清 华 大 学

综 合 论 文 训 练

题目：高可用分布式持久内存文件系统设计与实现

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：高　健

指导教师：舒继武　教授

2020 年 4 月 12 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____ 导师签名：_____ 日　期：_____

中文摘要

可用性是分布式文件系统的重要属性之一。近年来，持久性内存（NVM）技术出现并高速发展，给这一领域带来了新的机遇。通过合理地利用 NVM 高速、字节粒度访问的特性，系统能够以更低的开销、更高的速度提供高可用的文件存储服务。

本文提出一种面向 NVM 存储的分布式文件系统。该系统使用纠删码提供可用性保障，其性能不低于使用备份策略的传统分布式文件系统，并且在相同容错条件下显著节省所需的存储空间。测试显示。

关键词：NVM；RDMA；分布式文件系统；可用性

ABSTRACT

Availability is a key feature of distributed file systems. In recent years, the advent and rapid development of non-volatile memory (NVM) technology brings new chances to this field. By leveraging the high-performance and byte-addressable features of NVMs, the system can provide file storage service with lower cost, higher speed, and together with high availability.

In this paper, we present a new distributed file system for NVM-based storage, which uses erasure codes to guarantee its availability. It performs no worse than traditional systems using replication as their availability strategy, while using far less storage with the same fault-tolerance ability. Evaluations show that.

Keywords: NVM; RDMA; distributed file system; availability

目　录

主要符号对照表

HDD	机械硬盘（Hard Disk Drive）
SSD	固态硬盘（Solid State Drive）
DFS	分布式文件系统（Distributed File System）
NVM	持久性内存（Non-volatile Memory）
RDMA	远程直接内存访问（Remote Direct Memory Access）

第 1 章　引言

1.1　研究背景

文件是用于数据处理的最基本的抽象之一。日益增长的互联网规模正带来日益增长的文件存储需求；单机文件系统早已无法应对如此巨大的数据量，分布式文件系统（Distributed File System，DFS）应运而生。相比于单机系统，DFS 具有高性能、高并发度和良好的可扩展性，已经成为近年来的热点研究方向。

分布式文件系统通常包含三个最主要的部分：存储、计算和网络通信，系统的性能由这三者共同决定。由于它们各自的性能互不相同，平衡三者的性能从而优化整个系统成为了近年来分布式文件系统方向研究的主流方向。另外，组成分布式系统的单机彼此之间在物理上分离。在共同提供单机所无法比拟的数据处理能力的同时，它们也可能各自独立地发生故障，从而使得整个系统发生故障的概率大大提高。因此，分布式文件系统通常必须引入额外的存储、计算和网络通信，从而提供可用性保障，即是说，系统在某些节点故障时应该仍然能正常工作。如何在三者性能损失和系统可用性之间权衡，是 DFS 设计中另一个必须考虑的问题。

传统分布式文件系统通常使用机械式磁盘（Hard Disk Drive，HDD）或固态硬盘（Solid State Drive，SSD）作为其持久性存储介质。它们的优点是容量较大：如 Seagate 公司已经生产出容量达 16TB 的商用 HDD，SanDisk 公司也已制造出容量达 8TB 的 SSD 原型；缺点则是访问延迟也较大：HDD 的随机访问由于寻道、磁盘旋转等延迟的存在通常需要花费数个毫秒，SSD 的平均访问延迟也通常达到数百微秒。这种访问特性使得存储部分成为整个分布式文件系统的瓶颈，也使传统 DFS 集中于优化其对存储介质的访问。这类 DFS 的典型例子如 Ceph[1]、HDFS[2] 和 Gluster[3]。

近年来，持久性内存（Non-volatile Memory，NVM）技术的迅速发展打破了这种设计范式。与传统的持久性存储介质相比，NVM 同样具有断电不丢失数据的优点；同时，它支持字节粒度访问，读写延迟也相对较低。对比之下，HDD 必须以 512 字节的扇区为单位读写数据，SSD 则通常必须以 4KB 的数据页为单位进行读写。

