起来以充分利用两者的优势; 另一方面, 基于多种分布式文件系统的研究成果, 人们对体系结构的认识不断深入, Grid 的一些研究成果等也推动了分布式文件系统体系结构的发展。这一时期, IBM 的 StorageTank, Cluster 的 Lustre, Panasas 的 PanFS, 蓝鲸文件系统 (BWFS) 是这种体系结构的代表。而且随着信息数字化的深入, 各种应用对存储系统提出了更多的需求:

- a. 大容量 (Capacity), 现在的数据量比以前任何时期更多, 生成的速度更快;

- b. 高性能 (High Performance), 数据访问需要更高的带宽;
- c. 高可用性 (High Availability), 不仅要保证数据的高可用性, 还要保证服务的高可用性。
- d. 可扩展性 (Scalability), 应用在不断变化, 系统规模也在不断变化。这就要求系统提供很好的扩展性, 在容量、性能、管理方面都能适应应用的变化。
- e. 可管理性 (Manageability), 随着数据量的飞速增长, 存储的规模越来越庞大, 存储系统本身也越来越复杂, 这给系统的管理运行带来了很高的维护成本。
- f. 按需服务 (On-Demand), 能够按照应用需求的不同提供不同的服务, 如不同的应用, 不同的客户端环境, 不同的性能等。

IBM 公司在 GPFS 的基础之上发展进化来的 Storage Tank[17]以及基于 Storage Tank 的 TotalStorage SAN File System[45]又将分布式文件系统的设计理念和系统架构向前推进了一步。它们除了具有一般的分布式文件系统的特性之外, 采用 SAN 作为整个文件系统的数据存储和传输路径。它们采用带外 (out-of-band) 结构, 将文件系统元数据在高速以太网上传输, 由专门的元数据服务器来处理 and 存储。文件系统元数据和文件数据的分离管理和存储, 可以更好地利用各自存储设备和传输网络的特性, 提高系统的性能, 有效降低系统的成本。Storage Tank 采用积极的缓存策略, 尽量在客户端缓存文件元数据和数据。即使打开的文件被关闭, 都可以在下次使用时利用已经缓存的文件信息, 整个文件系统由管理员按照目录结构划分成多个文件集 (fileset)。每一个文件集都是一个相对独立的整体, 可以进行独立的元数据处理和进行文件系统备份等。不同的文件集可以分配到不同的元数据服务器处理, 形成元数据服务器机群, 提供系统的扩展性、性能、可用性等。在 TotalStorage 中, 块虚拟层将整个 SAN 的存储进行统一的虚拟管理, 为文件系统提供统一的存储空间。这样的分层结构有利于简化文件系统的设计和实现。同时, 它们的客户端支持多种操作系统, 是一个支持异构环境的分布式文件系统。在 SAN File System, 采用了基于策略的文件数据位置选择方法, 能有效地利用系统的资源、提高性能、降低成本。

Panasas 公司的 PanFS[20]是一个基于对象存储 (Object-based Storage) 的 Linux 分布式机群文件系统。它是该公司的存储机群的核心部分。由于采用了基于对象的磁盘设备, 元数据服务器的很大一部分负载 (据分析有 90%) 都转移给了基于对象的存储设备。同时由于存储设备可以很好地理解和应用对象 (也就是文件或者目录), 因此可以更好地组织对象的数据布局, 也能很好地满足对象的性能要求。文件系统的数据由客户端通过网络直接和对象磁盘交换; 元数据通过元数据服务器进行管理。客户端文件系统向应用提供符合 POSIX 标准的文件接口, 可以缓存数据, 并且对数据进行分片存储。元数据服务器维护整个系统的元数据一致性, 维护系统缓存的一致性。因为文件系统的很大一部分工作已经由对象磁盘分担, 元数据服务器可以服务于更多的客户端, 提供更好的性能。

Cluster File Systems 公司的 Lustre[21]是面向下一代存储的分布式文件系统。它采用基于对象的磁盘作为整个文件系统的存储设备, 但是又不同于对象磁盘, 因为它可以通过软件的形式来模拟。它使用了 Sandia 开放的 Portals 网络传输协议, 支持多种网络 (TCP/IP, Quadrics, Myrinet, InfiniBand 等)。它也采用数据和元数据分开传输和存放的机制提供系统的性能。Lustre 创造性的带意图的锁 (intent lock), 明显地减少客户机和文件系统元数据服务器之间的消息传递, 也缩小了每次操作的延迟。它采用标准的 XML、LDAP、SNMP

等协议来进行系统管理；还加入了失效接替等特性，提高了系统的可用性。

这个阶段的系统都在研究中，但从中也可以看出一些发展的趋势：如体系结构的研究逐渐成熟，表现在不同文件系统的体系结构趋于一致；系统设计的策略基本一致，如采用专用服务器的方式等。每个系统在设计的细节上各自采用了很多特有的先进技术，也都取得了很好的性能和扩展性。另外，在协议方面的探索也是研究的热点之一，如 Direct Access File System[18]利用了远程内存直接访问（Remote Direct Memory Access, RDMA）的特性，借鉴了 NFS 第四版本和 Common Internet File System（CIFS）[48]等协议，设计了一套新的网络文件访问协议，同时添加了一些 NFS 和 CIFS 中不存在的协议，改进了一些关于锁的语义。根据测试结果，对比 NFS 等传统网络文件系统，DAFS 取得了较好的性能。

