HAMMERSPACE

并行NFS（pNFS）

高性能工作负载专用协议

Hammerspace Asia
是我们的微信服务号
发布内容偏重于
公司新闻与行业方案
扫码可关注

订阅号 “悍宙尔空间”
发布内容偏重于
技术细节
扫码可关注

目录

- “重新”认识NFS
- NFS与pNFS发展及动态
- NFSv4.2特性与I/O机制
- pNFS架构、特性与I/O模型
- 错误恢复与特殊操作
- Hammerspace的pNFS实践
- 参考文献与资源

“重新”认识NFS

每当某项技术变得普及且一直稳定运行时，它就会被视为理所当然。NFS（网络文件系统）——这项去年刚满40周年的文件共享协议——便是其中之一。除非您在IT领域工作已久，否则NFS似乎一直都存在；而像所有基础设施一样，它仅在发生罕见故障时，才会引起人们的注意。

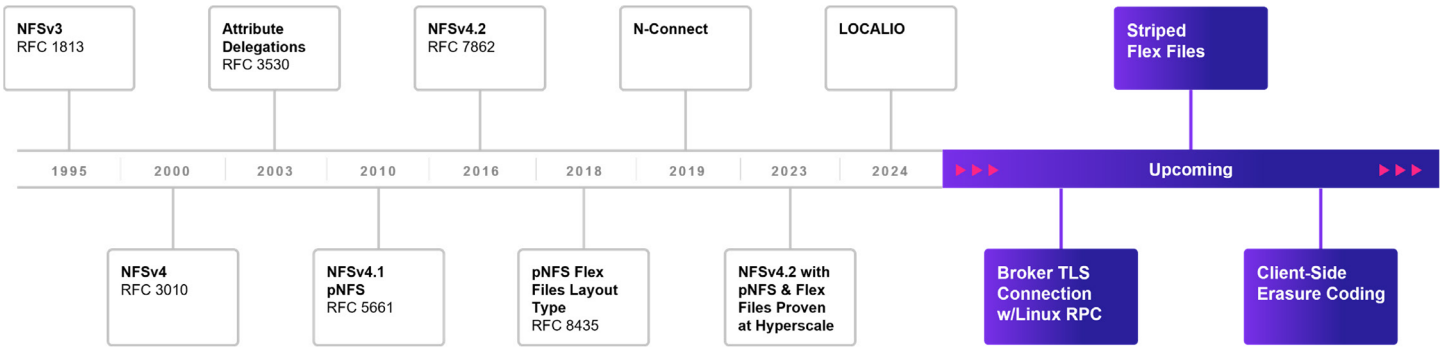
遗憾的是，由于NFS如此普及且为人熟知，许多IT从业者——甚至存储专家——对其能力和潜力的认知仍停留在过时阶段。传统观点长期认为：NFS适用于通用文件共享，但它“不为速度而生”（"Not For Speed"）。真正的高性能文件共享需要放弃标准，部署复杂昂贵且需精心维护的第三方并行文件系统，对吧?

这正是专有文件系统和横向扩展存储设备厂商希望世人相信的论调，但事实并非如此。一批专业人士始终在推动NFS向前发展，其能力早已远超诞生初期的局限。如今的NFS已具备与HPC级并行文件系统正面对抗的实力，无论在性能还是可扩展性上皆如此。没错——标准NFS（随每个Linux发行版内置）足以胜任当今最严苛的应用场景，包括AI及其他GPU计算工作负载。

本文是对现代NFS的技术综述，详细阐释其架构与特性如何被优化以实现可扩展的并行高性能。它描述了当前的技术前沿，并展望了正在标准流程中推进的增强功能。同时探讨了Hammerspace如何基于此为基础，通过数据编排（data orchestration）、全局命名空间（global namespace）等能力扩展了自身的数据服务范畴。

NFS 与 pNFS 的一些历史回顾

在深入探究NFSv4.2与pNFS的技术细节前，我们有必要回溯一些历史，来真正理解NFS的演进历程。如下图所示，其发展过程中的重要时间节点可清晰地展现这一路的里程碑轨迹。



上图 - NFS 发展的重要时间节点

NFS起源于1984年的Sun Microsystems，而至今最流行的版本v3发布于1995年。NFSv3与其前代协议一样采用无状态（stateless）设计，依赖辅助协议处理挂载（mounting）和文件锁定（file locking）。这种架构导致其在网络传输中产生“过度通信”（chatty），不仅难以穿越防火墙，还引入了复杂的故障模式。

NFSv4则采用了截然不同的方法：它使用有状态（stateful）模型，并将所有功能整合至单一协议且运行于单一端口。此外，该版本引入了关键概念——复合操作（compound operations，简称compounds）和委托（delegations），后文将详细探讨这些机制。在v4基础上，NFSv4.1进一步增加了重要安全增强功能，并首次引入并行NFS（pNFS），这是实现NFS高性能的关键突破。尽管意义重大，但pNFSv4.1仍受困于过度的元数据请求（getattr）通信问题及其他缺陷，使其无法完全达到与HPC级并行文件系统竞争所需的性能水平。

NFSv4.2以“通过向客户端推送更多功能来提升可扩展性”为目标，新增了大量可选特性。其核心创新是实时通信通道（real-time communication channel），使客户端能主动向元数据服务器反馈存储服务器节点的性能数据与健康状况。2018年，pNFS通过名为灵活文件布局（Flexible File layout，FlexFiles）的新布局类型实现了增强。该布局支持将任意NFSv3存储系统作为pNFS后端存储节点——由于NFSv3支持具有普适性（ubiquitous），这意味着任何新建或现有NFSv3存储均可无缝接入pNFS架构。FlexFiles将pNFS转变为真正开放的解决方案，使传统存储环境无需改造即可升级至并行文件系统性能。**凭借这些突破性进展，pNFSv4.2在性能与可扩展性上，可比肩HPC行业专用的并行文件系统。**

标准固然重要且必要，但在其落地实施之前，终究只是纸面规范与空谈。以 NFS 为例，这些高性能特性的开发始终与 Linux 社区紧密协作，并通过 IETF（互联网工程任务组）流程提交，以确保厂商中立性。

而最终成果是：这些改进现已纳入标准 Linux 内核，并自 2019 年起被主流发行版广泛集成。通过 pNFSv4.2，当前所有主流 Linux 发行版均内嵌了一个高性能并行文件系统，其性能表现已达到甚至超越了最大规模 HPC 环境中使用的同类系统。

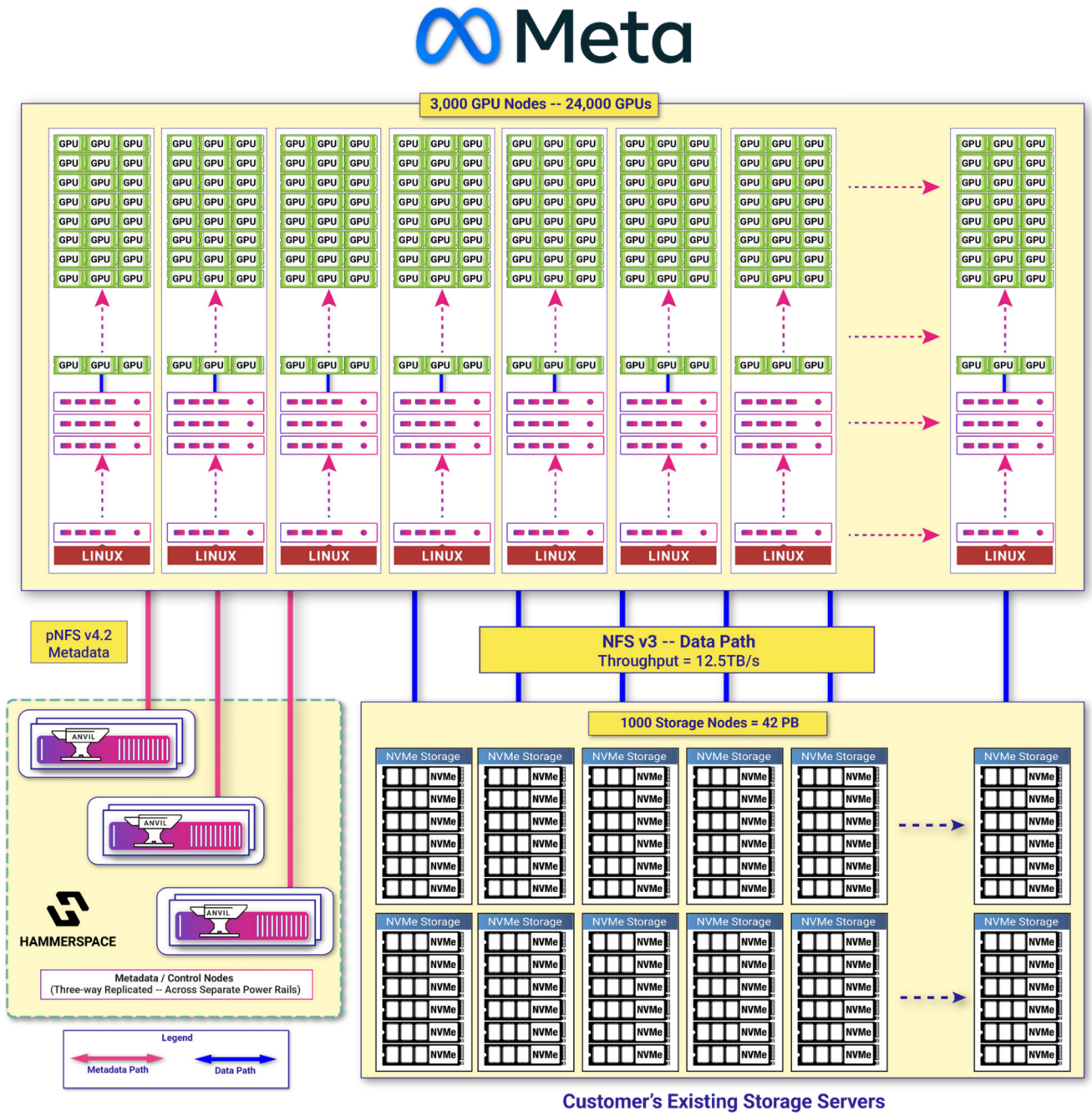
以下表格总结了 NFSv3（当提及 "NFS" 时大多数人首先联想到的版本）与 NFSv4.2（包含 pNFS 与 FlexFiles）之间的关键差异：

特征	NFSv3	NFSv4.2 (pNFS)
性能	单服务器	HPC级性能：支持RDMA（远程直接内存存取）及并行数据传输至多个存储服务器节点
扩展性	单服务器	横向扩展存储服务器节点的统一命名空间：跨所有存储节点提供单一逻辑视图
效率	无状态	有状态协议：大多数操作可缓存状态，显著减少网络往返次数
安全性	基础	访问控制列表（ACLs）：高级传输中的加密与严格的客户端身份认证
可管理性	简便	大规模场景下的简洁性：即使跨数据中心部署，仍可保持HPC级性能；将MOUNT操作整合进NFSv4.2协议
客户端	标准	标准：自2017年起，所有Linux发行版均内置支持

上图 - 基于pNFS和FlexFiles的NFS v4.2和NFS v3的比较

实践是检验真理的标准

资深IT专业人士往往会持怀疑和保守态度。这种怀疑和保守态度是可以理解的，因为技术炒作与现实之间通常存在差距。pNFSv4.1未能实现HPC级性能，这让业界许多人留下了持久的负面印象。但自那以来持续的改进——包括pNFSv4.2与FlexFiles的推出——彻底改变了这一现状，如今该协议已足够稳定且性能优越，可在全球最大的GPU计算环境中部署，用于超大规模AI工作负载。在深入探讨NFSv4.2与pNFS的工作原理之前，我们先来看一个实际案例，展示其在大规模生产环境中的应用：



