带制程差异的非易失内存文件系统索引节点磨损优化研究

重庆大学硕士学位论文

（学术学位）

学生姓名：王鑫鑫

指导教师：刘　铎 教授

学科门类：工　学

学科名称：计算机科学与技术

研究方向：内存文件系统

答辩委员会主席：陈　武 教授

授位时间：2020 年 6 月

Exploring Process Variation Aware Wear Leveling Mechanism for Inodes of NVM-based In-memory File System

A Thesis Submitted to Chongqing University

In Partial fulfillment of the requirement for the

Master's Degree of Engineering

By

Xinxin Wang

Supervised by Prof. Duo Liu

June, 2020

摘　要

作为新一代持久化数据存储，新型非易失性内存 NVM（Non-Volatile Memory）为存储系统的设计带来了新的机遇与挑战。目前已有多个以 NVM 为存储介质的非易失内存文件系统。由于未充分考虑 NVM 存储单元耐受度受限的特点，使得这些文件系统在大量文件操作后，极易造成对 NVM 设备的磨损不均衡现象，尤其是文件索引节点所在的元数据区域。硬件制程差异导致的耐受度差异使得这一问题更加严峻，即写次数相同时，耐受度低的存储单元更容易被写穿。

本文针对导致上述问题的原因进行分析和讨论，将文件系统的索引节点区域划分为多个耐受度不同的存储域，设计并提出制程差异感知的索引节点空间管理策略以及磨损均衡机制 Contour。借助索引节点虚拟化技术，在存储域之间以及存储域内部执行索引节点的动态迁移，以在耐受度不同的存储单元之间实现磨损率均衡。主要研究内容可分为如下几个部分：

① 针对现有索引节点组织结构导致的头尾磨损不均等问题，提出域间差异感知的空间管理策略，使索引节点均匀分布在耐受度不同的存储域之间，进而达到缓解磨损不均衡、降低索引节点迁移开销的目的。

② 针对存储域之间的耐受度差异，提出域间磨损均衡机制。首先根据文件系统历史运行情况将索引节点与存储域进行动态匹配，以尽可能将访问热度较高的索引节点保存至耐受度较高的存储域。其次以运行周期为单位，对每个存储域的可用写次数加以限制，以达到周期性磨损率均衡的目的。

③ 针对存储域内的索引节点写差异，提出域内磨损均衡机制。设计索引节点迁移算法以在存储域内执行索引节点的动态迁移。存储域最优匹配算法可在相邻存储域之间分散写操作，进而避免对单个存储域造成过度磨损。

本文基于真实非易失内存文件系统 SIMFS，实现了 Contour 的原型并编译到 Linux 内核中。使用标准测试工具 Filebench、FIO 以及数据库应用程序 MySQL 对 Contour 的磨损分布和性能开销进行评估。实验结果表明，Contour 在四种不同的负载模式下，以物理页为单位的磨损率标准差平均优于 NoWL 和当前最优的 PCV 磨损均衡机制 644.8 倍和 11.0 倍，能够有效均衡索引节点区域的磨损分布，且平均性能开销仅为 3.85%。

关键词：非易失性内存；内存文件系统；制程差异；索引节点；磨损均衡

Abstract

With the development of emerging non-volatile memory (NVM), many NVM-based in-memory file systems are proposed to exploit the advanced characteristics of NVM. Since the limited write endurance of NVM is not fully considered, these file systems may cause non-uniform wear distribution on NVM after performing a large number of file operations, especially for the inode section. Endurance variation caused by NVM's process variation makes things worse since uniform write distribution will cause the weakest storage domain to be worn out much earlier.

In this thesis, we discuss the wear problem for the inode section, and propose a wear leveling mechanism, called Contour, including process variation aware space management scheme and wear leveling mechanism for inodes. Based on the inode virtualization technology, inodes can be moved dynamically between and within domains to balance the wear rate of inode slots from different domains. The main contributions of this thesis include:

① The existing single-linked list of inodes may cause unbalanced wear across different storage domains. We propose a process variation aware space management scheme to distribute the inodes evenly among the storage domains.

② We propose a process variation aware wear leveling mechanism to balance the wear rate of different storage domains. First, dynamically migrate hot inodes to storage domains with higher endurance. Second, limit the number of writes to each storage domain in round to regularly balance the wear rate.

③ Slots in the same storage domain may have different wear rate. We design an algorithm to perform dynamic migration of inodes within storage domains. The best-fit algorithm can distribute writes between adjacent storage domains, thereby avoiding excessive wear on a single storage domain.

We implement the prototype of Contour in the NVM-based file system SIMFS, and compile it into the Linux kernel. Extensive experiments were conducted using standard benchmarks including Filebench, FIO and MySQL. The experimental results show that under four different workloads, Contour's standard deviation of page wear rate is better than that of NoWL and PCV by 644.8 times and 11.0 times, respectively. The average performance overhead of Contour is 3.85%.

Key words： NVM; In-memory File System; Process Variation; Inode; Wear leveling

目 录

1 绪 论

1.1 研究背景

大数据时代背景下，随着数据量的不断积累和技术的迭代更新，大量新型技术手段被逐渐提出并广泛应用，以此来满足使用者的特殊需求并不断提升用户体验，例如个性化推荐、自动驾驶、虚拟现实等。这些应用场景中存在的普遍特点是用户对响应时延的敏感度较高，这就要求上述应用在使用过程中必须要保证足够低的应用程序响应延迟。而与之相反的是，现有机器学习、图像识别等复杂算法的运行时间较长，无法满足用户对于实时应用的时延需求。现有主流操作系统仍使用基于磁盘介质的文件系统。当需要对大量历史数据进行处理时，受到磁盘介质本身读写带宽较低的限制，使得在数据得到处理之前需要消耗大量的时间用于将其从磁盘读入运行内存，从而极大地增加了算法的运行时间。

新型非易失性内存（Non-Volatile Memory, NVM）[1-4]的提出为这些复杂技术的广泛应用提供了新的机遇和挑战。一方面，学术界和工业界设计并实现了多个以 NVM 为存储介质的非易失内存文件系统[5-12]。区别于磁盘文件系统，这类文件系统充分利用 NVM 可字节寻址且可持久化存储的特性，优化了传统面向磁盘设备的 I/O 软件栈[13-14]，以提高文件系统的读写性能。替换低速的磁盘设备，而将新型非易失性内存作为文件系统的存储介质，能够极大程度上提高文件系统的读写性能，从而有效降低应用程序的响应时延。

另一方面，NVM 的缺陷同样不可忽略。与闪存设备相类似[15-19]，NVM 设备上的存储单元在写入新的内容之前，需要首先对已有数据进行擦除，并且这些存储单元存在耐受度限制[20-22]，即其可擦除次数是有限的。与此同时，由于硬件制造工艺的不同，导致 NVM 设备上的多个存储单元之间可能存在不同的耐受度[23-24]。这一特点得写次数相同时，耐受度较低的存储单元往往具有更加严重的磨损程度。由于现有非易失内存文件系统在设计和实现过程中并未对上述 NVM 硬件缺陷进行充分的考虑，使得文件系统使用过程中极易出现对 NVM 设备磨损不均衡的现象，尤其是索引节点所在的元数据区域，进而缩短设备的使用寿命。

在上述背景下，本文将研究问题定义为：在具有制程差异的 NVM 存储设备上，非易失内存文件系统应如何实现索引节点区域的磨损均衡？探讨分析造成文件系统索引节点区域磨损严重不均衡的根本原因，并尝试从空间管理策略和磨损均衡机制两个角度进行优化和改进，进而最大限度提升设备的使用寿命。

1.2 国内外研究现状

本节首先对新型非易失性内存 NVM 的典型特征进行说明，并简要介绍基于 NVM 的相关研究工作及其进展。其次，针对 NVM 写耐受度受限的问题，列举了现有部分非易失内存文件系统中所推荐采用的磨损均衡思路。之后针对由于制程差异而导致 NVM 设备上存在的耐受度差异特征，阐述其相关研究进展。

1.2.1 新型非易失性内存及相关研究

近年来，新型非易失性内存 NVM，如相变存储器(PCM, Phase Change Memory)[1]、3D Xpoint[2]、铁电随机存储器(FeRAM, Ferroelectric Random Access Memory)、阻变随机存储器(RRAM, Resistive Random Aceess Memory)[3]、磁性随机存储器(MRAM, Magnetic Random Access Memory)[4]等，受到学术界和工业界的广泛关注，并逐渐发展成为存储级内存[25-26]。NVM 能够直连内存总线，并且可按字节进行寻址。区别于动态随机存储器 DRAM，其主要特征在于能够支持数据的持久化存储，即保存在 NVM 中的数据不会由于发生掉电故障而丢失。该特征使得 NVM 能够作为计算系统的内存，并且成为未来最有可能替换 DRAM 的非易失性存储设备之一，尤其是在数据量较大、以及对功耗和持久化有特殊要求的应用场景当中[27-28]。

另一方面，NVM 也可作为计算机系统的外存，或与 DRAM、闪存等介质共同组成混合存储结构，以提高系统的整体性能。区别于低速的磁盘或闪存介质，NVM 设备所提供的内存级别读写延迟，能够有效降低或避免数据从低速存储设备读入到运行内存的时间开销，同时功耗更低。

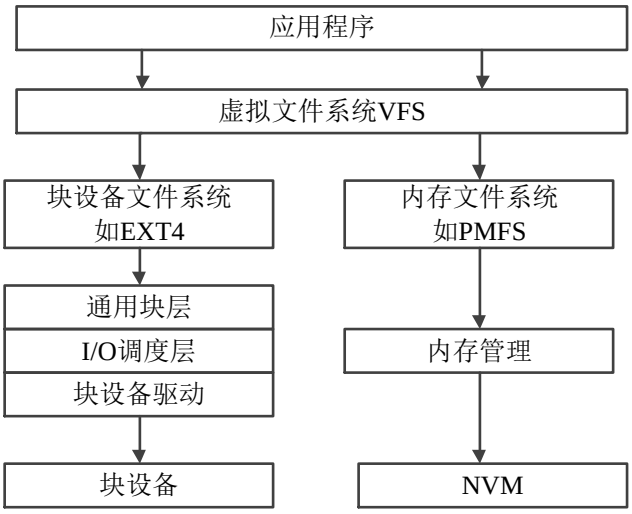


图 1.1 文件系统架构对比

Fig.1.1 Comparison of file system architecture

非易失内存文件系统是 NVM 的典型应用之一。如图 1.1 所示，区别于面向磁盘等传统低速设备而设计优化的文件系统，非易失内存文件系统的软件流程中无须涉及到对复杂且耗时的通用块层、I/O 调度层和块设备驱动层的调用，从而简化 I/O 流程，提高文件系统性能。

目前已有多种基于 NVM 内存架构的非易失内存文件系统[5-12]。其中 Intel 提出的内存文件系统 PMFS[9]主动绕开页高速缓存，直接在用户缓冲区和文件数据区之间进行数据传输操作，以降低软件路径开销。NOVA[10]是一款基于日志的非易失内存文件系统，通过结构化日志（Log-structured）的使用能够同时提供对文件数据和元数据的原子操作。重庆大学沙行勉教授团队提出的 SIMFS 文件系统[5]则提出了“文件虚拟地址”的概念，借助进程内核地址空间和内存管理单元 MMU，加速从文件虚拟地址到物理地址的转换，进而大幅提高文件系统读写性能。

与此同时，NVM 也存在部分缺陷，相关研究可分为如下几类。NVM 设备的读写延迟与 DRAM 仍有部分差异。以相变存储器 PCM 为例，PCM 读操作的延迟与 DRAM 为同一数量级，但写操作延迟约为 DRAM 的 10 倍[27]。由于 NVM 硬件技术尚未成熟，因此目前已有相关研究工作[9, 29-31]致力于如何更好地使用 DRAM 模拟 NVM，尤其是写时延的模拟，以尽可能反映出 NVM 硬件的真实性能情况，进而使得文件系统性能更加准确。

其次，NVM 设备上的存储单元存在耐受度限制，即存储单元的写次数是有限的。若某个存储单元的累计写次数超出其耐受度的限制，则会导致这个存储单元被写穿，进而缩短设备的使用寿命。目前相关研究工作主要通过减少对 NVM 的写次数或写入数据量[32-33]、引入磨损均衡机制等方式，对该问题进行优化和改进。NVM 持久化特性可能引入的数据一致性问题也被广泛研究。非持久化的 CPU 缓存与非易失内存之间的更新粒度有所差异，并且现有处理器可能对写 NVM 操作进行重新排序以优化性能。因此发生掉电故障时，CPU 缓存中的数据可能无法及时地写回至 NVM，进而导致数据不一致问题。目前相关工作主要从基于 NVM 构建的持久索引、文件系统以及持久性事务等方面进行数据一致性研究[28]。

1.2.2 非易失内存文件系统中的磨损均衡

非易失内存文件系统运行过程中，任何对文件元数据或数据的更新操作，均会产生对底层存储设备的磨损，并且不同区域磨损程度差异较大。因此磨损均衡问题在非易失内存文件系统的设计和实现过程是不可忽略的。目前已有部分相关工作指出其对该问题的思考及其采用的解决办法。

其中，微软提出的 BPFS[12]文件系统中提到可通过定期将虚拟内存页面与物理内存页面进行随机交换的方法来实现物理页层面的磨损均衡，而页内则采用随机翻转标志位的方法避免对同一个存储单元的频繁磨损。由于不同应用程序所导致

的文件读写模式不同，且不同区域之间磨损差异较大，因此规则相对复杂，仅以随机的方式不能够满足磨损均衡的实际需求。HMVFS[34]指出可通过结构化日志限定文件元数据段和数据段被写入存储设备的顺序，以此来简化磨损均衡方案的设计。Chen 等人提出以索引节点为单位在 NVM 物理页之间进行迁移的磨损均衡机制 PCV[35]，通过给定的阈值来保证物理页写次数的相对均衡。Marching[36]则以保存索引节点的容器为单位，通过判断特定条件是否满足以不断增加或减少滑动窗口的容量大小。运行过程中不断将更新频繁的索引节点与窗口中其他索引节点进行交换，以实现所有容器间的写磨损均衡。这些研究工作由于并未充分考虑 NVM 硬件存在的耐受度差异，因此往往无法实现合理的磨损均衡效果。

1.2.3 制程差异感知的磨损均衡

硬件制程差异导致 NVM 上多个存储单元的耐受度不同。类似地，闪存设备也存在耐受度差异的问题，且已有相关研究工作[15-18]致力于提升闪存性能并延长设备使用寿命。但这些技术往往实现在闪存设备的控制器硬件当中，而非软件实现，并且闪存设备的读写粒度与 NVM 可字节寻址的特征存在较大差异，因此面向闪存设备的磨损均衡技术无法被直接应用于 NVM 设备。

