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 5 月 28 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

可用性是文件系统的重要属性。持久性内存（NVM）是一种新型、高速的存储介质，基于 NVM 的分布式文件系统技术近年来发展迅速，有必要为其寻找适合的可用性策略。

本文从节省 NVM 存储成本的角度出发，提出一个使用纠删码的文件系统原型。该系统实现了针对 NVM 存储环境优化的纠删码方案，并对其性能表现作了综合测试。实验结果表明，与传统的数据备份方案相比，纠删码在保持相同可用性水平的同时，减少了 50% 的存储空间开销；同时，其数据写入性能也与之相近或更优，其中写延迟最多降低 10%，写带宽最多提升 11%。结果证明了纠删码能以较低开销取得较高性能，适合作为基于 NVM 的分布式系统的可用性策略。

关键词：纠删码；可用性；持久内存；分布式文件系统

ABSTRACT

Availability is a key feature of file systems. Non-volatile memory (NVM) is a new type of persistent storage with notably high speed, and distributed file system technology based on NVM are developing rapidly in recent years. It is necessary to find a proper availability solution for those systems.

Born from the idea of saving NVM storage costs, this paper presents a file system prototype that leverages erasure codes. It implements coding strategies optimized for distributed NVM storage environment and tests them thoroughly. Results shows that, comparing to conventional data backup mechanisms, erasure codes achieve the same availability level with 50% less space usage. It also provides similar or better performances, reducing the write latency by up to 10% and improving the write bandwidth by up to 11%. The results demonstrated that erasure codes can perform well with lower costs and is suitable for NVM-based distributed file systems.

Keywords: erasure coding; availability; non-volatile memory; distributed file system

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）
NVM	持久性内存（Non-volatile Memory）
DFS	分布式文件系统（Distributed File System）
RDMA	远程直接内存访问（Remote Direct Memory Access）
CPU	中央处理器（Central Process Unit）
DRAM	动态随机存取存储器（Dynamic Random Access Memory）
IOPS	每秒读写次数（Input/Output Operations Per Second）

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。

DFS 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。由于三者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 DFS 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，DFS 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 DFS 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 DFS 的瓶颈。基于传统存储介质的 DFS 的典型例子如 Ceph[1]、HDFS[2] 和 Gluster[3]。它们通常针对系统中性能最低的存储介质读写进行优化，同时占用 CPU 进行大量运算（例如，Ceph 定期执行的 compaction 操作会带来巨大 CPU 负载）。为了保证可用性，这些文件系统通常会存储数据的若干个备份，防止单节点故障导致数据无法访问。

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 μs[4]，而其 Optane™ DC NVM 产品的 4KB 随机读延迟约为 0.3 μs，随机写速度更是低至 0.06 ~ 0.09 μs[5]。

在这种情况下，传统 DFS 对存储介质读写的优化就显得无足轻重；而同时，CPU 的计算资源和网络带宽反而成为了新的瓶颈。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。发展适合 NVM 特性的新型高可用 DFS 势在必行。

基于 NVM 的分布式文件系统的一个核心问题是其存储成本。由于 HDD、SSD 技术都早已成熟，单位容量价格十分低廉，传统 DFS 可以存储数据的多个副本，而不必担心因此产生过多的额外花销。NVM 单位容量的价格则较为昂贵，如果直接应用既有的可用性策略，则势必产生巨大的存储成本。如何在保证 DFS 高效率的同时尽可能减少存储开销，仍然是一个开放问题，既往研究几乎没有涉及。

纠删码是一种较为新型的数据容错策略，它在电子通信领域已经得到广泛应用：只需额外传输很少的额外数据，就能提供较高的容错能力。因此，在节省存储开销方面，纠删码有着巨大的潜力。在文件系统领域，也已有许多工作使用了纠删码，但它在基于 NVM 的分布式系统中表现如何尚未有定论。因此，本课题的目标是在 NVM+RDMA 环境下实现纠删码，并以此为文件系统中的数据提供可用性保障，最后测试其性能表现。

1.2 研究现状

本小节从基于 NVM 的文件系统和纠删码两方面介绍学术界目前的研究情况。

1.2.1 基于 NVM 的文件系统

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。已有的成果如加州大学圣迭戈分校提出的 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 技术的已有成果如清华大

学提出的 Octopus[8]、加州大学圣迭戈分校提出的 Orion[9] 和华盛顿大学提出的 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。

然而，上述这些研究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。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。

1.2.2 基于纠删码的文件系统可用性策略

纠删码早在 1950 年代就出现在通信领域，其代表是 Hamming 码，可以识别和纠正通信中的任意单比特错误。在存储领域，纠删码已经被应用于前沿研究和工程实际中。

前沿研究方面，上海交通大学提出的 Cocytus[11] 在一个内存键值存储引擎中应用了纠删码策略；乔治·梅森大学提出的 InfiniCache[12] 构建了一个大规模分布式数据缓存系统，同样应用了纠删码，以提供数据可用性保障、提升效率和降低成本；密歇根大学提出的 Hydra[13] 则构建了一个使用纠删码策略的分布式内存分配器。这些研究表明了在高性能系统中使用纠删码的可行性和有效性。

工程实际方面，微软公司早在 2012 年就在其 Windows Azure 云存储系统中部署了一个低开销的纠删码策略[14]，用于应对 Azure 存储服务器频繁的下线更新，使系统始终保持可用。Facebook 公司也使用了一个称为 f4 的纠删码存储系统[15]，用于以较低的空间开销存储 Facebook 上大量的图片和流媒体文件。这些纠删码的应用实例表明了纠删码用于文件系统时带来的明显优势。

然而，这些系统或者不是标准的分布式文件系统，不能代表纠删码在文件系统中的性能表现；或者基于传统块设备设计，并且已经有一定的历史，不能适应基于 NVM+RDMA 的最新存储环境。尚未有已发表的工作研究如何在高性能存储和网络环境下应用纠删码策略。

1.3 本课题解决的问题

既往研究已经揭示了纠删码作为分布式存储系统的可用性保障的潜力。但是，尚未有研究在 NVM+RDMA 环境下构建一个使用纠删码的 DFS，通过实证的方式检验纠删码的可行性和有效性。本课题首先作出了填补这一空白的尝试。