因此，基于大容量、低速存储介质的传统 DFS 在 NVM 的这些新特性面前不

再适用。同时，NVM 也为高效、低成本地实现高可用性提供了新的机遇。发展适合 NVM 特性的新型高可用分布式文件系统势在必行。

1.2 NVM 对 DFS 提出的挑战

NVM 的读写延迟相比 SSD 低三到四个数量级。例如，Intel 公司 Optane SSD 产品的 4KB 随机读写延迟约为 214 us[4]，而其 Optane DC NVM 产品的 4KB 随机读延迟约为 0.3 us，随机写速度更是低至 0.06 ~ 0.09 us[5]。在这种情况下，传统分布式文件系统对存储介质读写的优化就显得无足轻重；同时，计算资源（即 CPU）和网络带宽反而成为了新的瓶颈。

NVM 的主要缺点就是单位容量的成本远高于 HDD 和 SSD；例如，Intel 公司的 760p SSD 产品每 GB 价格约为 0.16 美元，而其 Optane DC NVM 产品每 GB 价格高达 10.42 美元，是前者的 65 倍。传统 DFS 通常不会特意节省廉价的硬盘存储空间，而基于 NVM 的 DFS 则不得不考虑这一问题。

1.3 研究现状

NVM 技术在近两三年发展迅速。早期的一些工作提出使用 NVM 代替 HDD、SSD 等低速存储介质，从而提高单机文件系统的性能。已有的成果如 NOVA[6] 和它的改进 NOVA-Fortis[7]。NOVA 利用了 NVM 的字节粒度访问的特性，将文件系统日志存储在 NVM 中，减少了日志持久化带来的开销。NOVA-Fortis 在此基础上还提供了数据完整性的保障。

另外，也已有多项研究开始使用 NVM 作为 DFS 中的持久性存储介质。由于数据存储的延迟已经极大地降低，传统的基于 TCP/IP 网络协议栈的网络通信成为了新的性能瓶颈。新兴的远程内存直接访问（Remote Direct Memory Access，RDMA）技术支持用户态直接访问远端节点的内存，减少了大量的数据复制，很大程度上缓解了这一问题。综合利用 NVM 和 RDMA 技术的已有成果如 Octopus[8]、Orion[9] 和 Assise[10]。它们从不同方面入手优化文件系统的性能。例如，Octopus 改进了分布式事务协议来降低网络请求的开销；Orion 借鉴了 NOVA，通过一个高效的元数据服务器降低维持一致性的开销；Assise 则实现了完善的缓存机制，通过多级备份实现快速的错误恢复。

由于针对基于 NVM 和 RDMA 技术的 DFS 的研究还处于较为早期的阶段，上述这些研究都只是针对较为单一的方面进行优化。虽然部分工作[9-10] 已经报

告了较好的可用性，但它们使用的仍是备份策略，忽视了当前高成本 NVM 带来的节省存储空间的需求。文献调研表明，尚未有已发表的工作研究如何以较低的 NVM 空间开销保证系统的高可用性。

1.4 本课题解决的问题

本课题的目标是在兼顾系统高可用性的同时，尽可能地降低 NVM 上的空间开销。

具体而言，本课题提出一种适合上述 NVM 特性的新型分布式文件系统。它在充分利用 NVM 高带宽、字节粒度访问等特性的同时，利用纠删码技术的优势，以较低的存储代价实现高可用性。

第 2 章　技术选型简介

本章介绍本文涉及的的存储、网络和分布式文件系统相关技术，同时详细介绍本课题使用的技术选型。

2.1　持久性内存技术

持久性内存（NVM）指的是在断电时不会丢失数据的随机访问存储器。NVM 的一个重要特性是支持字节粒度的随机访问，这使它与必须以 4KB 或其他大粒度读写的闪存（Flash Memory）等技术区别开来。

常见的 NVM 技术包括铁电存储器（FeRAM）、磁阻存储器（MRAM）、相变存储器（PCM）、铁电栅场效应晶体管存储器（FeFET Memory）等。NVM 技术在近年来发展迅速。例如，美国 Ramtron、Texas Instruments 等公司已经量产 FeRAM 存储设备，美国 Everspin 公司也已量产容量为 4Mb 的 MRAM 存储器。由于它们的存储容量都较小，因此 NVM 技术真正得到商用是在 2019 年，由 Intel 公司宣布推出使用 PCM 技术的 Optane DC 持久性内存产品；其容量可达 256GB，真正具备了取代传统存储设备的潜力。

