

**本科生毕业设计[论文]**

**非易失性内存文件系统的优化研究**

院 系 计算机科学与技术学院

专业班级 CS1410

姓 名 彭 周 旋

学 号 U201414833

指导教师 冯 丹

2018年 4 月 8 日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

摘 要

新兴的非易失性内存技术既具有与传统DRAM主存相近的低延迟和字节寻址特性，又具有类似硬盘等外存储设备的非易失存储特性，已逐渐对系统架构、系统软件、库和应用程序等产生深远的影响，并引起广泛的关注和研究。

非易失内存文件系统（Persistent Memeory File System，简称PMFS）是一个针对非易失内存（Persisitent Memory，以下简称PM）设计的轻量级的、符合POSIX文件系统标准的文件系统，该文件系统利用了PM的可字节寻址特性避免了面向块存储的开销，并使得应用程序通过内存映射I/O能够直接访问PM。PMFS的一些设计采取了简化的策略（如分配器），存在性能上的问题。本文以PMFS为基础，重点对PMFS的分配器等进行优化研究，使得PMFS获得更好的性能和效率。

PMFS的分配器采用线性链表的方式组织文件系统的数据块，并在已分配的数据块上建立索引，在分配据块时通过索引来加速查找过程。由于PMFS中分配器的频繁调用，这种时间复杂度趋近于O(n)的组织方式存在严重的效率问题。本文中提出一种优化方案可以将分配过程的时间复杂度降为常数时间，并且不提高回收过程的时间复杂度。在优化方案中，取消在已分配块上建立的索引，而在空闲块上建立索引，根据PMFS所支持的三种处理器页面大小，将连续的空闲块组根据长度拆分成三条链表，在分配块时可直接在对应的链表取出表头结点即可满足要求，无需遍历查找，以达到提高效率的目的。实验结果表明，这种优化的方案以较小的空间代价，取得了性能上的明显提升。

**关键词：**非易失内存；PMFS；文件系统；分配器

**Abstract**

Emerging non-volatile memory technology not only has low latency and byte addressing characteristics similar to traditional DRAM main memory, but also has non-volatile memory characteristics like external hard disk and other storage devices, and has gradually evolved into system architecture, system software, Libraries and applications have had far-reaching effects and have attracted widespread attention and research.

The Persistent Memeory File System (PMFS) is a lightweight, POSIX-compliant file system designed for non-volatile memory (PM). The byte-addressable nature of PM avoids the overhead of block-oriented storage and enables applications to directly access PM through memory mapped I/O. Some designs of PMFS have adopted simplified tactics (such as distributors) and there are performance problems. Based on PMFS, this paper focuses on the optimization research of the PMFS distributors, which makes PMFS get better performance and efficiency.

The PMFS allocator uses a linear linked list to organize the data blocks of the file system, and builds an index on the allocated data blocks. The indexes are used to speed up the search process when data blocks are allocated. Due to the frequent call of the distributor in the PMFS, there is a serious efficiency problem in the organization of this time complexity approaching O(n). In this paper, an optimization scheme is proposed to reduce the time complexity of the allocation process to a constant time, and does not increase the time complexity of the recovery process. In the optimization scheme, the index created on the allocated block is canceled, and the index is created on the free block. According to the three processor page sizes supported by the PMFS, the continuous free block group is split into three linked lists according to the length. When allocating a block, the header node can be directly taken out from the corresponding list to meet the requirement without traversing the search, so as to improve the efficiency. Experimental results show that this optimized solution achieves a significant improvement in performance with a small space cost.

**Key Words：**Non-volatile Memory; PMFS; File System; Allocator

**目 录**

[**1 绪论** 3](#_Toc514195277)

[1.1 课题背景 3](#_Toc514195278)

[1.2 国内外研究现况及发展趋势 2](#_Toc514195279)

[1.3 本课题主要工作 3](#_Toc514195280)

[1.4 本文组织结构 4](#_Toc514195281)

[**2 PMFS文件系统概述** 1](#_Toc514195282)

[2.1 PMFS的应用背景 1](#_Toc514195283)

[2.2 PMFS在内核中的架构及空间布局 1](#_Toc514195284)

[2.3 PMFS分配器的设计与不足 4](#_Toc514195285)