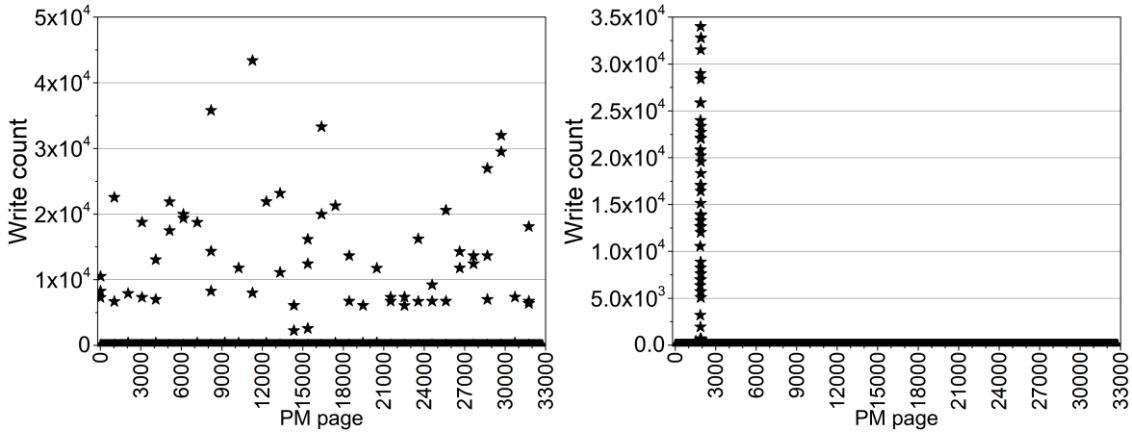
目前基于非易失性内存耐受度差异的相关研究工作中，WRL[37]是一种实现在硬件层面的基于数据迁移的磨损均衡机制。该机制在运行周期中首先记录并分析一定数量的历史写操作，以得到当前应用程序访问数据的冷热分布情况。其次结合硬件的磨损状态进行最优化建模，进而以最小的数据迁移代价尽可能地延长设备的使用寿命。Yun 等人提出的磨损均衡方案[38]在避免将热数据映射到耐受度较低的存储单元的同时，使用布隆过滤器（Bloom filter）以减少磨损均衡方案中写计数器引入的额外开销。这两种方案均依赖于对历史数据访问冷热程度的分析和预测，因此在恶意程序的攻击下无法发挥理想的效果。TWL[39]则提出将耐受度高低不同的两个物理页进行“绑定”。当需要写入新的数据时，以较高的概率写入耐受度较高的页，以较低的概率写入耐受度较低的页。这样的做法使得该方案既能够对耐受度差异有所感知，并且能够抵挡恶意磨损程序的攻击。此外，Zhao 等人[40-41]指出单个存储单元中保存的信息密度越高，总容量越大，耐受度则大大降低。因此其在运行过程中根据数据访问的冷热情况，动态地调整单个存储单元中所允许的最大信息密度，从而得以在存储容量和耐受度之间进行权衡。

1.3 研究动机和意义

非易失内存文件系统中索引节点所在区域的磨损分布不均衡问题尤为突出。在常用的文件操作中，文件索引节点的更新频率远高于其他区域。以写文件（write）操作为例，除了将最新的数据内容写入文件之外，还需要对索引节点中的文件大

小、修改时间等属性进行更新。创建（create）和删除（delete）文件操作，需要对索引节点所在存储单元进行写入或擦除，修改权限、修改所有者（chmod/chown）等操作同样仅需要修改文件的索引节点。

以文件系统 SIMFS 和 PMFS 为例，选取文件系统测试工具 Filebench[42]提供的 OLTP 负载进行时长 1 分钟的读写测试，文件系统索引节点所在区域的磨损分布结果如图 1.2 所示。OLTP 负载常用于模拟数据库的典型读写模式，配置有 50 个读进程、10 个写进程以及 1 个日志进程。其中写进程和日志进程分别对多个数据文件和单个日志文件进行频繁写操作。从图中可以看出明显的磨损分布不均衡现象。文件系统 SIMFS 和 PMFS 中存在磨损的页面个数占总页面个数的比例仅分别为 1.26%和 1.18%。以图 1.2(a)为例，文件系统 SIMFS 运行 OLTP 负载 1 分钟后，仅有 59（34）个页面的磨损次数大于 1000（10000），而剩余页面的写次数极少，或不存在写磨损。



(a) OLTP on SIMFS (b) OLTP on PMFS

图 1.2 SIMFS 和 PMFS 文件系统运行 OLTP 负载后的磨损分布情况

Fig.1.2 Write distribution of OLTP on SIMFS and PMFS

NVM 设备上不同存储单元之间的耐受度差异使得索引节点区域的磨损不均衡问题更加严重。写次数相同时，耐受度低的存储单元磨损程度更加突出，严重缩短了设备的使用寿命。与此同时，现有相关研究工作可大致分为两类。一类由于未充分考虑不同存储单元之间的耐受度差异，因而无法实现理想的均衡效果。另一类通常依赖于硬件实现或无法应用于内存文件系统中对索引节点的管理。该类机制可扩展性较差，往往仅以单一的方式（如粒度、迁移策略）进行调整，并且由于缺乏对数据类型的感知，使其所能够实现的磨损均衡效果有一定的局限性。为此，本文主张在文件系统这一系统软件层面实现细粒度、且更加灵活的磨损均衡机制，针对制程差异环境下的文件系统索引节点磨损不均衡问题进行研究。

通过对该问题的研究和探讨，能够进一步暴露出现有非易失内存文件系统对底层存储设备的磨损缺陷，并且在制程差异环境下为文件系统的改进提供了新的思路和挑战。从实践的角度看，本文在文件系统层面提出的磨损均衡机制能够在引入极少量性能和空间开销的同时，有效均衡索引节点区域的磨损分布。这使得面向大数据的复杂应用程序或算法在从磁盘文件系统迁移至非易失内存文件系统的过程中，能够获得极大性能提升的同时，有效延长设备的使用寿命。

1.4 论文主要研究内容

本文针对制程差异环境下非易失内存文件系统中的索引节点磨损不均衡问题进行研究，提出制程差异感知的索引节点磨损均衡机制 Contour。结合非易失内存文件系统中的索引节点组织结构以及访问模式，对导致磨损不均衡问题的根本原因进行分析和探讨，并分别对索引节点空间管理策略和磨损均衡机制进行优化。首先，本文提出域间差异感知的多链表空间管理策略，以避免传统基于单链表的空间管理策略中，频繁分配和回收操作可能导致的头尾磨损分布不均、操作效率降低等问题。

其次，针对存储域之间的磨损不均衡现象，本文提出域间磨损均衡机制。为了将访问热度较高的索引节点尽可能保存至耐受度较高的存储域，本文根据历史运行情况为索引节点和存储域分别计算权重，并将两者进行动态匹配。之后以运行周期为单位，动态调整存储域的匹配优先级，以实现周期性地磨损率均衡。域内磨损均衡机制用于实现存储域内的磨损均衡。索引节点迁移算法以尽可能少的迁移次数在存储域内执行索引节点的动态迁移，存储域最优匹配算法则可将当前存储域的写操作分散至相邻存储域，进而避免对单个存储域造成过度磨损。

最后，将 Contour 实现并成功编译至 Linux 内核中。借助文件系统测试工具 Filebench[42]、FIO[43]和数据库应用程序 MySQL[44]对所提出的磨损均衡机制进行了实验测试。测试主要从磨损分布、性能开销和参数敏感度三个方面进行。实验结果表明，在上述多个组件的共同帮助下，Contour 能够以较低的开销，在存储域和物理页层面均达到较优的磨损均衡状态。并且随着运行时间的增加，磨损分布依然呈现出近似水平增加的变化趋势。

1.5 论文组织结构

本文的组织结构如下：

第一章首先介绍了非易失性内存 NVM 的相关研究背景，指出制程差异环境下非易失文件系统中存在的索引节点磨损不均衡问题。之后给出国内外研究现状、论文的研究动机以及论文组织结构。

第二章对非易失性内存制程差异进行概述，并介绍了文件系统中典型的索引节点组织结构以及现有索引节点虚拟化技术。

第三章详细介绍了磨损均衡机制 Contour，包括空间管理策略、域间磨损均衡机制以及域内磨损均衡机制。最后给出实现过程以及关键问题分析。

第四章说明了测试对象、测试工具和测试方法，并分别对文件系统的磨损分布、性能开销和参数敏感度进行实验验证和结果分析。

第五章对全文进行总结和展望。

2 技术背景

本章分别对非易失性内存制程差异、文件系统中索引节点的常用组织结构、以及索引节点虚拟化技术进行概述。

2.1 非易失性内存制程差异概述

除写次数存在限制外，硬件制程差异导致的耐受度差异也是 NVM 的一大特性[23-24, 45-46]。硬件制造工艺上的细微差异，导致同一块非易失性内存上多个存储单元的耐受度有所不同。现有大部分磨损均衡方案通过迁移操作均衡多个存储单元的写次数，以延长设备的使用寿命。但由于忽略了不同存储单元之间的耐受度差异，使得写次数相同的多个存储单元，磨损程度却不同。

制程差异导致的耐受度差异特征，使得非易失性内存可被划分为多个连续但耐受度不同的存储域（Storage Domain），且同一存储域内的多个存储单元具有相同的耐受度[23, 46]。多个连续存储域的耐受度服从线性分布，例如大小 2GB 的非易失性内存，多个存储域的耐受度分布范围是3.0×10^6至1.7×10^8，即最大的耐受度数值约为最小耐受度的 56.7 倍[45-46]。因此在不考虑耐受度差异的磨损均衡方案中，同等量级的写次数将使得耐受度低的存储域磨损程度更加严重，也即更容易被磨损穿，从而无法发挥延长设备使用寿命的效果。

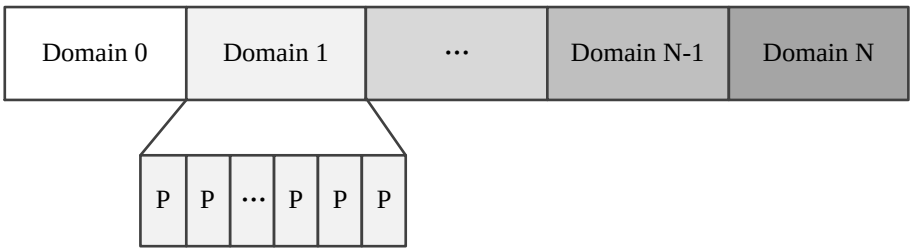


图 2.1 非易失性存储设备的存储域划分情况

Fig.2.1 Storage domain distribution of NVM device

耐受度存在差异的非易失性内存设备，其存储域的大小取决于设备的具体参数信息。本文以 4MB[45]作为单个存储域的大小，即单个存储域包含有 1024 个物理页（4KB 为一页）。在这种情况下，存储域划分的具体情况如图 2.1 所示。图中的每个物理页 P 均被划分为多个大小相同的存储单元，用于保存索引节点。文件系统中的索引节点区域包含有多个耐受度不同的存储域 Domain 0-N。例如，对于 256GB 的非易失性存储设备，索引节点所在的区域大小约为 2.56GB（文件系统通

常保留 1%的空间用于保存索引节点），也即该区域中包含约 656 个耐受度不同的存储域。

如上所述的 NVM 硬件耐受度差异特性，使得已有磨损均衡方法有了进一步优化和改进的空间。在现有磨损均衡方法的基础上，不仅要实现同一存储域内多个存储单元写次数的分布均衡，同时也要保证不同存储域的磨损程度尽可能一致，才能够有效发挥磨损均衡的效果，最终延长设备使用寿命。

2.2 文件系统中索引节点的常用组织结构

索引节点（Inode）是文件系统中最重要的元数据之一，每个索引节点即对应一个文件。索引节点保存了文件大小、创建/访问/修改时间等关键属性信息。上层应用程序通常使用绝对或相对路径来访问文件，如/home/a。底层文件系统会根据每一级的路径名称，以及当前文件系统内的索引节点组织结构，对该路径进行逐层解析，直至找到该文件对应的索引节点，并将打开后的文件描述符返回给应用程序。当对文件内容或文件属性信息有所修改时，需要由文件系统对相应的内容进行更新，并将最新的索引节点内容即时写回至对应的存储位置。

因此在访问更新文件的过程中，不同的索引节点组织结构对应着不同的访问路径，从而对索引节点所在区域的磨损也有所差异。本节给出文件系统中常用的两种索引节点组织结构，通过比较其在存储介质上的具体存储形式，以进一步分析造成索引节点磨损不均衡问题的主要原因。

2.2.1 基于 Array 的索引节点组织结构

文件系统中每个索引节点均有唯一的索引编号（Index number, ino)。访问索引节点的核心在于给定其索引编号，如何确定该索引节点的具体存储位置。因此最简单的方案是将文件系统中的所有索引节点保存在一段连续的物理区间，并记录该存储域的起始物理地址。访问某个索引节点时，只需要根据索引编号计算出偏移地址，便可得到其在物理设备上的具体存储位置。文件系统 NOVA[10]、SIMFS[5]和 SCMFS[7]均使用该结构保存索引节点。

该组织结构实现简单，无需额外的维护代价，并且能够始终以 O(1)的时间复杂度对某个索引节点进行访问。其缺点在于扩展性较差。文件系统在初始化阶段便需预留一定大小的空间用于存储索引节点，比如总空间大小的 1%。因此运行过程中索引节点所在存储域的大小是固定的。对于某些大型系统而言，当文件系统需要保存的文件数量超出当前索引节点区域的最大限制后，必须要对索引节点所在区域进行扩充才可继续运行。因此，基于 Array 的索引节点组织结构访问效率较高，实现方便，但扩展性相对较差。

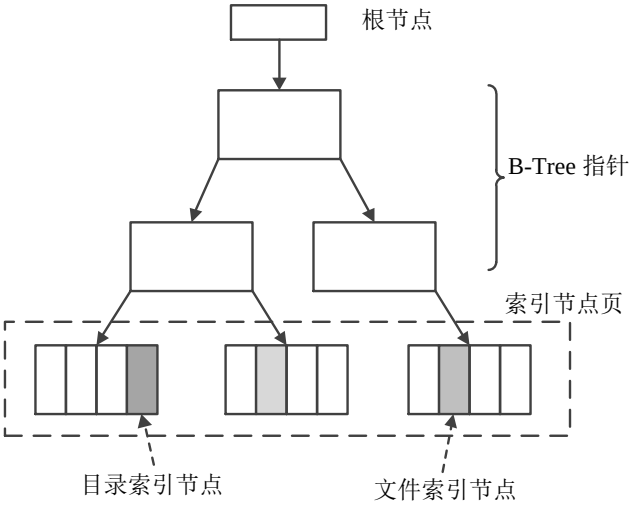


图 2.2 索引节点的树形组织结构

Fig.2.2 B-Tree structure of inode

2.2.2 基于 B-Tree 的索引节点组织结构

区别于预留固定区域保存索引节点的方法，PMFS[9]、SanGuo[8]、HiNFS[11]等文件系统则使用了基于树形的索引节点组织结构，其结构示例如图 2.2 所示。从图中可以看出，文件系统中索引节点所在的物理页面，以 B-Tree 结构进行组织。该组织结构虽实现相对复杂，但其主要优点在于可扩展性较强。当文件系统需要保存更多的文件，也即索引节点时，仅需分配新的空闲物理页，将其插入到树形结构中并重新调整即可。该结构可实现 O(logN)的索引节点访问时间复杂度，其中 N 为当前文件系统中索引节点所在物理页的个数。

索引节点组织结构的不同，使得文件系统对索引节点区域的磨损分布也会有所差异。大量文件读写操作后，基于 Array 的组织结构使得所有对索引节点的磨损均固定在存储设备上的某个固定区域。而基于树形结构的组织形式由于树中叶子节点上的物理页面是运行过程中动态分配的，因此保存索引节点的物理页可能分布在设备的多个不同位置，即最终的磨损分布相对分散。综上所述，两种典型的索引结构组织结构各有优缺点，并且可能产生不同的磨损分布特征。