上图 – Meta公司的 pNFSv4.2 部署

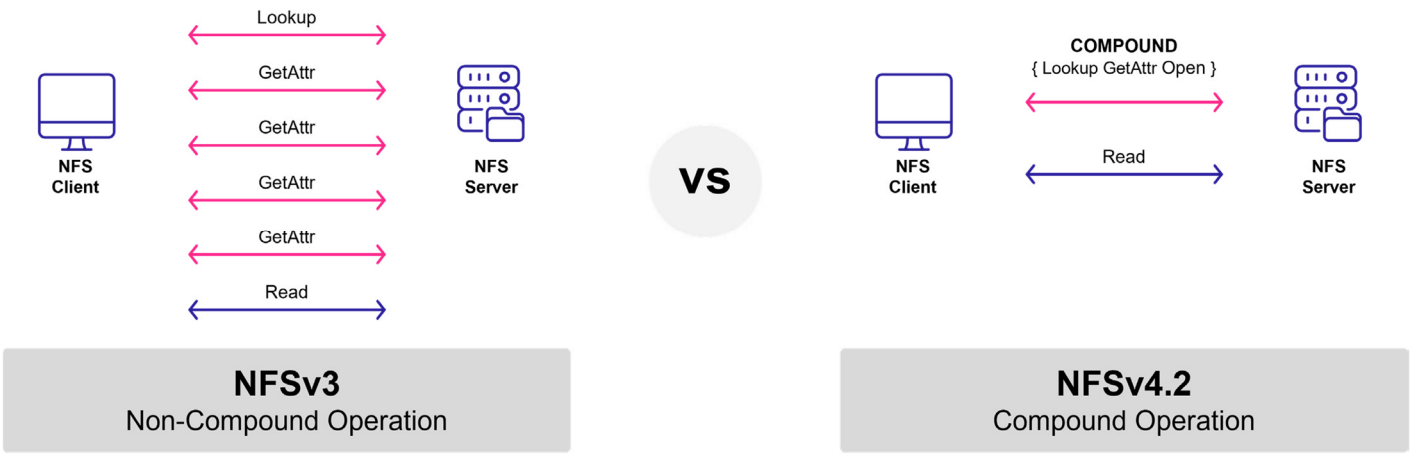
Meta 通过使用 pNFSv4.2， 为训练大型 AI 模型的 GPU 集群提供数据支持。该系统运行在 Meta 的标准基础设施之上，**由通用硬件通过标准以太网（非 RDMA）连接组成**。GPU 服务器客户端和 NVMe 存储服务器均无需安装特殊客户端软件——仅需标准 Linux NFS 即可。在选择 Hammerspace 和 pNFS 之前，Meta 测试了众多其他方案，包括主流厂商提供的专有系统。没有任何其他解决方案能够在性能和厂商中立性方面满足其要求，甚至相差甚远。这一实际案例证明，基于标准的 pNFS 方案可在任意规模（包括超大规模）下实现极限性能。如需了解此部署的更多细节，请参见客户案例。

NFSv4.2的 特性与 I/O

与 NFSv3 相比，NFSv4.2 的增强能力源于核心架构设计与细微优化的结合。本节将详细描述这些关键因素，但有一个例外：pNFS 值得单独成节（见下文），不仅因其重要性，更因其尚未被广泛认知。此处深入探讨的原因在于：尽管大多数 IT 从业者熟悉 NFS，但部署过 pNFS 的人极少。以下讨论的功能均在 NFSv4 的演进过程中逐步引入，但已被全部包含于 v4.2 版本中。本文后续内容中，除非涉及版本差异，否则统称“NFS”；若需强调版本，则会标注具体版本（如 NFSv3）。

复合操作（Compound Operations）

NFSv4 引入的一项重大增强功能是 COMPOUND RPC（复合远程过程调用）。在 NFSv3 及更早版本中，每个操作均为独立的 RPC 过程。而 NFSv4 及后续版本仅包含两个 RPC：NULL 和 COMPOUND。通过 COMPOUND，客户端可将多个操作打包为单个请求，显著减少与服务器的往返次数。与 NFSv3 相比，这是效率层面的根本性提升，尤其对延迟敏感型负载效果显著。复合操作类似于客户端发送至服务器执行的小型程序。服务器按顺序执行 COMPOUND 中的操作，甚至可将前一步的结果传递至后续步骤。仅当遇到错误或所有步骤执行完毕后，服务器才会向客户端返回结果。



上图 - NFSv4.2 Compound Operations

锁定与状态管理

NFSv3 在技术上是一个无状态协议，但锁定机制本质上需要维护状态信息。所以在 NFSv3 中实现锁定时，需依赖一个独立有状态锁管理器协议（Lock Manager Protocol）。然而，锁管理器协议的实现是可选的，且其锁定为“建议性锁”（Advisory Lock）。但可选且建议性的锁定机制对实际应用帮助有限——为防止冲突并确保数据一致性，应用程序需要明确知晓自身是否独占文件。

由于 NFSv4.2 是有状态协议，其锁定功能可直接整合在核心协议之中，而非依赖独立的网络锁管理器（NLM）协议。NFSv4.2 的锁定为标准功能，且锁定为“强制性锁”（Mandatory Lock），这对应用程序的实用性显著提升。所有状态信息由服务器统一管理，即使部分责任临时委派给客户端，服务器仍保留最终控制权。

NFSv4.2 的状态类别包括以下四类：

- **字节范围锁（Byte-range Locks）：允许锁定文件的特定字节范围。**
- **共享预留（Share Reservations）：授予客户端访问文件的权限，并可拒绝其他客户端同时访问同一文件。**
- **委派（Delegations）：授权客户端在特定规则下独立操作，从而缓存文件（未来可扩展至目录）的数据与元数据。**
- **布局（Layouts）：在使用 pNFS 时为客户端提供数据访问权限。**

上述状态由元数据服务器分别授予客户端，并在客户端完成操作后主动释放，或由元数据服务器主动召回（Revoke）或撤销（Recall）。每个客户端还拥有一个全局计时器（称为“租约”Lease）——客户端的任何操作均会延长其租约，但若客户端离线，元数据服务器将在租约到期后将其状态设为失效，防止“僵死”客户端阻断其他客户端访问数据。

若元数据服务器发生故障，其恢复后会进入一个短暂的“宽限期”（Grace Period），在此期间客户端可重新申领其状态信息。

为使元数据服务器能通知客户端必要的变更（如委派召回或锁定释放），NFS 也提供了回调机制（Callback Mechanism）。

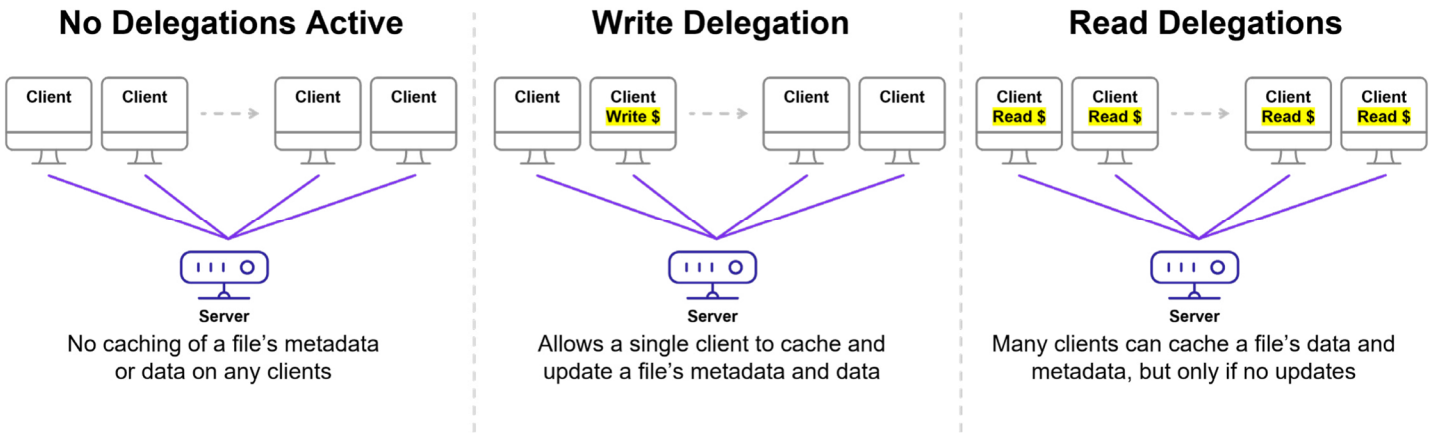
委派（Delegations）机制

客户端缓存文件数据与元数据的能力是性能优化的核心。这一机制在 NFSv4.2 中并非新功能，但相比早期版本有显著提升。

在多数场景中，实际文件冲突（例如两个客户端同时写入同一文件的同一字节范围）的发生频率远低于非冲突操作（如单客户端写入、多客户端读取）。NFSv4.2 的设计思路并非针对罕见冲突优化，而是通过委派机制，最大限度减少常见场景下的客户端与服务器通信开销。

委派机制使元数据服务器能够将特定操作的责任临时委派给客户端。当客户端完成相关操作后，会将委派权限归还给元数据服务器。如果在客户端归还委派之前，元数据服务器收到来自另一客户端的冲突请求，它将通过回调机制（callback）从第一个客户端处收回委派。此时，该客户端需先刷新（flush）其所有变更，然后通过归还委派来确认回收操作。

目前，文件数据和属性的缓存通过委派机制实现。未来将引入目录委派（directory delegations），以进一步提升客户端缓存状态的能力，并最大限度减少网络传输流量。



上图 – 委派与客户端缓存

委派机制使客户端能够比以往更加积极地进行缓存，因为客户端与元数据服务器之间就文件系统上的冲突操作保持着双向通信。客户端无需持续向元数据服务器查询状态，而只需监听（通常极少发生）回收请求，在无回收请求时即可独立执行操作。

由于元数据服务器负责管理所有状态信息，因此必须具备在客户端因任何原因未能及时归还委派时的恢复能力。一旦客户端的租约（lease）到期，元数据服务器可单方面撤销该客户端的所有状态信息，包括尚未归还的委派。客户端则有责任处理状态被撤销后的后续事宜，根据需要验证或重新申领相关状态。

认证与安全

在 NFSv3 中，安全机制较为基础。在身份认证方面，元数据服务器将“客户端”视为一整台机器或源 IP 地址。客户端主机负责对终端用户进行身份验证，而元数据服务器则默认该验证已正确执行。而 NFSv4 弥补了这一安全漏洞，支持从客户端的终端用户到 NFSv4.x 服务器的端到端身份认证。

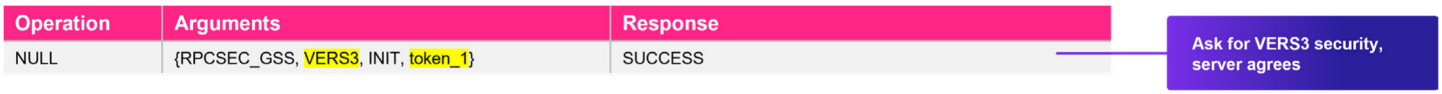
由于 NFSv4.x 服务器可能支持多种安全机制，甚至对文件系统的不同部分采用不同的安全策略，因此在客户端在挂载文件系统之前，必须进行安全协商。NFS 允许客户端查询服务器对其文件系统资源访问所需的安全机制。基于该查询的响应，客户端可选择一种双方均可接受的安全机制。

NFSv3 仅支持基于 POSIX 模式位（POSIX mode bits） 的文件访问控制机制。尽管如此，大多数 NFSv3 服务器实现（如 Linux）通过独立的 RPC 程序支持 POSIX 草案 ACL（POSIX Draft ACLs），这些 ACL 定义于未正式通过的 POSIX 1003.1e/1003.2c 第 17 版草案标准。Linux 内核于 2002 年正式引入了对这类 ACL 的支持。

NFSv4.x 则原生支持更丰富的 Windows 风格 ACL（Access Control Lists），提供更精细的权限控制（如针对单个用户或用户组的细粒度权限设置），同时显著简化了 Windows/Linux 混合环境中权限互操作性问题。

下图展示了一个典型的安全协商流程：客户端向元数据服务器发起安全策略提案（Security Proposal）；元数据服务器接受提案后，双方基于协商一致的机制完成身份验证与权限绑定。