具体而言，本课题解决的问题是：证明在 NVM 作为存储介质、RDMA 提供高速高带宽网络的环境下，纠删码是一种合适的可用性方案，其能在节省存储空间的同时，又保证较好的性能。

为了实现上述研究目标，本课题采取了以下的研究方案：首先，设计并实现一个基于 NVM 和 RDMA 的新型分布式文件系统。该系统采用模块化的设计方式，针对 NVM+RDMA 环境对现有的纠删码策略进行优化、移植。完成系统的设计后，从理论上证明，相较于传统数据备份策略，使用纠删码能带来更低的存储开销。完成系统的实现后，对该系统开展综合的测试，与传统数据备份策略对比，证明纠删码能提供与之相当的可用性保障，同时也能提供与之相当或更优的性能。

第 2 章　持久性内存文件系统可用性技术简介

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。

2.1　持久性内存技术

在本文中，持久性内存（NVM）特指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。

字节粒度访问特性使 NVM 提供了极大的灵活性，但这也造成它的生产技术难度较大，且成本昂贵。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。例如，美国 Ramtron、Texas Instruments 等公司已经量产用于工业设备的 FeRAM 存储设备，美国 Everspin 公司也已量产容量为 4Mb 的 MRAM 存储器。对于分布式文件系统领域来说，这些产品的存储容量都过于低下，缺少实用价值。

虽然没有适合的 NVM 设备，但近年来 NVM 技术仍然发展迅速，文件系统领域的许多最新成果使用了 NVM 作为存储介质（之一）。这是因为 Linux 内核已经提供了针对 NVM 的支持，并且提供了将 DRAM 内存空间映射为 NVM 设备的接口，大部分工作利用该接口通过大容量 DRAM 模拟 NVM。由于 NVM 的访问特性与 DRAM 并不完全相同，内存空间映射并不能解决这个问题，因此也有工作[16] 利用远程节点内存模拟 NVM，通过网络延迟模拟 NVM 的读写延迟。

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。真实产品的出现也使得对 NVM 读写特性的详细测试成为可能。

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。为了解决这一问题，文件系统势必要付出额外的空间存储冗余数据；由于冗余数据的存在，系统也因而必须付出额外的 CPU 和网络开销。

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。不同的数据冗余技术在 CPU、网络通信和存储空间开销之间作出不同的权衡，因此也适合不同的场景。本课题中涉及的数据冗余技术为以下 2 种：数据备份，以及纠删码。

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。因此，系统最多能容忍 $p = r - 1$ 个节点同时故障。实际使用时常常取 $r = 3$，对应的策略称为三副本策略。

数据备份的优点是简单、直接；在写数据时，只需要将数据原封不动写入到

若干台存储设备中，读数据时则从其中任意选择一个，读出原始数据即可。因此，它除了在写入时需要进行额外的网络访问，几乎不会产生其他的开销。

数据备份的缺点则是会占用大量的存储空间。例如，使用三副本策略时，全部存储空间只有三分之一可能被有效利用，其余部分必须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经济而行之有效的手段；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。表 2.1 给出了典型的 HDD、SSD 和 NVM 目前的价格，NVM 单位容量的价格相比 SSD 高出 65 倍。如果在 NVM 上随意应用数据备份策略，则必然导致昂贵的存储开销。

表 2.1　存储介质单位价格对比

存储介质	设备型号	每 GB 价格（美元）
HDD	Seagate Ironwolf 1TB NAS Raid	0.07
SSD	Intel® SSD 760p 1024GB	0.16
NVM	Intel® Optane™ DC Persistent Memory 256GB	10.42

本课题实现了一个简单的三副本策略，作为性能测试结果的基线对照。

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 $n = k + p$ 份数据，分别位于物理上分离的 n 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。因此，系统能

Reed-Solomon 编码是最典型的纠删码策略之一，其对应的编码策略通常记为 $RS(k, p)$。一般来说，它构建一个 Vandermonde 矩阵 V 用于编码：

$$V = \begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{k-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_p & a_p^2 & \cdots & a_p^{k-1} \end{pmatrix} \tag{2-1}$$

其中 $a_1, a_2, \cdots, a_p$ 互不相同且均不为 0。容易看出，数据备份可以看成是纠删码的一个特例：当 $a_1 = 1, k = 1$ 时，纠删码策略 $RS(1, p)$ 退化为 $r = p + 1$ 的数据备份策略。

实际操作时，可以用其他列满秩矩阵代替 Vandermonde 矩阵来计算 $\boldsymbol{V}$，有时能带来性能上的提升。

纠删码的优点是节省存储空间。为了衡量它节省空间的比例，我们定义存储效率等于数据的实际大小和它耗费的存储空间的比值。存储效率越高，节省的空间就越多。例如，三副本策略能容忍至多 2 个节点同时故障；其代价是将一份数据重复存储在 3 个位置上，存储效率为 33.3%。然而，任意的 $\boldsymbol{RS}(k,2)$ 同样能容忍 2 个节点同时故障；其代价是每 k 份原始数据都存储额外的 2 份校验数据，存储效率等于 $\frac{k}{k+2}\times 100\%$。显然，该值随 k 的增大而增大；只要 $k>1$，换言之，只要纠删码策略不退化为数据备份策略，它就能带来更高的存储效率。实践中，k 能够取到 10[12]，对应的存储效率为 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 码来说，它的编码和解码过程都涉及到矩阵运算，而数据备份策略只需进行网络传输，几乎完全没有计算开销。对此，已有的一些观点[8-9] 认为，相比于传统的块存储设备和基于 TCP/IP 协议栈的网络，NVM 和 RDMA 的引入已经极大地降低了存储和网络通信的延迟；为了继续提升性能，有必要减少耗费在 CPU 计算上的时间。根据这种观点，如果使用反而增加 CPU 计算开销的纠删码，将会是一个不经济的选择。

本文的目标在于证明，上述观点并不完全正确。使用纠删码虽然会引入额外的 CPU 开销，但其性能表现仍然能与传统的数据备份策略媲美。

第 3 章　分布式文件系统中的纠删码优化

3.1　文件系统架构设计

为了完成研究目标，本课题设计并实现了一个分布式文件系统原型。逻辑上，其结构可分为两层：文件系统层、可用性策略层。

(1) **文件系统层**：文件系统层处理文件系统操作逻辑。由于本课题的核心目标是证明纠删码的可行性和有效性，因此在文件系统逻辑方面，直接复用了已有的工作 LocoFS[17]。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，其逻辑简单，方便移植。LocoFS 包含三类存储节点：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，负责存放中心化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 有多个，负责存放文件数据。外部节点通过 LocoFS 客户端与存储集群交互，执行各种文件系统操作。