[2.4 PMFS分配器的优化思路 8](#_Toc514195286)

[2.5 本章小结 8](#_Toc514195287)

[**3 PMFS分配器的优化设计** 1](#_Toc514195288)

[3.1 分配器调用过程优化分析 1](#_Toc514195289)

[3.2 分配器组织方式优化分析及设计 2](#_Toc514195290)

[3.3 分配/回收策略优化分析及设计 4](#_Toc514195291)

[3.4 优化方案整合及效率分析 5](#_Toc514195292)

[3.5 本章小结 6](#_Toc514195293)

[**4 PMFS分配器的优化实现** 1](#_Toc514195294)

[4.1 组织方式的优化实现 1](#_Toc514195295)

[4.2 分配/回收策略优化实现 3](#_Toc514195296)

[4.3 本章小结 6](#_Toc514195297)

[**5 性能测试与分析** 1](#_Toc514195298)

[5.1 测试环境 1](#_Toc514195299)

[5.2 测试方法 1](#_Toc514195300)

[5.3 测试结果及分析 3](#_Toc514195301)

[5.4 本章小结 5](#_Toc514195302)

[**6 总结和展望** 1](#_Toc514195303)

[**致 谢** 1](#_Toc514195304)

[**参考文献** 1](#_Toc514195305)

[**附录** 1](#_Toc514195306)

# 1 绪论

1.1 课题背景

随着非易失内存技术的兴起和普及，计算机存储体系结构正面临着全新的发展机遇。传统的存储体系按访存延时从低到高的顺序由缓存，主存和辅存等三个层次构成，然而，非易失内存的的出现将改变这一局面。非易失内存在存储体系中可以扮演多种角色：1）由于非易失内存可以提供与DRAM在同一个数量级上的性能，因此非易失内存可以作为当前主流的DRAM主存的替代品；2）作为存储体系结构中位于主存（DRAM）和外存（机械硬盘或固态硬盘）之间的一个全新的存储层级；3）作为一个独立的内存区域，与DRAM和硬盘等并行存在[9]。非易失内存已经对系统架构、系统软件、库和应用程序产生深远的影响[10,11,12,13]。由于非易失内存在存储体系中的灵活应用，计算机如何管理非易失内存的话题也逐渐得到重视。传统上，操作系统将易失性的内存（虚拟内存管理器VMM）和外存（文件系统，块设备驱动）的管理分开，而非易失内存兼顾内存的字节寻址和外存的非易失存储的特性，非易失内存的管理有多种选择：1）扩展VMM用于管理非易失内存；2）为非易失内存实现一个块设备从而使用现有的的文件系统来管理；3）实现一个针对非易失内存优化的、不经过块层的文件系统。Subramanya R Dulloor等人采取了第三种策略，实现了一个轻量级、符合POSIX文件系统标准的非易失内存文件系统，即PMFS[6]。该文件系统利用了非易失内存可字节寻址的特性避免了面向块存储的开销，并且使得应用程序通过内存映射I/O能够直接访问非易失内存。相较于另外两种方式，PMFS具有许多优势：1）支持传统应用程序。PMFS实现了许多存储密集型应用程序所依赖的完整的符合POSIX标准的文件系统接口；2）支持轻量级的文件系统。通过针对PM进行优化和避免块层，PMFS消除了复制开销，并为传统的应用程序提供了大量的好处；3）优化的内存映射I/O。PMFS通过直接将PM的页面映射到应用程序的地址空间中去以避免内存映射时的复制页面开销，同时PMFS实现了诸如透明的大页面支持等特征，以进一步优化内存映射I/O。本课题的研究正是基于PMFS展开的，尽管PMFS已经取得了可喜的成绩，但是其本身仍然存在许多可优化的地方，对于PMFS的研究也远远没有到尽头。本课题提出对PMFS的并行性（在Subramanya R Dullor论文中未提到的）进行研究，并尝试进行优化。对PMFS的进一步研究和优化，有利于非易失内存相关的技术以及PMFS本身更快的发展和普及。