2.5 国内研究情况

中科院计算所研究的“曙光”系列机群服务器上的文件系统 COSMOS[22]和 DCFS [23]，为曙光超级服务器提供了一个分布式的文件系统环境。为了获得较好的 I/O 性能，DCFS 机群文件系统中的元数据服务器采用二级的树状结构、文件属性缓存、集中式目录缓存管理等技术，存储服务器主要采用了多线程和缓存技术。DCFS 将整个名字空间根据目录关系划分成多个子空间，分配到不同的元数据服务器管理，取得较好的性能、扩展性和可用性。

从以上的分析中我们可以看出，最初远程文件访问和共享的需求，促使了 NFS、AFS 的出现；NFS 结构中单个服务器在容量和性能方面的限制，使得使用多个存储服务器结构的分布式文件系统出现了，同时能够在更大范围的广域网上都具有很好扩展性的系统也出现了。之后，随着网络技术和磁盘技术的发展，分布式文件系统又将 SAN、对象磁盘等先进技术引进来，出现了基于 SAN 的分布式文件系统和基于对象存储的分布式文件系统。整个分布式文件系统中单个节点和整体的聚集速度在快速地增加。

3 体系结构

分布式文件系统的体系结构在多年来经历了多次变化。这其中有软硬件技术发展的作用，也有应用需求变化的因素。下面我们根据不同的标准分析一下多年来分布式文件系统设计和实现过的主要的体系结构。

3.1 数据访问方式——带内模式和带外模式

在传统的分布式文件系统中，所有的数据和元数据都存放在一起，通过服务器提供。这种模式一般称之为带内模式（in-band mode）。随着客户端数目的增加，服务器就会成为整个系统的瓶颈。因为系统所有的数据传输和元数据处理都要通过服务器，不仅单个服务器的处理能力有限，存储能力受到磁盘容量的限制，吞吐能力也受到磁盘 I/O 和网络 I/O 的限制。

于是一种新的分布式文件系统的结构出现了，那就是利用 SAN 技术，将应用服务器直接和存储设备相连接，大大提高数据的传输能力，减少数据传输的延时。在这样的结构里，所有的应用服务器都可以直接访问存储在 SAN 中的数据，而只有关于文件信息的元数据才经过元数据服务器处理提供，减少了数据传输的中间环节，提高了传输效率，减轻了元数据服务器的负载。每个元数据服务器可以向更多的应用服务器提供文件系统元数据服务。这种模式一般称之为带外模式（out-of-band mode）。最近的 Storage Tank、CXFS、Lustre、BWFS 等都采用这样的结构，因此它们可以取得更好的性能和扩展性。区分带内模式和带外模式的主要依据是关于文件系统元数据操作的控制信息是否和文件数据一起，都通过服务器转发传送。前者需要服务器转发，后者直接访问。

3.2 系统服务器的方式

分布式文件系统结构上有两种大的方式：一种是专用服务器结构（Dedicated Server），另外一种是无服务器结构（Serverless）[24]。

Serverless 方式指系统中没有专用的系统服务器，所有的用户服务器都被当作系统服务器来使用。基于这种系统结构，所有的用户服务器都既是存储系统的系统服务器，又是存储系统的用户服务器。此时，一个用户服务器不仅要处理本系统的数据需求，而且，还要服务于其它系统的数据请求。这样的系统包括 xFS、CFS/Veritas[52]和 Fragipani。这种系统结构可以提供很高的性能上的可扩展性，但系统非常复杂，而且造成系统的管理困难。

此外，由于有多个系统服务器，数据的一致性变为了一个重要的问题。例如，为了解决已知性问题，xFS 使用哈希方式来寻找文件所在的服务器，而 CFS 则使用对于每个文件实行一主多从及令牌的办法。

Dedicated Server 方式采取专用服务器（群）。主要有以下几种形式：

单个系统服务器，NFS over NASD/CMU 和 Transoft File System/HP 就属于这种情况。由于只有一个系统服务器，数据的一致性得到很大的缓解，但其可扩展性受到很大限制。

多个系统服务器，SFS、GPFS、StorageTank、Lustre、BWFS 和 Tiger Shark 就属于这种情况。这种系统一般具有较好的可扩展性，但由于多个系统服务器的原因，数据一致性很难处理。在 SFS 中，一个文件只能由一个系统服务器来服务。文件到其相应的服务器的映射是由一个静态 Hash 函数完成的。由于一个文件只能映射到一个服务器，文件访问的同步及数据的一致性都可以得到较好地解决，而静态映射难以达到好的负载平衡效果。

在 Lustre、CXFS 的实现中，在某一时刻只有一个活动的提供元数据服务的文件系统服务器。在 Storage Tank 中，管理员根据文件目录结构将整个名字空间划分成多个文件集合（file set）。在某个特定的时刻，任何一个文件集合只能被映射到元数据服务器机群中的某一个，由它提供元数据管理服务。