(2) **可用性策略层**：可用性策略层对文件系统层提供 4KB 粒度的数据访问接口。它屏蔽底层 NVM 设备和 RDMA 通信的细节，向上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，可用性策略层自动将其映射为对真实 NVM 设备的读写或 RDMA 访问请求。通过该层提供的抽象，只需做很少的修改，就能为文件系统应用纠删码、多副本等不同的可用性策略。

系统的架构如图 3.1 所示：

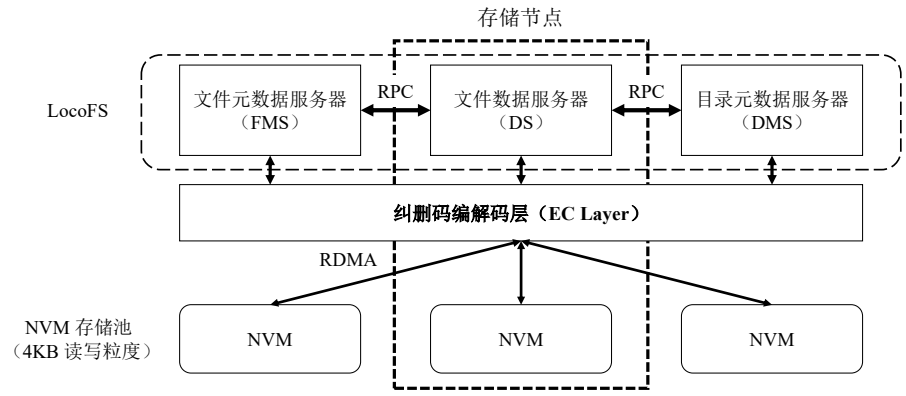


图 3.1　系统架构示意图

以下各小节将详细说明文件系统的细节设计和优化方案。

3.2 纠删码策略实现细节

本课题中数据以 4KB 为粒度访问，每个 4KB 数据块都被独立地编码。每个 4KB 数据块被切分为 k 个大小等于 $\frac{4}{k}$ KB 的编码单元，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。这种实现方式的优势是读写逻辑简单，并且在有节点故障的情况下，读写操作逻辑基本不受影响。

显然，相较于传统的数据备份策略，上述纠删码方案在节省存储开销方面有明显优势。在上述方案下，为了容忍任何 2 个存储节点同时发生错误，任何一个 4KB 数据块实际占用 6KB 存储空间。为了达到相同容错水平，数据备份策略需要将每个 4KB 数据块备份 2 次，合计占用 12KB 存储空间。纠删码方案节省了 50% 的存储空间。

3.2.1 朴素的 Reed-Solomon 编码实现

如 2.3.2 小节所述，Reed-Solomon 码策略 $RS(k,p)$ 需要一个 $p\times k$ 大小的列满秩矩阵 V。实际编码时，为了统一处理编码块将 V 拼接在 k 阶单位阵下方，得到一个大小为 $n\times k$（$n=k+p$）的编码矩阵；编码时，用该矩阵左乘 k 阶原始数据向量 D，得到 p 阶校验和向量 C：

$$\begin{pmatrix} I_k \\ V \end{pmatrix} \times D = \begin{pmatrix} D \\ C \end{pmatrix} \tag{3-1}$$

为了编码时的简便，矩阵乘法操作通常在伽罗华域 $GF(2^8)$ 上进行。此时，加法操作变为按位异或；乘法操作虽然较为复杂，但可以利用查表法高效实现。重复执行矩阵乘法，即可实现对任意相同长度数据的编码。

在解码时，需要首先根据待解码数据的来源，取出编码矩阵中的 k 行构成一个 k 阶方阵，然后计算其逆矩阵。计算逆矩阵的代价随 V 的选取有所不同，本课题使用了 $GF(2^8)$ 下的 Cauchy 矩阵作为 V，求逆只需要 $O(k^2)$ 的时间复杂度。

综合使用上述策略，就能以基本最高的效率实现朴素的 Reed-Solomon 码编解码操作。本课题据此完整地实现了任意参数的 Reed-Solomon 码编解码算法，用作其他纠删码优化方案的基线对照。使用 $RS(4,2)$ 编码 16KB 数据进行测试，一

次编码操作平均耗时 159.2 μs。显然，在高速 RDMA 网络通信环境、一次网络请求仅需数微秒的情况下，该方案效率十分低下，会让计算成为整个系统的瓶颈。

3.2.2 基于 SIMD 指令集优化的 Reed-Solomon 编码

我们观察到，朴素的 Reed-Solomon 码实现的运行耗时与待编码数据长度成正比，重复的矩阵乘法运算占用了大量时间。由于本课题中文件系统规定必须以 4KB 页为单位进行数据读写，待编码数据在存储上必定连续，这使我们使用单指令多数据流指令（SIMD）对编码进行优化。

Intel 的大部分 CPU 产品提供了三种 SIMD 指令集支持，分别为 SSE、AVX 和 AVX2。SSE 是 Intel 于 1999 年推出的 SIMD 指令集，支持 128 位的向量处理；AVX 是 Intel 于 2007 年推出的 SIMD 指令集，支持 256 位浮点 SIMD 运算支持；AVX2 则是于 2014 年推出的 AVX 的扩展，增加了 256 位整点 SIMD 运算支持。

Intel 在其存储加速库（Intel Storage Acceleration Library，ISA-L）中提供了开源的向量运算实现，其中就包括伽罗华域 $GF(2^8)$ 上的向量乘法和加法运算。本课题利用 ISA-L 实现了 SIMD 指令加速的纠删码编解码算法。使用 $RS(4,2)$ 编码 16KB 数据进行测试，分别使用 SSE、AVX 和 AVX2 指令集加速，一次编码操作平均耗时分别为 3.31 μs、2.98 μs 和 2.37 μs。