1.2 国内外研究现况及发展趋势

当前国际上针对非易失内存设计的文件系统主要有SCMFS，BPFS，Object-based File System以及PMFS。Wu Xiaojian等人根据SCM（Storage Class Memory）特性提出了专门设计应用于SCM设备的文件系统，简称SCMFS[15]，其中SCM设备直接连接到内存总线上，既用作内存设备，又可以作永久性存储设备。SCM使用传统文件系统接口，挂载在系统虚拟地址空间上，并通过内存管理模块MMU[16]完成虚拟地址到物理地址的映射，大大简化了SCMFS的开发。

基于可字节寻址的持久RAM（BPRAM）文件系统BPFS由Condit J等人提出[17]，BPFS根据BtrFS组织元数据的机制，使用两层树结构的空间框架，并使用短电路影子分页的技术提供可靠性保证。BPFS在文件管理方面比传统文件系统具有更好的性能，有较好的数据持久性和一致性，但其实现需要有相应的硬件支持，移植性不好，且文件系统实现使用的PCM比DRAM访问速度慢，使得文件系统影响了系统的整体性能。

Object-based File System由UCSC（University of California，Santa Cruz）的Yangwook Kang等人提出[18,19]，采用基于对象的方式设计文件系统，具有较好的可移植性。它将文件管理与存储管理分离，让存储管理根据应用与设备特性做优化，取代文件系统成为所有相关应用的底层，减少软件层次，简化应用程序设计。

Subramanya R Dullor等人提出的PMFS是一个轻量级的、符合POSIX文件系统标准的非易失内存文件系统。在PMFS的实现中，Subramanya R Dullor等人做了大量的研究和努力，使得PMFS相较于一些传统的文件系统在文件I/O、文件应用、内存映射I/O等方面的性能上取得了令人欣喜的成绩。PMFS的突出贡献包括利用处理器的分页和内存排序功能进行了优化，例如针对一致性而使用的精细日志记录，为更快的内存映射I/O而实现的透明大页面支持等；实现了一个硬件原语（PM写屏障）用于提供强一致性保证；使用处理器现有功能保护非易失内存免于杂散写入等。

而目前针对非易失内存的研究趋势，主要可以分为两类：（1）兼容传统主存控制器的非易失内存（Non-Volatile Memory, NVM）；（2）连接在I/O总线上的存储级内存（Storage-Class-Memory, SCM）[20]。NVM的特点是可直接替换传统的DRAM内存或两者同时使用构成混合主存，其优势是NVM主存由硬件控制，只需针对NVM主存的特性对主存控制器进行适当的改进而无需引入系统级的改动。SCM是对介于传统DRAM主存和HDD外存之间的存储层次所使用的存储设备的统称。相比于DRAM，具备非易失、低价格、低功耗的优势；相比于HDD具备无运动部件、低延迟、高吞吐率的优势[10]。

1.3 本课题主要工作

本论文主要研究非易失内存文件系统PMFS的分配器的组织方式和分配/回收策略，并探寻其中可进行并行性优化的要素，对二者提出优化的设计方案，并进行实现。

（1）优化分配器的组织方式。PMFS分配器将所有的数据块（包括已分配和空闲的）组织成一条线性链表，并在连续已分配块上建立的索引，当分配器进行块的分配或回收时就根据索引依次遍历该链表，直到找到满足需求的块。而在优化方案中，分配器在空闲块上建立索引，并根据空闲块的长度拆分为三条链表，消除了分配块时遍历空间带来的时间开销。

（2）优化的分配策略。PMFS分配器分配空闲块时每次都需要进行遍历，并且根据找到的空闲块位置采取左对齐、右对齐或居中等不同的策略，而居中策略会导致产生更多的碎片。在优化方案中，由于新的组织方式的良好特性，在分配时每次都直接从表头取空闲块而无需遍历，并且每次在需求块小于空闲块长度时都可以采取左对齐策略，减少碎片化。

1.4 本文组织结构

全文一共分为六章，组织内容安排如下：

第一章阐述非易失内存及非易失内存文件系统的研究背景和意义，并介绍了国内外关于非易失内存文件系统的研究现况，以及非易失内存技术的发展趋势，说明了本文的主要工作。

第二章简要介绍了非易失内存文件系统PMFS，着重讲述PMFS的应用背景、总体布局，研究PMFS的分配器组织方式和分配/回收策略，发现PMFS分配器性能方面的不足，针对组织方式和分配策略提出优化方案。