3.3 文件与系统服务器的映射

文件映射对于数据一致性有着相当的影响。文件到其相关的系统服务器的映射可分为两种：

单一映射，在任意时刻，一个文件最多只能映射到一个系统服务器。这种映射可以大大简化数据的一致性，这是由于每个文件只由一个系统服务器负责。当需对文件进行锁或契约操作时，这些操作都只限于相关的某一个服务器，而不是分布式的操作。但它也带来了几个复杂的问题，其中最重要的就是可扩展性问题。当采取单一映射时，对于一个文件而言，存储系统所能支持的访问是被相关的单一系统服务器的性能所限制。Lustre、StorageTank、BWFS 均采用单一映射的方式。

多维映射，在任意时刻，一个文件可以映射到多个系统服务器。这种映射会增加简化数据的一致性的复杂性。这是由于每个文件是由多个系统服务器负责。当需对文件进行锁或契约操作时，这些操作都必须通过所有相关的服务器协调。这样的操作就变成了分布式的操作，从而增加了系统的复杂度。但多维映射可以提供更高的系统可扩展性。

3.4 有状态和无状态

应用程序调用文件系统的接口，实现对数据的存取和管理。用户的每一次访问，都是对多个文件系统接口多次调用的结果。应用程序对文件系统接口的调用，是有顺序，有前后关系的。这些调用一般以 open 开始，close 结束，构成一次会话（session）。应用程序调用

open 以后，获得一个打开文件的句柄（handle），关于文件的所有信息，都有文件系统（操作系统的一部分）记录。

在分布式文件系统中，根据打开文件的状态由谁记录可以分为有状态的（stateful）分布式文件系统和无状态的（stateless）分布式文件系统。在有状态的分布式文件系统中，元数据服务器跟踪每一个打开的文件，记录文件的操作信息，比如当前的读写指针，当前的文件锁等信息。在无状态的分布式文件系统中，元数据服务器不记录任何打开文件的信息，包括读写指针和文件锁等信息。NFS 本身是属于无状态的分布式文件系统，当然它有一个分开的锁管理程序来处理关于文件锁信息。最新的 NFS 第四版本加入了基于会话的语义，服务器需要记录一定的状态信息。有状态的和无状态的模式各有优缺点：由于前者不需要在每次进行通信时由客户机将状态信息带到服务器上，因此可以减少通信的数据量和通信次数，服务器也可以更好地进行客户机之间的协调；而后者则具有更好的可用性和失效恢复，因为文件的状态都记录在客户机，某个客户机的失效不影响其他客户机的操作。服务器的失效重启以后，服务器不需要进行很多的修复，客户机可以重做刚才的操作，快速恢复。

4 关键技术

分布式文件系统向用户提供和本地文件系统相同的访问接口，却可以让用户访问和管理远程的数据。分布式文件系统中的应用可能来自很多不同的节点，它所管理的数据也可能存储在不同的节点上，同时系统中可能存在多个提供元数据操作的元数据服务器，这些都视具体实现而定。分布式文件系统中有很多设计和实现与本地文件系统存在巨大的差别，也是巨大的挑战，这主要是数据、管理的物理分布和逻辑分布造成的。下面主要讲述分布式文件系统设计和实现中所要面对和解决的主要问题。

4.1 全局名字空间（Global Name Space）

全局名字空间是指每一个文件和目录在文件系统中都有一个统一的、唯一的名字，在所有的应用服务器上，用户都可以用相同的名字来访问该文件或者目录而无需关心文件的实际存储位置和给其提供服务的元数据服务器的位置。当用户访问的文件从一个存储位置迁徙到另一个新的位置以后，用户也无需知道。他可以继续用原来的名字来访问此文件或者目录。

全局名字空间在只有一个元数据服务器的分布式文件系统中比较容易实现，因为所有的名字服务都是由一个服务器提供，它具有全局知识，可以了解和管理全局所有的数据，其中包括名字的组织、映射和查找。整个系统只有一个元数据服务器的分布式文件系统实现有 Lustre、CXFS 等。虽然它们的系统中还有一个备份的元数据服务器，但是正常时却只有一个元数据服务器工作。当然这样的实现可能存在系统瓶颈，特别是元数据处理比较频繁的时候。所以 Lustre、CXFS 都在研究如何实现多个元数据服务器协同工作。Storage Tank 已经实现了这一点。它将整个文件系统划分成多个文件集（fileset），一般是一个目录及其包含的所有文件和子目录构成一个文件集。整个文件系统可以有多个文件集合，文件集可以嵌套。文件集是一个相对独立的整体，可以独立地进行备份、操作等。Storage Tank 中一个文件集只能由一个元数据服务器提供服务和管理。多个文件集可以映射到元数据服务器，共同组成一个全局名字空间。文件集和元数据服务器的映射关系由管理员指定。由多个元数据服务器协同提供名字服务和元数据服务，可以有效地进行负载平衡，也可以提高整个系统的可用性。因为一个服务器的失效不会影响其他服务器的运行，因此除了已经失效的服务器所负责的那一部分系统还可以部分对外提供服务。