*此为最简化的场景，实际部署中可能涉及多轮协商或动态策略调整。



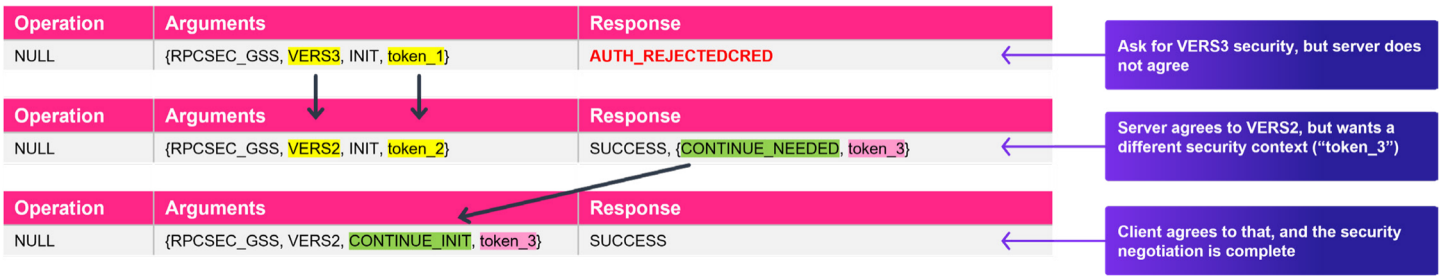
上图 - 简单的安全协商流程

而下图则展示了一个更为复杂的安全协商流程：

首次提案：客户端提出初始安全策略提案，但被元数据服务器拒绝；

二次提案：客户端调整策略后提出第二个提案，元数据服务器在满足特定条件（如加密强度、身份验证方式）的前提下接受该提案；

协商完成：客户端确认提案后，双方基于协商一致的机制完成后续操作。

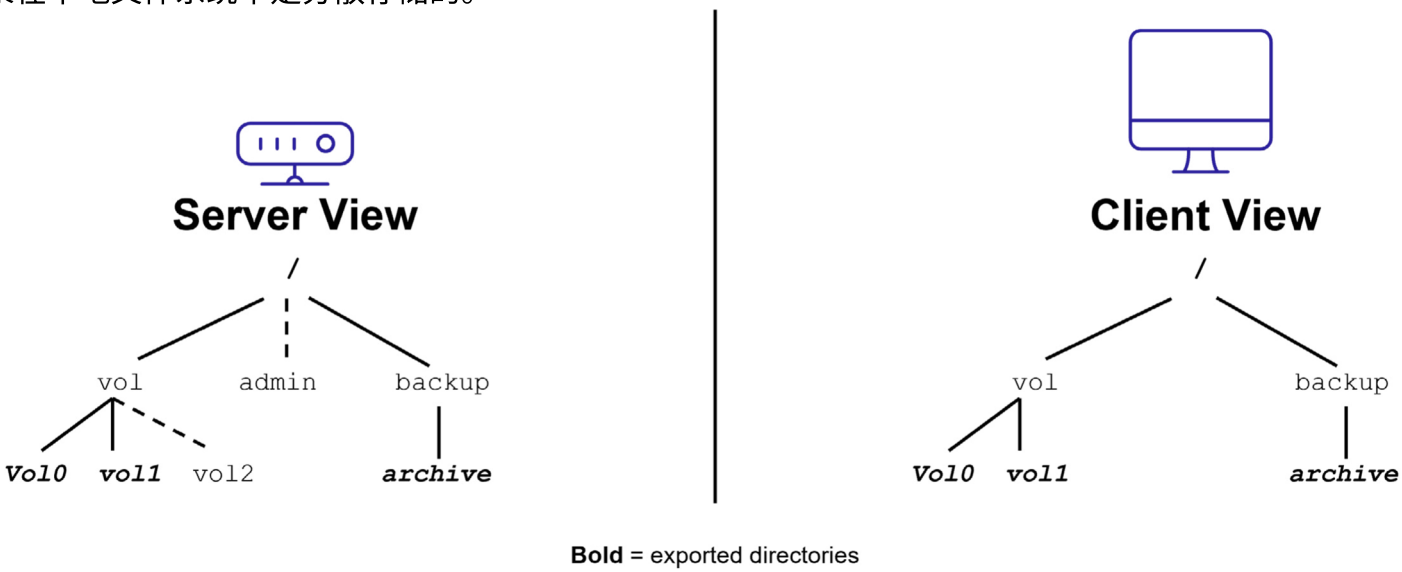


上图 – 更为复杂的安全协商流程

NFS 安全机制的完整描述涉及大量技术细节（如 Kerberos 集成、加密算法协商、权限映射规则等），已超出本文的讨论范围。然而，这一复杂的协商流程会在后台自动完成，用户在后续操作中无需干预，可将其视为默认行为（transparent to end-users）。

挂载机制（Mounting）

在 NFSv3 中，客户端挂载 NFS 导出目录（export）需依赖独立的挂载协议（Mount Protocol）。而在 NFSv4 及后续版本中，这一功能被直接集成到核心协议中，其操作也非常简洁。NFS 协议定义了一个名为 ROOTFH（Root Filehandle） 的常量，表示每个服务器提供的逻辑文件系统树的根节点。客户端可通过 LOOKUP 操作，从该根文件句柄出发，定位服务器上的任意其他文件句柄（filehandle）。若元数据服务器导出了多个文件系统，这些文件系统将被整合为一个统一的虚拟文件系统树，称为伪文件系统（pseudo-file system）。这一设计使得客户端能够无缝访问所有导出目录，即使这些目录在本地文件系统中是分散存储的。



上图 -伪文件系统（pseudo-file system）示意

关于本文档中的协议表格结构说明：

- 每张图表表示客户端与元数据服务器之间**一次完整的往返通信（round trip）**。
- 若图表标题行（粉红色）下方显示多个操作，则表示这些操作属于一个 **复合操作（compound operation）**，即多个协议请求/响应合并处理以提升效率

表格列定义：

- **Operation（操作）列**：包含 NFS 客户端发起的请求，相关参数可见**Arguments（参数）列**
- **Response（响应）列**：包含服务器返回的响应结果

颜色编码规则：

- **粉色标题表格**：表示 **pNFSv4.2 元数据通信**（客户端与 pNFS 元数据服务器之间的交互）
- **蓝色标题表格**：表示 **NFSv3 数据通信**（客户端与 pNFS 存储服务器之间的数据传输）

依赖关系标识：

- **黑色粗箭头**：表示操作间的依赖关系：
- 从请求指向对应的响应（请求 → 响应）
- 或从响应指向后续的请求（响应 → 新请求）

***注释说明**：表格右侧的 **注释框**（Comment boxes）会用通俗易懂的语言解释技术细节，帮助读者理解协议交互的逻辑。

完成身份认证后，客户端必须获取文件系统根目录的实际文件句柄（filehandle），如下图所示：PUTROOTFH 操作将当前文件句柄设置为根文件句柄（root filehandle）；随后的 GETFH 操作返回当前文件句柄的值。这一操作序列即可“隐形”完成服务器整个导出目录树（export tree）的挂载。

Operation	Arguments	Current FH	Response
PUTROOTFH		{rootfh}	NFS_OK
GETFH		{fh2}	NFS_OK, {fh2}

Just PUT the predefined “rootfh” constant, then pull the real value back.

Figure 8 - Get Root Filehandleto Mount

并行 NFS（Parallel NFS）

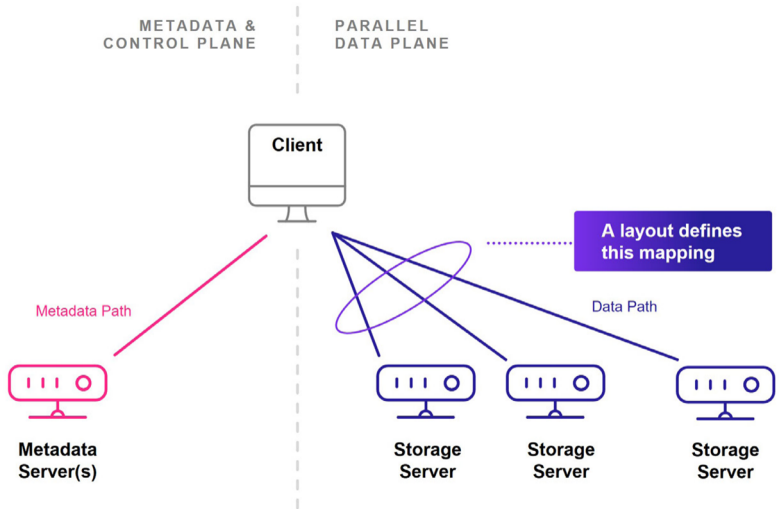
快递公司拥有不止一辆运输卡车是有原因的：他们唯有通过并行作业，才可能完成海量包裹的派送。数据传输亦是如此。每台服务器和网络连接都有性能上限，若想突破这些限制，就必须并行使用多台服务器和多条传输路径。pNFS（Parallel NFS） 正是为此而生——它能够高效利用多个服务器和多条路径，实现远超单台服务器或单一路径的数据传输速度。

尽管存在一些可独立使用或与 pNFS 结合使用的技术手段，但它们不足以最大化性能。例如：使用 NFS 的 nconnect 挂载选项实现 TCP 连接多路复用（操作相对简单）；或是通过 LACP 聚合多个物理网络接口（操作复杂，需重新配置交换机）。这些方法虽有一定帮助，但无法从根本上突破瓶颈。

pNFS 架构

许多人尚未意识到的是，pNFS 的价值远不止于实现数据的并行传输。它还实现了元数据通信路径与数据路径的分离。这使得客户端能够绕过传统“NAS 头节点”等中间代理服务器，直接与存储服务器通信，进行数据的读写操作。如果你熟悉 Lustre 等高性能并行文件系统，这一设计应当并不陌生——它们正是通过分离元数据与数据路径，并允许客户端并行直连存储服务器，才实现了惊人的高吞吐性能。这是一种经过验证的架构。

然而，在 pNFS 出现之前，实现此类架构意味着必须面对复杂且难以维护的专用软件，并在每个需要访问的系统上安装定制化客户端。如今，这一切已成为历史。pNFS将同样高性能的架构，以标准化的方式集成于主流Linux系统之中，无需额外部署专有软件即可使用。



右图 - pNFS 架构

布局（Layouts）

在客户端能够直接与存储服务器通信之前，需要满足两个条件：1. 了解如何访问存储服务器；2. 获得在存储服务器上执行操作的权限（因为这是一个共享文件系统）。这两项信息由元数据服务器通过名为布局（Layouts） 的数据结构提供。

pNFS 的布局设计初衷

从诞生之初，pNFS 就旨在支持多种存储后端类型，包括块存储（Block Storage）、文件存储（File Storage） 和对象存储（Object Storage）。每种存储类型对应不同的布局定义，为 pNFS 客户端提供与该类型存储交互所需的特定信息。规范（Specification）还预留了未来扩展性，允许新增布局类型以适应新的存储技术。

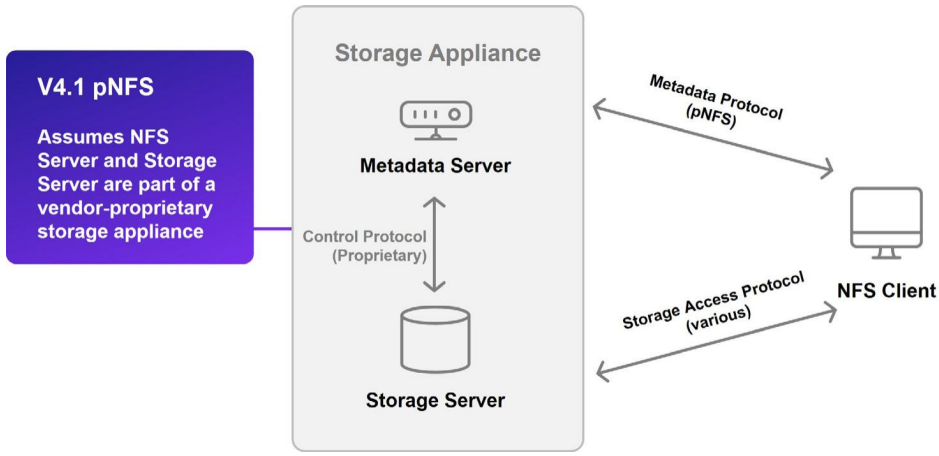
布局的作用：分段授权与条带化

布局是针对特定字节范围（Byte Range） 进行的，而非总是针对整个文件。从而形成的优势包括：允许多个客户端同时操作同一文件的不同部分（并行处理）；支持条带化（Striping），即客户端可将文件分段写入多个存储目标（如多个存储服务器），提升性能。

pNFS 系统的三大组件如右图所示包含：

客户端（Clients）：存储服务的使用者。元数据服务器（Metadata Servers）：处理所有元数据，并协调客户端行为。存储服务器（Storage Servers）：存储数据的物理设备（规范中称其为数据服务器）。