2.3 索引节点虚拟化技术

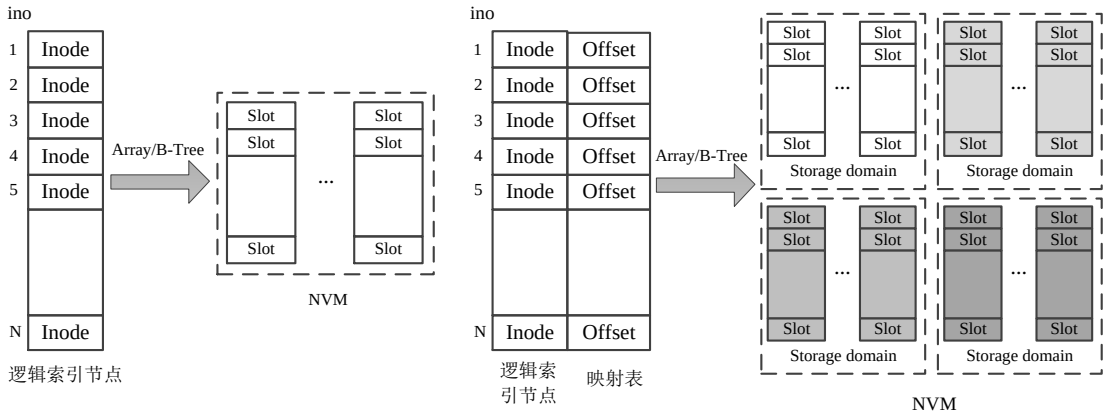
导致索引节点磨损严重的主要原因在于索引节点在整个生命周期中极少发生迁移，也即逻辑索引节点始终与某个固定的物理存储单元保持绑定关系。因此本节介绍索引节点虚拟化技术[35]，以解除逻辑索引节点与索引节点存储单元之间的对应关系，进而在此基础上实现基于迁移的索引节点磨损均衡方法。

如前文所述，文件系统中每个索引节点均有唯一的索引编号（ino）。应用程序

访问指定路径的文件时，文件系统需根据解析路径得到的 ino，找到该索引节点在物理设备上的具体存储位置。索引节点号码与存储位置之间的映射关系是由文件系统中索引节点的组织结构决定的。现有非易失内存文件系统中，从索引节点号码到实际存储位置的转换过程如图 2.3(a)所示。

图中左侧一列表示当前运行内存中的索引节点（也称为逻辑索引节点），均由唯一的索引节点号码来标识。右侧一列则表示当前存储设备上的物理组织结构。逻辑索引节点应与物理介质上的索引节点存储单元保持一一对应。从图中可以看出，在传统的索引节点组织结构中，给定索引节点编号，文件系统可通过 Array 或 B-Tree 结构的映射关系，快速定位到其在物理设备上的具体存储位置。

索引节点虚拟化技术通过在逻辑索引节点和物理存储单元之间增加映射表的方式，来实现索引节点的“虚拟化”。虚拟化后逻辑索引节点与物理存储单元之间的映射关系如图 2.3(b)所示。不同存储域之间的耐受度差异根据颜色的深浅加以区分。从图中可以看出，对于每个逻辑索引节点，文件系统均在映射表（Deflection table）中保存了该索引节点实际存储的物理偏移，之后便可通过 Array 或 B-Tree 的形式定位到具体的存储单元。在映射表的帮助下，索引节点与存储单元之间的绑定关系允许被动态修改，并且修改过程对于上层应用程序是透明的。



(a) Conventional inode section (b) Virtualized inode section

图 2.3 索引节点虚拟化前后映射关系对比

Fig.2.3 Comparison of conventional inode section and the virtualized inode section

使用上述索引节点虚拟化技术后，给定索引节点编号，文件系统访问物理索引节点的过程可分为如下三个步骤：① 文件系统通过唯一的索引节点编号和映射表，查询得到该索引节点对应的偏移信息；② 文件系统根据当前索引节点的组织结构，得到该索引节点对应的虚拟地址；③ 文件系统使用索引节点的虚拟地址，借助处理器中的硬件内存管理单元访问存储单元中的物理索引节点。

本文后续方案所涉及到的索引节点迁移操作是基于索引节点虚拟化技术实现的。实现过程中，映射表是一个与文件系统的逻辑索引节点相对应的连续数组，保存在一段预先分配的物理区间上。对于文件数量为 100 万的文件系统，建立映射表所需要的额外空间大小约为 3.8MB，空间开销较低。性能上，该虚拟化技术仅在访问索引节点过程中增加了一次额外的 NVM 访问，因此其开销对于相对耗时的文件读写操作而言可忽略不计。

2.4 本章小结

本章首先对新型非易失性内存 NVM 的硬件制程差异进行概述，旨在说明由制程差异导致的耐受度差异分布特征，及其对磨损均衡方案的影响。其次对非易失内存文件系统中两种典型的索引节点组织结构进行分析和对比。之后介绍了现有索引节点虚拟化技术，以及该技术对于不同索引节点组织结构的适用性分析。

3 制程差异感知的索引节点磨损均衡机制

本章提出制程差异感知的索引节点磨损均衡机制 Contour，意为等高线，旨在耐受度差异环境下实现近似水平的写磨损率，以缓解非易失内存文件系统中的索引节点磨损不均衡现象。索引节点虚拟化技术使得索引节点能够被动态地迁移，进而允许文件系统可基于迁移操作实现不同的磨损均衡。文件系统的索引节点区域通常包含多个耐受度不同的存储域，每个存储域包含多个连续且耐受度相同的物理页或存储单元。

首先给出磨损率（Wear ratio）的定义。给定非易失性存储设备上的物理页（或存储域），其磨损率为当前页（或存储域）累计写次数与其耐受度的比值。磨损率越高，表示对应区间的累计磨损次数越接近其耐受度，也即磨损程度更加严重。由于不同存储域内物理页的耐受度有所差异，因此理想情况下的磨损均衡机制应保证所有物理页均保持一致的磨损率，而非磨损次数。为此，本章首先对导致索引节点区域磨损不均衡问题的根本原因进行分析和讨论，其次将磨损率均衡这一目标分为三个主要部分进行设计与实现。

① 域间差异感知的索引节点空间管理策略，见 3.2 节；② 域间磨损均衡机制，通过跨存储域的索引节点迁移操作以及周期性地写次数限制，使得耐受度不同的多个存储域能够保持基本一致的磨损率，见 3.3 节；③ 域内磨损均衡机制，通过存储域内的索引节点迁移算法，使得所有物理页能够均衡分担当前存储域的全部写磨损，见 3.4 节。最后给出 Contour 磨损均衡机制在非易失内存文件系统 SIMFS 中的具体实现，并对实现过程中遇到的关键问题进行分析和说明。

3.1 索引节点磨损不均衡问题分析

本节对导致索引节点磨损不均衡问题的三点主要原因进行分析和讨论。

① 索引节点需要频繁地即时写回

应用程序在执行文件操作时，往往涉及到对文件数据、元数据和日志等区域的更新。现有非易失内存文件系统通常使用硬件原子操作，如 pm_barrier[9]和 PCOMMIT[10, 47-48]等指令，以确保文件的数据一致性。以对文件索引节点的更新为例，文件系统运行过程中，文件的索引节点会通过虚拟文件系统读取至运行内存。虚拟文件系统对索引节点的更新，首先被写入到运行内存，即 DRAM。由于内存掉电后数据无法持久化保存，为保证文件的数据一致性，需要实时地将运行内存中的索引节点写回到持久化存储介质上。作为文件系统中最重要的元数据之一，频繁地索引节点写回操作使得索引节点所在区域的磨损明显高于其他区域。

② 不同索引节点之间磨损程度差异较大

文件系统对底层存储介质的磨损分布，一定程度上取决于上层应用程序的数据读写模式。以邮件服务器为例，每个客户端在访问过程中会频繁读取大量文件，并周期性地向日志文件中写入少量数据。因此长时间运行后，日志文件所对应的索引节点的磨损次数将远高于其他索引节点。对于文件服务器而言，由于每个客户端均会执行大量的文件创建和删除操作，而涉及到的文件读写操作极少，因此在文件系统层面将执行大量的索引节点分配和释放操作。

除负载模式不同之外，考虑到文件系统往往使用链表的结构来管理空闲索引节点存储单元，因而频繁地分配和删除操作同样可能会导致链表头部索引节点的磨损次数明显高于链表尾部的索引节点。

③ 索引节点极少迁移

经过前两项分析，可知索引节点的频繁即时写回和应用程序负载模式的差异，导致索引节点所在区域的磨损明显高于其他区域，并且不同索引节点的磨损状况差异较大。该问题的主要根源在于文件系统内的索引节点极少发生迁移。一个大小为 4KB 的物理页中可以保存多个索引节点，也即其包含多个索引节点存储单元。现有大多数非易失内存文件系统中，文件索引节点自创建之后便被保存在一个固定的物理位置。在对文件系统中的少量文件进行频繁更新的过程中，由于文件索引节点的位置始终保持不变，因此所有对该索引节点的更新操作，直接造成了对底层存储单元的磨损，进而导致严重的磨损不均衡现象。综上分析，可得出缓解索引节点磨损不均衡问题的基本思路。

首先，应从索引节点分配策略的角度来避免造成对存储单元的磨损不均。其次，由于索引节点自文件创建之后便始终保存在固定的存储单元，使得现有文件系统无法通过迁移索引节点的方式来实现磨损均衡。因此需要借助索引节点虚拟化技术，将索引节点与保存索引节点的物理存储单元进行解耦合，使得索引节点能够实现动态地迁移。在域间差异感知的空间管理策略的配合下，磨损均衡机制不仅要考虑硬件制程差异导致的耐受度差异，还要兼顾不同应用程序之间可能存在的负载模式差异。

3.2 域间差异感知的空间管理策略

本节从文件系统对空闲索引节点的管理和分配策略出发，首先介绍了文件系统中基于单链表结构的索引节点空间管理策略及其优缺点，其次针对存储域之间耐受度差异的特点，提出域间差异感知的多链表空间管理策略。

3.2.1 基于单链表的空间管理策略

如研究背景所述，文件系统常使用 Array 或 B-Tree 数据结构存储和管理当前

所有有效的索引节点。而在此之前，则首先需要分配空闲的索引节点。也即当应用程序创建文件或目录时，文件系统首先需要从当前的空闲区域中为其分配一个空闲的索引节点，之后再对其进行统一的组织和管理。目前文件系统中常用的是基于单链表结构的空闲空间管理策略。

以 SIMFS 为例，该文件系统的索引节点区域被划分为多个连续且相同大小的存储单元，用于保存索引节点。这些空闲的存储单元以节点的形式组成链表，供文件系统分配和回收。PMFS 等采用树形组织结构的文件系统并未预留保存索引节点所需的空间，而是将运行过程中分配的物理页动态地加入到树形结构中进行统一管理。由于保存索引节点的物理页均以叶子节点的形式出现在树形结构中，因此所实现的索引节点分配和回收效果与单链表结构相类似。具体地，PMFS 文件系统中采用位置标记的形式，记录上一次分配或回收操作的位置信息，并始终从该位置开始顺序遍历以查找空闲的存储单元。

基于单链表结构的空间管理策略存在两个主要问题。一是由于该方案并未考虑不同存储域的耐受度差异，使得最终的磨损不均衡现象较为严重。具体地，以文件服务器 Fileserver 负载为例，由于链表中节点数量较多，且该负载运行过程中包含了频繁地索引节点分配和回收操作，因此最终对单链表头部节点造成的磨损较为严重，而对尾部节点产生的磨损几乎为 0。并且文件系统运行时间越长，该问题越为突出。二是该方案的分配效率较低。当应用程序的多个线程并发执行大量文件创建和删除操作时，由于仅存在一个空闲链表，线程之间极易产生加锁冲突，从而导致操作效率降低。

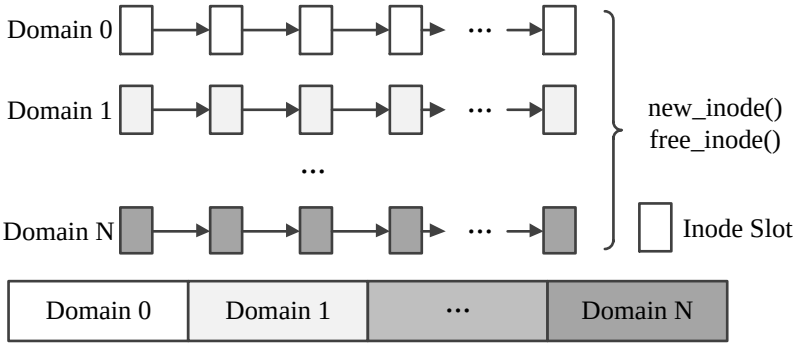


图 3.1 域间差异感知的多链表空间管理策略

Fig.3.1 Process variation-aware multi-linked list space management scheme

3.2.2 域间差异感知的多链表空间管理策略

为了避免单链表结构所导致的磨损分布不均衡、分配效率低等问题，本节提出域间差异感知的多链表空间管理策略。具体地，Contour 将全部索引节点存储单

元按照其所在存储域的耐受度情况，划分组成不同的链表，也即链表个数与存储域个数相同。基于多链表的空间管理策略如图 3.1 所示。

由于文件系统在创建索引节点时，无法得知该索引节点在后续时间内的访问和更新热度情况，因此创建文件时默认以轮询的方式在多个空闲链表中分配新的索引节点，以保证文件系统内的索引节点能够均衡地分布在不同存储域内。对应地，删除索引节点时则根据该存储单元的物理地址将其回收至对应的链表即可。同时，为了避免上述单链表中存在的头尾节点磨损不一致的问题，该策略中采取了头部分配、尾部回收的方案，以尽可能缓解磨损不均衡现象。

区别于基于单链表的分配管理策略，域间耐受度差异感知的空间管理策略能够在一定程度上缓解索引节点的磨损不均衡问题，并且有效降低单链表结构中出现资源冲突的可能性，尤其是在创建和删除操作较多的负载模式中。而在读写操作较多的负载模式中，如 OLTP、Webserver 和 MySQL 等，由于新的空间管理策略仅仅是改变了索引节点在创建时被保存的位置，而并未引入任何迁移操作，因此这类负载所造成的磨损分布有较为统一的变化特征。即以存储域或物理页为单位进行统计的写次数情况并未发生明显变化，而仅仅是部分索引节点由于其存储位置发生改变，导致写磨损分布更加分散。此外，由于文件系统内存储域个数较少，因此多链表方案所引入的管理开销可忽略不计。

3.3 域间磨损均衡机制

本节提出域间磨损均衡机制，以在多个耐受度不同的存储域之间实现同等水平的磨损率。首先定义 WBL（Write budget line）的概念。WBL 是一个可修改的磨损率参数，该参数与存储域耐受度（E_j）的乘积可用来表示一段时间内某个存储域（d_j）所允许的最大磨损次数，记为WB_j（Write budget）。

给定参数 WBL 的值（如 X%），耐受度不同的存储域所能够承受的磨损次数也不同。耐受度较高的存储域能够承受的磨损次数较多，反之则较少。因此为了使得多个存储域能够始终保持同等水平的磨损率，需要通过索引节点的迁移操作，使得各个存储域所承受的写次数不同。

一方面，由于不同应用程序所造成的索引节点冷热程度不同，并且访问热度较高的索引节点不能够总是被预先分配至耐受度较高的存储单元，因此 Contour 动态地将索引节点的访问热度与存储域的耐受度进行匹配，从而得以在存储域之间进行索引节点的迁移操作。

表 3.1 Contour 的符号定义

Table 3.1 The notations of Contour

符号	定义
N	表示存储域（Storage Domain）个数
N_p	单个存储域内物理页（Page）的个数
N_s	单个物理页内存储单元（Slot）的个数
I_i	第 i 个索引节点（Inode）
d_j	第 j 个存储域
E_j	第 j 个存储域的耐受度
T	表示时钟周期，Contour 将时钟周期的长度设置为 10s
R	表示运行周期，单个运行周期通常包含多个时钟周期 T
WBL^R	第 R 个运行周期的磨损率参数 WBL
$DF^T(I_i)$	第 i 个索引节点在第 T 个时钟周期内的写频率
DF_{avg}^T	前 T 个时钟周期内全部索引节点的平均写频率
α_i^T	第 i 个索引节点在第 T 个时钟周期的权重（Critical factor）
WI_j^T	表示第 j 个存储域在时钟周期 T 内的实际写次数
WB_j^R	表示第 j 个存储域在运行周期 R 内所允许的最大写次数