本课题目前使用 DRAM 模拟 NVM 设备。本课题选取的实验环境操作系统为 CentOS 7，其支持将一定容量的 DRAM 模拟为 NVM 空间，并提供与真实 NVM 相同的设备接口。

2.2　远程直接内存访问技术

远程直接内存访问（RDMA）指的是一种通过支持 RDMA 的网卡（即 RDMA NIC），直接读写远端节点内存的技术。与直接内存访问（DMA）技术类似，RDMA 在传输数据时无需操作系统参与，减少了上下文切换和数据复制的 CPU 开销，相较于传统 TCP/IP 协议栈而言，极大地降低了网络之间数据传输的延迟。

RDMA 技术支持许多十分有用的特性：

(1) 与 TCP 协议类似，RDMA 也支持可靠连接（Reliable Connection）；两个节点之间的 RDMA 连接建立后，如果其中一方意外断开，另一方能立即收到连接断开的消息。这使分布式系统能方便地检测到其中某个节点的故障；

(2) 与套接字类似，RDMA 支持发送（send）/接收（recv）原语，能够以通知远端节点的方式传送数据，即双边操作；RDMA 也支持读（read）/写（write）原语，能够在不通知远端节点的方式传送数据，即单边操作。由于双边操作适合实现远程过程调用，单边操作适合进行单纯的数据传输，文件系统可以使用合适的原语来优化其性能；

(3) RDMA 同时支持一种轻量级的通知远端节点的写原语（write-with-imm），这使得文件系统中的节点能高效地监控到达其上的写入操作并记录日志，从而方便地实现节点的错误恢复。

2.3 数据冗余技术

可用性的定义是系统在特定时间段内能正常提供服务的能力。在分布式系统中，组成系统的各个存储节点彼此在物理上独立，这使得某个节点发生故障时其他节点能不受波及，但也使得整个系统的故障率极大提升。例如，假如某份数据在整个系统中仅存有一个副本，则当存放该副本的节点发生故障时，数据就不再可用。在不应用数据压缩算法的前提下，解决这一问题势必要付出额外的存储空间存储冗余数据，从而也就必须付出额外的 CPU 和网络开销。

在 CPU、网络和存储空间开销之间作出权衡，通过存储冗余数据为系统提供可用性的技术即称为数据冗余技术。

2.3.1 数据备份

数据备份（Replication）是一种最为常见和简单的数据冗余技术。它的基本思想是，对于存入文件系统的一份数据，系统实际上将它存储为 r 个完全相同的副本，分别位于物理上分离的 r 台存储设备上。只要 r 份数据没有全部损坏，系统总能从中选择一份完好的数据副本用于提供服务。实际使用时通常取 $r = 3$，对应的策略称为三副本策略。

数据备份的优点是简单、直接；例如，写数据时只需要将数据原封不动写入到若干个设备中，读数据时则从中任意选择一个，读出原始数据即可。

它的缺点则是存储效率低下。例如，使用三副本策略时，全部存储空间只有三分之一被有效利用，其余都用来存储数据的备份。在大容量、价格低廉的 HDD 或 SSD 上，备份策略不失为一种行之有效的手段；然而在 NVM 上，它则会大量占用 NVM 本就十分昂贵的存储空间，进一步加大数据的存储成本。

本课题实现了数据备份策略，作为实验结果的基线对照。

2.3.2 纠删码

纠删码（Erasure Code）是另外一种常见的数据冗余技术，广泛使用于网络通信数据编码等领域。它的基本思想是将 k 份原始数据通过一定的方式进行编码，得到 p 份冗余数据，合计 $k+p$ 份数据，分别位于物理上分离的 $k+p$ 台存储设备上。当其中至多 p 份任意数据出错时，均可以通过算法重构出原始的 k 份数据。典型的纠删码策略如 Reed-Solomon 编码，对应的编码策略通常记为 $RS(k,p)$。容易看出，上述数据备份策略实际上等价于 $RS(1,r-1)$。

纠删码的优点是节省存储空间、存储效率高。例如，三副本策略能容忍至多 2 个节点同时故障，存储效率为三分之一。然而，任意的 $RS(k,2)$ 同样能容忍 2 个节点同时故障，它的存储效率等于 $\frac{k}{k+2}$。实践中 k 能够取到 10[11]，对应的存储效率约为 83.3%，为三副本策略的 2.5 倍。