这三类组件之间需要两两通信，通信过程依赖多种协议：

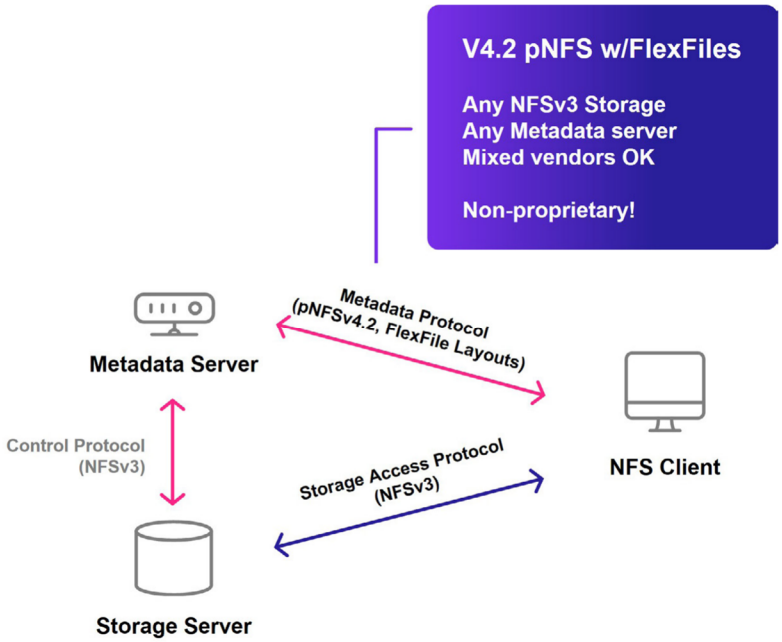


上图– 以往pNFSv4.1时期的通信协议

客户端与元数据服务器：通过 pNFS 协议交互。客户端与存储服务器：根据存储类型，可能使用 NFS、块协议（如 iSCSI）或对象协议（如 S3）。元数据服务器与存储服务器：pNFSv4.1 规范未定义此控制协议（Control Protocol）。

这一设计看似奇怪，实则源于一个假设：传统 NAS 系统中，元数据服务器和存储服务器通常集成在同一个存储设备（如 NAS 头节点 + 后端存储）中。此场景下，控制协议仅在设备内部使用，无需对外暴露，因此无需标准化。这一设计的连锁影响包括：厂商锁定（Vendor Lock-in）——存储厂商需自行定义控制协议，导致不同厂商的元数据服务器（NAS头节点）无法与彼此的存储服务器兼容。客户被迫从同一厂商购买元数据服务器（NAS头节点）和存储服务器，无法混用不同厂商的硬件。而存储设备厂商也偏好这种绑定销售模式，而客户也习惯了这种“打包购买”的存储采购方式。

灵活文件布局（Flexible File Layout）



上图 - pNFSv4.2时期的灵活文件布局（Flexible File Layout）通信协议

让我们重新审视这一假设：**为什么元数据服务器和存储服务器不能来自不同厂商？**实现这一点所需的是一个标准控制协议。如果该控制协议仅使用已有的工具和代码，任何通用存储服务器都可以被使用。所有独特性和创新将集中在元数据服务器内部，而存储服务器则成为标准商品化设备。灵活文件布局（又称为 FlexFile 布局或简称为FlexFiles）正是为此诞生的。其通过标准的 NFSv3 操作 定义控制协议。从鱼鹭在积灰的 NetApp 设备，到搭载 NVMe 硬盘的通用 Linux 服务器，或任何支持 NFSv3 的设备都可以成为 pNFS 存储服务器。

FlexFiles 控制协议

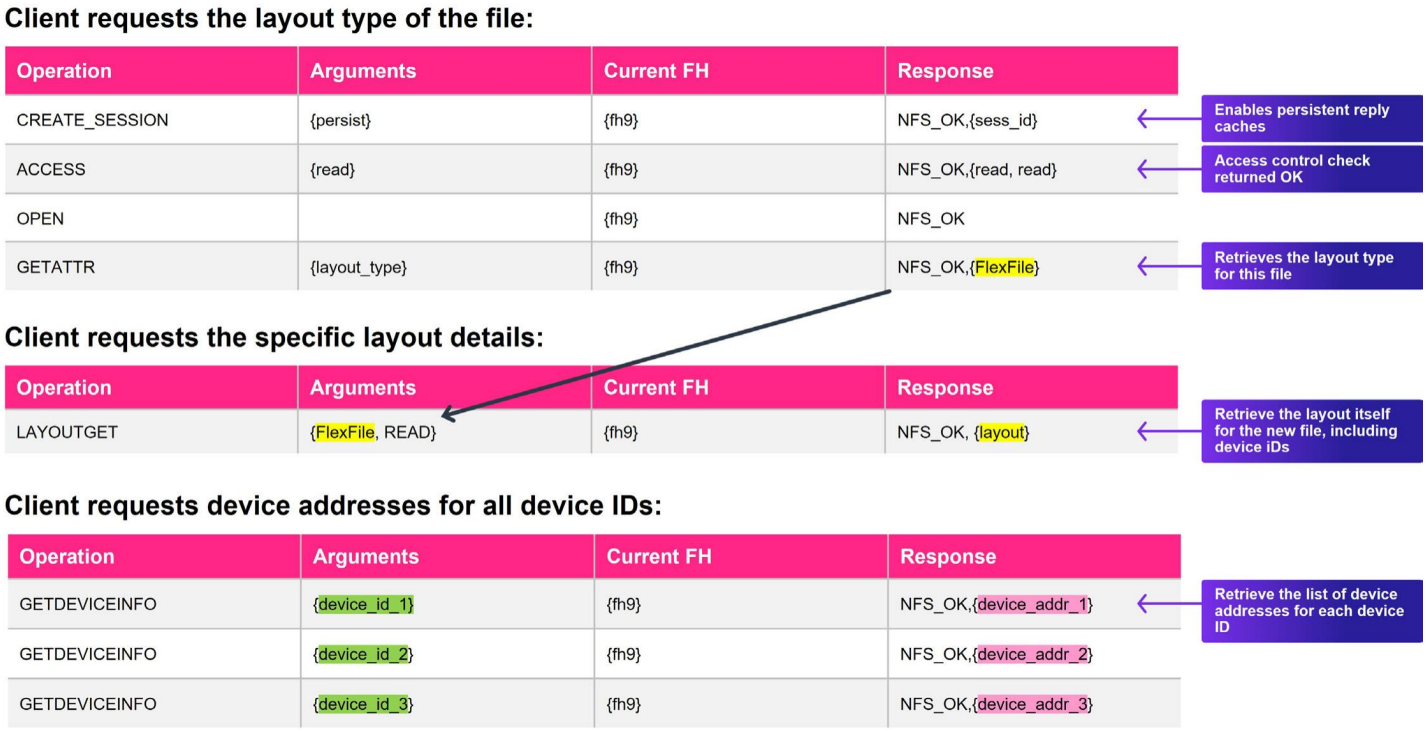
FlexFiles控制协议的核心功能是管理对存储服务器上文件的访问权限。这种机制不仅需要授权客户端访问文件，更重要的是能通过“围栏机制”（fencing）阻止客户端在非授权情况下访问数据。在FlexFiles布局中，只有pNFS元数据服务器拥有对后端NFSv3存储服务器的root权限，并完全掌控存储服务器上的文件目录。这种设计通过隔离机制将所有客户端与数据隔离，再通过合成用户标识（UID）和组标识（GID）的创新机制动态授予客户端访问权限。文件的UID拥有者具备读写权限，GID拥有者则仅限只读。当元数据服务器向客户端分配布局时，会将对应的UID/GID作为RPC凭证嵌入布局中。当需要撤销访问权限时，元数据服务器仅需修改文件的合成UID/GID，即可自动使客户端持有的凭证失效。客户端检测到访问被拒绝后，会主动归还布局并重新向元数据服务器申请新布局，此时元数据服务器将根据当前权限状态分配新的凭证。

需要特别说明的是，上述UID/GID的动态管理机制与终端用户挂载文件系统时的访问权限控制是相互独立的——后者通过pNFS的元数据路径单独处理。这种设计本质上是通过NFSv3已有的数据通道机制，在通用硬件和第三方存储基础上，构建出一个共享式并行文件系统。其创新之处在于：既利用了NFSv3协议的广泛兼容性，又通过元数据服务器对访问凭证的动态控制实现了存储资源的共享与隔离。这种方案使得用户可以灵活组合不同厂商的存储设备，同时将系统核心创新集中在元数据管理层面，而存储服务器则降级为一种标准化组件。

pNFS 读写 I/O 示例

让我们通过一个简单的例子来了解如何使用 pNFS 和 FlexFile 布局打开文件进行读取。下方图表代表一个复合操作（Compound Operation），即客户端与服务器之间的一次往返通信。蓝色表格表示客户端与元数据服务器（例如 Hammerspace Anvil 节点）之间的 NFSv4.2 对话，绿色表格表示客户端与存储服务器之间的 NFSv3 对话。

在下图所示的第一个复合操作中，客户端会创建会话、验证访问权限、请求打开文件，并获取该文件的布局类型。一旦客户端收到布局类型后，即可在第二个复合操作中请求具体的布局信息。布局中包含的内容之一是设备 ID 列表。当客户端获取到设备 ID 后，会请求之前未见过的设备 ID 对应的设备地址（这些地址会被缓存起来）。通过这些地址，客户端便具备了直接与存储服务器通信所需的条件，这一过程发生在第四个也是最后一个复合操作中。



上图 - 使用 FlexFiles 布局的 pNFSv4.2 协议打开文件进行读取

写操作的流程与读操作类似，如下页中的表格所示。在第一个复合操作中，客户端提交对文件的打开请求，并进一步指定仅在文件尚不存在时才创建该文件。同时，客户端会向服务器提供一个“布局提示”（Layout Hint），表明希望接收的布局类型——本例中为FlexFile布局。最后，客户端请求服务器实际分配的文件句柄（Filehandle）和布局类型，服务器将返回这些信息。

随后，客户端利用返回的文件句柄和布局类型，请求具体的布局信息。接着，客户端会获取布局中所有未交互过的设备ID对应的设备地址。最终，在最后一个复合操作中，客户端将数据直接写入存储服务器。

Create the file while asking for a layout:

Operation	Arguments	Current FH	Response
PUTFH	{fh6}	{fh6}	NFS_OK
OPEN	{fh6, "file_1", guarded, layout_hint}	{fh7}	NFS_OK
GETFH		{fh7}	NFS_OK, {fh7}
GETATTR	{layout_type}	{fh7}	NFS_OK,{FlexFile}

Filehandle for the directory

"guarded" means create the file if-and-only-if it does not yet exist

Hint to the server on the type of layout desired for this file, eg: mirrored

Get the details of the assigned layout:

Operation	Arguments	Current FH	Response
LAYOUTGET	{FlexFile, RW}	{fh7}	NFS_OK, {layout}

Retrieve the layout type assigned to the file by the server

Client requests device addresses for any device IDs it hasn't seen before:

Operation	Arguments	Current FH	Response
GETDEVICEINFO	{device_id_1}	{fh7}	NFS_OK,{device_addr_1}
GETDEVICEINFO	{device_id_2}	{fh7}	NFS_OK,{device_addr_2}