第三章提出对PMFS分配器进行优化的各种不同策略，经过详细的分析和论证后确定最终优化方案，并详细阐述了PMFS组织方式和分配策略的优化设计方案。

第四章根据上述提出的优化方案，详细讲述了PMFS分配器组织方式和分配策略优化的具体实现。

第五章介绍了PMFS文件系统的测试方法和测试平台，对优化前后的性能进行了测试和对比，并详细分析了文件系统测试结果。

第六章对全文进行总结并展望未来工作。

# 2 PMFS文件系统概述

2.1 PMFS的应用背景

由于新兴的非易失内存技术兼具大容量、字节编址、低延迟及存储非易失等特性，及其在传统存储体系结构层次中的参与方案的多样性和灵活性，计算机对于非易失内存的管理也呈现出多种选择。传统上，操作系统将内存和外存的管理分开，用虚拟内存管理器（Virtual Memory Manager，VMM）管理内存，用文件系统和块设备驱动等来管理外存储设备。然而非易失内存兼具内、外存的特性，在管理上，既可以扩展虚拟内存管理器来管理非易失内存，也可以为现有的文件系统（如Ext4）使用非易失内存实现块设备驱动，还可以在不经过块层的情况下实现一个针对非易失内存的文件系统。

正是在上述背景下，SR Dulloor等人设计了一个针对非易失内存进行优化的、符合POSIX标准的文件系统，即PMFS。并且提出PMFS作为管理非易失内存的系统软件层相较于其他方式具有许多优势。一方面，PMFS实现了完整的符合POSIX标准的文件系统接口，对传统的应用程序有更好的支持；另一方面，通过针对非易失内存进行优化和避免块层，PMFS消除了复制开销，支持轻量级的文件系统。此外，在传统的文件系统实现中，内存映射I/O将会先复制访问的页面到DRAM中，而PMFS可以通过直接将非易失内存的页面映射到应用程序的地址空间中以避免这种开销，PMFS还实现了其他的一些特征，以进一步优化内存映射I/O。

PMFS的开源源码由Intel发布在github网站上（网址为：<https://github.com/linux-pmfs/pmfs>）[2]，基于开发环境和兼容性的考虑，PMFS的源码与Linux-3.11内核源码是一起发布的，作为内核文件系统的一部分，其主要的实现代码位于/pmfs-master/fs/pmfs目录下。

2.2 PMFS在内核中的架构及空间布局

PMFS及其所管理的非易失内存在系统体系结构中的架构如图2-1所示。在该架构中，PMFS假设处理器能够同时支持易失性的DRAM和非易失内存PM，并且非易失内存作为主存扩展的一部分。操作系统的VMM继续管理DRAM，而PMFS负责管理PM[24]。PMFS通过mmap实现应用程序对PM的直接访问，并且将为简化使用PM而设计的编程模型和库集成为PMLib。

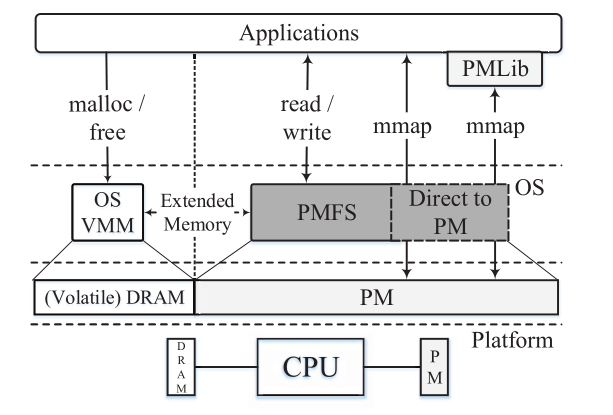


图2-1 非易失内存系统体系架构

PMFS设计之初有三个目标：1）优化字节寻址的存储。PMFS利用PM的字节寻址能力来避免传统文件系统基于块设计的开销；2）实现应用程序对PM的高效访问。PMFS通过避免不必要的复制和软件开销优化了文件读、写以及mmap；3）保护PM免于杂散写入。PMFS为了性能将整个PM映射到内核虚拟地址空间中，但这样可能导致PM遭受杂散写入而永久性损坏。为避免如此，PMFS使用处理器中的写保护控制功能实现了一个低开销写保护方案原型。与传统的基于块的文件系统对比，PMFS的设计避免了块设备层，并因此消除了页面复制的开销，最关键的是，PMFS提供给应用程序对PM的直接访问，图2-2展示了一个使用PMFS的系统的高级软件架构，包括与传统文件系统的对比。