4.2 缓存一致性 (Cache Coherence)

缓存一致性是指在各个应用服务器访问分布式文件系统时，它们所访问的文件和目录内容之间的一致性关系。按照 POSIX 语义的标准，在本地文件系统中，如果一个进程修改了某个文件或者目录的属性或者内容，其他进程可以在这之后马上察觉到这种改变。但是在分布式系统，要实现严格的 POSIX 是非常困难的，同时也会导致性能急剧下降。因为用来同步文件属性和内容的开销不仅会消耗计算和网络资源，也会造成很大的延时。因此各种分布式文件系统都或多或少的降低 POSIX 语义的严格性。

NFS 客户端会缓存文件或者目录的属性信息，并且保持一定的时间。它没有缓存文件的数据。只有在下次访问该文件或者目录的属性，并且时间间隔已经到期的情况下，NFS 客户端才会去向服务器端重新刷新这些属性。对于文件或者目录的某些信息，比如文件的访问时间 (Access Time)，NFS 不会在文件每次被访问时候都去向服务器更新。因为这样会大大增加客户端和服务端之间的通信量和通信延时。GPFS 也没有保证文件的访问时间的 POSIX 语义。

在 AFS 服务器会记录客户端的缓存行为，客户端缓存文件的属性和数据。当有其他客户端试图修改已经被别的客户端缓存的文件属性或者数据时，服务器会采用回调 (call-back) 的机制通知客户端这些缓存已经失效。

分布式文件系统对客户端的写操作一般采用两种方式处理：写回 (write-back) 和写透 (write-through)。前者缓存客户端写的数据，然后在适当的时候再写回到数据实际应该存储的物理位置；后者是在用户发出写请求时，直接将数据写到该写的地方，然后才将控制返回。前者的好处在于可能减少数据的实际传输量。因为基于数据访问局部性的现象，最近被访问的数据很可能在不久的将来被再次访问。有些文件可能是临时文件，写入以后不久可能就会被删除，有些文件在写入以后，很可能马上就被再次修改。不过这样的方式，很可能影响其它也在同时读该文件的其他主机上的进程，因为它们难以读到最新的数据。

同时为了支持客户端对文件加锁的操作，一般有两种处理方式：集中锁和分布式锁。如果一个分布式文件系统中只有一个锁管理器，就称之为集中锁。如果有多个管理器，就称之为分布式锁。集中锁的好处是实现相对容易，不足之处就是效率相对低下，而且一旦锁管理器失效，整个系统使用锁的服务都会被迫停止，直到锁管理器恢复正常运行。按照文件锁的粒度，一般有如下三种层次：整个文件、文件的一段 (segment)、文件的任何一部分。第二种是将文件按照某个特定的大小分成多个段，锁的粒度只能是一个段或者是多个段。第三种情况下，用户可以对文件的任何部分加锁，甚至是现在文件中不存在的部分，比如锁的范围已经超出了文件的大小。这三个层次的锁设计和实现难度依次升高，但应用程序对锁的冲突访问的发生则依次降低，从而应用程序访问文件系统的并发度可以依次提高。关于这些锁的赋予和撤销，在[25]中有详细研究。

4.3 安全性

在传统的分布式文件系统里边，所有关于文件的数据和元数据都要经过元数据系统进行存取。因此系统的安全性都可以在元数据服务器那里集中控制。这样的情况下，安全性比较容易实现，一般采用简单的身份验证和访问授权的形式即可，比如 NFS、CIFS 等。

如果应用服务器可以直接访问存储设备，那么系统设计就需要一定的改变来确保所有数据的安全存取。NASD[26]通过在设备上添加一定的软件，利用设备本身的处理能力对用户的请求进行验证。用户在第一次访问某个文件的数据时，由元数据服务器对其进行身份验证和授权，之后通过特定的访问接口和磁盘设备进行交互以达到有限制地访问数据的目的。

Lustre 采用多种方式来实现系统中文件的安全访问。比如在一个可以信任的域内，采用简单的基于用户 ID 和组 ID 的形式，也可以再加入一定的身份验证。Lustre 还采用给数据加密的方式保护数据。由于 Lustre 采用基于对象的存储，凭借对象存储对于安全的控制也能很好的控制系统的的核心安全。

现在越来越多的分布式文件系统采用 SAN 作为其存储环境。SAN 的直接磁盘访问方式也给系统的安全实现带来了很大的挑战。Storage Tank 利用系统中的一个可以信赖的验证服务器，采用了独特的“着色+色彩”的模型，实现了分布式文件系统的数据安全[25]。在此模型中，磁盘设备无需跟验证服务器进行任何的信息交互，只需要对应用服务器提供的“票据”进行验证，简化了磁盘的设计和实现。同时这种方法跟具体的应用环境无关，可以在分布式文件中使用，也可以在电子图书馆等使用共享存储设备 (share-disk) 的地方发挥作用。