Retrieve the layout itself for the new file, including device iDs

Retrieve the list of device addresses for each device ID

Write data directly to storage server:

Operation	Arguments	Current FH	Response
WRITE	{0, 1MB}	{fh91}	NFS_OK

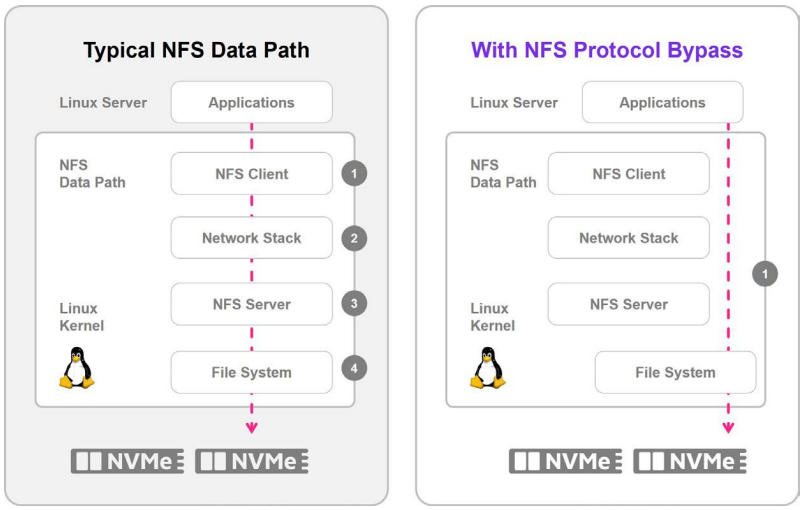
Write the first 1MB

上图 -使用 pNFSv4.2 和 Flex Files 创建并写入文件

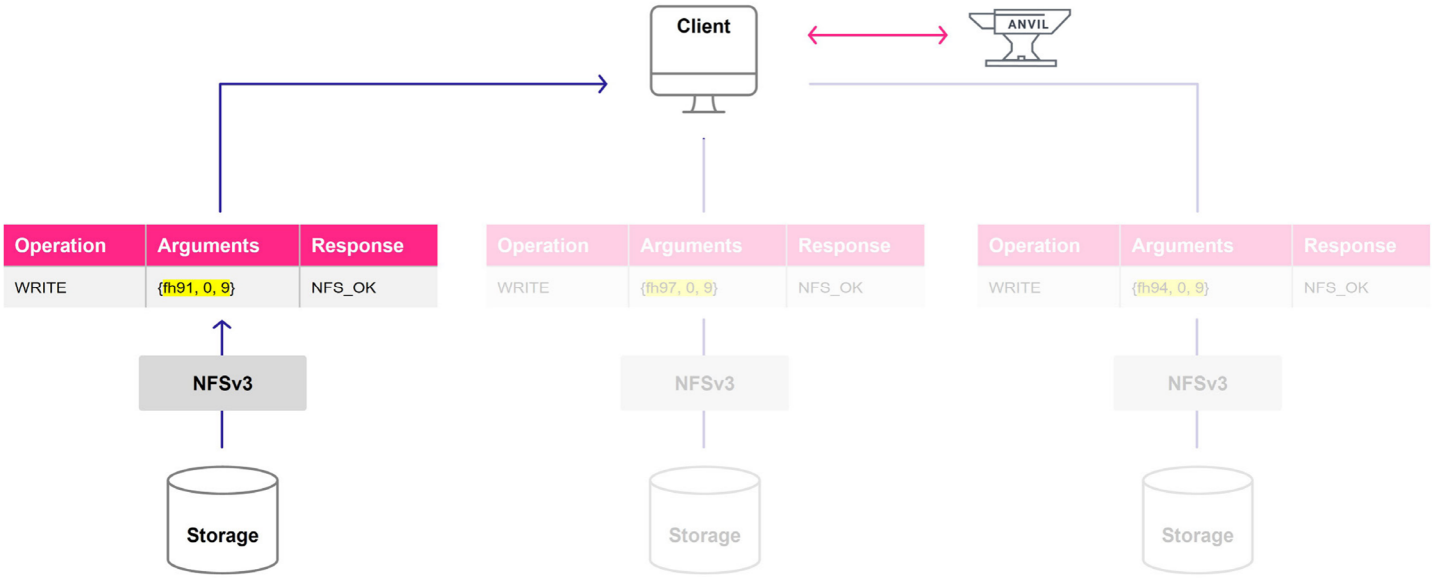
使用客户端镜像、条带化和纠删码的 pNFS

RFC8435（定义 FlexFiles 布局的标准）引入了客户端镜像（Client-Side Mirroring）功能。传统镜像由服务器负责创建、验证和维护数据副本，而客户端镜像则通过元数据服务器（如 Hammerspace 的 Anvil）定义镜像数量和位置，并在布局中提供可用镜像列表。客户端读取时可从列表中选择任意镜像（当前 Linux NFS 客户端默认使用第一个），Hammerspace 则通过动态调整镜像顺序实现负载均衡。

未来，Hammerspace 将进一步优化：将与客户端位于同一节点的本地硬盘置于镜像列表顶部，以发挥 **LOCALIO**（NFS 协议旁路）优势。该技术允许 NFS 客户端与服务器运行在同一主机时，**直接访问本地硬盘**，绕过网络和 NFS 协议栈，从而显著降低延迟。这种优化通过消除不必要的网络"跳数"提升性能，适用于本地硬盘访问场景（如 Hammerspace 存储节点）。

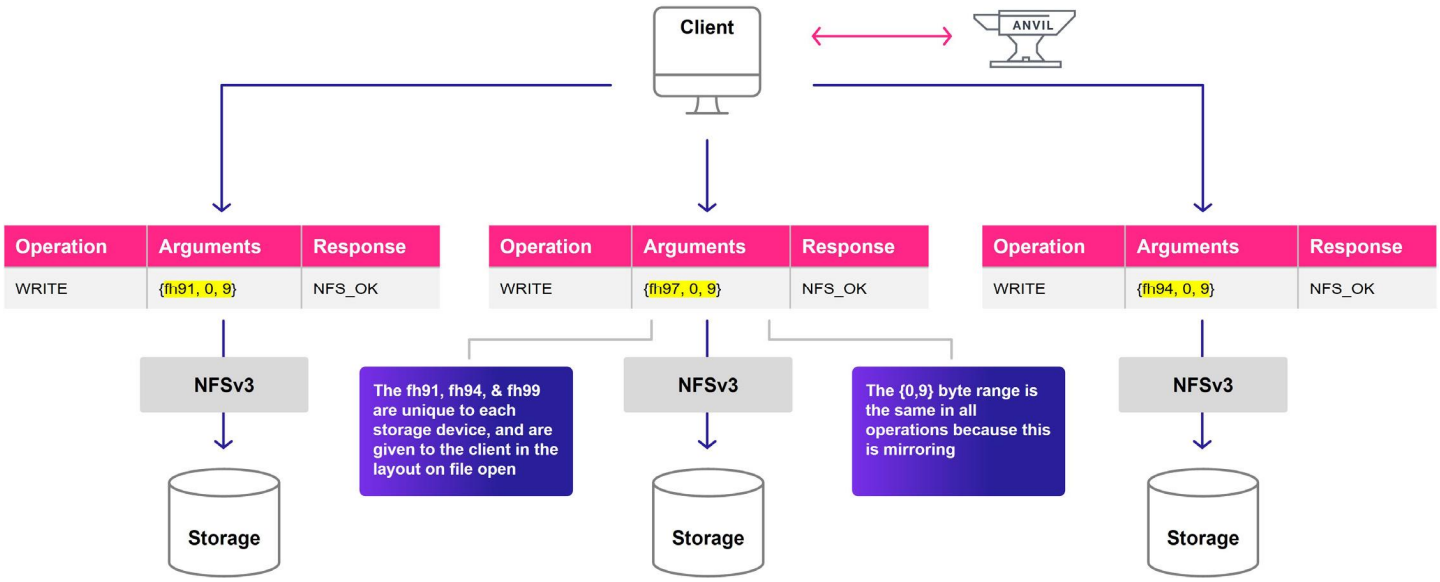


上图 - NFS 协议旁路（又称 LOCALIO）



上图 – 客户端镜像从三重镜像文件的读取

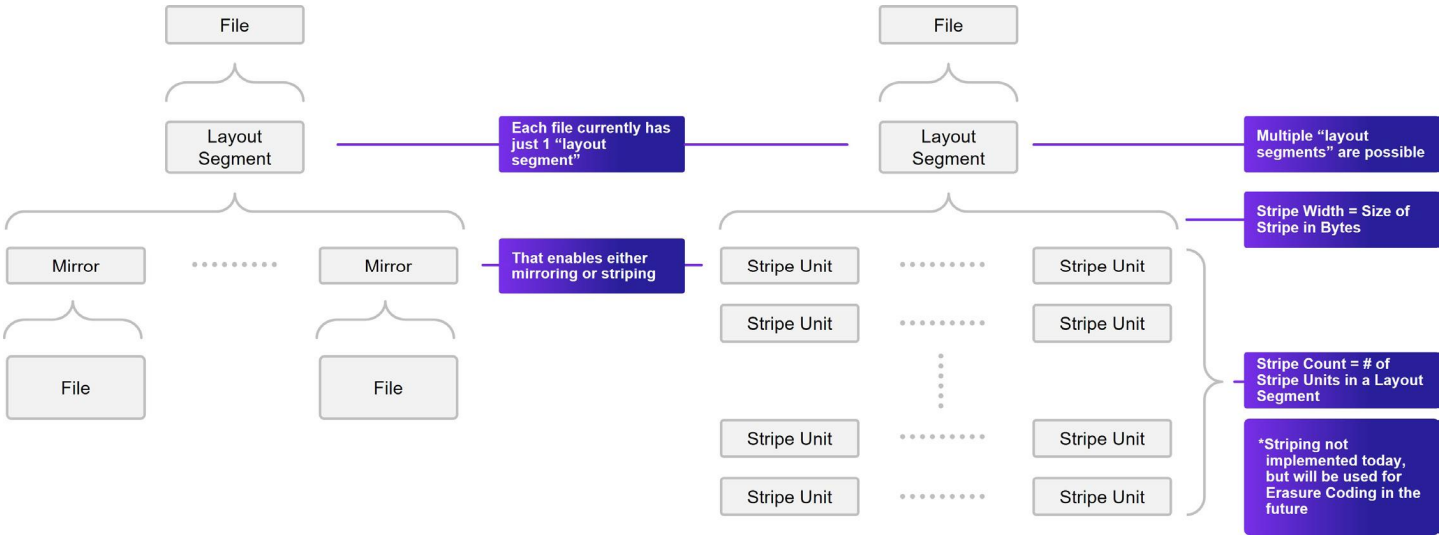
对于写操作来说，客户端需将数据同时写入所有镜像副本。只有在所有镜像均成功写入后，该写操作才会被标记为完成。