纠删码的缺点则是需要引入额外的计算量。例如，对于 Reed-Solomon 编码来说，它的编码和解码过程都涉及到矩阵运算。编码 k 份长度为 L 的原始数据的计算复杂度为 $O((k+p)kL)$；解码则首先需要花费至少 $O(k^2)$ 的时间计算解码矩阵，再花费 $O(pkL)$ 的时间解码原始数据。虽然实践中 k、p 并不大，但它表明纠删码策略会引入正比于数据量的 CPU 开销，而数据备份策略几乎没有这部分开销。

本课题实现了纠删码策略，并且将会通过实验证明：纠删码编解码虽然带来了额外的 CPU 计算量，但它对系统的影响是可忽略的。

第 3 章　文件系统架构设计

本课题设计并实现的文件系统大致包含两层：文件系统层和纠删码层。

由于本课题的核心目标是证明利用纠删码提供可用性保障的可行性和评估其相对效率，因此文件系统层直接复用了已有的工作 LocoFS[12]。LocoFS 是一个针对元数据操作进行优化的分布式文件系统原型，它的各种逻辑相对简单，方便移植。但是，LocoFS 针对传统低速块设备存储和低速网络设计，对 CPU 资源的使用较为随意；因此本课题在移植 LocoFS 时，重写了其网络通信逻辑，同时优化了读写路径，尽可能降低了网络带宽占用和 CPU 开销。

文件系统的节点分为三类：目录元数据服务器（DMS）、文件元数据服务器（FMS）和文件数据服务器（DS）。其中，DMS 只有一个，负责存放目录元数据和进行目录操作；FMS 可以有多个，负责存放文件元数据和进行文件元数据操作；DS 也可以有多个，负责存储文件数据。在实现时，任何一个 DS 都可以同时充当客户端，与元数据服务器通信，进行各种文件系统操作。

纠删码层对文件系统层提供 4KB 页粒度的数据访问接口。它对用户屏蔽底层 NVM 设备，向用户层提供一个连续的虚拟 NVM 空间。当用户读写某一个虚拟页时，纠删码层自动计算其对应的数据块位置；如果是读操作，则通过 RDMA 读操作取得对应的数据块，解码出原数据块后返回给用户；如果是写操作，则先对数据块进行编码，然后将数据块和纠删码块通过 RDMA 写操作写入到对应的位置上。