4.4 可用性

分布式文件系统一般都是由多个节点组成，需要集体协作才能对外提供服务。但是随着系统规模的逐渐扩大，系统中软硬件的失效概率也会大大增加，比如磁盘损坏，节点失效，网络断开等。一般采用 RAID 技术保证磁盘的有效工作，提供比较稳定的数据源。对于节点的失效，一般针对不同的节点类型有不同的处理方法：对于元数据服务器，一般采用失效接替减少系统暂停服务的时间（比如 Lustre、Storage Tank 等），采用日志技术进行文件系统的快速恢复（CXFS 等）；对于系统中的存储节点，采用数据复制技术（replica），或者采用类似 RAID 在各个存储服务器之间做数据冗余存储等；对于网络失效，采取把网络划分成多个互相不同通信的几个部分，如 GPFS、Storage Tank 规定只有包含半数以上可用节点的部分才能继续工作，以避免造成数据不一致性。

4.5 可扩展性

应用对分布式文件系统在性能和容量方面的要求在不断的增长。分布式文件系统一般都是通过扩充系统规模来取得更好的性能和更大的容量。但是对于各种系统结构，系统在规模上可能存在不同限制。比如 NFS 或者 CIFS 采用单个服务器，那么整个系统的性能和容量都存在瓶颈，难以突破单个服务器系统的极限。于是 Slice File System 通过在客户机和服务器之间添加 μ proxy 提供系统性能，Zebra 通过管理多个存储设备获得更大的容量，Storage Tank、TotalStorage 里的元数据服务器集群帮助系统可以动态添加更多的存储设备，服务于更多的客户端文件访问。设法保证系统的性能随着系统的规模能够线性增长，是分布式文件系统一直努力的目标。

另外一方面，扩展性还体现在系统规模的增长不会带来系统管理复杂度的过度增加。降低控制系统的管理成本，简化系统的管理流程，都是分布式文件系统实现的关键技术。Storage Tank、CXFS、GFS 等将系统分为多个层次，每个层次实现不同的功能，简化整个系统的设计和管理难度。其中很重要的一个技术方面就是存储空间的虚拟化（Virtualization）。虚拟化存储提供给文件系统的是一个共享的虚拟存储空间，屏蔽了存储实现的细节，同时具备了比如容错、动态负载平衡等特点。分布式文件系统通过利用这些特性，就可以取得很好的扩展性。

4.6 其他关键技术

此外，文件系统的快照和备份技术、热点文件处理技术、元数据集群的负载均衡技术、分布式文件系统的日志技术[42]等，都是目前分布式文件系统中需要解决的技术难点。

5 蓝鲸分布式文件系统

蓝鲸分布式文件系统（BWFS）是中科院计算所工程中心研发的基于网络存储的分布式文件系统。BWFS 采用基于 IP 的网络环境，多个存储设备形成一个共享的虚拟存储池，提供大容量、可扩展的存储空间。同时，存储设备具有很好的 I/O 处理性能，通过千兆以太网（Gigabit Ethernet）提供外界访问。有很高的传输速度和相对较低的成本，具有很好的性价比。BWFS 的体系结构设计参考了当今分布式文件系统最新的发展趋势，吸取了很多其他本地文件系统和分布式文件系统的高级特性，克服了在某些分布式文件中存在的瓶颈，使其能够真正满足海量数据并发访问的需求。下图是系统结构图：

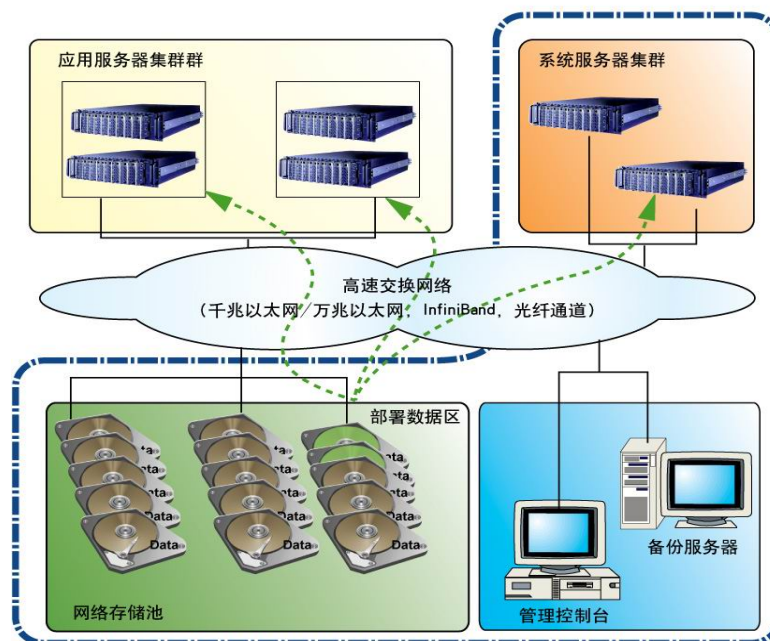


图 BWFS 系统结构图

BWFS 为机群环境中的应用节点提供单一映象的文件系统接口。文件系统的元数据由专用的元数据服务器（MS）集群处理，绑定服务器负责文件和 MS 的映射关系以及 MS 集群的负载均衡。文件系统的数不需要经过元数据服务器的处理，直接在应用服务器和存储设备之间传送。这种带外传输的模式，使元数据服务器专注于处理元数据请求，提高了元数据服务器的性能，有效克服了数据传输瓶颈。同时，蓝鲸文件系统全局动态分配数据块和索引节点资源，使得系统具有很好的扩展性。通过在客户端缓存尽量多的元数据信息，可以明显减少客户端和元数据服务器之间的请求，大大提高系统的性能。