可见，SIMD 指令的引入大大降低了纠删码的编解码开销（加速比超过 50），使 NVM+RDMA 存储环境下纠删码的运用成为可能。另外也可看出，越新的指令集的运行效率也越高。因此，本课题选用 AVX2 指令集对所有纠删码操作进行加速。

3.2.3 基于分布式 NVM 存储优化的 Clay Code 编码

Reed-Solomon 码设计和实现简单，但有时难以满足实际需求。例如，朴素的 Reed-Solomon 码允许通过 k 份存活的数据恢复其余所有 p 份数据；但这同时表明任意 k 份数据线性无关，在执行数据恢复时，必须首先从 k 个节点中分别完整读取 1 份数据（合计 k 份）。实际应用中，为了降低存储代价、提升存储效率，k 的取值一般较大；因此，一个故障节点在执行恢复时将产生巨大的网络开销，有可能严重影响系统的执行效率。

不过，已有工作[18] 表明，可以通过设计合适的编码方式，减少进行恢复时需要读取的数据。如果从 d（$d \geqslant k$）个节点中读取数据，那么只需从每个节点中分别读取 $\frac{1}{d-k+1}$ 份数据即可。此时总共需要读取的数据量为 $\frac{d}{d-k+1}$ 份；显然 $\frac{d}{d-k+1} \leqslant k$ 恒成立，并且当且仅当 $d = k$ 时取到等号。

Clay Code[19] 是 2018 年由印度学者发表的一个 Reed-Solomon 码变种，它针对上述问题实现了最优化，同时具有设计和实现相对简单的优点。这一工作基于 Ceph 实现；由于 Ceph 使用高延迟的传统块设备作为存储介质，CPU 计算开销并系统瓶颈，因此原本的实现没有针对纠删码编解码进行优化。本课题希望评估朴素 Reed-Solomon 码以外的各种纠删码策略在 NVM+RDMA 环境下的可行性。因此，本课题选取了 Clay Code 作为非朴素纠删码策略的代表，基于 ISA-L 实现了指令集优化的 Clay Code 编解码算法。

3.2.4 基于分布式 NVM 和 RDMA 网络的前-k 读取策略实现

上述设计方案中，读取每个 4KB 数据页时，总是需要从 k 个不同的节点处分别读取对应的数据块进行解码。在实际运用中，出于各种无法预测的原因，总会有少数节点的响应延迟明显高于平均值，这些响应即被称作尾延迟（Tail Latency）。这时，系统就不得不等待尾延迟节点返回后再将结果返回给用户，系统性能也将受制于这些尾延迟节点。

为了降低尾延迟，我们可以回顾 Reed-Solomon 码的特性：在 k 个原始数据块和计算出的 p 个校验数据块中，任意读取 k 个都能恢复出原始数据。由于尾延迟现象是由偶然出现的个别节点导致的，我们可以应用一个非常简单的策略来过滤掉它们：随机选择位于不同存储节点上的 $k+\Delta$ 个数据块（$1 \leqslant \Delta \leqslant p$），选择其中延迟最小的 k 个节点读取数据。实践中难以随时检测各个节点的访问延迟，因此我们只需同时读这 $k+\Delta$ 个数据块，在读出 k 个后立即开始解码。这被称作前-k 读取策略。

由于 Δ 可以取得较小（通常为 1 或 2），并且其值基本无需随 k 变化，这一优化手段只会额外占用很少的网络带宽。因此，这一策略简单、有效，并且开销很低，在已有工作中已经得到广泛应用[12-13,20]。

然而，已有工作通常基于传统 TCP/IP 网络构建，在此前提下使用该策略取得了较好的效果。本课题希望评估在 RDMA 网络环境下使用该策略的可行性和有效性，以及其对性能带来的实际改进。因此，本课题同样实现了该策略，并且为了简单起见固定取 $\Delta = 1$。

3.3 可用性分析

如上所述，本课题实现了两种纠删码策略，分别为朴素的 Reed-Solomon 码，及其变种 Clay Code。后者仅针对数据恢复过程进行优化，在可用性方面本质上

与前者相同。因此，以下仅讨论朴素 Reed-Solomon 码 $RS(k, p)$ 的情况。

本课题直接使用了 LocoFS 的文件系统架构。如图 3.1 所示，节点分为 DMS、FMS 和 DS 三类，这三类节点分别使用不同的可用性策略。

对于 DMS 和 FMS 而言，它们仅用于存储元数据，并且读写粒度较小。如果在其上应用纠删码会导致严重的写放大，传统的数据备份策略更加适合这种情况。由于这并非本课题的研究重点，本课题没有实现对 DMS 和 FMS 的数据备份。然而，已有工作[21] 早已提供了成熟的基于主从备份的解决方案，并且过往研究也证明了此方案的可行性[9]。采用已有研究成果，系统能容忍 DMS 和 FMS 中任意 p 个节点同时故障。因此，本课题虽然不在 DMS 和 FMS 上实现相应的可用性策略，但这并不影响课题的核心结论。

对于 DS 而言，它们用于存储文件数据。本课题针对 DS 使用纠删码提供可用性保障。文件系统中所有存储文件内容的 4KB 页都经过 $RS(k, p)$ 编码，并分别存储在 $k + p$ 个节点上。在故障节点数不超过 p 的情况下，对任何一个数据页的读写都能成功；故障节点在恢复后，也能通过至少 k 个其他存活节点自主执行恢复数据，并在成功后重新加入存储集群。因此，系统能容忍任意 p 个 DS 节点同时故障。

综上所述，系统在理论上能容忍任意 p 个节点同时发生故障。

第 4 章　文件系统性能测试

本课题实现的文件系统部署在 3 台配置相同的服务器组成的微型集群上进行测试。测试环境的细节如下：每台服务器搭载 4 个 18 核 Intel® Xeon® Gold 6240M CPU，频率为 2.60 GHz。每台服务器装有 200 GB 内存，以及两个大小为 800GB 的 Intel® Optane™ DC 持久性内存；3 台服务器上合计 6 个 NVM 设备，用于模拟 6 个不同的存储节点。每台服务器安装有两个带宽为 100Gbps 的 Mellanox InfiniBand 网卡（实验时只使用其中一个），服务器之间通过它们建立 RDMA 网络连接。