整个系统的架构如下图所示：

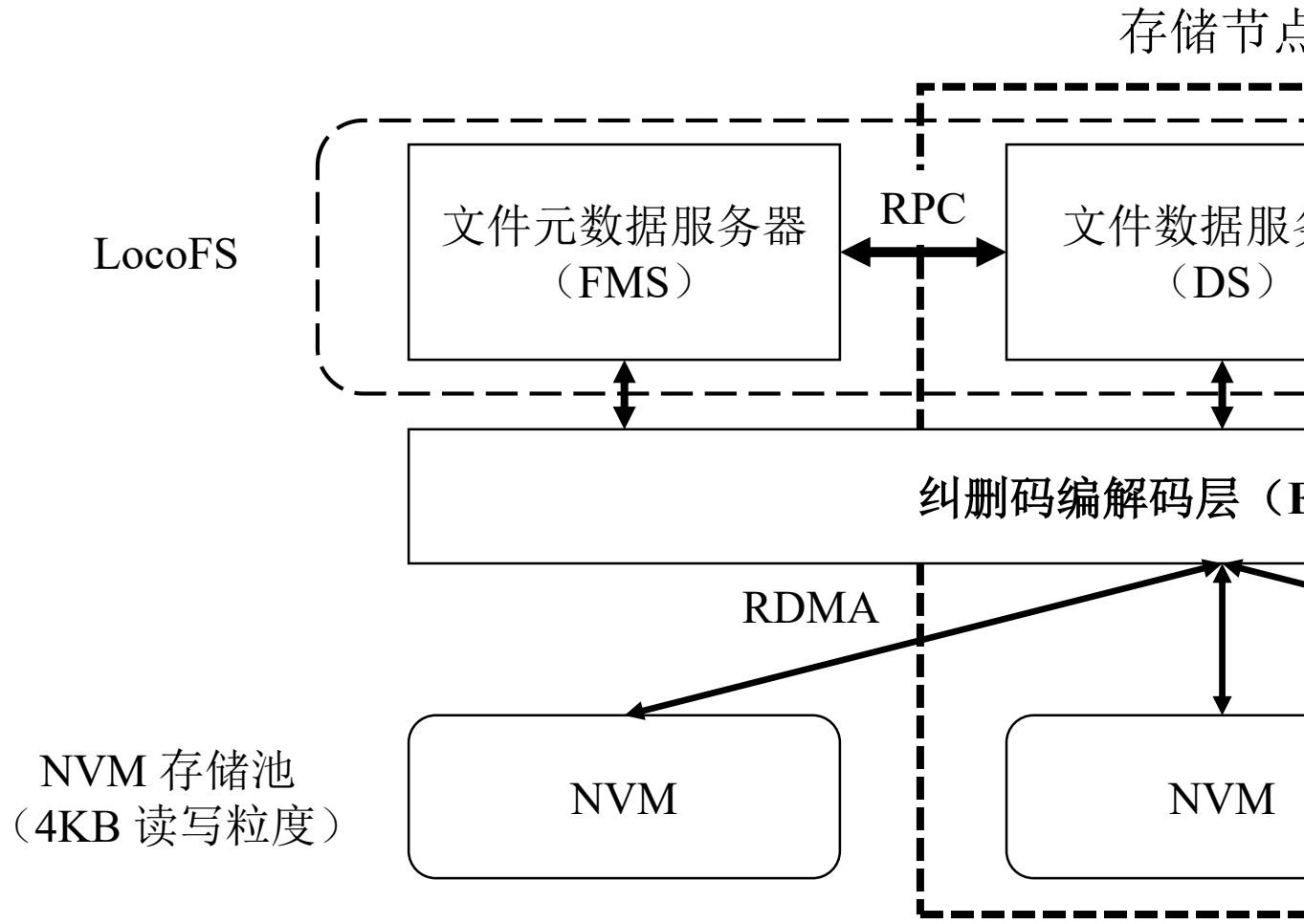

图 3.1　系统架构示意图

以下各小节将详细说明文件系统的细节设计和优化方案。

3.1　纠删码策略的选用

根据被编码数据块的大小不同，纠删码策略又可细分为两种：

(1) **分组**：被编码的数据块单元大小等于 4KB。以 $RS(4,2)$ 为例，分组策略对 4 个 4KB 原始数据块进行编码，产生 2 个 4KB 的校验数据块，然后将它们分别存放于 6 个物理上分离的存储设备上。分组的优势是读写时网络访问次数少；劣势是写入操作较为复杂，同时需要额外处理系统降级时的读操作。

(2) **切块**：被编码的数据块单元大小等于 $\frac{4}{k}$ KB，其中 k 是纠删码的参数。以 $RS(4,2)$ 为例（$k=4$），切块策略把一个 4KB 原始数据块切分为 4 个 1KB

数据块，对其编码，产生 2 个 1KB 的校验数据块，然后将 6 个 1KB 数据块分别存放于 6 个物理上分离的存储设备上。切块的优势是读写逻辑简单，系统降级情况下读操作不受影响；劣势是读写时需要的网络访问次数较多。以上两种策略的读写

插图索引

表格索引

参考文献

[1] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]//Proceedings of the 7th symposium on Operating systems design and implementation. 2006: 307-320.

[2] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[C]//2010 IEEE 26th symposium on mass storage systems and technologies (MSST). 2010: 1-10.

[3] DAVIES A, ORSARIA A. Scale out with glusterfs[J]. Linux Journal, 2013, 2013(235):1.

[4] INTEL. Intel optane ssd dc d4800x product brief[EB/OL]. 2020[2020-03-31]. https://www.intel.cn/content/www/cn/zh/products/docs/memory-storage/solid-state-drives/data-center-ssds/optane-ssd-dc-d4800x-series-brief.html.

[5] YANG J, KIM J, HOSEINZADEH M, et al. An empirical guide to the behavior and use of scalable persistent memory[C]//18th USENIX Conference on File and Storage Technologies (FAST '20). 2020: 169-182.