蓝鲸分布式文件系统采用 Lazy-Binding, Logical-Binding, Local-Binding-Cache 等技术来提供多个元数据服务器机群之间的负载平衡，尽量发挥每一个元数据服务器的最大性能。系统针对不同属性文件或者目录采用不同的资源分配策略，以保证任务的平均分配。系统还可以根据运行时的实际负载情况，动态调整各个元数据服务器之间的负载分配以及各个存储服务器之间存储空间分配。

蓝鲸分布式文件系统采用针对元数据的分布式日志，保证系统元数据的一致性，缩短系统灾难恢复的时间。系统采用双机热备进行失效接替管理，减少系统单点故障发生的几率。系统采用智能化的备份方案保证用户数据的可用性。

蓝鲸分布式文件系统支持数百个应用节点，多个元数据服务器和网络存储服务器。在大

规模的机群环境下，系统的可管理性显得非常重要。蓝鲸分布式文件系统中，采用动态部署系统轻松部署运行环境和切换应用服务器，降低管理的难度和成本。

采集自系统中各个节点有关系统运行的实时信息，为系统故障分析和性能调整提供了依据。系统可以自动完成一部分负载动态平衡的功能和故障的自我诊断与恢复。

5.1 可扩展性

蓝鲸分布式文件系统一直十分注重系统可扩展性的设计，包容系统规模、系统整体容量、系统性能各个方面。蓝鲸目前可以支持上千个应用节点的并发访问；最大容量可以扩充至 512TB；可以提供每秒若干 GB 的聚集数据传输能力。这样的扩展性是建立在良好的体系架构之上的，因为系统可以根据性能的要求，配备多个网络存储服务器和多个元数据服务器。

6 展望

各种计算机技术在不断地发展，应用也在不断地发展，促使分布式文件系统也需要不断发展。以下一些技术和方向将对未来分布式文件系统的设计产生较大影响。

6.1 网络技术

计算机中很多部件的性能或者容量都在某种程度上遵循着摩尔定律发展，但是它们之间还是有很大的差异的。分布式文件系统中涉及到的软硬件技术主要包含 CPU 的处理速度，内存的容量和访问速度，I/O 总线的传输速度，磁盘的容量、访问延迟和传输速度，网络的传输带宽和访问延迟等。这么多因素中间，从今天的发展角度看，唯有网络的性能在跳跃式发展，其性能和发展速度已经远远超过了磁盘技术的发展。光纤通道，千兆以太网，万兆以太网，InfiniBand 等网络传输协议标准的确立和产品的出现，使得网络已经不再是系统性能的瓶颈。

同时，为了克服利用 TCP/IP 网络协议传输数据带来的巨大 CPU 开销和性能限制，远程内存直接访问（RDMA）技术已经在出现并且得到了一定的应用。传统的数据传输访问和消息交换机制在 RDMA 中已经不再使用，新的基于 RDMA 协议的文件系统访问协议将会出现。

总之，利用高速互连网络和 RDMA 快速传送数据，将是未来分布式网络文件系统的-一个发展方向。

6.2 数据库技术

随着系统规模越来越大，数据的组织和检索对于系统结构和性能的影响越来越重要。许多本地文件系统的实现中，已经借鉴了数据库的技术，如 JFS[28], Befs[53], XFS, Reiser[50] 等；文件系统的日志技术也来自于数据库技术。对于分布式文件系统，数据库技术将对文件系统的磁盘组织、元数据组织、查询方式、分布式日志等产生较大影响。

6.3 虚拟存储技术

分布式文件系统将支持越来越多的应用。如何在一个文件系统中针对不同的应用提供不同的服务，包括针对不同的访问模式（频繁的创建删除、分段读等），不同的文件大小，不同的性能要求（如持续带宽），不同的组织方式（大目录小文件、小目录大文件等）等；在规模增大的同时，如何更好地满足个性化的需求，将是分布式文件系统需要解决的问题。

6.4 智能存储设备

智能存储设备利用本身的计算能力可以提供更多的语义级支持,如在某部分存储区域上查找某个字符串,将某部分存储区域拷贝到另外的存储区域,做某部分存储区域的快照等。在分布式文件系统的设计中,利用智能存储设备提供的功能,来提高系统的性能、可靠性,也是分布式文件系统的一个发展方向。

7 总结

通过简要回顾分布式文件系统的发展和当前的实现情况,并分析对比各个系统体系结构的优缺点,我们发现分布式文件系统逐渐趋向成熟。它们采用的很多技术和理念,都利用了当时最先进的软硬件技术。每一种系统在历史上都较好地解决了它所面临的问题,取得了不错的效果。技术的进步,应用需求的发展,是分布式文件系统不断进步的原动力。现在的分布式文件系统,已经不再是单纯的文件系统,而是融合了很多其他方面计算机技术的综合系统,其发展更加注重系统的性能、扩展性、可用性等因素,同时结合存储管理,也更加强调降低整个系统的总体拥有成本,包括系统的管理成本。