上图 – 客户端镜像对三重镜像文件的写入

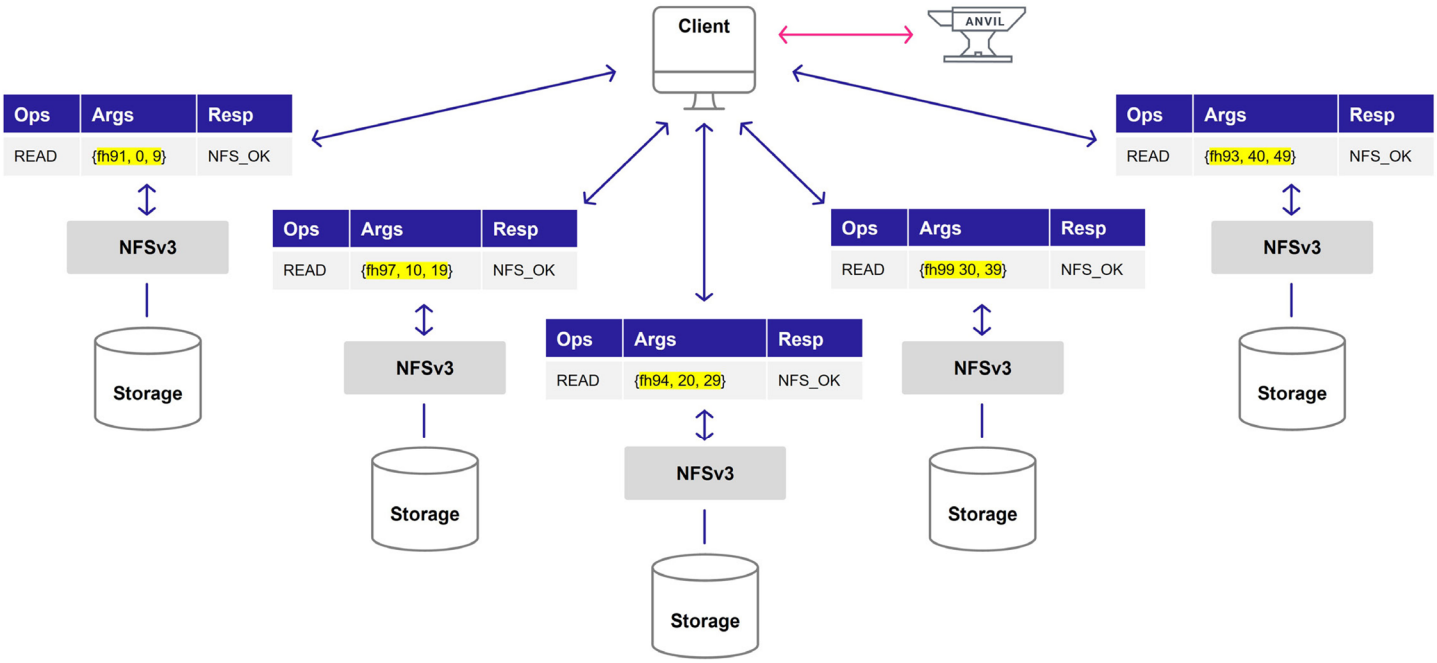
若客户端在写入一个或多个镜像时遇到错误，会通过 LAYOUT ERROR 操作（详见下一节）向元数据服务器上报错误。随后，元数据服务器负责修复文件的镜像副本，这一过程称为 “resilvering”（镜像重建）。

由于客户端镜像的性能可随客户端数量线性扩展，并避免了元数据服务器成为瓶颈，其在大型或高负载存储环境中的性能显著优于传统服务器端镜像方案。



上图 - 客户端镜像与条带化布局

除了客户端镜像外，协议标准还包含客户端条带化（Client-Side Striping） 的规范。条带化（Striping）具有多重优势，包括提升大文件性能：通过将文件并行条带化分布到多个存储服务器，可显著提高大文件的读写效率；高级数据保护：支持客户端使用纠删码（Erasure Coding） 替代简单的条带化，将数据分片并冗余编码后分布到多个存储服务器，从而实现更高效的数据保护机制。



上图 - 跨越五台存储服务器的客户端条带化读写

当前，Linux NFS 客户端（内核）尚未实现客户端条带化（Client-Side Striping）和纠删码（Erasure Coding）功能，但 Hammerspace 正在积极推进这些特性的融入，并计划在未来的版本中将其贡献给开源社区。

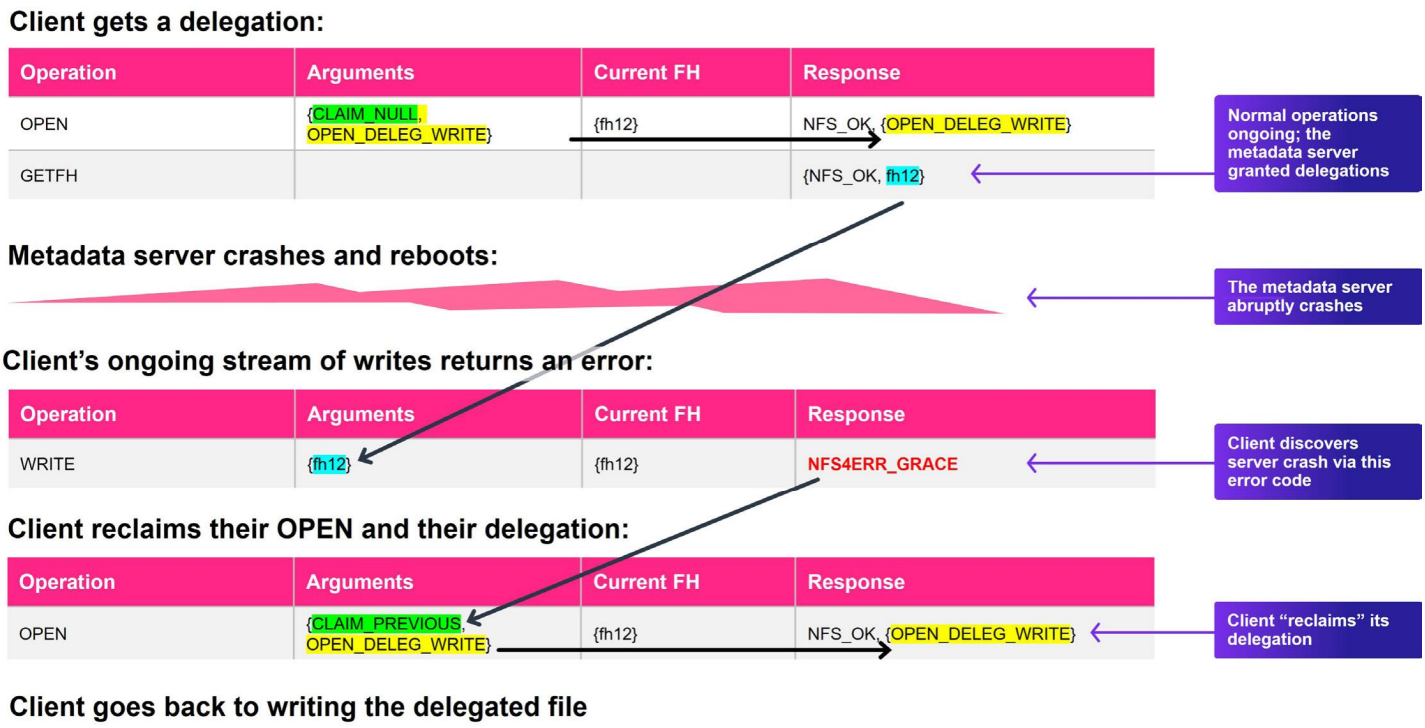
错误恢复与特殊操作

在任何数据处理系统中，防止数据丢失始终是最高优先级的事项。各层级的技术与协议必须具备足够完善的机制，既能处理正常运行状态，也能应对各类故障场景。对于NFS而言，这意味着需要优雅地恢复客户端与服务器间的通信中断（包括系统崩溃/重启事件）。

NFS客户端与服务端崩溃恢复

如前所述，布局（Layout）机制内置了客户端故障处理能力。例如，当客户端失联且其租约（Lease）到期时，元数据服务器可按需召回或撤销布局。此举可避免因客户端挂起或消失导致的锁滞留问题。每次客户端启动时，它会从服务器获取一个新的唯一客户端ID。若客户端崩溃后重启并丢失状态，它将获得新客户端ID，而元数据服务器将释放与旧ID关联的所有锁。

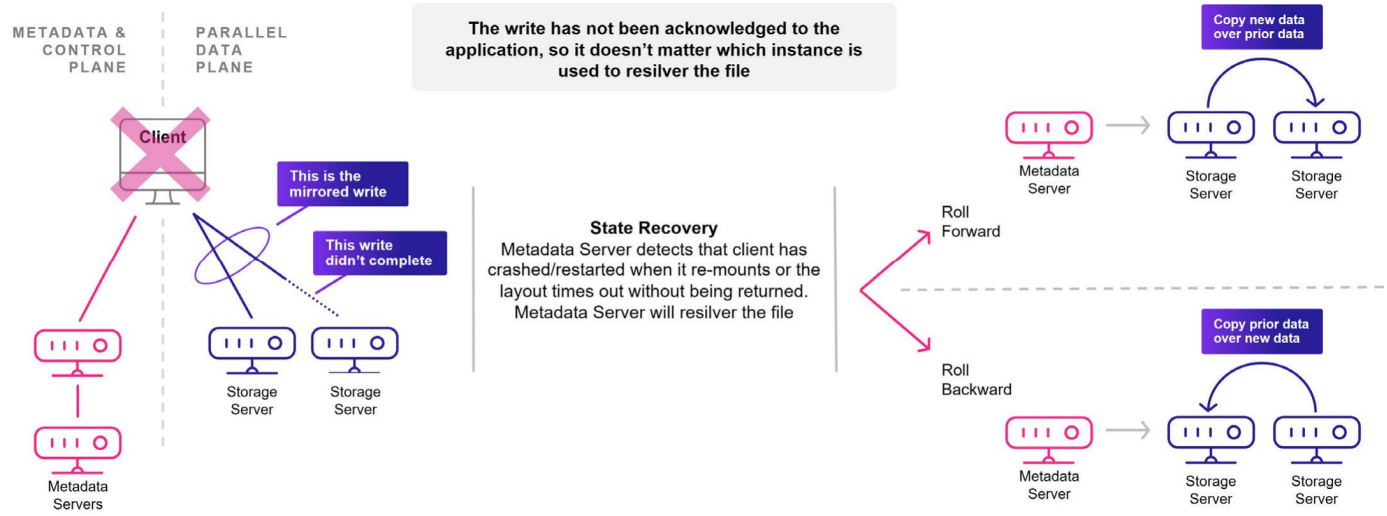
NFS还支持客户端在元数据服务器崩溃/重启后的状态恢复：在特定的宽限期（Grace Period）内，服务器将拒绝某些非恢复类请求（例如对已打开文件的RENAME或REMOVE操作），并返回NFS4ERR_GRACE错误；但允许执行安全操作（如READ）。正常操作将在所有客户端恢复其锁状态后恢复（通常仅需数秒），或当完整的45秒宽限期结束后自动恢复。



上图 - 元数据服务器崩溃/重启后的状态恢复

pNFS 镜像或条带化写入的恢复

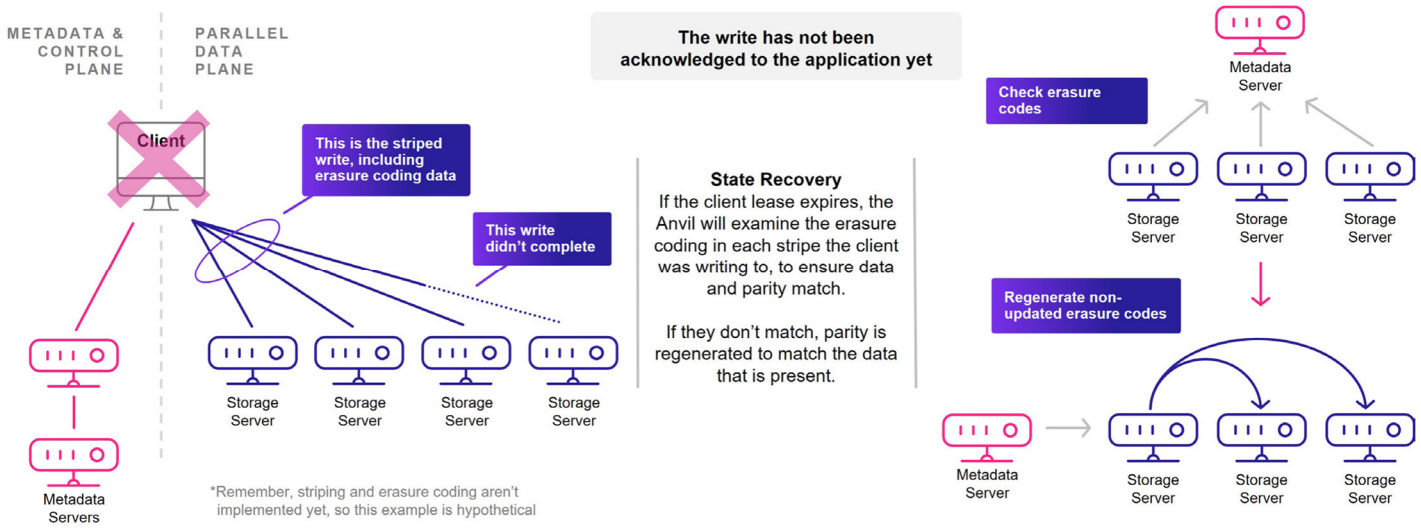
镜像和条带化会在恢复过程中存在特殊问题。在 pNFS 中，客户端需负责写入所有镜像和条带。然而若客户端发生故障，导致部分镜像或条带未被成功写入，必须进行恢复操作以确保所有镜像和条带的一致性。这一恢复过程将由元数据服务器完成。