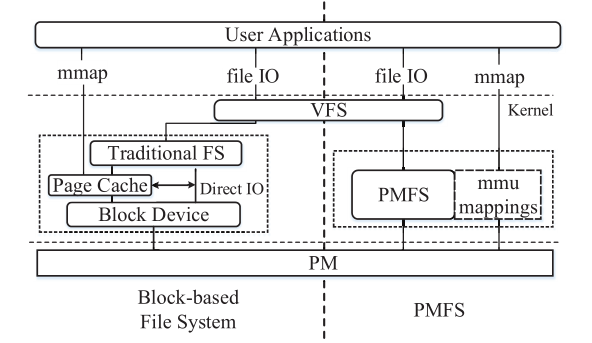


图2-2 PMFS与传统文件系统的对比

PMFS的空间布局如图2-3所示。超级块及其冗余备份之后是日志（PMFS-Log）和动态分配的页面。和许多其他文件系统一样，PMFS中的元数据使用B树进行组织，B树是索引大量稀疏数据的最佳选择之一。B树用于表示inode表和inode中的数据。

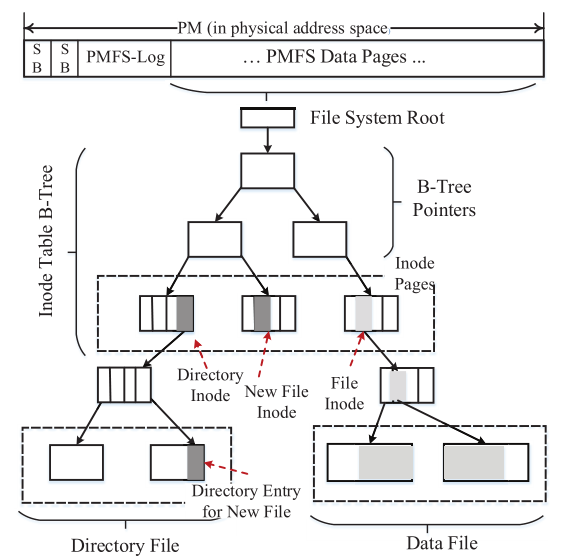


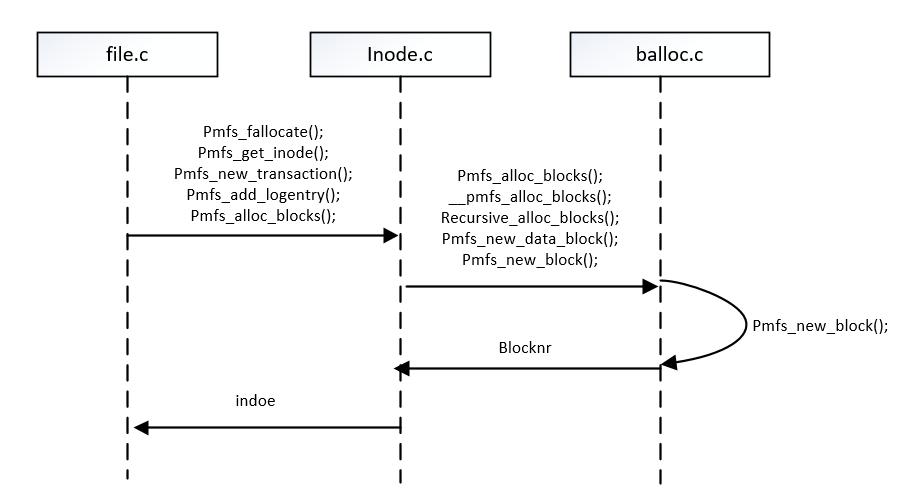
图2-3 PMFS的数据布局

2.3 PMFS分配器的设计与不足

**2.3.1 分配器的调用过程**

文件系统有四大对象：超级块、目录项、文件、索引节点，基本囊括了一个文件系统中所需要的最为关键的信息和相关的操作，而这四个对象的创建和回收都离不开分配器的参与，分配器负责最基础的数据单元的分配和回收。下面将结合PMFS的源码，以文件的创建流程为例来分析分配器的调用过程。