[6] XU J, SWANSON S. Nova: A log-structured file system for hybrid volatile/non-volatile main memories[C]//14th USENIX Conference on File and Storage Technologies (FAST '16). 2016: 323-338.

[7] XU J, ZHANG L, MEMARIPOUR A, et al. Nova-fortis: A fault-tolerant non-volatile main memory file system[C]//Proceedings of the 26th Symposium on Operating Systems Principles. 2017: 478-496.

[8] LU Y, SHU J, CHEN Y, et al. Octopus: An rdma-enabled distributed persistent memory file system[C]//2017 USENIX Annual Technical Conference (USENIX ATC '17). 2017: 773-785.

[9] YANG J, IZRAELEVITZ J, SWANSON S. Orion: A distributed file system for non-volatile main memory and rdma-capable networks[C]//17th USENIX Conference on File and Storage Technologies (FAST '19). 2019: 221-234.

[10] ANDERSON T E, CANINI M, KIM J, et al. Assise: Performance and availability via nvm colocation in a distributed file system[J]. arXiv preprint arXiv:1910.05106, 2019.

[11] WANG A, ZHANG J, MA X, et al. Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache[C]//18th USENIX Conference on File and Storage Technologies (FAST '20). 2020: 267-281.

[12] LI S, LU Y, SHU J, et al. Locofs: A loosely-coupled metadata service for distributed file systems [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2017: 1-12.

[13] 薛瑞尼. ThuThesis: 清华大学学位论文模板[EB/OL]. 2017[2019-04-27]. https://github.com/xueruini/thuthesis.

致　谢

衷心感谢导师 xxx 教授和物理系 xxx 副教授对本人的精心指导。他们的言传身教将使我终生受益。

在美国麻省理工学院化学系进行九个月的合作研究期间，承蒙 xxx 教授热心指导与帮助，不胜感激。感谢 xx 实验室主任 xx 教授，以及实验室全体老师和同学们的热情帮助和支持！本课题承蒙国家自然科学基金资助，特此致谢。

感谢 LaTeX 和 ThuThesis[13]，帮我节省了不少时间。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

附录 A　外文资料的调研阅读报告

Literature Review of RDMA-Enabled Distributed File Systems on NVM

High-performance, byte-addressable non-volatile memories (NVMs) developed dramatically and have been the cutting-edge of storage systems in recent years. Prior to its advent, distributed system designers find that the system's performance is mainly limited by the access latency of the underlying block device. Both hard-disks and solid-state drives (SSDs) are much slower than DRAM, leading to complex, specialized designs for better efficiency[9].

However, the appearance of NVM brings us low-cost, high-capacity and byte-addressable fast persistent data storage. Media access latency no longer determines the whole system's performance, which forces designers to rethink those tradeoffs with networking overhead and CPU involvement[9]. With remote direct memory access (RDMA), NVM access over the InfiniBand network is also efficient enough, promising a bright future for RDMA-based NVM distributed storage systems[10].

Many new file systems are designed to fully exploit the features of NVM and RDMA for better performance. They share the same design principle, that the overheads of kernel/user crossing and network accesses should be reduced to minimal. All these systems must provide data consistency, while they can also have additional features like availability, fault tolerance, and failure recovery. Due to the CAP Theorem, they must make different tradeoffs among these features, resulting in the diversity of these systems. *Octopus*[8], *Orion*[9], and *Assise*[10] are three typical examples of NVM-specialized distributed file systems. Octopus was first published in 2017, and Orion and Assise were both published in 2019.

A.1　Octopus

Octopus exploits the fact that a remote procedure call (RPC) is much more expensive than an RDMA verb. From this fact, Octopus stated that the client should actively fetch data from the server, instead of incurring an RPC request and simply waiting for

the server to prepare the data and send it back. We may call it the Client-Active Principle and we will see it later. Octopus also proposed improved transaction protocols. Transaction is a fundamental mechanism in storage systems to ensure atomicity, concurrency, and crash consistency. To support transactions, original approaches consist of no less than 2 remote procedure calls (RPCs) and incurs high overheads, while Octopus proposed a new protocol named Collect-Dispatch Transaction which needs only 1 RPC and 2 RDMA verbs. As we mentioned above, RDMA verbs incur far less overhead, and therefore the new protocol is faster.