上图 -客户端崩溃后的镜像写恢复

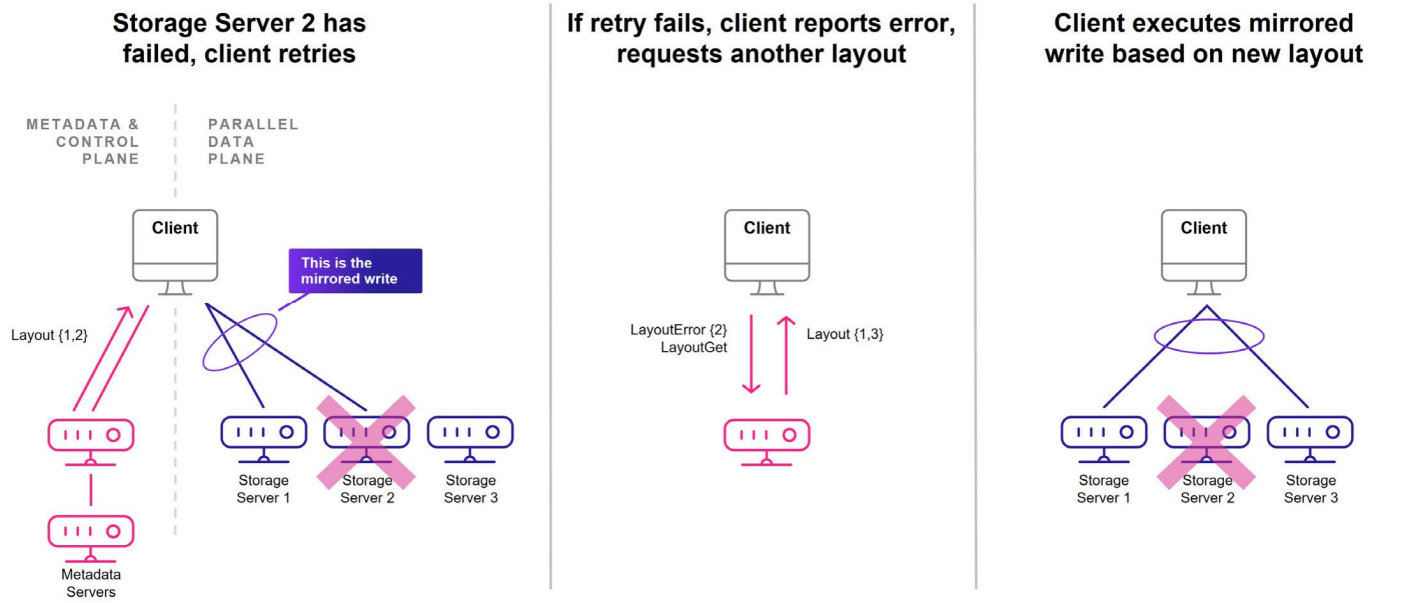
镜像恢复相过程对简单。由于客户端在写入过程中发生故障，应用程序将无法收到写入成功的确认。元数据服务器会在客户端重启并重新挂载共享目录，或其租约过期时检测到客户端崩溃。随后，元数据服务器需确保所有镜像一致，方法是完成剩余镜像的更新，或回滚客户端已修改的内容。这一过程称为对镜像进行的“镜像重建（resilvering）”。

而对于条带化写入操作，情况则更为复杂，相关细节仍在讨论之中。当使用纠删码时，如果客户端已成功写入足够的数据，元数据服务器可能已经具备完成写入操作的能力。这一示例（假设性场景，因客户端侧纠删码尚未实现）如下图所示。



上图 - 客户端崩溃后的条带化写恢复

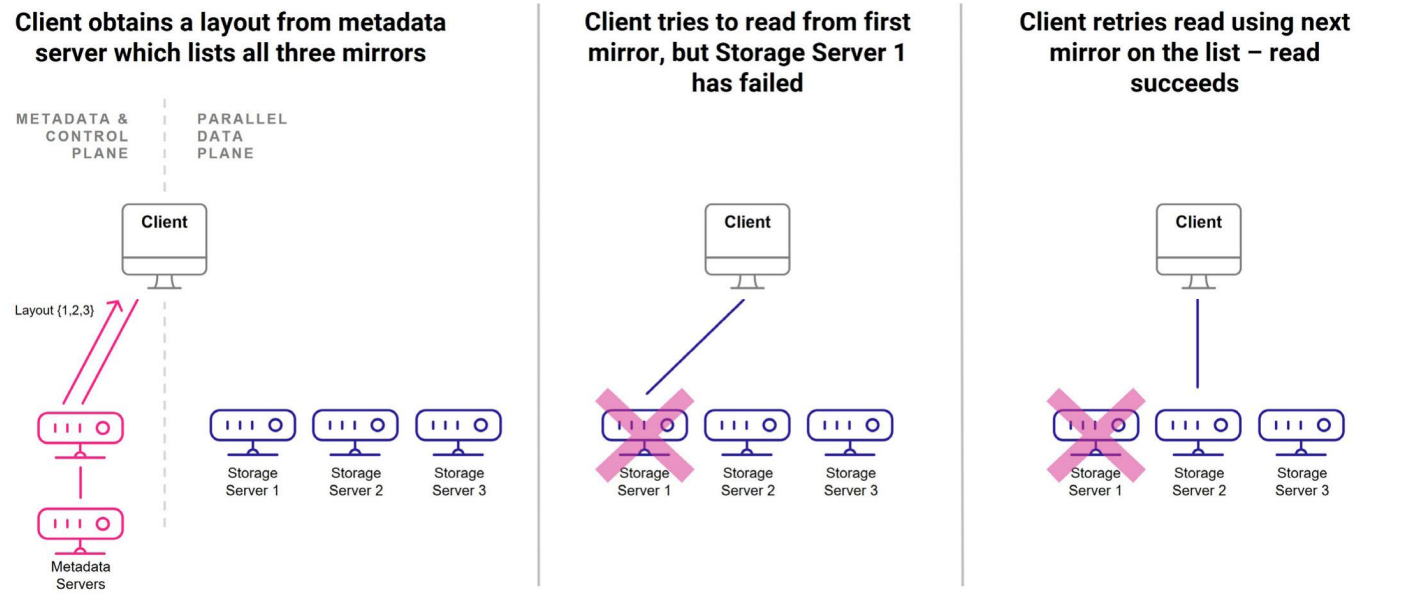
若在镜像写入过程中存储服务器发生故障，客户端通常能够透明地恢复并完成写入，而应用程序并不感知。如果重试失败，客户端会通过 LAYOUTERROR（下节详述）向元数据服务器报告错误，请求新布局，并根据新布局完成写入。



上图 -镜像写入期间存储服务器故障的客户端恢复

pNFS 镜像读取过程中存储服务器故障的恢复机制与此类似，但更简单。请注意，客户端收到的布局会包含所有镜像的列表，但只需一个镜像即可读取数据。

客户端会首先尝试从列表中的第一个镜像读取。如果该存储服务器不可用，客户端将转向布局中的下一个存储服务器进行读取。



上图 - 镜像读取期间存储服务器故障的客户端恢复

pNFSv4.2存储状态报告

在 pNFS 中，客户端会直接与存储服务器进行通信，这可能导致通信盲区的问题。元数据服务器与存储服务器通过控制协议通信，因此元数据服务器只能从自身视角检测存储服务器的部分问题。对于客户端在 I/O 操作中与存储服务器交互时遇到的问题，元数据服务器无法感知，且 pNFSv4.1 的原始定义中，未提供客户端可上报存储服务器通信问题的机制。

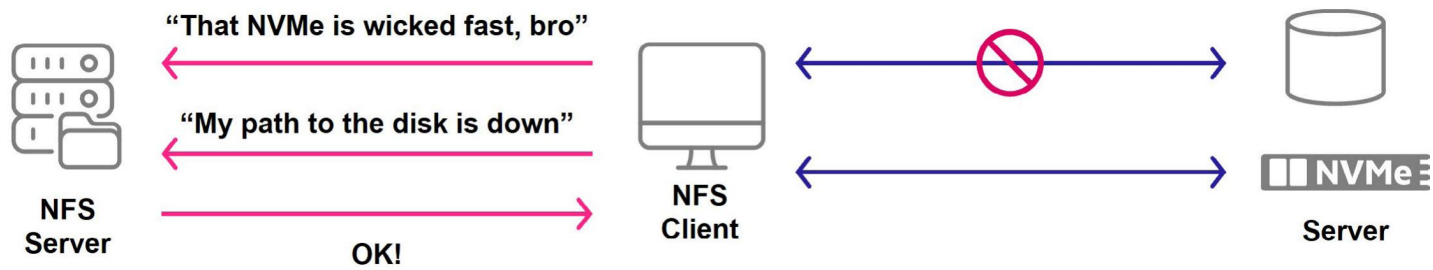
例如，若客户端与存储服务器之间的网络路径中断，而元数据服务器通过另一条网络路径仍能与该存储服务器通信，则无法检测到此问题。在 pNFSv4.1 中，客户端无法将此类问题反馈给元数据服务器。

pNFSv4.2 引入了新的 LAYOUTERROR 操作，使客户端能向元数据服务器上报与存储服务器交互中的问题。LAYOUTERROR 的设计初衷是仅报告持久性的错误——客户端仍需自行处理临时性错误。元数据服务器可根据 LAYOUTERROR 消息中的信息调整行为，例如避免下发包含客户端无法访问的存储目标的布局。

这一机制已能满足基本需求，但如果客户端不仅能上报错误，还能够报告与存储服务器交互的性能特征，岂不是一个更优的机制？若元数据服务器能理解客户端感知到的存储性能（包括网络性能），则在下发布局时可做出智能决策，例如优先选择高性能存储目标，或在多个存储服务器间实现负载均衡。

所以，pNFSv4.2 引入另一个新操作 LAYOUTSTATS ——以实现此类功能。LAYOUTSTATS 允许客户端将特定布局的性能信息反馈给元数据服务器，后者可聚合这些数据并据此调整策略。

综上，LAYOUTSTATS 与 LAYOUTERROR 提供了关键的遥测数据，使 pNFSv4.2 元数据服务器能够智能绕过故障并优化性能。此能力在早期 NFS 版本中完全无法实现。