在PMFS源码目录下，与文件的相关实现在file.c文件中，通过函数名称猜想和分析，创建文件的流程，其入口应当在pmfs\_fallocate()这一函数中。该函数有四个参数：1）文件指针file；2）创建文件的模式mode；3）偏移地址offset；4）文件长度len。首先通过pmfs\_get\_inode()获取索引节点，随后通过pmfs\_new\_transaction()启动一个新的事务，并添加一条新的日志记录（pmfs\_add\_logentry()），在根据偏移地址和长度等信息计算出所需的块数量后，调用pmfs\_alloc\_blocks()函数分配块，后者的实现在inode.c中，其实现函数为\_\_pmfs\_alloc\_blocks()，而该函数主要是用于维护索引节点的B树结构，最后通过pmfs\_new\_data\_block()或recursive\_alloc\_blocks()函数，调用分配器的pmfs\_new\_block()函数，完成块的分配。整个流程的示意图如图2-4所示。

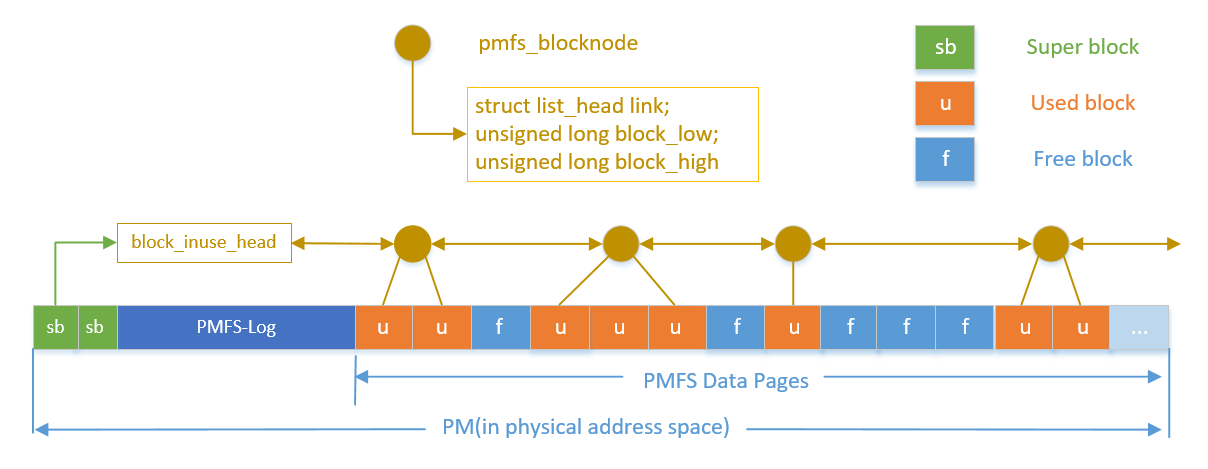


**图2-4 分配器的调用过程示意图**

根据上述过程分析可知，当文件系统需要给文件分配多个块时，为了维护索引节点的B树结构，分配器的块分配采用递归调用方式，这意味着每分配一个块就需要调用一次分配器，在进行文件创建和删除时，分配器的调用频率非常高，因此分配器的设计和效率显得尤为关键，否则将导致性能低下，下面将结合源码仔细分析PMFS分配的组织方式和分配策略的实现。

**2.3.2 分配器的组织方式**

大多数现代的文件系统使用基于范围的分配（如ext4，btrfs），而一些较旧的分配器是间接的基于块的（如ext2）[21]。PMSF的分配是基于页面的，支持所有处理器页面大小（4K,2M,1G），以实现透明的大页面支持。默认的情况下，PMFS分配器使用4KB页面作为数据文件B树的元数据（内部）节点，但数据（叶）节点可以是4KB,2MB或1GB。