8 参考文献

- [1] Satyanarayanan, M. *A Survey of Distributed File Systems*. In Annual Review of Computer Science. Annual Reviews, Inc, 1989
- [2] Postel, J., and Reynolds, J. *RFC 959: File Transfer Protocol (FTP)*. Tech. rep., Internet Request for Comments, October 1985.
- [3] Fabio Kon, *Distributed File Systems Past, Present and Future A Distributed File System for 2006* March 6, 1996.
- [4] R. Sandberg, *Sun Network Filesystem Protocol Specification*, Technical Report, Sun Microsystems, Inc., 1985.
- [5] S. Shepler, B. Callaghan. *RFC 3530: Network File System (NFS) version 4 Protocol*. The Internet Society, 2003.
- [6] M. Satyanarayanan, J. H. Howard, D. N. Nichols, R. N. Sidebotham, A. Z. Spector and M. J. West, *The ITC Distributed File System: Principles and Design*, Proceedings of the 10th Symposium on Operating System Principles (SOSP), Orcas Island, Washington, U.S., ACM Press, December 1985.
- [7] Braam, P. J. *The Coda Distributed File System*, Linux Journal, June 1998.
- [8] Peter Braam, Philip Nelson, *Removing Bottlenecks in Distributed Filesystems: Coda InterMezzo as examples*, www.inter-mezzo.org.
- [9] Randolph Y. Wang and Thomas E. Anderson. *xFS: A Wide Area Mass Storage File System*. In Proceedings of the Fourth Workshop on Workstation Operation Systems, pages 71 -- 78, October 1993.
- [10] E. K. Lee and C. Thekkath. *Petal: Distributed virtual disks*. In Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), pages 84-92, October 1996.
- [11] Chandramohan A. Thekkath, Timothy Mann, Edward K. Lee, *Frangipani: A Scalable Distributed File System*, Symposium on Operating Systems Principles (SOSP), 1997.
- [12] Steve Soltis, Grant Erickson, Ken Preslan, Matthew O'Keefe, and Tom Ruwart, *The Design and Performance of a Shared Disk File System for IRIX*, Fifteenth IEEE Symposium on Mass Storage Systems, 1998.
- [13] Kenneth W. Preslan, Andrew P. Barry, etc. *A 64-bit, Shared Disk File System for Linux*, Storage Conference, 1999.
- [14] Frank Schmuck and Roger Haskin, *GPFS: A Shared-Disk File System for Large Computing Clusters*, Proceedings of the Conference on File and Storage Technologies (FAST'02). 28-30 January 2002, Monterey, CA,

pp. 231–244. (USENIX, Berkeley, CA.)

- [15] Christos Karamanolis, Mallik Mahalingam, Dan Muntz, Zheng Zhang, *DiFFS: a Scalable Distributed File System*, Hewlett-Packard Company 2001.
- [16] Hyeran Lim, Vikram Kapoor, Chirag Wighe and David H.-C. Du, *Active Disk File System : A Distributed, Scalable File System*, Proceedings of the 18 th IEEE Symposium on Mass Storage Systems and Technologies, San Diego, April 2001, pp. 101-115.
- [17] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, B. Hillsberg, *IBM Storage Tank—A heterogeneous scalable SAN file system*, IBM SYSTEMS JOURNAL, VOL 42, NO 2, 2003.
- [18] Matt DeBergalis, Peter Corbett, etc., *The Direct Access File System*, Proceedings of FAST '03: 2nd USENIX Conference on File and Storage Technologies.
- [19] Laura Shepard and Eric Eppe, *SGI® InfiniteStorage Shared Filesystem CXFS™: A High-Performance, Multi-OS Filesystem from SGI*, White Paper, 2691 [08/21/2003], ' 2003 Silicon Graphics, Inc. www.sgi.com
- [20] PANASAS WHITE PAPER, *Object Storage Architecture*, <http://www.panasas.com>, 10/19/2003.
- [21] www.lustre.org
- [22] 史小冬, 孟 丹, 祝明发, *COSMOS:一种可扩展单一映像机群文件系统*, 南京大学学报 (自然科学), 2001.10
- [23] 吴思宁, 贺劲, 熊劲, 孟丹, *DCFS 机群文件系统服务器组的设计与实现*, 2002 全国开放式分布与并行计算学术会 (DPCS2002), 2002.
- [24] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. *Serverless network file systems*. In Proceedings of the 15th Symposium on Operating Systems Principles, pages 109--126, Copper Mountain Resort, Colorado, December 1995. ACM.
- [25] R. Burns. *Data management in a Distributed File System for Storage Area Networks*. PhD Thesis. Department of Computer Science, University of California, Santa Cruz, March 2000.
- [26] Garth A. Gibson, David F. Nagle, etc., *NASD Scalable Storage Systems*, Proceedings of USENIX 1999, Linux Workshop, Monterey, CA, June 9 - 11, 1999.
- [27] Theodore Y. Ts'o, Stephen Tweedie, *Planned Extensions to the Linux Ext2/Ext3 Filesystem*, Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, Monterey, California, USA June 10-15, 2002.
- [28] Juan I. Santos Florido, *Journal File Systems*, Published in Issue 55 of *Linux Gazette*, <http://www.linuxgazette.com/issue55/florido.html>, July 2000.
- [29] Howard, J.H. *An Overview of the Andrew File System*, Proceedings of the USENIX Winter Technical Conference Feb. 1988, Dallas, TX
- [30] Howard, Kazar, M.L. J.H. *Scale and Performance in a Distributed File System*, ACM Transactions on Computer Systems Feb. 1988, Vol. 6, No. 1, pp. 51-81.
- [31] Satyanarayanan, M. *Integrating Security in a Large Distributed System*, ACM Transactions on Computer Systems, Aug. 1989, Vol. 7, No. 3, pp. 247-280
- [32] Satyanarayanan, M. *Scalable, Secure, and Highly Available Distributed File Access*, IEEE Computer, May 1990, Vol. 23, No. 5
- [33] Satyanarayanan, M. *Coda: A Highly Available File System for a Distributed Workstation Environment*, Proceedings of the Second IEEE Workshop on Workstation Operating Systems Sep. 1989, Pacific Grove, CA
- [34] Roger Haskin and Frank Schmuck, *The Tiger Shark File System*, Proceedings of IEEE 1996 Spring COMPCON, Santa Clara, CA, Feb. 1996.
- [35] Roger Haskin, *The Shark Continuous Media File Server*, Proceedings of IEEE 1993 Spring COMPCON, San Fransisco, CA, Feb. 1993, pp. 12-17.