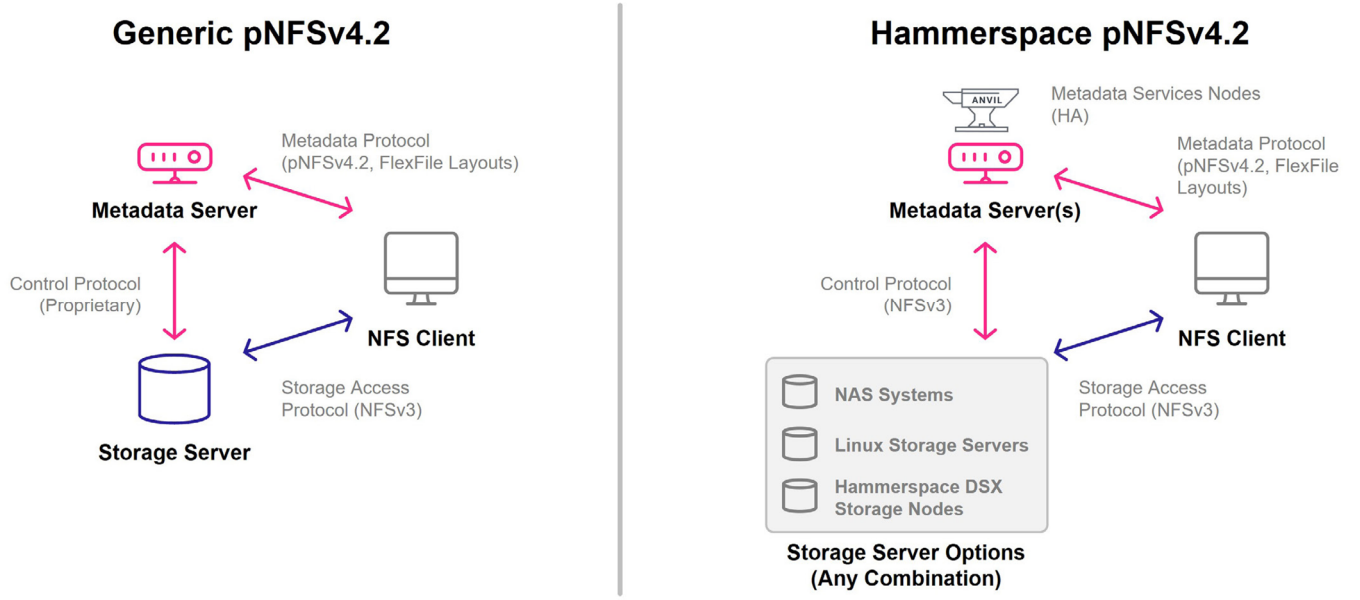
上图 - LAYOUTERROR 与 LAYOUTSTATS 操作

Hammerspace与pNFS

本文的设计初衷是聚焦 NFS 协议，旨在向读者介绍当前的前沿技术。但这很可能让你产生疑问：“Hammerspace 在其中扮演什么角色？”

在 pNFSv4.2 架构中，Hammerspace 提供元数据服务器角色的软件，并可选地提供部分或全部存储服务器的软件，运行于客户提供的 x86 服务器硬件上。Hammerspace 包含两种节点类型，分别称为 Anvils（元数据节点）和 DSX（Data Services eXtension，数据服务扩展）节点。

在 pNFSv4.2 中，Anvils 节点充当元数据服务器。DSX 节点负责前端数据访问协议，并承担数据移动任务。DSX 节点还可通过连接块存储设备（如 HDD 或 NVMe 闪存），作为存储服务器（包括 pNFSv4.2 系统中的存储服务器）运行。



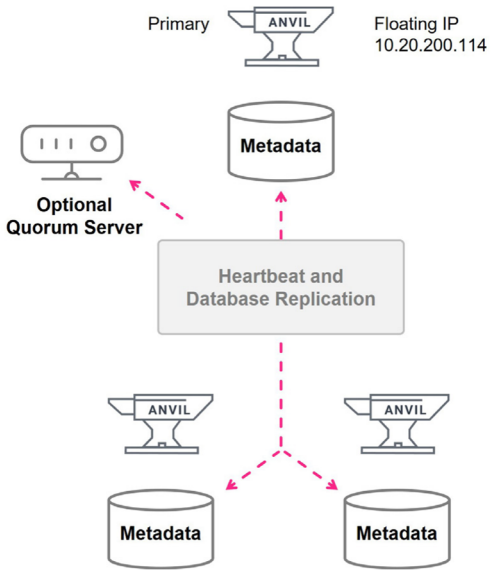
上图 – 通用 pNFSv4.2 vs. Hammerspace实现的pNFSv4.2

Anvil 集群

Hammerspace 中的所有元数据（包括 pNFS 元数据）均由一对 Anvil 节点集群处理。客户端通过一个始终指向主 Anvil 节点的浮动 IP 与 Anvil 集群进行通信。

所有 Anvil 节点之间同步复制元数据数据库，并通过心跳机制监测集群健康状态。

Anvil 集群最多可包含三个 Anvil 节点和一个可选的仲裁服务器（quorum server）。在 Anvil 节点故障切换时，仲裁服务器提供额外的“投票权”，以确保集群一致性。单个仲裁服务器可为多个 Hammerspace 集群提供服务。



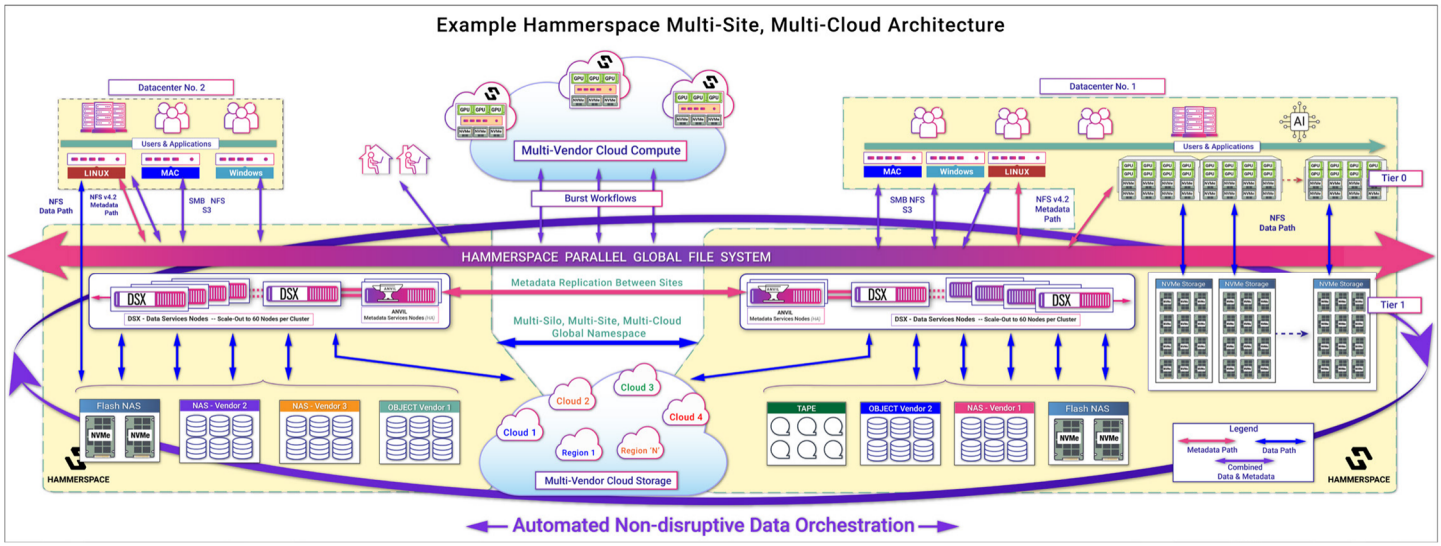
上图 - Hammerspace Anvil集群

Hammerspace 基于pNFS构建

Hammerspace 拥有远超基础 pNFS 功能的多项能力，其中许多功能基于 pNFSv4.2 的架构特性实现。具体包括：

- **通过 pNFSv4.2、NFSv3/v4、SMB、S3 以及容器使用的CSI，同时（或独立）导出共享资源**
- **为任意厂商的存储提供统一的控制与管理平面**
- **通过快照、克隆、版本控制、WORM（一次写入多次读取）及复制（包括镜像与纠删码）实现数据保护**
- **支持块存储、文件存储与对象存储的后端架构**
- **原地融合现有存储数据——无需繁琐迁移，数据无需迁移**
- **透明且无中断的实时数据迁移**
- **支持丰富的标签与元数据功能，驱动工作流与报告生成**
- **基于目标（Objective）驱动下的多项操作，实现自动化的数据编排、保护、分层及其他数据放置任务，支持工作流自动化**
- **构建真正的多协议全局命名空间，覆盖多站点与多云环境，支持云爆发（cloud-bursting）与混合云场景**

这一丰富的功能集使 Hammerspace 成为了完备的企业级数据平台，以企业友好的方式提供基于标准的并行文件系统性能。如需了解 Hammerspace 更完整的能力，请参阅 Hammerspace.com 上的技术白皮书。



上图 – Hammerspace的 架构概览

结论

大多数IT专业人员——即使是在存储领域经验丰富的——对NFS仍持有过时的看法。尽管已有更新的技术对协议进行了变革和使其现代化，NFSv3仍是目前部署最广泛的版本。

pNFSv4.2 将 NFS 转变为了一种并行文件系统，其性能与扩展性可线性增长至以往需复杂 HPC 文件系统（需定制客户端及厂商锁定的专有硬件）才能实现的水平。pNFS 提供与这些系统相同的高性能架构（元数据与数据路径分离），但通过通用硬件及现代 Linux 发行版自带的标准客户端和服务器软件即可实现。复合操作与委托机制等创新改进了性能，同时消除了 NFS 曾以“冗长通信”著称的缺陷。

pNFS 的 Flexible Files 布局类型允许任何支持 NFSv3 的存储（从现有 NAS 设备到运行通用硬件的 Linux 存储服务器）加入 pNFS 集群。

客户端镜像（以及即将实现的条带化与纠删码技术）通过让客户端承担更多数据处理任务，进一步提升了扩展性。

客户端与元数据服务器之间的遥测反馈循环为系统注入实时智能，在大规模复杂环境中显著增强了容错能力。

Hammerspace 基于这些标准构建，提供全套数据服务，包括跨站点与云环境的数据编排与多协议全局命名空间能力。

通过 Hammerspace 部署 pNFSv4.2，可实现面向高性能计算（HPC）场景的性能与扩展性，同时保留企业所需的易用性与丰富的数据管理功能。

参考资料与资源

了解 NFS 协议深层细节的最佳方式是阅读定义该协议的标准文档。这些文档被称为 RFC（征求意见稿），由互联网工程任务组（IETF）通过正式化流程进行的创建和维护。以下是 NFS 的主要 RFC 文档。请注意，Hammerspace 深度参与了标准制定流程，协助定义新功能，并推动 NFS 向更强大、更可扩展的方向发展。

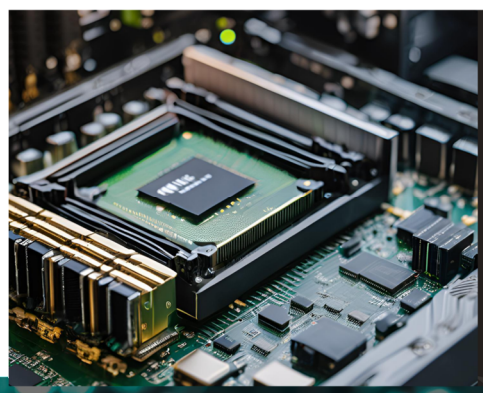
除了撰写 RFC 文档外，Hammerspace 团队的多位成员还担任 Linux 内核维护者，致力于确保在新功能定义完成后，其支持功能被集成到标准 Linux NFS 客户端和服务器代码中。

Topic	RFC Link	Date	Notes
NFSv3 Definition	RFC1813	6/1995	Co-authored by Brian Pawlowski, now at Hammerspace
NFSv4 Definition	RFC7530	3/2015	Introduction of Compounds & Delegations, enhanced Locking Co-authored by Tom Haynes, now at Hammerspace
NFSv4.1 Definition	RFC8881	8/2020	Update of NFSv4.1, including pNFS (which was introduced in the original NFSv4.1 definition, RFC5661, in 1/2010)
NFSv4.2 Definition	RFC7862	11/2016	Authored by Tom Haynes, now at Hammerspace
Flexible Files Layout Definition	RFC8435	8/2018	Co-authored by Tom Haynes, Hammerspace

Table 2 - Key NFS RFCs

有关 pNFS 的更多信息，可访问 **Hammerspace pNFS 资源中心**，其中包含一篇探讨如何利用 pNFS 支持 AI/DL 工作负载的白皮书。

更多白皮书、技术简报、客户案例研究及其他资源，请访问 **Hammerspace 资源页面**。



如需了解更多-可联系
ralphie.yu@hammerspace.asia
或 18810311799
手机・微信同号
HAMMERSPACE