另一方面，为了避免对单个存储域造成过度磨损，若某一存储域的写次数超出参数 WBL 所允许的最大值，则在后续的运行周期内限制对该存储域的写操作，进而将该存储域内的索引节点迁出，以达到均衡的目的。具体地，域间磨损均衡机制可分为两个部分：索引节点与存储域的匹配算法，和索引节点迁移触发机制，分别见 3.3.1 和 3.3.2 小节。本文所提出的磨损均衡机制所涉及的全部符号定义如表 3.1 所示。

3.3.1 索引节点与存储域的匹配算法

每一次索引节点迁移均会对非易失性存储设备造成额外的磨损，因此在实现磨损均衡的过程中，应当尽可能减少索引节点的迁移次数。举例而言，对于文件系统中磨损次数最多的索引节点，应尽可能地将其迁移至耐受度较高的存储域，以避免后续的额外迁移。从这个角度出发，本小节提出索引节点与存储域的动态匹配算法。根据索引节点的潜在磨损热度以及存储域的可用写次数将两者进行动态匹配，从而在后续每一次的迁移过程中始终将索引节点迁移至与其匹配的存储域中去，进而在实现存储域层面磨损均衡的基础上，尽可能减少迁移所带来的磨损开销和性能开销。

为了能够有效区分不同索引节点的冷热程度，首先使用写频率来标识索引节

点的相对更新热度。索引节点I_i在时钟周期 T 内的写频率记为$DF^T(I_i)$，时钟周期的长度设置为t。时钟周期T之后文件系统中所有索引节点的平均更新频率用DF_{avg}^T表示，其计算方式如公式（3.1）所示。

$$DF_{avg}^T = \frac{WI_{total}^T/t + DF_{avg}^{T-1}}{2} \tag{3.1}$$

其中WI_{total}^T表示时钟周期 T 内所有存储域的写次数，即$\sum_{j=1}^{N} WI_j^T$，也即文件系统内全部索引节点的写次数。DF_{avg}^T的初始值为 1。借助平均更新频率DF_{avg}^T，可通过公式（3.2）计算单个索引节点在全部索引节点中的权重（Critical factor），α_i^T表示第 i 个索引节点（I_i）在第 T 个时钟周期内的权重。

$$\alpha_i^T = \frac{DF^T(I_i)}{DF_{avg}^T} \tag{3.2}$$

若某个索引节点的更新频率远高于当前的平均更新频率，即α_i^T远大于 1，那么可以近似地认为该索引节点在接下来的时钟周期内将会对其所在的存储单元产生相对较高地磨损，也即该索引节点应当被迁移到耐受度更高的存储域。因此以时钟周期为单位，可根据计算得到的权重将每个索引节点匹配至耐受度合适的存储域。并且当某个索引节点的权重不断增加至超出当前存储域的限定范围时，可将其迁移至耐受度更高的存储域以避免潜在的磨损。

索引节点与存储域的匹配过程如下。对于包含有 N 个具有不同耐受度存储域的文件系统，可设置 N 个对应的权重区间。文件系统中索引节点权重与存储域的映射关系如公式（3.3）所示。

$$\mathrm{F}(\alpha_i^T) = \begin{cases} 1, & if\ \alpha_i^T \in A_1 \\ 2, & if\ \alpha_i^T \in A_2 \\ \cdots & \\ N, & if\ \alpha_i^T \in A_N \end{cases} \tag{3.3}$$

其中$A_j(j = 1,2,\ldots,\mathrm{N})$表示第 j 个存储域所对应的权重区间。当索引节点$I_i$的权重数值处于某个存储域对应的权重区间内时，可认为该索引节点与存储域相匹配。以存储域为单位的权重区间划分方式与索引节点权重相类似，可通过对每个存储域的最大可磨损次数进行归一化得到。具体过程如下：

① 以存储域为单位，计算所有存储域的平均可磨损次数，并将其作为基准值；② 按照基准值，对每个存储域的可磨损次数从小到大进行归一化，第 j 个存储域归一化之后的值记为NWB_j；③ 根据归一化的结果，设置每个存储域对应的权重区间。其中$A_j = (NWB_{j-1}, NWB_j]$，$A_0 = [0, NWB_1]$，$A_N = (NWB_{N-1}, +\infty]$。

3.3.2 索引节点迁移触发机制

上一小节通过对最大可磨损次数进行归一化，得到每个存储域的权重区间，并根据索引节点的历史更新热度实现索引节点与存储域之间的动态匹配。默认情况下，当索引节点的权重始终保持在当前存储域所对应的权重范围时，也即其α_i^T满

足条件$NWB_{j-1} < \alpha_i^T \leq NWB_j$时，可认为该索引节点暂时不需要被迁移。

对应地，根据公式（3.2）可得到时钟周期 T 内对该索引节点预期的写频率区间，如公式（3.4）所示。

$$NWB_{j-1} \cdot DF_{avg}^{T-1} < DF^T(I_i)_{exp} \leq NWB_j \cdot DF_{avg}^{T-1} \tag{3.4}$$

时钟周期内，若索引节点I_i的写次数超过预期的最大值时，文件系统选择迁移该索引节点。以时钟周期 T 为单位，索引节点I_i所期望的写次数最大值（记为$WI^T(I_i)_{max}$）为$\mathrm{t} \cdot NWB_j \cdot DF_{avg}^{T-1}$。若单个时钟周期内的写磨损次数超过期望的最大值，则立即重新计算该索引节点的权重，并为其匹配新的存储域。此时该索引节点权重的计算公式为：

$$\alpha_i^T = \frac{WI^T(I_i)_{max}}{\Delta t_i \cdot DF_{avg}^{T-1}} = \frac{t \cdot NWB_j}{\Delta t_i} \tag{3.5}$$

其中Δt_i表示当前时钟周期 T 内经过的时间长度。计算得到该索引节点最新的权重数值之后，即可通过公式（3.3）给出的映射关系重新匹配新的存储域。选中目标存储域后，文件系统通过算法 3.1 为索引节点在目标存储域中寻找可用的存储单元，并自动完成迁移操作，具体执行过程见 3.4 节域内磨损均衡机制。此外，如前文所述，为了尽可能地避免迁移所引入的磨损和性能开销，当某个索引节点的权重低于所在存储域的权重区间最小值时，文件系统选择暂不处理。

在上述域间磨损均衡机制中，使用 WBL 参数来控制多个存储域写磨损的均衡分布，该参数以运行周期（Round）为单位进行设置。当所有存储域的剩余可磨损次数低于某个阈值（如 5%）时进入下一个运行周期。通常一个运行周期包含有多个时钟周期。进入下一个运行周期之前，首先需要根据历史运行情况来计算所有存储域在接下来的周期内的最大可磨损次数WB_{total}^R，计算方法如公式（3.6）所示。

$$WB_{total}^R = \sum_{j=1}^{N} \left(WBL^R \cdot E_j \right) + \left(WB_{total}^{R-1} - WI_{total}^{R-1} \right) \tag{3.6}$$

其中$E_j (j = 1,2,\cdots,N)$表示第 j 个存储域的耐受度，$WBL^R \cdot E_j$表示第 j 个存储域在运行周期 R 内的最大可磨损次数。WB_{total}^{R-1}和WI_{total}^{R-1}分别表示在上一个运行周期内所有存储域的最大可磨损次数和实际磨损次数。若两者之差小于 0，则表示在上一运行周期内某个存储域的写次数超过了预期的最大值，因此需要在后续周期内降低该存储域的最大写次数，即限制对该存储域的写操作。

另外，需要加以说明的是，由于任何方案都无法保证磨损分布在任意时刻都是绝对均衡的，因此 Contour 中磨损率参数 WBL 的主要作用是保证多个存储域的磨损率以某种粒度（或步长）进行均衡。理论上，该参数越小，能够实现的均衡效果就越好，反之则均衡效果相对较差。综合考虑多次迁移所引入的磨损和性能开销后，本文在之后的实验过程中将参数 WBL 的值设置为 1%。此外，为了验证所提出磨损均衡机制对参数 WBL 的敏感程度，本文在 4.4 节中针对 WBL 参数的

不同取值进行了对比实验和结果分析。

3.4 域内磨损均衡机制

本节提出域内磨损均衡机制，通过迁移索引节点的方式实现存储域内多个存储单元的磨损均衡。上一节给出的域间磨损均衡机制以索引节点为主要对象，基于权重的方式实现索引节点与存储域的动态匹配，以及索引节点在存储域之间的迁移。除此之外，由于对索引节点的写操作均会对其所在的存储单元造成直接地磨损，因此还需要对存储单元的写次数和耐受度进行监测，以避免将某个存储单元写穿，或造成存储域内的磨损不均衡。

存储域内的磨损均衡意味着所有存储单元均承受了相同的写次数，也即具有相同的磨损率。因此，可将存储域内的磨损均衡问题定义如下：给定 N 个索引节点和N_p个物理页，其中每个物理页均包含N_s个耐受度相同的存储单元，求这些索引节点应如何分布，才能使得写磨损在存储域内分布均衡，即所有物理页写次数（或写磨损率）的标准差最小。

由于文件系统运行过程中每个索引节点的写次数可能是实时变化的，并且索引节点的访问冷热程度可能会随时间的变化而不同，因此上述问题无法通过静态的“背包问题”来解决，而是需要借助索引节点的动态迁移，来实现存储域内部的写次数均衡分布。为此，本节提出索引节点迁移算法和存储域最优匹配算法，分别见 3.4.1 和 3.4.2 小节。

3.4.1 索引节点迁移算法

本节提出索引节点迁移算法。该算法的主要作用是当存储域内某个存储单元处于“过磨损”状态，即当存储单元的累计写次数超出其最大可磨损次数的限制时，算法可将该索引节点迁移至存储域内的其他单元，进而避免造成域内的磨损不均衡现象。另一方面，该算法也被用于域间磨损均衡机制。根据索引节点的权重为其匹配目标存储域后，可通过索引节点迁移算法在目标存储域内寻找适当的存储单元，进而得以完成迁移操作。

算法 3.1：索引节点迁移算法

输入： WP_{avg}: 当前存储域内物理页写次数的平均值;

WS(I_i): 索引节点I_i所在存储单元的写次数;

输出： 迁移操作后，索引节点I_i的偏移地址信息$Offset_i$;

1： if WS(I_i) $\leq \alpha$ then

2： continue;

3： else

4： for P_p has free slot and $WP_p \leq WP_{avg}$ do

算法 3.1：索引节点迁移算法

```
5：        if a slot Slot_s on P_p satisfies WS_s < α then
6：            Slot_cand ← Slot_s;
7：            break;
8：    if Slot_cand = NULL then
9：        for P_p has free slot and WP_p < α · N_S do
10：        if a slot Slot_s on P_p satisfies WS_s < α then
11：            Slot_cand ← Slot_s;
12：            break;
13：if Slot_cand = NULL then
14：    if WP_avg · N_p ≥ WB_j then
15：        Slot_cand ← a free slot in the best-fit available domain;
16：    else
17：        I_cand ← the least recently written inode;
18：        Slot_cand ← Slot(I_cand);
19：        Swap I_i with I_cand;
20：        Exchange Offset_i and Offset(Slot_cand);
21：        return;
22：Migrate I_i into Slot_cand;
23：Offset_i ← Offset(Slot_cand);
```

首先给出如下两点分析。第一，为了尽可能限制迁移操作的次数，当某个存储单元的累计写次数低于其在运行周期内所允许的最大写次数时，算法不应主动迁移该存储单元上保存的索引节点。第二，索引节点迁移后，对该索引节点的更新操作将会对新的存储位置产生进一步磨损。因此在迁移过程中，应同时考虑目标存储单元的磨损情况以及索引节点的访问特征，尽可能地为其选择适合的位置，以达到理想的磨损均衡效果。

索引节点迁移算法的输入包括：① 当前存储域内所有物理页写次数的平均值，记为WP_{avg}；② 索引节点I_i所在存储单元的累计写次数$\mathrm{WS}(I_i)$。将WP_{avg}作为基准值，主要目标是尽可能地将存储域内所有物理页的写次数与WP_{avg}保持一致。具体执行过程如算法 3.1 所示。

$$\alpha = \frac{E_j \sum_{r=1}^{R} WBL^r}{N_p \cdot N_s} \tag{3.7}$$

对于索引节点I_i，首先需要判断其所在的存储单元是否处于“过磨损”的状态，

即当前存储单元上的累计写次数是否已经超过了其在运行周期内所允许的最大写次数。本文使用阈值α作为判断依据，其计算方法如公式（3.7）所示。对于单个存储域，阈值α表示当前存储域内每个存储单元在过去的所有周期内累计的所允许最大写次数。若某个存储单元的写次数超过该阈值，则认为当前存储单元存在“过磨损”的情况。此时认为该存储单元上保存的索引节点理应被迁移至当前存储域内其他的存储单元上。

算法 3.1 中，若索引节点I_i所在存储单元的累计写次数$\mathrm{WS}(I_i)$超过了阈值α，则表示索引节点I_i应被迁移。接下来的 4-21 行表示为索引节点I_i寻找一个适合的存储单元以完成迁移操作。为使得存储域内所有物理页的写次数尽可能均衡，查找过程被分为三个步骤。

首先，算法在写次数低于平均值（WP_{avg}）的物理页当中寻找存储单元，并且目标存储单元不应处于“过磨损”状态。若未找到适合的存储单元，则进入第二步查找。算法的 8-12 行表示所查找物理页的写次数不应超过其在运行周期内累计的所允许最大写次数，并且在满足条件的物理页中寻找适合的存储单元。前两步的检索过程使得索引节点迁移的目标物理页的写次数尽可能小，从而避免对磨损程度较高的页产生进一步地磨损。

若仍无法找到适合的物理页以及存储单元，则需要考虑当前索引节点所在的存储域是否已经处于“过磨损”的状态。“过磨损”状态的判断依据是比较存储域的累计写次数是否超过其在所有运行周期内所允许的最大写次数。若当前存储域已经“过磨损”，则认为索引节点I_i应被及时迁移至邻近的其他存储域，而非继续保存在当前位置。因此算法的 14-15 行调用存储域最优匹配算法，尝试将索引节点迁移至相邻的存储域，具体执行过程见 3.4.2 节。

反之，若当前存储域仍处于正常磨损状态，则算法的 17-21 行选择当前存储域中更新次数最少的索引节点I_{cand}，并将交换索引节点I_i和I_{cand}。在算法的 22-23 行，若存在适合的目标存储单元，则将索引节点I_i迁移至目标存储单元，也即$Slot_{cand}$，更新映射表中的偏移信息即可完成索引节点的迁移过程。

上述索引节点迁移算法以运行周期为单位计算阈值α，并且该阈值在运行周期内部保持不变。上述迁移算法在最差情况下需要对存储域内保存的全部索引节点进行迁移，因此该算法的时间复杂度为$\mathrm{O}(N_p \cdot N_s)$，其中$N_p$和$N_s$分别为单个存储域内物理页的个数和单个物理页内存储单元的个数。

3.4.2 存储域最优匹配算法

本节提出存储域最优匹配算法，以在相邻存储域之间分散写操作。上一节给出的索引节点迁移算法中，若索引节点所在的存储域已经处于“过磨损”状态，算法会主动将对应的索引节点迁移至相邻存储域，以避免对单个存储域的过度磨