测试时，通过一个客户机与上述集群通信，完成各种文件系统操作，并测试其性能。

4.1　不同可用性策略的开销

本节测试不同可用性策略的时间开销。在文件系统其它部分不变的情况下，在可用性策略层实现不同的策略，并测试其耗时。

一共测试四种不同的情况：无可用性策略（None）、三副本（Repl）、朴素 Reed-Solomon 码 $RS(4,2)$（RS），以及 Clay Code 编码（Clay）。每种情况下的时间开销分为两部分：耗费在 RDMA 上的网络通信开销，以及耗费在纠删码编码上的计算开销。的考虑到不同读写粒度可能对可用性策略的开销带来不同影响，分别以 4KB、16KB 和 64KB 作为读写粒度进行测试。由于测试的读写数据量之间相差较大，因此将测得的开销数据向三副本策略归一化。

测试结果如图 4.1 所示：

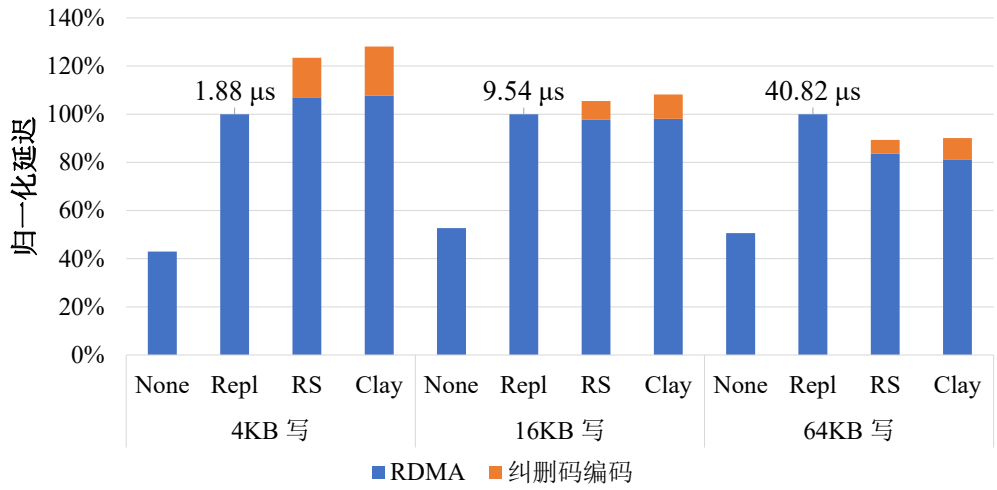


图 4.1　不同可用性策略的归一化开销

可见，虽然纠删码的编码带来了额外的 CPU 计算开销，在 4KB 读写时其延迟高出三副本策略 20% ~ 40%；但在读写粒度较大时，其网络通信量较小、并行度高，在 RDMA 通信上节省了较多时间，读写延迟反而更低；以 64KB 为粒度读写时，$RS(4,2)$ 和 Clay Code 带来的延迟均比三副本策略低 10% 左右。测试表明，纠删码在文件系统访问粒度较大时有良好的表现；而在访问粒度较小时，相比于传统数据备份策略，纠删码虽然会引入较大比例的额外延迟，但这些延迟的绝对值很小（本例中不超过 1 µs），对系统整体性能的影响并不大。

4.2　读写带宽和延迟

本节测试文件系统的读写延迟和读写带宽，表征文件系统的整体性能。读写以 4KB 粒度进行。

测得的 4KB 读写操作延迟如表 4.1 所示：

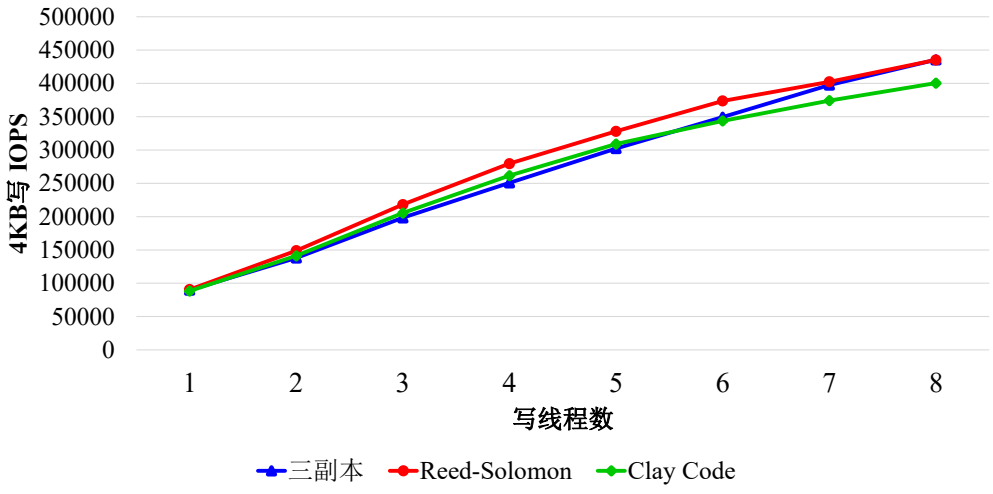
表 4.1　不同可用性策略下的 4KB 读写延迟（单位：µs）

可用性策略	读延迟	写延迟
三副本	18.92	33.46
Reed-Solomon 编码	22.15	38.75
Clay Code 编码	22.30	40.33

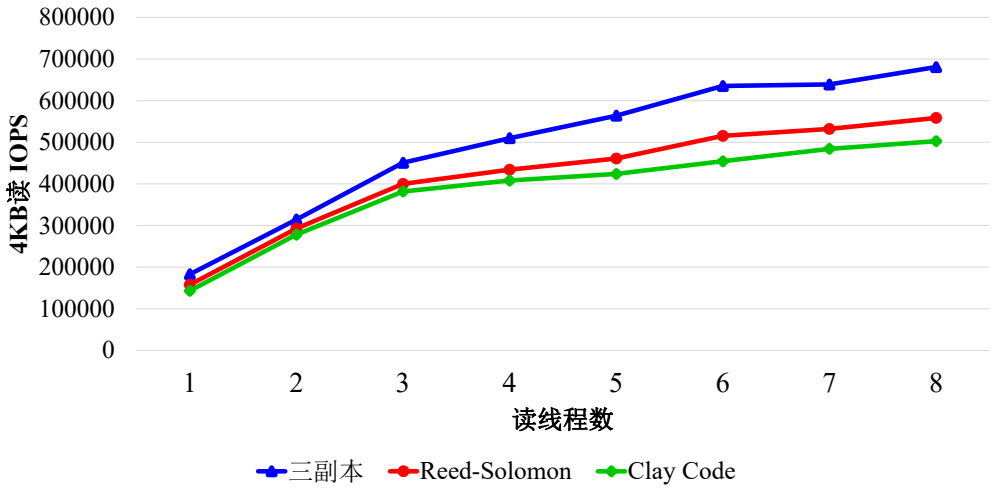
读写延迟由 4.1 节所述的可用性策略开销，以及文件系统的元数据操作开销共同构成。可以明显看出，元数据操作开销占了总开销的绝大部分。虽然这些开