However, Octopus also has its drawbacks:

- it cannot tolerate any client's failure;
- it uses a simplified file system model and only supports small file and directory sizes[9];
- it locates files at different clients based on hash values and therefore is insensitive to data locality;
- it makes no use of DRAM, accelerating write operations but significantly slowing down read operations[10].

Later works, like Orion and Assise, pay more attention to these problems.

A.2 Orion

Orion further exploits the speed of RDMA requests and aggressively uses RDMA wherever it can. Inherited from NOVA[?], it is log-structured and ensures strong data consistency by atomic operations on a centralized metadata server (MDS). The MDS maintains a log for each inode. When the client needs to update a file, it first fetches the pointer to the corresponding inode log from the server, then updates the log locally. Next, the client writes the updated part to the MDS using an RDMA write request, and finally notifies the server by an optimized RPC. For every file request, the client consults the MDS to ensure data consistency. If it finds any conflict, it blocks the request and rebuilds in-memory data structures and retries. DRAM cache is used to exploit data locality. The system is based on NOVA and Mojim: both are previously published distributed file systems, while the former provides efficiency and the latter provides availability and crash consistency. As a result, Orion outperforms existing distributed file systems in file

system operation (FSop) latencies and I/O throughput. Orion is also well-scalable at the ”rack-scale”, i.e. around 10 clients, under the assumption that it will not be limited by the MDS bandwidth bottleneck.

We can see that Orion’s designers treat the Client-Active Principle as common sense, as they never mention Octopus when describing their system’s design. By the time of the advent of Orion, characteristics of NVMs, RDMA requests, and RPCs are already well-known. State-of-the-art distributed file systems implement the most efficient strategy en masse, using small-sized RPCs for acknowledgment and RDMA requests to send large-sized data for best efficiency. We will also see this in Assise.

A.3 Assise

Assise put more efforts on system availability and crash consistency. It implements a novel distributed cache layer, named CC-NVM, to provide these features. For access speed, it takes advantage of the fact that NVM is much faster than local cold storage (e.g. HDD or SSD), even if located remote. It builds a three-layer cache hierarchy consisting of DRAM (for applications), local NVM (for client nodes), and remote NVM. To guarantee that applications can always get access to the newest data, it employs the LRU policy in CC-NVM. For crash consistency, it uses a replication chain to ensure the prefix semantics – that is, the system can always recover after a failure to a state which is the result of a prefix of the operation sequence. When doing replication, a node writes data to the log area of its successor in the replication chain, incurs a small RPC to notify the successor to continue the chain, and then waits for an acknowledgment RPC back.

Assise also supports replicas at both application and client node levels. It uses heartbeat messages to detect crashes, and once a crash occurs, Assise immediately fails-over to the replica process or client node. Process crashes only need a quick restart to get fixed, but node failures take more time for we must reboot the OS. To speed it up, Assise stores a snapshot of a freshly booted OS in the NVM and recovers the system to the snapshot when necessary. As a result, Assise also outperforms existing systems in FSop latencies and throughput mainly because of its good cache layer design. It is also highly available and recovers fast from failure. Although Assise is published several months later than Orion, its authors did not compare the two systems together because

Orion's source code is currently not publicly available.

We see the Client-Active Principle again at the chain-replication part of Assise. A node continues the chain by an RDMA write and a small RPC for notification. The successor node is not responsible for its own replica, because this brings large RPC overheads if we do so. In about two years, this principle changes from a new concept to common sense. It shows that NVM-based distributed systems are really developing very rapidly.

A.4 Summary

By observing Orion and Assise we still see several problems in the designing of a distributed system. The arrangement of metadata is a key point, and Orion takes the approach of a centralized MDS while Assise distributed metadata to the clients. The former one is limited by the bandwidth of the MDS and the latter one introduces consulting overheads when accessing metadata. Also, we must consider load balancing among the clients, which may require splitting files to chunks to support large files and make full use of the network bandwidth. Further, there is a chance to further improve the availability of the system. Although erasure codes can be applied in a distributed file system to improve availability, few have considered this before and most researchers still stick to the traditional replication approach. Our aim is to improve Octopus by not only overcoming its own shortcomings, but also making thorough investigations about the abovementioned problems and implementing an optimal solution to make it state-of-the-art.