损。选择目标存储域时采用的算法是存储域最优匹配算法。

域间磨损均衡机制中也存在对该匹配算法的调用。若第 j 个存储域（d_j）在某一周期内的累计写次数超出最大值限制，导致该存储域在后续运行周期内的可用写次数为 0，则根据 3.3 节给出的存储域权重划分方法以及动态匹配机制可得出，每一次对该存储域内索引节点的更新操作均会触发索引节点的向外迁移。因此，为了避免所有索引节点全部迁移至耐受度最低的存储域，文件系统采用了该存储域最优匹配算法，以使得写磨损尽可能地均衡分布在相邻的位置上。

存储域最优匹配算法的运行过程如算法 3.2 所示。

算法 3.2：存储域最优匹配算法

输入: cur_domain: 当前存储域;

N: 存储域的个数;

输出: target_domain: 目标存储域;

1：for d_j whose endurance is **larger than** $d_{cur\_domain\_index}$ do

2：　　if d_j has aviliable write budge then

3：　　　　if a slot $Slot_s$ on d_j is free and not be over written then

4：　　　　　　target_domain = j;

5：　　　　　　return;

6：for d_j whose endurance is **smaller than** $d_{cur\_domain\_index}$ do

7：　　if d_j has aviliable write budge then

8：　　　　if a slot $Slot_s$ on d_j is free and not be over written then

9：　　　　　　target_domain = j;

10：　　　　　　return;

11：target_domain = cur_domain;

12：return;

该算法的输入是索引节点当前所在的存储域 cur_domain，输出为迁移操作的目标存储域 target_domain。算法的主要目的是在相邻的存储域中为索引节点选择合适的存储位置，因此查找过程首先从当前存储域向耐受度增加的方向进行遍历，如算法 1-5 行所示。对于剩余可写次数大于 0 的存储域，若该存储域内存在空闲的索引节点存储单元，且该存储单元未处于“过磨损”状态，则表示算法已匹配到合适的目标存储域，此时返回并且查询过程结束。

若未找到满足条件的存储域，则接下来的 6-10 行从当前存储区向耐受度减少的方向进行遍历，遍历过程同理。这样的做法使得算法能够尽可能地将索引节点

迁移至磨损率较低的存储域，也即将对当前存储域的过多地写操作分散到相邻的存储域，从而实现磨损均衡的效果。

若上述过程仍无法确定目标存储域的位置，则默认不迁移索引节点，也即将索引节点继续保存在当前所在的存储域，直至下一次迁移条件被触发。

3.5 磨损均衡机制的实现

本节给出索引节点磨损均衡机制 Contour 在文件系统中的具体实现，并对方案实现过程中遇到的关键问题进行分析和讨论。

3.5.1 总体实现

本文选取典型的非易失内存文件系统 SIMFS 实现了所提出的索引节点磨损均衡机制。该文件系统使用 Array 数据结构保存文件系统中的所有索引节点。实现过程中使用 Intel 提供的非易失性内存模拟工具 PMEM[49]，划分一段连续的物理内存 DRAM，以模拟非易失性存储设备。

① 文件系统初始化

在文件系统初始化阶段，首先划分 1%的空间用于保存索引节点及其映射表。其次，按照 4MB 的大小将索引节点区域划分为多个连续且耐受度递增的存储域，且每个存储域内的所有存储单元均具有相同的耐受度。

为了能够得到尽可能理想的磨损均衡效果，同时排除不同索引节点分配方案对方磨损分布结果的影响，本文将域间差异感知的空间管理策略实现在文件系统中，也即采用“多链表+轮询”的方式实现空闲索引节点的管理和分配。文件系统初始化阶段，每个存储域内的索引节点存储单元均以单链表的形式进行组织，也即空闲链表的个数等于索引节点区域所包含存储域的个数。当需要分配新的索引节点时，优先从剩余节点数量最少的链表中进行分配。默认情况下，为了尽可能避免对链表头部节点造成过多的磨损，分配操作从链表头部进行，而回收操作则将索引节点回收至链表尾部。在基于树形索引节点组织结构的文件系统中，可通过位置标记的形式实现同等效果，而无须对多链表进行显式管理。

文件系统为每个索引节点、存储单元、物理页以及存储域建立写计数器，以保存其写次数。这些以不同粒度为单元的计数器，一方面能够为磨损均衡机制的正常运行提供数据支持，比如索引节点的更新频率、物理页的平均写次数等。另一方面可用于磨损分布结果的验证。具体地，可通过 Linux 内核提供的/proc 虚拟文件系统将这些计数器以文件的形式进行导出，进而得以观察到文件系统在运行不同负载后的磨损分布情况。

在初始化阶段文件系统需要为这些计数器分配对应的存储空间。计数器默认保存在运行内存 DRAM 中，不存在写次数限制或磨损不均衡问题。文件个数为 100

万时，保存这些计数器所需要的内存空间大小仅约为 8.13MB，空间开销较小，且这些计数器所在的内存不存在耐受度受限的问题。实现时以时钟周期为单位，将这些计数器写回 NVM，进而避免系统发生掉电故障时丢失计数器的内容。

② 磨损均衡机制的实现

域间磨损均衡机制负责将索引节点与存储域进行动态匹配，并根据匹配结果完成跨存储域的索引节点迁移。其中存储域的权重区间划分以运行周期为单位，索引节点的权重计算以时钟周期为单位。新的运行周期开始时，首先根据历史磨损情况为每个存储域计算下一个周期内所允许的最大写次数，并通过归一化得到权重划分区间，如公式（3.3）所示。基于树形索引结构的文件系统，运行过程中可能涉及存储域的动态分配，因此分配新的存储域时同样需要执行归一化并重新调整权重区间。

实现时将域间磨损均衡机制中时钟周期 T 的时间长度设置为 10s。即文件系统初始化操作完成后，立即启动一个名为 period_kthread 的内核线程，该线程以 10s 为间隔，计算文件系统内所有索引节点的平均更新频率DF_{avg}^{T}。之后以该数值作为基准，当某个索引节点的写次数达到所在权重区间限定的最大阈值时，文件系统自动为该索引节点计算新的权重，并重新匹配存储域，进而触发迁移过程。

域内磨损均衡机制主要负责同一存储域内多个存储单元的磨损均衡。在对索引节点不断更新的过程中，一旦其所在的存储单元处于“过磨损”状态，则首先按照算法 3.1 为其寻找新的存储单元，并触发迁移。

图 3.2 给出了从应用程序更新文件内容到文件系统完成索引节点迁移的详细流程。首先，上层应用程序对某个文件进行更新时，文件系统首先增加该文件所对应的索引节点、索引节点所在的存储单元、物理页以及存储域的写次数。其次，若该索引节点的写次数超出其所在存储域所预期的最大写次数时，则触发域间磨损均衡机制。该机制首先进行索引节点和存储域重新匹配，之后完成从原存储域到新存储域的迁移过程，并更新相关的元数据信息。

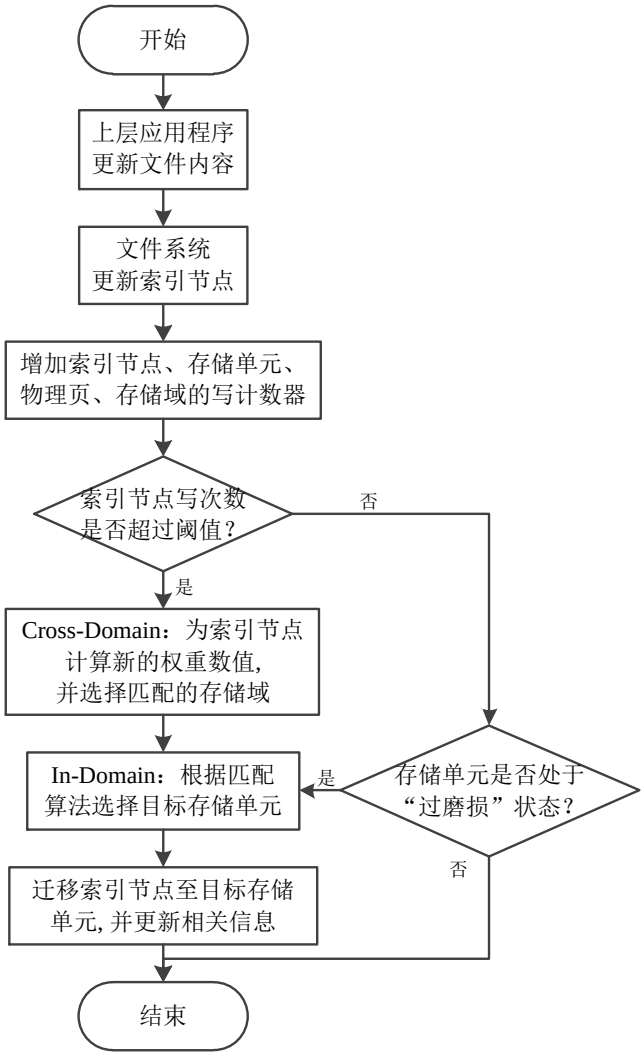


图 3.2 索引节点磨损均衡机制 Contour 的示例流程图

Fig.3.2 Example process of Contour

若该索引节点的写次数并未超出阈值，则接下来检查其所在的存储单元是否处于“过磨损”的状态，也即该存储单元的写次数是否超过了当前所有运行周期内累计所允许的最大写次数。若满足条件，则触发域内磨损均衡机制。该算法在当前或相邻的存储域中为其寻找新的存储位置，触发迁移并更新相关数据信息。若不满足条件，则不执行任何操作，直至下一次迁移条件被触发。

3.5.2 关键问题分析

索引节点磨损均衡机制的引入，使得文件系统运行过程中任一索引节点均有可能被动态地迁移至新的存储位置。索引节点的迁移操作对于上层应用程序而言是透明的，但从实现文件系统的角度来看，需要着重考虑如下两个关键问题：① 空闲索引节点链表的并发访问控制问题；② 索引节点迁移前后内存中数据结构的一致性问题。本小节对上述两个关键问题进行逐一分析和说明。

实际使用过程中文件系统的索引节点数量较多，且写操作频繁，导致磨损均衡机制触发的迁移次数较多。而索引节点的迁移操作必须是原子操作。首先需要从空闲索引节点链表 A 中分配一个空闲节点，将数据从原位置拷贝至新的存储位置后，再将旧存储单元释放并回收至空闲链表 B。

为了优先保证高并发情况下上述索引节点迁移操作的正确性，实现过程中必须使用严格的锁逻辑对链表的分配和回收流程进行控制。当存在多个线程同时对多个链表进行加锁时，还应尽可能避免因多线程资源竞争导致的死锁问题。此外，Contour 在实现时采用了链表头尾加锁的方法，以尽可能降低加锁带来的性能损失。其中分配和回收操作分别获取不同的互斥锁，以分别在链表头部和尾部完成操作。

与此同时，索引节点迁移前后内存中某些数据结构的一致性问题也尤为重要。区别于传统的文件系统，在引入索引节点磨损均衡机制之后，索引节点的动态迁移可能导致编码过程中的某些数据结构失效，从而导致后续代码段运行错误。

以 SIMFS 文件系统为例，其使用 struct nvmm_inode 数据结构表示物理索引节点。当两个连续的代码段 A 和 B 先后对同一个该结构的实例进行操作时，可能出现代码段 B 运行错误的现象。这是由于代码段 A 中可能包含有对索引节点的更新操作，也即可能会触发索引节点迁移，导致索引节点被迁移至新的物理位置。这就使得代码段 B 中默认使用的实例并没有指向最新的存储位置，反而可能指向了旧的索引节点或非法地址，从而导致运行错误。因此实现过程中应在类似的代码段 B 中首先重新访问映射表，获取到最新的索引节点物理地址，才能继续运行。

3.6 本章小结

本章首先对导致索引节点磨损不均衡问题的根本原因进行分析，并提出耐受度差异感知的多链表空间管理策略。其次，域间磨损均衡机制主要针对索引节点更新频率与存储域耐受度之间的不匹配现象而提出。通过存储域之间的索引节点迁移操作，实现周期性地存储域写磨损率均衡。域内磨损均衡机制主要关注于单个存储域内多个存储单元之间的磨损均衡问题。充分考虑索引节点潜在的读写特征以及当前存储单元的磨损状态，以使用最少的迁移次数实现尽可能最佳的域内均衡状态。最后给出了磨损均衡机制 Contour 在文件系统 SIMFS 上的实现思路及关键问题分析，并介绍了详细的工作流程。

4 实验与分析

本章对所提出方案的磨损均衡效果和性能开销进行详细测试和分析。首先对包括设备配置信息、测试对象和工具、测试负载等在内的测试环境进行介绍，见4.1 节。之后 4.2 节从 Domain 和 Page 两个不同粒度的存储单元出发，对比三种文件系统在不同运行时长下的磨损分布效果。最后 4.3 和 4.4 节分别对磨损均衡机制的性能开销，以及该方案对参数的敏感程度进行实验评估。

4.1 测试环境

4.1.1 测试主机环境

测试实验是在配置有 Intel Xeon E5-2640 v4 2.40GHz 处理器和 256GB 内存的惠普服务器上进行的。使用英特尔提供的 PMEM 非易失性内存模拟工具划分并模拟 64GB DRAM 作为文件系统的持久性内存。文件系统的索引节点区域大小为128MB，包含 32 个物理连续的存储域，可支持约一百万个文件。测试所使用的操作系统为 Ubuntu16.04，内核版本为 Linux 4.4.30。

4.1.2 测试对象与测试工具

本文将基于 Linux 4.4.30 内核版本的 SIMFS 文件系统[5]作为测试时的文件系统实例。该文件系统通过文件虚拟地址以及文件页表结构提高读写性能，并且使用基于 Array 的索引节点组织结构。经测试，该文件系统各项功能正确，足以支持本文所需的磨损分布和性能验证实验。

本文将实现有索引节点磨损均衡机制 Contour 的文件系统与另外两种版本的文件系统进行对比。其中包括：① 初始版本的 SIMFS 文件系统，也即未实现任何磨损均衡机制的文件系统版本，记为 NoWL；② 实现有 PCV 磨损均衡机制[35]的 SIMFS 文件系统，记为 PCV。该机制的核心思想是使用写次数阈值来保证所有存储单元的写均衡。当某个存储单元的写次数超出整体平均值和阈值的限制时，采用特定的查询算法为其匹配新的存储单元，并执行迁移操作。

实验使用 Filebench[42]和 Flexible I/O(FIO)[43]两种测试工具，分别对三种文件系统的磨损分布和性能开销情况进行测试和对比。Filebench 和 FIO 均是典型的文件系统测试工具，通过模拟不同类型的工作负载、读写模式等，可对文件系统功能和性能进行全方位地测试。

对于磨损分布情况的验证，本文首先选取了 Filebench 工具中自带的三种典型应用负载。其中 OLTP 负载用于模拟典型的数据库读写负载模式。测试时同时启动 50 个读线程、10 个写线程和 1 个写日志线程。其中日志线程仅对固定的日志文件

进行写操作，而剩余线程则随机选择文件进行读或写操作。Fileserver 负载用于模拟真实场景中的文件服务器，同时运行 50 个访问线程。其中每个线程循环执行一系列的文件访问操作，包括创建、删除、打开文件、读写文件等。Webserver 负载用于模拟配置有 100 个读写线程的网络服务器。该服务器上的每个线程按照打开文件、读文件、和关闭文件的顺序循环执行，并在每一轮操作执行完成之后向该服务器上的日志文件中写入少量数据。区别于 Fileserver 负载，Webserver 负载只有当线程对日志文件进行写操作时才会对索引节点产生磨损，因此两者所造成的磨损分布情况也有所差异。