销绝大部分由 LocoFS 引入，针对其进行优化理应能获得更低的读写延迟；但由于这并非主要研究目标，因此本课题并未予以实现。

测得的 4KB 读写操作带宽如下：



(a) 4KB 写



(b) 4KB 读

图 4.2　4KB 读写带宽曲线

写数据方面，本课题实现的文件系统在写线程数较少时，其性能表现了良好的线性增长趋势。并且，Reed-Solomon 码的表现略优于三副本策略，写带宽提升最大达到 11%。这是由于其写入的数据量相较于三副本策略少 50%，节省了 RDMA 带宽，因此取得了较好的表现。在使用 Clay Code 编码时，系统的写带宽相对较低，相比三副本而言最多降低了 8%。考虑到纠删码策略相比于三副本节省了 50% 的 NVM 存储空间，这一性能下降是完全可以接受的。

读数据方面，本课题实现的文件系统在读线程数少于 3 时表现良好的性能线性增长，但在并发数继续增加后性能增长进入瓶颈。相较于三副本策略，使用 Reed-Solomon 码时，系统的读带宽降低了 6%～18%；使用 Clay Code 编码时，系统的写带宽降低了 11%～30%。三副本策略在读数据时表现更好，是因为它没有纠删码解码开销；并且，虽然其读取的数据量与纠删码策略相同，但只需从一个节点读取一次；而纠删码需要从多个节点各读取一次，引入了额外的网络通信开销。

4.3 文件系统可用性及稳定性

在第 3 章中，我们从理论上论证了本课题中的文件系统能提供一定的可用性保障：3.3 节证明了该系统能容忍部分节点故障，3.2.4 节证明了该系统能容忍部分节点访问延迟的突然升高。本节针对上述两点进行测试，证明文件系统具备预想的可用性等级。

4.3.1 纠删码策略的可用性

本小节测试文件系统在节点失效时的工作状况。客户端连续进行约 60 秒的 4KB 文件读写；在进行到约 24 秒时，手动结束一个节点上的文件系统进程。由于一台服务器使用 2 个 NVM 设备模拟 2 个远程节点，因此这相当于模拟两个节点同时发生故障，达到了系统的最大容错能力。合计测得约 273.1 万次读写的延迟数据，经过均匀采样后，绘制曲线如下：

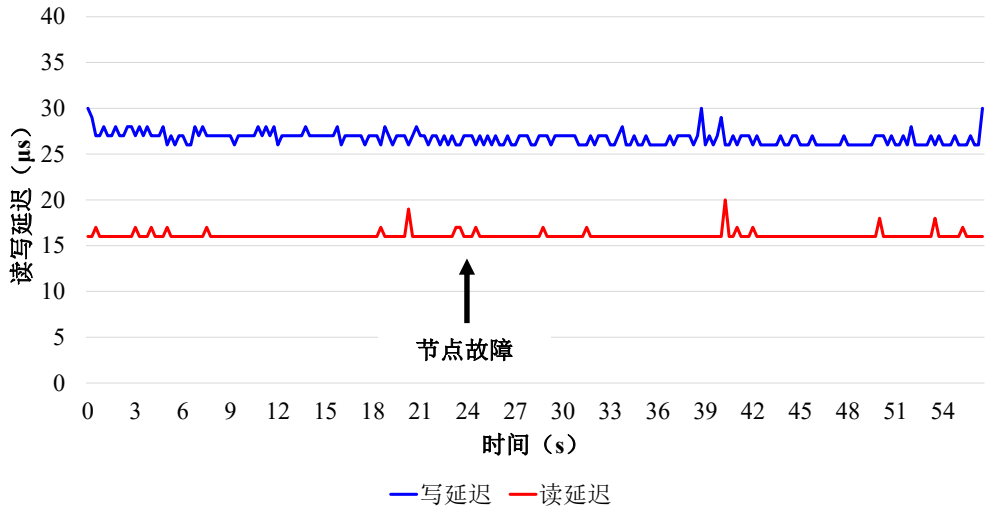
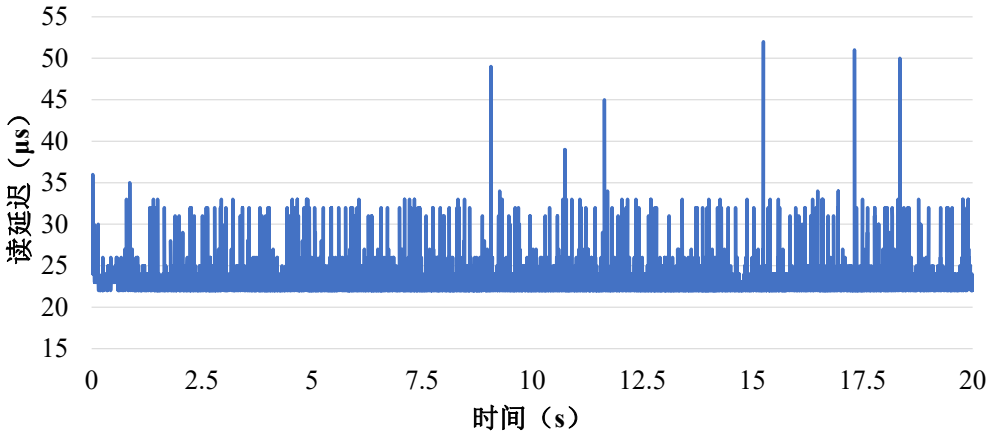