- [36] Roger Haskin and Frank Stein, *A System for the Delivery of Interactive Television Programming*, Proceedings of IEEE 1995 Spring COMPCON, San Fransisco, CA, Mar. 1995, pp. 209-214.
- [37] John H. Hartman and John K. Ousterhout. *The Zebra striped network file system*. ACM Symposium on Operating System Principles (Asheville, NC), pages 29--43, 5--8 December 1993.
- [38] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout, *Caching in the Sprite Network File System*. ACM Transactions on Computer Systems, 6(1), February 1988.
- [39] Fred Dougliis, John K. Ousterhout, M. Frans Kaashoek, Andrew S. Tanenbaum, *A Comparison of Two Distributed Systems: Amoeba and Sprite*, ACM Transactions on Computer Systems, 4(4), Fall 1991.
- [40] J. K. Ousterhout, A. R. Chersonson, F. Dougliis, M. N. Nelson, and B. B. Welch. *The Sprite Network Operating System*. IEEE Computer, 21(2):23-36, February 1988.
- [41] Brent B. Welch: *Measured Performance of Caching in the Sprite Network File System*. Computing Systems 4(3): pages 315-342 , 1991.
- [42] Michael L. Kazar, Bruce W. Leverett, Owen T. Anderson, Vasilis Apostolides, Ben A. Bottos, Sailesh Chutani, Craig F. Everhart, W. Antony Mason, ShuTsuTu, and Edward R. Zayas. DEcorum file system architectural overview. In Proceedings of the Summer USENIX Conference, pages 151--164, June 1990.
- [43] Mary Baker, John Ousterhout, *Availability in the Sprite Distributed File System*, ACM Operating Systems Review, 25(2), pages 95-98, April 1991.
- [44] Darrell C. Anderson, Jeffrey S. Chase, Amin M. Vahdat, *Interposed Request Routing for Scalable Network Storage*, Proceedings of the Fourth Symposium on Operating System Design and Implementation, October 2000.
- [45] Charlotte Brooks, Ravi Khattar, Satoshi Suzuki, Mats Wahlstrom, *IBM TotalStorage: Introducing the SAN File System*, IBM International Technical Support Organization, November 2003.
- [46] <http://bebits.com/>
- [47] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck, *Scalability in the XFS File System*. Proceedings of the USENIX, 1996.
- [48] Paul Leach and Dan Perry, *CIFS: A Common Internet File System*. Microsoft Interactive Developer, November 1996.
- [49] EMC Celerra HighRoad, EMC white paper.
http://www.emc.com/pdf/products/celerra_file_server/HighRoad_wp.pdf
- [50] www.namesys.com
- [51] Anupam Bhide, et al, *File Virtualization with DirectNFS*, Proc. of the 10th NASA Goddard Conference on Mass Storage Systems and the 9th IEEE Symposium on Mass Storage Systems, pp. 43-58, Maryland, USA, 2002.
- [52] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, *Wide-area cooperative storage with CFS*. Symposium on Operating Systems Principles (SOSP'01), October 2001.

作者: **黄 华** **中国科学院计算技术研究所工程中心** **博士研究生**
杨德志 **中国科学院计算技术研究所工程中心** **博士研究生**
张建刚 **中国科学院计算技术研究所工程中心** **副研究员 硕导**