此外，为了验证磨损均衡机制在真实应用场景下所发挥的效果，测试过程还使用典型的数据库应用程序 MySQL 进行了磨损分布验证。MySQL 中自带的数据库测试工具 mysqlslap[50]可模拟多个用户同时对单个数据库进行大量查询和插入操作，常用于对 MySQL 数据库进行压力测试。本文设置了 50 个并发访问线程，累计对单个 MySQL 数据库执行 4 亿次查询和插入条目操作，以观察其对底层设备造成的磨损分布情况。上述四种负载模式的详细参数配置如表 4.1 所示。负载运行过程中会对文件系统中索引节点所在区域造成一定的磨损，因此可通过观察磨损次数以及磨损率在存储域或物理页上的分布情况得到不同文件系统的磨损均衡效果。

表 4.1 Filebench 和 MySQL 工作负载的参数配置

Table 4.1 Characteristics of Filebench and MySQL workloads

	# of files (Data/Log)	Average file size	Threads	I/O size (r/w)
OLTP	200K/1	10MB	61	2KB/2KB
Fileserver	200K/0	4KB	50	1MB/16KB
Webserver	200K/100	4KB	100	1MB/256B
MySQL	30M/20	8KB	50	32B

其次，使用测试工具 FIO 和 Filebench 对三种文件系统的性能开销进行对比测试，并将未实现磨损均衡机制的 NoWL 文件系统性能作为基准值。其中 Filebench 以不同工作负载测试得出的 IOPS（ops/s）作为评价指标，FIO 测试则能够得到不同写粒度下的文件系统写带宽，包括顺序写和随机写两种模式。

4.1.3 磨损验证和性能模拟方法[①]

本文主要从磨损分布和性能开销两个方面对磨损均衡机制进行评价。

[①]该部分涉及的磨损验证方法和写时延模拟方法已发表在中文期刊《计算机科学》

为了较直观地反映测试程序在文件系统索引节点区域造成的磨损分布情况，实验过程中采用了基于写计数器的写磨损验证方法。为文件系统索引节点区域的每个存储域、物理页以及存储单元建立对应的更新计数器，并在文件系统更新索引节点时增加相应的计数器。测试程序运行一定时间后，通过内核的临时文件系统/proc 将磨损分布数据以任意单位进行导出。

其次，由于 NVM 硬件技术尚未成熟，在使用 DRAM 模拟非易失性内存的过程中，往往需要增加额外的软件时延，以弥补两者之间的写时延差异。为此 Contour 在文件系统保证数据一致性的软件流程中，根据对 NVM 的读写粒度和访问次数，基于 x86 平台的读取时间戳指令 RDTSC 引入部分软件时延，从而使得文件系统性能测试结果更加接近于硬件真实情况。

4.2 磨损均衡效果验证

本节分别从 Domain 和 Page 两个层面，对不同文件系统所造成的磨损分布情况进行对比和分析。其中 Domain 层面的磨损分布结果旨在说明文件系统所造成磨损分布在不同运行时长下的变化趋势，见 4.2.1 小节。以 Page 为单位的磨损分布统计结果，见 4.2.2 小节，选取了运行时长为 10 小时的磨损分布情况进行说明，旨在对比不同方案在较长时间运行后能够取得的磨损均衡效果。

4.2.1 Domain 层面磨损分布对比分析

图 4.1 以负载模式为单位，记录了三种文件系统运行负载后的存储域写磨损率分布情况，运行时长分别为 1 小时、5 小时和 10 小时。文件系统的磨损均衡效果可通过观察图中对应曲线的水平程度得出。曲线走势越接近水平，也即磨损率的标准差越小，表明磨损均衡效果越好。

首先以 OLTP 负载产生的磨损分布结果为例进行分析。如图 4.1(a)所示，运行该负载 10 小时后，NoWL、PCV 和 Contour 最终得到存储域写磨损率的最大值分别为 89.32%、70.21%和 27.09%。同时三种文件系统得到的磨损率标准差分别为 22.6、14.4 和 0.53，也即运行 OLTP 负载 10 小时后，Contour 的磨损均衡效果能够分别达到 NoWL 和 PCV 文件系统的 42.2 和 26.9 倍。

其次，为了观察不同文件系统在不同运行时长下的磨损分布变化趋势，使用 WG（Wear gap）表示磨损率最大值和最小值之间的差值。从图中可以看到，分别运行 OLTP 负载 1 小时、5 小时和 10 小时后，Contour 文件系统的 WG 分别为 3.46%、9.48 和 2.13%。对应地，NoWL 文件系统的 WG 则分别为 9.69%、47.64%和 89.32%。该数值的不断增加，表示随着文件系统运行时间越久，其所造成的磨损不均衡问题也更加严重。

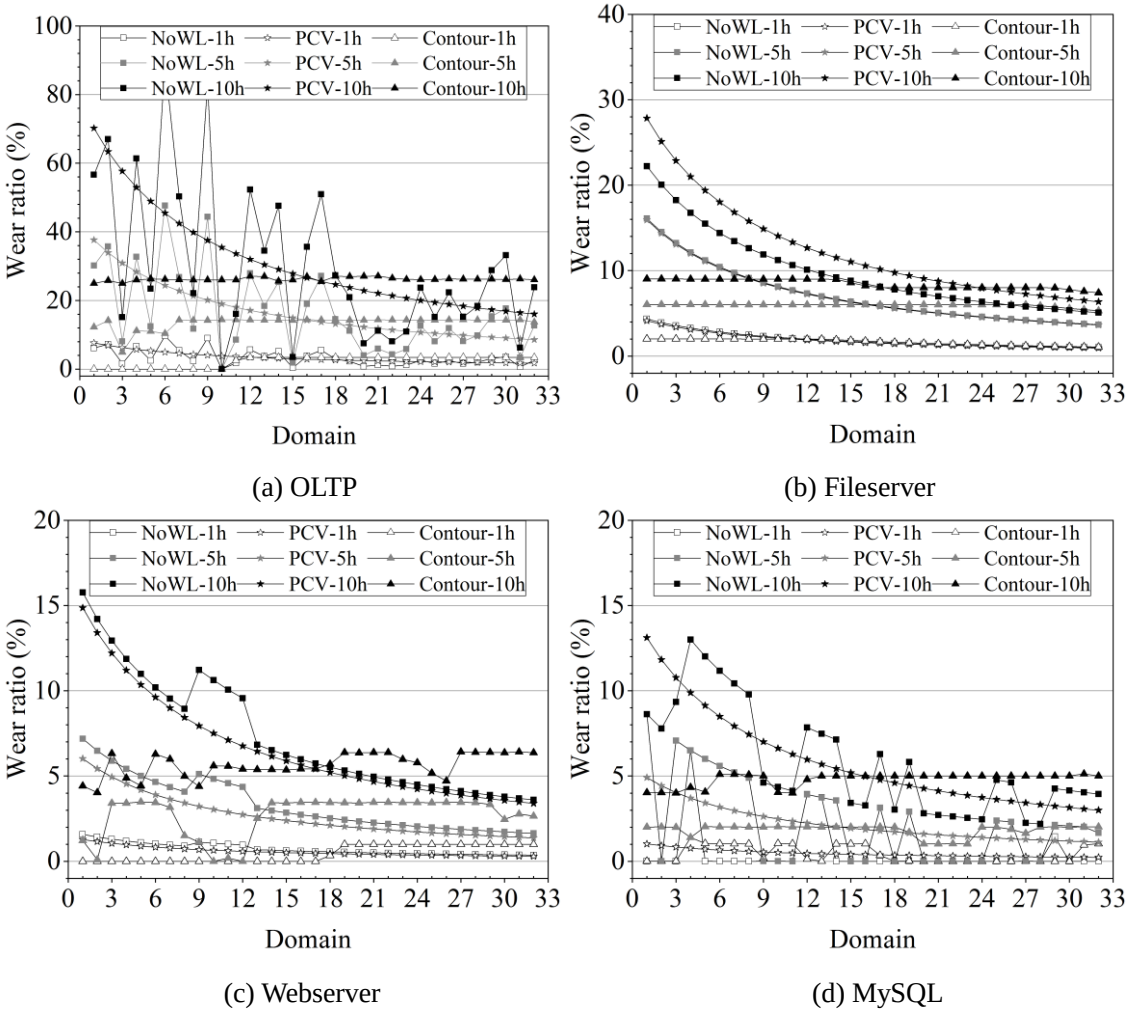


图 4.1 不同文件系统造成的存储域磨损率变化情况

Fig.4.1 Comparison of wear ratio of domains in NoWL, PCV, and Contour

类似地，文件系统 PCV 在运行上述三种时长后得到的 WG 值分别为 5.9%、29.04%和 54.17%。其采用的磨损均衡机制以物理页为单位，通过在页面之间迁移写次数较多的索引节点，从而对所有物理页的写次数进行均衡。由于其并未考虑存储单元的耐受度差异，使得耐受度较低的存储单元磨损程度更加严重，尤其是在增加文件系统的运行时长后。通过上述分析可得出结论，OLTP 负载模式下，本文提出的磨损均衡机制 Contour 能够在不同运行时长下均得到相对均衡的磨损分布结果，且运行时间越长，取得的磨损均衡效果越明显。

通过分析 Fileserver、Webserver 和 MySQL 负载的实验结果，可得出类似结论。以 MySQL 负载为例，Contour 文件系统运行得到的 WG 始终小于 1.4%，而 NoWL 文件系统运行 1 小时、5 小时和 10 小时后得到的 WG 分别为 6.5%、8.63%和 10.81%，PCV 文件系统的 WG 则分别为 0.79%（1 小时）、3.79%（5 小时）和 10.12%（10 小时）。也即随着 MySQL 负载运行时间的增加，NoWL 和 PCV 两种文件系统所引

入的磨损不均衡现象愈加明显，而 Contour 文件系统则能够始终保持相对均衡的状态。运行 10 小时后，NoWL、PCV 和 Contour 三种文件系统得到的存储域磨损率最大值分别为 13.0%、13.11%和 5.12%。与此同时，三种文件系统的磨损率标准差分别为 3.1、2.7 和 0.39，也即对于 MySQL 负载，Contour 能够实现的磨损均衡效果分别约为 NoWL 和 PCV 文件系统的 7.9 和 6.8 倍。

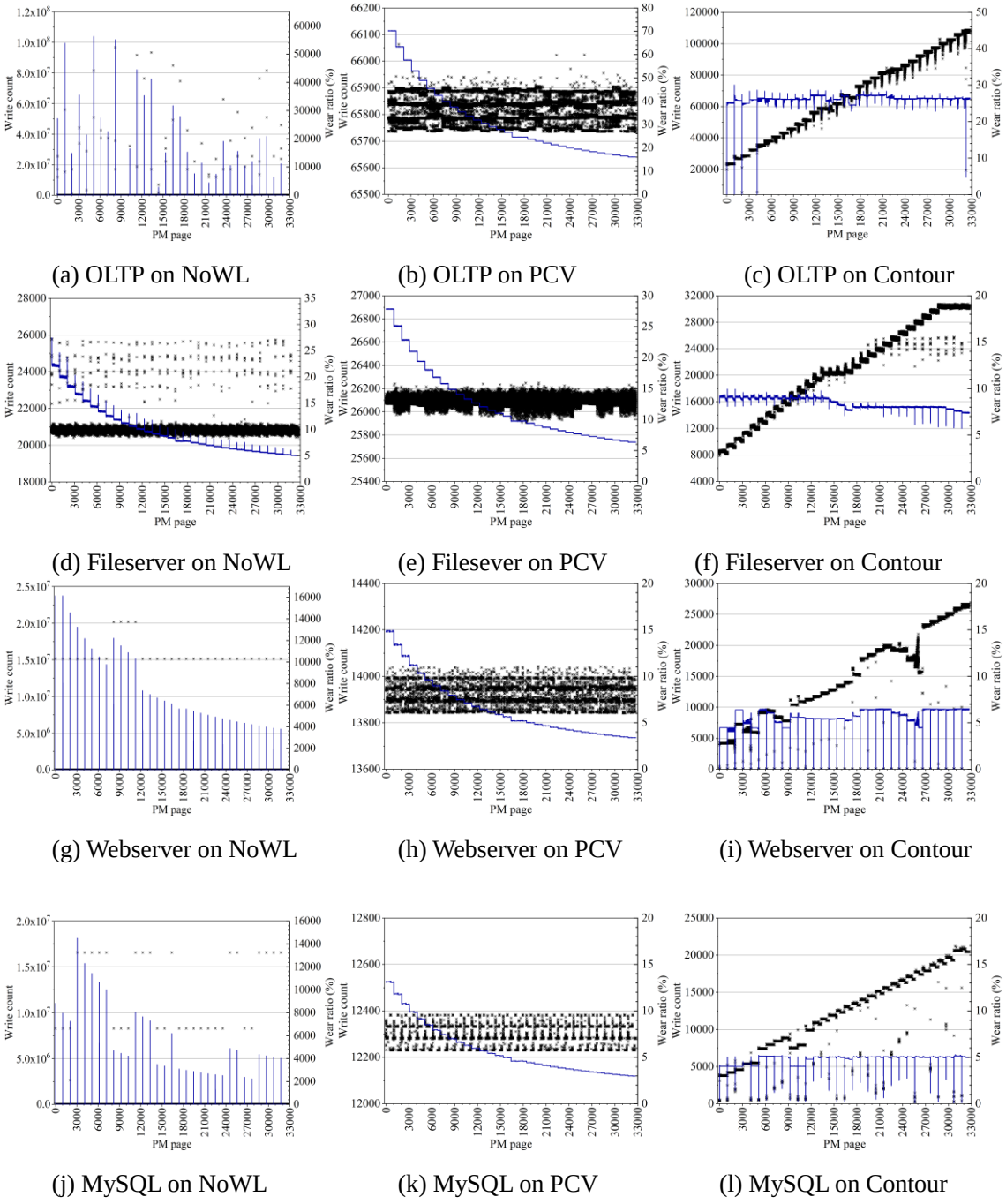
需要说明的是，在上述以存储域为单位的磨损率分布结果中，Fileserver 和 Webserver 负载所造成的磨损率变化曲线相对平缓，而 OLTP 和 MySQL 负载造成的磨损率曲线波动较大。这主要是由于不同负载对文件的访问模式不同而导致的。Fileserver 和 Webserver 在测试过程中会执行较多的读操作，而仅仅对少量文件进行写操作。与此相反，OLTP 和 MySQL 负载在运行过程中包含对文件的大量写操作，因此其造成的磨损率变化曲线也有所不同。

另外，图 4.1(b)中 Fileserver 负载运行 10 小时后，PCV 文件系统造成的磨损率高于 NoWL 文件系统的主要原因在于 Fileserver 负载运行时会执行大量的文件创建和删除操作，在“多链表+轮询”的空间管理策略的帮助下，其造成的磨损已相对均衡。而磨损均衡机制 PCV 的引入使得运行过程中增加了许多无效的索引节点迁移操作，从而导致对存储单元的磨损次数有所增加。

4.2.2 Page 层面磨损分布对比分析

为进一步观察存储域内的磨损分布情况，图 4.2 给出三种文件系统分别运行 10 小时后以物理页为单位统计得到的写次数和写磨损率分布结果，包含 OLTP、Fileserver、Webserver 和 MySQL 四种测试负载程序。横坐标以物理页为单位，纵坐标包括物理页的写次数，以及该物理页的磨损率。图中曲线代表所有物理页的写磨损变化情况，点表示某个物理页对应的写次数。Contour 文件系统分别运行上述负载 10 小时后，物理页写磨损率的标准差均低于 0.95，而其他两种类型文件系统得到的标准差波动较大。