图 4.3　节点故障前后的 4KB 读写延迟曲线

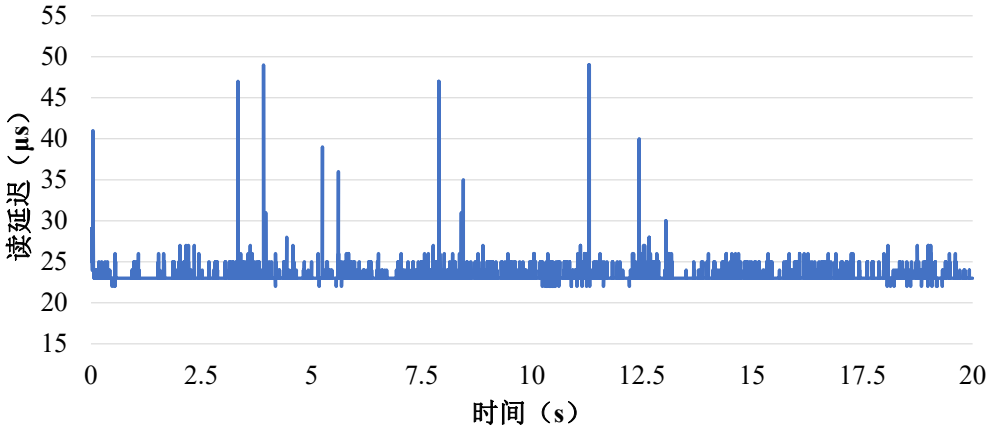
可见，在节点故障前后，系统的读写延迟没有明显改变，客户端仍然能正常执行文件系统操作。这表明系统的容错能力符合预期。

4.3.2 前-k 读取策略对读取性能稳定性的影响

本小节测试前-k 读取策略对读操作性能稳定性的影响。客户端进行两次约 20 秒的 4KB 文件读取，其中一次使用前-k 读取策略，另一次不使用该策略，而是随机选择恰好 k 个节点读取数据，作对照组。每次均测得约 112.0 万次读取延迟数据，绘制曲线如下：



(a) 禁用前-k 读取



(b) 使用前-k 读取

图 4.4　禁用和使用前-k 读取策略时的读操作延迟曲线

在不使用该策略时，读延迟平均值为 22.38 μs，读延迟曲线产生了明显抖动，有大量的读操延迟了 30 μs。使用该策略时，读延迟平均值为 22.16 μs，延迟曲线虽然仍然出现了一定的抖动，但少有延迟超过 30 μs 的情况发生，读性能的不稳

定性明显较小。可见，在 RDMA 网络环境下，前-k 读取策略虽然不能明显降低读操作延迟，但能明显使性能表现更加稳定。

第 5 章　结论

本课题的初衷在于证明，对于使用 NVM 作为持久性存储介质、RDMA 作为网络通信协议栈的分布式文件系统，为了提供数据可用性保障而应用纠删码策略是一个可行的选择。为此，本课题实现了一个基于 NVM+RDMA，并且使用纠删码提供高可用性保障的文件系统原型。接下来，本课题对纠删码编解码和传输延迟、文件系统读写延迟和带宽，以及文件系统可用性水平进行了测试。测试结果表明，使用纠删码虽然引入了额外的计算开销，但相较于传统多副本策略，文件系统的读性能有最多 30% 的降低，而写性能有最多 11% 的提升。这表明纠删码策略的性能并不逊色于传统的数据备份策略，揭示了分布式文件系统中应用纠删码作为可用性策略的潜力。

然而，综观本课题的设计、实现与测试，其中仍然存在一些主要问题：

(1) 本课题利用 LocoFS 实现了一个文件系统，但 LocoFS 本身的操作延迟较大，不能代表分布式文件系统领域的最新技术水平。由于这与我的主要研究目标关系不大，同时也因为技术和时间有限，本课题并未将文件系统操作逻辑的优化作为研究重点，而是直接使用了现成的 LocoFS 源代码。最近发表的一些工作使用了更加合理的空间分配策略[9]，也引入了完善的缓存机制以减小延迟[10]；这使它们的写延迟降低到约 10 μs，读延迟则降低到 3 ～ 4 μs。因此，在数据读写延迟与纠删码的编解码延迟处于同一个数量级时，本课题的研究结论在何种程度上仍然有效，尚需要进一步研究；

(2) 本课题在测试时使用的服务器集群规模非常小。实际的系统中通常包含大量的节点，例如，一项已有工作在多达 300 ～ 400 个节点构成的缓存系统上应用了 Reed-Solomon 编码[12]。在这一规模的系统中，通常还需要引入一致性哈希等机制，方能有效处理空间分配、存储节点增删等情况。因此，基于 NVM+RDMA 的大规模分布式系统中纠删码应当如何应用，尚需要进一步研究；

(3) 本课题实现的纠删码策略只有 2 种，且均为原型设计。然而，工程中实际使用的是更为复杂的纠删码编码方案。例如，1.2.2 小节中提到了 Facebook 公司的 f4 系统，该系统虽然使用朴素的 Reed-Solomon 编码，但使用了复杂的存储空间分配模式，以应对实际应用中不同规模的故障；同样于该小节中提

到的 Windows Azure 存储系统则使用了一个称为 LRC（Local Reconstruction Code）的编码策略，它利用大多数故障都是单节点故障的统计规律，对数据进行二级编码，有效降低了编码和数据恢复的开销。因此，使用更为复杂的纠删码系统时，本课题的研究结论在何种程度上仍然有效，尚需要于进一步研究。

综上所述，本课题的研究成果尚有一些不完整之处，今后应当按照这些方向加以改进。在有一定改进空间的同时，本课题确实地证明了使用纠删码构建高可用分布式持久内存文件系统的可行性和有效性，完成了预设的研究目标。

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/OL]//BERSHAD B N, MOGUL J C. 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA. USENIX Association, 2006: 307-320. http://www.usenix.org/events/osdi06/tech/weil.html.

[2] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[C/OL]//KHATIB M G, HE X, FACTOR M. IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010. IEEE Computer Society, 2010: 1-10. https://doi.org/10.1109/MSST.2010.5496972.

[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/OL]//NOH S H, WELCH B. 18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020. USENIX Association, 2020: 169-182. https://www.usenix.org/conference/fast20/presentation/yang.