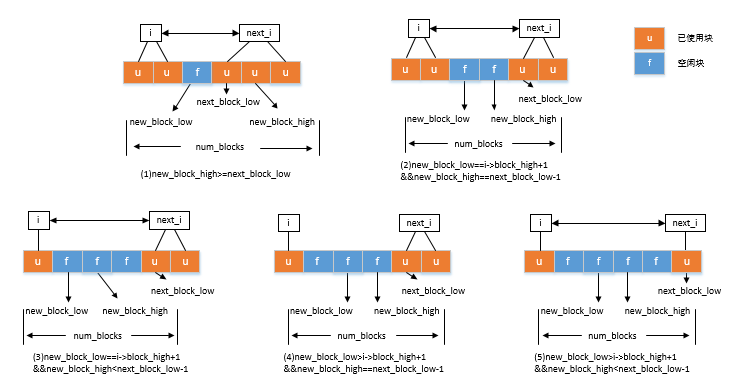


**图2-5 分配器的块组织方式**

在前面已经知道PMFS的数据布局，在超级块及其冗余备份和PMFS日志结构之后便是PMFS全部数据页面，在物理地址上以4KB大小的数据块为最小单元，并由低地址向高地址从0开始为数据块编号，由分配器进行管理和分配。由于文件系统中分配/回收操作的频率，使用日志记录来维护PMFS分配器数据结构的一致性会导致较高的开销，因此PMFS分配器采用空闲链表来维护分配器的结构。结合分配器的源码balloc.c文件分析可知，分配器在编号索引的基础上使用了成组链接法，用pmfs\_blocknode这一数据结构将所有已分配块组成双向循环链表，并将表头存放在超级块信息结构体pmfs\_sb\_info的block\_inuse\_head中，而双向循环链表的实现则是使用了内核的list\_head这一结构体实现的[22]，其好处是该结构体已实现了大量的宏定义调用，方便使用。在pmfs\_blocknode结构中包含了连续已分配块的低地址索引block\_low和高地址索引block\_high,以及一个list\_head结构的link，每一个pmfs\_blocknode结点代表一个或连续若干个已分配的数据块，其示意图如图2-5所示。

**2.3.3 分配/回收策略**

在已经了解了PMFS分配器块组织方式以后，下一步就是考察分配器的分配/回收策略。在balloc.c文件中找到了相应的实现函数pmfs\_new\_block()和pmfs\_free\_block()，通过参数可以知道，分配和回收块都是通过块的编号blocknr来进行索引的，在整个非易失内存的物理地址上，以4KB为单元块大小，给所有的数据块从0开始编号，因此最简单、最直观的方法便是每次从PMFS数据页面的起始地址开始遍历整个地址空间，直到找到合适的块，显然这也是效率最为低下的方法，其时间复杂度是O(N)。在上一节的分析中，已经知道分配器利用pmfs\_blocknode数据结构在已分配的数据块上使用成组链接法，其目的就是为了加速块的查找过程，避免直接遍历整个地址空间。下面结合源码，以分配过程为例，具体分析分配器的策略。



**图2-5 分配策略**

分配块的具体实现在pmfs\_new\_block()中，该函数有一个参数btype用于指示所申请的块的类别，前面已经提到分配器的分配的数据节点可能有三种类别：4K，2M，1G，通过pmfs\_get\_numblocks()函数可以解析块的类别并返回所需要块的数量num\_blocks，分配器用两个pmfs\_blocknode类型的指针（i,next\_i）遍历已分配块上的双向循环链表,其表头存储在pmfs\_sb\_info结构的block\_inuse\_head字段中。i和next\_i分别指向两个相邻的已分配的连续块地址，而两者之间存在着数量未知的空闲块，通过这种遍历方式避免了地址空间的全遍历，而是以一种跳跃式的方式进行遍历，从而加速了查找过程。