首先以图 4.2 (a)-(c)给出的 OLTP 负载结果为例进行分析。OLTP 负载运行过程中多个线程会对测试文件和日志文件进行大量读写操作，因此如图 4.2(a)所示，NoWL 文件系统造成的磨损不均衡现象较为严重，且其 WG（Wear gap，即最大磨损率与最小磨损率之间的差值）约为 Contour 文件系统的 1874 倍。实现了磨损均衡机制 PCV 的文件系统最终得到的写次数相对均衡，如图 4.2 (b)所示。但由于该方案并未考虑不同存储单元的耐受度差异，导致物理页的写次数大致相同，写磨损率却不同。耐受度较低的存储单元的磨损情况更为严重，这进一步缩短了设备的使用寿命。Contour 文件系统物理页写磨损率的标准差为 0.92，也即对于 OLTP 负载，Contour 最终得到的写磨损率标准差分别是 NoWL 和 PCV 文件系统的 1006 倍和 15.8 倍。



(a) OLTP on NoWL (b) OLTP on PCV (c) OLTP on Contour

(d) Fileserver on NoWL (e) Filesever on PCV (f) Fileserver on Contour

(g) Webserver on NoWL (h) Webserver on PCV (i) Webserver on Contour

(j) MySQL on NoWL (k) MySQL on PCV (l) MySQL on Contour

图 4.2 不同文件系统造成的物理页写磨损分布情况

Fig.4.2 Comparison of page-level write distribution in NoWL, PCV and Contour at 10h

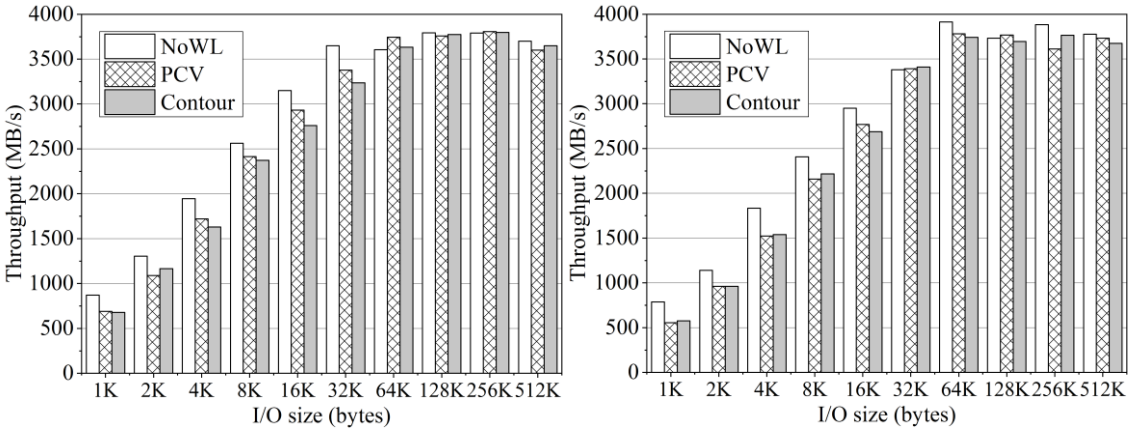
对于 Fileserver、Webserver 和 MySQL 负载，Contour 同样能够实现更加均衡的磨损分布效果。Contour 文件系统运行这三种负载后写磨损率的标准差分别为 0.56、0.95 和 0.49。NoWL 文件系统由于并未对索引节点进行迁移，因此该方案在 Webserver 和 MySQL 负载中可能导致部分物理页的写次数超出其耐受度，也即被磨损穿。而 Fileserver 负载由于其访问模式的特殊性，使得最终的磨损分布相对均

衡。NoWL 文件系统得到的物理页最大写磨损率平均为 Contour 的 1741 倍，并且三种负载得到的写磨损率标准差分别可达 Contour 文件系统的 8.1 倍、287 倍和 418 倍。

PCV 文件系统在 Fileserver、Webserver 和 MySQL 负载上得到的物理页写磨损结果如图 4.2 (e)、(h)、(k)所示。与 OLTP 负载的结果相类似，PCV 文件系统最终导致物理页的写次数相同，但磨损率不同。该文件系统在三种负载上的写磨损率标准差分别为 5.7、3.1 和 2.7。相对于 Contour，其标准差分别高出 9.2、2.2 和 4.5 倍。因此 Contour 文件系统在这三种负载上仍能够得到优于 PCV 的磨损均衡结果。综上分析可得结论，Contour 文件系统能够在物理页层面实现优于其他两种对比方案的磨损均衡效果。

4.3 性能开销评估

本节使用测试工具 FIO 和 Filebench 对上述三种文件系统的写性能进行对比，以说明不同磨损均衡机制引入的文件系统性能开销。其中 FIO 测试可得到不同粒度（I/O size）写文件操作的带宽，包括顺序写和随机写两种模式。Filebench 测试可通过运行 OLTP、Fileserver 和 Webserver 三种负载得到 IOPS（ops/s）。



(a) Sequential write by FIO (b) Random write by FIO

图 4.3 三种文件系统写性能对比

Fig.4.3 File write performance comparison of NoWL, PCV, and Contour

FIO 的测试结果如图 4.3 所示。相对于没有实现磨损均衡的 NoWL 文件系统，顺序写模式下 PCV 和 Contour 文件系统分别引入了约为 6.8%和 8.1%的平均写性能开销。随机写模式下两种文件系统引入的平均性能开销则分别为 8.9%和 8.6%。同时，从图中可以看出当写粒度增加至超过 32KB 时，PCV 和 Contour 两种文件系统所引入的性能开销趋于稳定且相对较小。这是因为随着写粒度的不断增加，导致

磨损均衡机制所引入的时间开销在写操作总时长中的占比不断减小，最终对性能的影响较弱。

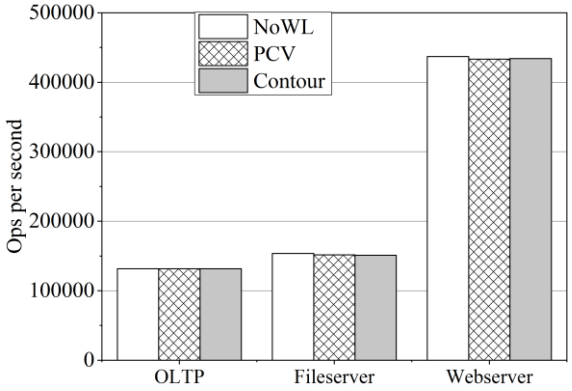


图 4.4 Filebench 性能对比

Fig.4.4 Filebench performance comparison of NoWL, PCV, and Contour

Filebench 的测试结果如图 4.4 所示。从图中可以看出，三种文件系统分别运行 OLTP、Fileserver 和 Webserver 负载得到的 IOPS 相差极小。其中 PCV 和 Contour 在三种负载下测试得到的 IOPS 平均为 NoWL 文件系统的 99.9%、98.4%和 97.32%。也即两种磨损均衡机制所引入的性能开销是可忽略不计的。

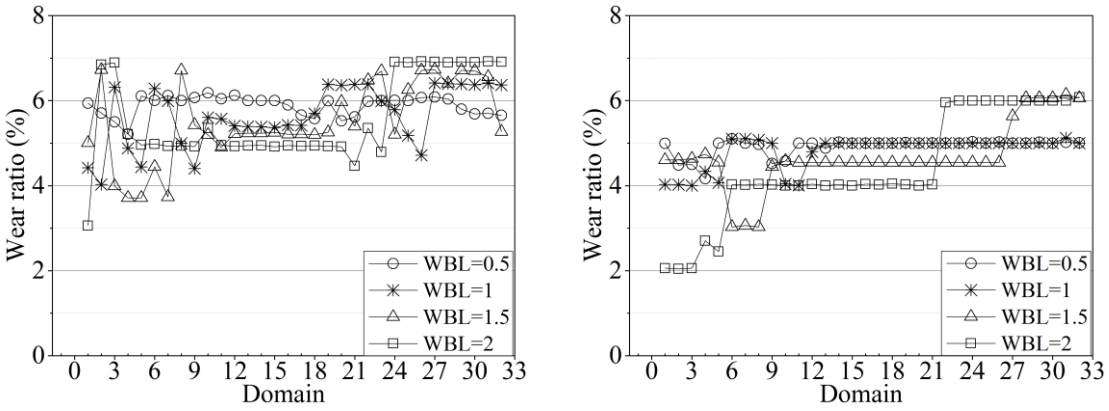
综上分析可得出结论，Contour 与 PCV 两种磨损均衡机制最终引入的性能开销相差不超过 1.27%，并且 Contour 能够实现更加均衡的磨损分布效果。除性能开销外，在文件系统中引入磨损均衡可能会带来额外的磨损开销。Contour 文件系统中磨损开销的来源主要分为两个部分，一是由迁移索引节点而导致的额外写操作，二是迁移过程中对映射表的更新操作。

如前文所述，Contour 在设计和实现过程中尽可能地避免没有必要的迁移操作，以此尽可能减少对映射表和存储设备的磨损。通过数据分析得到 Contour 在运行四种测试负载 10 小时后，对存储设备所造成的额外磨损平均低于 1%，并且该数值高出 PCV 方案 0.06%。区别于 PCV，Contour 重点关注耐受度较低的存储单元，并使用统一的磨损率参数 WBL 来保证所有存储单元具有一致的磨损率，而非磨损次数。因此 Contour 需要相对较多的迁移次数，将索引节点从耐受度较低的存储单元迁移至耐受度较高的位置。这一思路最终使得 Contour 能够实现优于 PCV 的磨损均衡效果，并且引入的额外开销是可忽略不计的。

4.4 参数敏感度分析

如前文所述，域间磨损均衡机制以运行周期为单位，以磨损率参数 WBL 为步

长来保证多个存储域的磨损率均衡。为了说明该参数对 Contour 所造成磨损均衡效果的影响，本节对该参数的不同赋值进行调节分析。以 Filebench 中的典型负载 Webserver 和数据库应用程序 MySQL 为例，分别测试 WBL 值为 0.5%、1%、1.5%和 2%时的存储域磨损率分布和性能开销情况。



(a) Webserver (b) MySQL

图 4.5 Websever 和 MySQL 负载对 WBL 参数的敏感程度

Fig.4.5 Sensitivity of tuning write budget line in Webserver and MySQL

测试结果如图 4.5 所示，其中横坐标表示文件系统索引节点区域所在的 32 个存储域，纵坐标表示运行 10 小时后每个存储域的磨损率。从图 4.5(a)可以看出，Webserver 负载模式下 WBL 参数数值越大，写磨损率分布曲线波动程度越大。WBL 值为 0.5%、1%、1.5%和 2%时，磨损率的标准差分别为 0.0023、0.0071、0.0094 和 0.0103，也即磨损率标准差随 WBL 参数的增加而增加。与此同时，存储域磨损率的最大值和最小值之差也从 0.97%（WBL=0.5%）增加至 3.86%（WBL=2%）。因此从存储域磨损率的角度来看，WBL 参数数值越小，Webserver 负载最终得到的磨损分布效果越均衡。

通过分析图 4.5(b)中给出的 MySQL 负载结果可得出相似的结论。WBL 值为 0.5%、1%、1.5%和 2%时，MySQL 负载所造成的磨损率标准差分别为 0.0021、0.0039、0.0077 和 0.013，也即随着 WBL 参数的增加，磨损率曲线波动程度也随之增加。因此通过以上分析可知，WBL 参数越小，Contour 所取得的磨损均衡效果越好。

需要说明的是，上述 Webserver 和 MySQL 磨损测试结果中，存储域磨损率的最大值仅分别为 6.92%和 6.14%，使得图中曲线波动效果不够突出。导致磨损率数值较低的原因主要有两个方面。一方面是由于该实验结果是通过运行 Contour 磨损均衡机制而得出的，也即在初始文件系统中所造成的磨损程度已经远高于该数值。另一方面，该结果是通过模拟真实应用场景下对物理存储设备进行大量写操作而

得出的。理论上模拟的总写次数越多，得到的数据和结论就越准确。考虑到上述测试所造成的累计写次数最大值分别可达 4.86 亿次（Webserver）和 4.09 亿次（MySQL），因此综合分析认为本次测试结果能够一定程度上反映磨损均衡方案 Contour 对参数 WBL 的有效敏感情况。

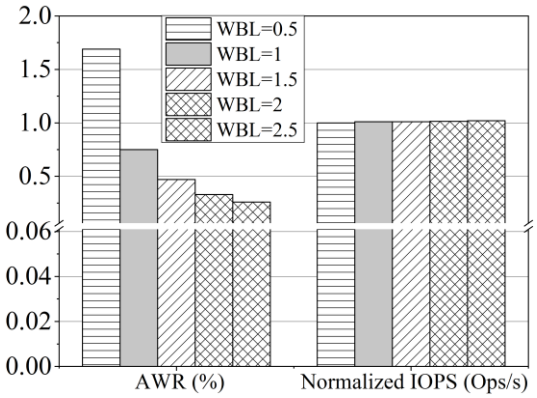


图 4.6 不同 WBL 参数下 Webserver 负载的开销对比

Fig.4.6 Overhead analysis of tuning write budget line in Webserver

图 4.6 给出了参数 WBL 在四种不同取值下 Webserver 负载所得到的磨损开销和性能开销情况。其中，磨损开销指的是由于索引节点迁移操作而对 NVM 产生的额外磨损。从图中可以看出，随着 WBL 数值的不断增加，所引入的磨损开销也越小。当该参数数值较大时，每个存储域在当前运行周期内所允许的最大写次数也相对较多，从而避免了许多不必要的迁移，也即使得迁移次数减少。对应地由于迁移操作而引入的额外磨损开销也就不断减少。不同 WBL 参数设置所引入的性能开销可忽略不计。以 WBL 值为 0.5%时的 IOPS 为基准值，WBL 值为 1%、1.5%和 2%时的 Webserver 性能提升分别为 1%、1.08%和 1.46%。

综上分析可以发现，当 WBL 数值大于 1%时存储域的写磨损率变化波动较大，并且随着该参数数值的增加，Contour 文件系统引入的磨损开销和性能开销逐步减小。因此为了在磨损均衡效果和性能开销之间进行权衡，Contour 在测试过程中将 WBL 值设置为 1%。此外，虽然本文在测试时为所有负载设置了相同的 WBL 参数且取得的磨损均衡效果相对较好，但由于不同负载模式可能仍然对应着不同的参数最优设置。因此根据负载模式的变化对参数进行动态调节的方法也是目前可研究的热点问题之一，以此期望在不同负载模式下均能够取得最优的磨损分布结果。

4.5 本章小结

本章对三种文件系统的磨损分布和性能开销进行实验对比。首先通过运行

Filebench 的三种典型负载和数据库应用程序 MySQL 进行磨损分布测试，并分别从 Domain 和 Page 两个物理层面观察不同文件系统造成的磨损分布情况。其次使用 FIO 和 Filebench 两种测试工具对文件系统的性能开销进行测试。实验结果表明，相对于现有磨损均衡机制 PCV，本文提出的磨损均衡机制 Contour 能够在不同运行时长下均取得较优的磨损均衡效果，并且引入的性能开销是可忽略不计的。此外对比了参数 WBL 在不同取值下对磨损分布和性能开销的影响，以说明磨损均衡机制对该参数的敏感程度。

5 总结与展望

5.1 总结

本文针对制程差异环境下非易失内存文件系统中索引节点磨损不均衡问题进行研究。分析得到导致该问题的三点主要原因：① 索引节点需要频繁地即时写回；② 应用程序负载模式不同，并且现有索引节点空间管理策略存在缺陷；③ 索引节点极少发生迁移。并分别从索引节点空闲空间管理策略和磨损均衡机制两个方面进行优化和改进。

首先，为避免传统基于单链表的空间管理策略可能带来的缺陷，本文提出了域间差异感知的多链表空间管理策略。多链表的引入一定程度上提高了节点分配、删除操作的效率，并且该策略能够有效缓解链表头尾节点磨损分布不均的问题。