[6] XU J, SWANSON S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories[C/OL]//BROWN A D, POPOVICI F I. 14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016. USENIX Association, 2016: 323-338. https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu.

[7] XU J, ZHANG L, MEMARIPOUR A, et al. NOVA-Fortis: A fault-tolerant non-volatile main memory file system[C/OL]//Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017. ACM, 2017: 478-496. https://doi.org/10.1145/3132747.3132761.

[8] LU Y, SHU J, CHEN Y, et al. Octopus: an RDMA-enabled distributed persistent memory file system[C/OL]//SILVA D D, FORD B. 2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017. USENIX Association, 2017: 773-785. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu.

[9] YANG J, IZRAELEVITZ J, SWANSON S. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks[C/OL]//MERCHANT A, WEATHERSPOON H. 17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019. USENIX Association, 2019: 221-234. https://www.usenix.org/conference/fast19/presentation/yang.

[10] ANDERSON T E, CANINI M, KIM J, et al. Assise: Performance and availability via NVM colocation in a distributed file system[J/OL]. CoRR, 2019, abs/1910.05106. http://arxiv.org/abs/1910.05106.

[11] ZHANG H, DONG M, CHEN H. Efficient and available in-memory KV-store with hybrid erasure coding and replication[C/OL]//BROWN A D, POPOVICI F I. 14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016. USENIX Association, 2016: 167-180. https://www.usenix.org/conference/fast16/technical-sessions/presentation/zhang-heng.

[12] WANG A, ZHANG J, MA X, et al. Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache[C/OL]//NOH S H, WELCH B. 18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020. USENIX Association, 2020: 267-281. https://www.usenix.org/conference/fast20/presentation/wang-ao.

[13] LEE Y, MARUF H A, CHOWDHURY M, et al. Mitigating the performance-efficiency tradeoff in resilient memory disaggregation[J/OL]. CoRR, 2019, abs/1910.09727. http://arxiv.org/abs/1910.09727.

[14] HUANG C, SIMITCI H, XU Y, et al. Erasure coding in Windows Azure storage[C/OL]//HEISER G, HSIEH W C. 2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012. USENIX Association, 2012: 15-26. https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang.

[15] SUBRAMANIAN M, LLOYD W, ROY S, et al. f4: Facebook's warm BLOB storage system [C/OL]//FLINN J, LEVY H. 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014. USENIX Association, 2014: 383-398. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/muralidhar.

[16] GOGTE V, WANG W, DIESTELHORST S, et al. Software wear management for persistent memories[C/OL]//MERCHANT A, WEATHERSPOON H. 17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019. USENIX Association, 2019: 45-63. https://www.usenix.org/conference/fast19/presentation/gogte.

[17] LI S, LU Y, SHU J, et al. Locofs: a loosely-coupled metadata service for distributed file systems [C/OL]//MOHR B, RAGHAVAN P. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017. ACM, 2017: 4:1-4:12. https://doi.org/10.1145/3126908.3126928.

[18] DIMAKIS A G, GODFREY B, WU Y, et al. Network coding for distributed storage systems [J/OL]. IEEE Trans. Inf. Theory, 2010, 56(9):4539-4551. https://doi.org/10.1109/TIT.2010.2054295.

[19] VAJHA M, RAMKUMAR V, PURANIK B, et al. Clay codes: Moulding MDS codes to yield an MSR code[C/OL]//AGRAWAL N, RANGASWAMI R. 16th USENIX Conference on File and Storage Technologies, FAST 2018, Oakland, CA, USA, February 12-15, 2018. USENIX Association, 2018: 139-154. https://www.usenix.org/conference/fast18/presentation/vajha.

[20] LIU Z, BAI Z, LIU Z, et al. DistCache: Provable load balancing for large-scale storage systems with distributed caching[C/OL]//MERCHANT A, WEATHERSPOON H. 17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019. USENIX Association, 2019: 143-157. https://www.usenix.org/conference/fast19/presentation/liu.

[21] ZHANG Y, YANG J, MEMARIPOUR A, et al. Mojim: A reliable and highly-available non-volatile memory system[C/OL]//ÖZTURK Ö, EBCIOGLU K, DWARKADAS S. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015. ACM, 2015: 3-18. https://doi.org/10.1145/2694344.2694370.

[22] 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

感谢导师舒继武教授、陆游游助理教授为我的研究全程提供指导和建议。

感谢吕文豪学长为我提供了 LocoFS 的源代码和使用指导。感谢汪庆学长、朱博弘学长为我提供测试服务器，并且帮助我解决在使用过程中遇到的各种问题。

感谢 ThuThesis[22] 提供了功能齐全的模板，极大地方便了本文的撰写工作。

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[4].

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[4]. 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[1].

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*[2], *Orion*[4], and *Assise*[1] 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[4];
- 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[1].

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[3], 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.

参考文献

[1] Thomas E. Anderson, Marco Canini, Jongyul Kim, Dejan Kostic, Youngjin Kwon, Simon Peter, Waleed Reda, Henry N. Schuh, and Emmett Witchel. Assise: Performance and availability via NVM colocation in a distributed file system. *CoRR*, abs/1910.05106, 2019. URL http://arxiv.org/abs/1910.05106.

[2] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an rdma-enabled distributed persistent memory file system. In Dilma Da Silva and Bryan Ford, editors, *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*, pages 773–785. USENIX Association, 2017. URL https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu.

[3] Jian Xu and Steven Swanson. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In Angela Demke Brown and Florentina I. Popovici, editors, *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016*, pages 323–338. USENIX Association, 2016. URL https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu.

[4] Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A distributed file system for non-volatile main memory and rdma-capable networks. In Arif Merchant and Hakim Weatherspoon, editors, *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019*, pages 221–234. USENIX Association, 2019. URL https://www.usenix.org/conference/fast19/presentation/yang.

综合论文训练记录表

学生姓名		学号		班级	
论文题目					
主要内容以及进度安排	指导教师签字： 考核组组长签字： 年　　月　　日				
中期考核意见	考核组组长签字： 年　　月　　日				

指导教师评语	指导教师签字：_____ 年　　月　　日
评阅教师评语	评阅教师签字：_____ 年　　月　　日
答辩小组评语	答辩小组组长签字：_____ 年　　月　　日

总成绩：_____

教学负责人签字：_____

年　　月　　日