其次，本文提出制程差异感知的索引节点磨损均衡机制，并将索引节点磨损均衡这一目标分为两个部分进行设计和实现。域间磨损均衡机制中，基于 WBL 参数的磨损率均衡机制以运行周期为单位，限定存储域在单个周期内所允许的最大写次数，以对多个存储的写磨损率进行均衡。基于权重的索引节点与存储域匹配算法能够在运行过程中实现索引节点与存储域的动态匹配，进而得以在存储域之间完成索引节点的动态迁移。实验结果表明，该机制能够在存储域层面实现较好的磨损均衡效果，存储域磨损率的标准差平均优于 NoWL 和 PCV 文件系统 14.8 倍和 11.1 倍。

域内磨损均衡机制注重单个存储域内多个存储单元之间的写次数均衡。在存储域最优匹配算法的配合下，该算法能够在物理页层面实现较好的磨损均衡效果，测试所采用的四种负载可得到的物理页磨损率标准差平均仅为 0.62。

本文所提出的磨损均衡机制 Contour，通过对制程差异环境下的耐受度差异的特征进行针对性优化，进而能够以极小的性能开销，在文件系统层面实现较优的索引节点磨损均衡效果。与此同时，该方案可扩展性较强，能够有效适用于现有多种非易失内存文件系统。并且引入磨损均衡机制可进一步提高应用程序从磁盘文件系统迁移至非易失内存文件系统的可能性，有效缓解这些应用程序在进行大量数据读写操作时对存储设备造成的磨损不均衡现象，进而延长设备的使用寿命。

5.2 展望

本文所提出的空间管理策略和磨损均衡机制均实现在文件系统层面，能够有效缓解文件系统中索引节点所在区域的磨损不均衡问题。与此同时，方案设计和实现过程中也存在部分遗留问题有待进一步地研究和探讨。

首先，文件系统的挂载区间一般可分为索引节点区域、数据区域和日志区域等多个关键部分。由于索引节点所在区域的磨损不均衡问题最为严重，因此本文将文件系统的索引节点区域作为主要的研究对象。文件系统中其他区域和索引节点区域之间存在的部分差异，比如组织结构、存储粒度、读写模式等，使得本文所提出的空间管理和磨损均衡机制无法直接应用于其他区域，因此相关的磨损均衡机制仍需进一步优化和改进。

其次，从实验结果可以明显看出，由于应用程序负载模式的不同，最终对 NVM 设备所造成的磨损分布情况也有所差异。为了能够在不同负载模式下均取得最优的磨损分布结果，针对负载模式的变化而进行的参数动态设置，也是可研究的热点问题之一。此外，本文所提出方案在运行时长不断增加的情况下，存储域磨损率曲线能够实现近似水平向上移动的变化趋势。因此测试运行时间越长，越能够体现该方案的优越性。而由于受到时间的限制，本文实验过程中选择对多个应用负载分别进行时长为 10 小时的读写模拟测试。因此在后续的论文完善过程中，运行时长的进一步增加有助于对该方案的持续优化。

参考文献

[1] B. C. Lee, P. Zhou, J. Yang, et al. Phase-Change Technology and the Future of Main Memory[J]. IEEE Micro, 2010,30(1): 143.

[2] Intel Optane Memory[EB/OL]. https://www.intel.com/content/www/us/en/architecture-and-technolo -gy/optane-memory.html, 2019.

[3] M. Jung, J. Shalf, M. Kandemir. Design of a Large-Scale Storage-Class RRAM System[C]. Proceedings of the 27th international ACM conference on International conference on supercomputing. 2013: 103-114.

[4] S. S. P. Parkin, M. Hayashi, L. Thomas. Magnetic Domain-Wall Racetrack Memory[J]. Science, 2008,320(5873): 190-194.

[5] E. H. Sha, X. Chen, Q. Zhuge, et al. A New Design of In-Memory File System Based on File Virtual Address Framework[J]. IEEE Transactions on Computers, 2016,65(10): 2959-2972.

[6] E. Lee, S. H. Yoo, H. Bahn. Design and Implementation of a Journaling File System for Phase-Change Memory[J]. IEEE Transactions on Computers, 2015,64(5): 1349-1360.

[7] X. Wu, A. L. N. Reddy. SCMFS: A File System for Storage Class Memory[C]. Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. 2011: 1-11.

[8] K. Zeng, Y. Lu, H. Wan, et al. Efficient Storage Management for Aged File Systems On Persistent Memory[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). 2017: 1769-1774.

[9] S. R. Dulloor, S. Kumar, A. Keshavamurthy, et al. System Software for Persistent Memory[C]. Proceedings of the Ninth European Conference on Computer Systems. 2014: 1-15.

[10] J. Xu, S. Swanson. NOVA: A Log-Structured File System for Hybrid Volatile/Non-volatile Main Memories[C]. Proceedings of 14th USENIX Conference on File and Storage Technologies. 2016: 323-338.

[11] J. Ou, J. Shu, Y. Lu. A High Performance File System for Non-Volatile Main Memory[C]. Proceedings of the Eleventh European Conference on Computer Systems. 2016: 1-16.

[12] J. Condit, E. B. Nightingale, C. Frost, et al. Better I/O through Byte-Addressable, Persistent Memory[C]. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009: 133-146.

[13] 陈祥，肖侬，刘芳. 新型非易失存储I/O栈综述[J]. 计算机研究与发展, 2014(s1):18-24.

[14] 张鸿斌，范捷，舒继武等. 基于相变存储器的存储系统与技术综述[J]. 计算机研究与发展，

2014,51(8):1647-1662.

[15] M. Chang, S. Shen. A Process Variation Tolerant Embedded Split-Gate Flash Memory Using Pre-Stable Current Sensing Scheme[J]. IEEE Journal of Solid-State Circuits, 2009,44(3): 987-994.

[16] L. Shi, Y. Di, M. Zhao, et al. Exploiting Process Variation for Write Performance Improvement on NAND Flash Memory Storage Systems[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016,24(1): 334-337.

[17] Y. Di, L. Shi, K. Wu, et al. Exploiting Process Variation for Retention Induced Refresh Minimization On Flash Memory[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). 2016: 391-396.

[18] Y. Luo, S. Ghose, Y. Cai, et al. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2018,2(3): 1-48.

[19] 陆游游, 舒继武. 闪存存储系统综述[J]. 计算机研究与发展, 2013,50(1):49-59.

[20] M. K. Qureshi, V. Srinivasan, J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology[C]. Proceedings of the 36th annual international symposium on Computer architecture. 2009: 24-33.

[21] F. Huang, D. Feng, Y. Hua, et al. A Wear-Leveling-Aware Counter Mode for Data Encryption in Non-Volatile Memories[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). IEEE, 2017: 910-913.

[22] H. S. P. Wong, S. Raoux, S. B. Kim, et al. Phase Change Memory[J]. Proceedings of the IEEE, 2010,98(12): 2201-2227.

[23] W. Zhang, L. Tao. Characterizing and Mitigating the Impact of Process Variations On Phase Change Based Memory Systems[C]. Proceedings of 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2009: 2-13.

[24] A. P. Ferreira, S. Bock, B. Childers, et al. Impact of Process Variation On Endurance Algorithms for Wear-Prone Memories[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). IEEE, 2011: 1-6.

[25] R. F. Freitas, W. W. Wilcke. Storage-Class Memory: The Next Storage System Technology[J]. Ibm Journal of Research & Development, 2008,52(4-5): 439-447.

[26] C. Xu, P. Y. Chen, D. Niu, et al. Architecting 3D Vertical Resistive Memory for Next-Generation Storage Systems[J]. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2014

[27] 冒伟, 刘景宁, 童薇等. 基于相变存储器的存储技术研究综述[J]. 计算机学报,

2015(5):38-54.

[28] 肖仁智，冯丹，胡燏翀等. 面向非易失内存的数据一致性研究综述[J]. 计算机研究与发展, 2020.

[29] H. Volos, G. Magalhaes, L. Cherkasova, et al. Quartz: A Lightweight Performance Emulator for Persistent Memory Software[C]. Proceedings of the 16th Annual Middleware Conference. ACM, 2015: 37-49.

[30] Z. Duan, H. Liu, X. Liao, et al. HME: A Lightweight Emulator for Hybrid Memory[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). IEEE, 2018: 1375-1380.

[31] N. L. Binkert, B. M. Beckmann, G. Black, et al. The Gem5 Simulator[J]. Acm Sigarch Computer Architecture News, 2011,39(2): 1-7.

[32] Y. Joo, D. Niu, X. Dong, et al. Energy- and Endurance-Aware Design of Phase Change Memory Caches[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). 2010: 136-141.

[33] S. Cho, H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance[C]. Proceedings of 42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009). ACM, 2009: 347-357.

[34] S. Zheng, L. Huang, H. Liu, et al. HMVFS: A Hybrid Memory Versioning File System[C]. Proceedings of 32nd Symposium on Mass Storage Systems and Technologies (MSST). 2016: 1-14.

[35] X. Chen, E. H. Sha, Y. Zeng, et al. Efficient Wear Leveling for Inodes of File Systems On Persistent Memories[C]. Proceedings of the conference on Design, Automation & Test in Europe (DATE). IEEE, 2018: 1524-1527.

[36] H. Chang, Y. Chang, P. Hsiu, et al. Marching-Based Wear-Leveling for PCM-Based Storage Systems[J]. ACM Transactions on Design Automation of Electronic Systems, 2015,20(2): 1-22.

[37] J. Dong, L. Zhang, Y. Han, et al. Wear Rate Leveling: Lifetime Enhancement of PRAM with Endurance Variation[C]. Proceedings of the 48th Design Automation Conference. ACM, 2011: 972-977.

[38] J. Yun, S. Lee, S. Yoo. Dynamic Wear Leveling for Phase-Change Memories with Endurance Variations[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2015,23(9): 1604-1615.

[39] X. Zhang, G. Sun. Toss-Up Wear Leveling: Protecting Phase-Change Memories from Inconsistent Write Patterns[C]. Proceedings of the 54th Annual Design Automation Conference. ACM, 2017: 1-6.

[40] M. Zhao, L. Jiang, Y. Zhang, et al. SLC-enabled Wear Leveling for ML C PCM Considering Process Variation[C]. Proceedings of the 51st Annual Design Automation Conference. ACM, 2014: 1-6.

[41] M. Zhao, L. Jiang, L. Shi, et al. Wear Relief for High-Density Phase Change Memory through Cell Morphing Considering Process Variation[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015,34(2): 227-237.

[42] V. Tarasov, E. Zadok, S. Shepler. Filebench: A Flexible Framework for File System Benchmarking[J]. USENIX, 2016,41(1): 6-12.

[43] FIO: Flexible I/O Tester[EB/OL]. http://freecode.com/projects/FIO, 2019.

[44] MySQL[EB/OL]. https://www.mysql.com/, 2019.

[45] W. Zhou, D. Feng, Y. Hua, et al. Increasing Lifetime and Security of Phase-Change Memory with Endurance Variation[C]. Proceedings of IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2016: 861-868.

[46] J. Xu, D. Feng, Y. Hua, et al. An Efficient Spare-Line Replacement Scheme to Enhance NVM Security[C]. Proceedings of the 56th Annual Design Automation Conference (DAC). ACM, 2019: 1-6.

[47] R. Zwisler. Add Support for New Persistent Memory Instructions[EB/OL]. https://lwn.net/Articles/619851/, 2019.

[48] Intel Architecture Instruction Set Extensions Programming Reference[EB/OL]. https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf, 2019.

[49] Intel. Persistent Memory Programming[EB/OL]. https://pmem.io/2016/02/22/pm-emulation.html, 2019.

[50] Mysqlslap[EB/OL]. https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html, 2019.

附 录

A. 作者在攻读学位期间发表的论文

[1] **王鑫鑫**, 诸葛晴凤, 吴林. 面向非易失性内存文件系统的 NVM 模拟与验证方法[J]. 计算机科学, 2019.

B. 作者在攻读学位期间参加的科研项目

[1] 国家自然科学青年基金, 面向非易失性内存存储的性能和耐久性优化关键技术研究, 2019 年 1 月至 2021 年 12 月.

[2] 中国博士后科学基金面上资助(一等), 面向非易失性内存的文件系统设计与优化研究, 2017 年 11 月至 2019 年 10 月.

[3] 科技部国家高技术研究发展计划(863 计划), 面向大数据应用的新型内存计算系统软件及关键技术, 2015 年 1 月至 2017 年 12 月.

C. 学位论文数据集

<table>
<tr><td colspan="2">关键词</td><td colspan="2">密级</td><td colspan="2">中图分类号</td></tr>
<tr><td colspan="2">非易失性内存；磨损均衡</td><td colspan="2">公开</td><td colspan="2">TP311.1</td></tr>
<tr><td>学位授予单位名称</td><td colspan="2">学位授予单位代码</td><td>学位类别</td><td colspan="2">学位级别</td></tr>
<tr><td>重庆大学</td><td colspan="2">10611</td><td>学术学位</td><td colspan="2">硕士</td></tr>
<tr><td colspan="2">论文题名</td><td colspan="2">并列题名</td><td colspan="2">论文语种</td></tr>
<tr><td colspan="2">带制程差异的非易失内存文件系统索引节点磨损优化研究</td><td colspan="2">无</td><td colspan="2">中文</td></tr>
<tr><td>作者姓名</td><td colspan="2">王鑫鑫</td><td>学号</td><td colspan="2">20171402005t</td></tr>
<tr><td colspan="3">培养单位名称</td><td colspan="3">培养单位代码</td></tr>
<tr><td colspan="3">重庆大学</td><td colspan="3">10611</td></tr>
<tr><td>学科专业</td><td>研究方向</td><td>学制</td><td colspan="3">学位授予年</td></tr>
<tr><td>计算机科学与技术</td><td>内存文件系统</td><td>3</td><td colspan="3">2020</td></tr>
<tr><td>论文提交日期</td><td>2020 年 6 月</td><td>论文总页数</td><td colspan="3">53</td></tr>
<tr><td>导师姓名</td><td>刘铎</td><td>职称</td><td colspan="3">教授</td></tr>
<tr><td colspan="3">答辩委员会主席</td><td colspan="3">陈武</td></tr>
<tr><td colspan="6">电子版论文提交格式
文本（✓） 图像（） 视频（） 音频（） 多媒体（） 其他（）</td></tr>
</table>

致 谢

硕士阶段已接近尾声。在此，特向过去三年学习生活以及本论文完成过程中，给予我帮助和支持的老师、同学、家人们表示最诚挚的感谢。

首先，我要感谢沙行勉老师和诸葛晴凤老师在前期阶段对我的指导和帮助。他们所提供的良好学习工作氛围、优质的科研条件等，都为我后续的科研工作打下了坚实的基础。其次，感谢我的导师刘铎老师。在我融入研究团队及本论文的完成过程中，都给予了充分支持和理解，丰富的项目实践经历也开阔了我的眼界。

之后，我要感谢陈咸彰师兄在本论文完成过程中提供的帮助。从选题到论文的最终完成，每一次讨论都使我受益匪浅。不仅学习到了在遇到问题时该如何着手分析并解决，也对研究方向和内容有了更加积极和深刻的认识。

其次，我要感谢三年来实验室的师兄师姐们在学习和生活等多方面对我的帮助和影响，无论是初来乍到时的生活起居，还是刚入门时对研究内容的迷惑，在他们的帮助和共同努力下，我才得以顺利地融入整个团队。

还要感谢我亲爱的家人们、朋友以及各位同学们，我们之间的情谊以及他们所给予我的支持和帮助，使得我在面对困难时能够积极地走下去。

最后，向在百忙之中抽出时间对本论文进行评阅、以及参与答辩的各位专家表示衷心地感谢！

王鑫鑫

二O二O年六月 于重庆