在进行分配时，用new\_block\_low和new\_block\_high这两个索引值指示将用于分配的新的连续空闲块的低地址索引和高地址索引，二者的差值正好是所需要的块的数量num\_blocks，还有一个next\_block\_low用于指示紧邻的下一个pmfs\_blocknode的低地址，由于不知道当前找到的连续空闲块的长度能否满足需求，分配器可能面临以下多种不同的情况：1）new\_block\_high>=next\_block\_low，说明在i和next\_i之间的连续空闲块数量小于所需要的块数量num\_blocks，不足以分配，因而需要继续遍历；2）new\_block\_low=i->block\_high+1 && new\_block\_high=next\_block\_low-1，即i和next\_i之间的空闲数量块正好满足需要，这种情况是最理想的，因为不会产生碎片；3）new\_block\_low=i->block\_high+1 && new\_block\_high<next\_block\_low-1，即所需要的块不足以填满i和next\_i之间的空隙，此时分配器将采取左对齐策略，即将所计算的new\_block\_low到new\_block\_high范围内的空闲块分配出去，然后令i移动到new\_block\_high位置，实现左对齐操作；4）new\_block\_low>i->block\_high+1&&new\_block\_high=next\_block\_low-1，与3类似，此时分配器将采取右对齐策略，令next\_i移动到new\_block\_low位置，保留左侧空闲块；5）new\_block\_low>i->block\_high+1&&new\_block\_high<next\_block

\_low-1，这是最不理想的情况，因为分配时在左右两侧都会产生碎片，分配器此时采取的策略直接在空闲范围内的一段区域分配所需要的块，并产生一个新的pmfs\_blocknode结点。分配器策略示意图如图2-5所示。

在上述策略中，除了考虑各种对齐策略外，还有就是维护成组链接的双向链表结构，在必要的时候增减pmfs\_blocknode的结点，以及移动结点，这种维护操作的时间开销相对于遍历来说可以忽略不计。回收块函数pmfs\_free\_block()的实现中，采用了与分配块同样的策略，利用成组链接法加速查找过程，并且根据找到的需要回收的块的位置采用相对应的策略，以及维护成组链接的链表结构。

**2.3.4 分配器效率分析**

根据上述分析可以知道，PMFS分配器对于块的组织方式本质上是一种一维的线性链表，在不考虑成组链接法带来的加速效益时，其时间复杂度也是线性的，即O(N)。但由于成组链接法，时间复杂度的计算存在不确定性。最好的情况是已分配的块和空闲块各自占据连续的地址空间，这样遍历时间将大大减少，假若成组链接链表上的结点数为K，则时间复杂度为O(K)。而最坏的情况下，地址空间中存在大量的碎片，这时成组链接的索引能提高的效率十分有限，查找块的时间复杂度又会降为O(N)。因此分配器的效率关键依赖于地址空间中碎片化的程度。而已经知道PMFS中针对碎片化问题的处理办法，只是在分配和回收块时采用简单的相邻合并策略，在长期使用的文件系统中，必然导致较为严重的碎片化问题，因此总体来说PMFS分配器的效率是趋近于O(N)的。显然这种时间效率，对于在文件系统中需要被频繁调用的分配器来说，是远远不够的，会严重影响系统的性能，需要进行改进和优化。

2.4 PMFS分配器的优化思路

上文介绍了PMFS文件系统分配器的调用过程、分配器对于数据页面的组织方式、分配/回收策略以及分配器的效率分析等。从中可以看出，对于在文件系统中需要被频繁调用的分配器而言，其时间复杂度趋近于O(N)的查找效率存在严重的性能上的问题，需要进行优化。根据上述分析，优化的思路也大致可以从调用过程、组织方式和分配策略等三个方面开展。下面将给出几种未进行详细的可行性分析的优化方案，以拓宽优化的思路，取得更好的优化效果，具体的分析过程将在下一章进行。

首先，从调用过程来看，PMFS分配器的调用是一次一调用，频繁的调用过程势必导致额外的调度开销，因此可以考虑将单元分配（每次分配一个单元块）改为一次性分配，即每次调用分配器时就将该文件（或其他申请块的对象）所需要的所有块一次性分配完，大幅度减少了调用的次数和相应的开销。其次，从分配器对于数据页面上块的组织方式来看，其本质上仍然是一维的线性链表，尽管这种方式简单直观，但效率低下，并且已分配的块和空闲块都在该链表上进行操作，遍历查找无可避免。因此在组织方式上可以考虑将空闲块和已分配块的管理分开，比如分成两根链表，或者是将整个地址空间的块分成多组链表，增加维度，提高并行性等。最后，在分配器的分配/回收策略上，尽管采取了几种对齐和相邻合并的策略，但是碎片化的问题仍然严重，优化的思路应该是尽可能的减少碎片化出现的情况，提高地址空间的利用率。

2.5 本章小结

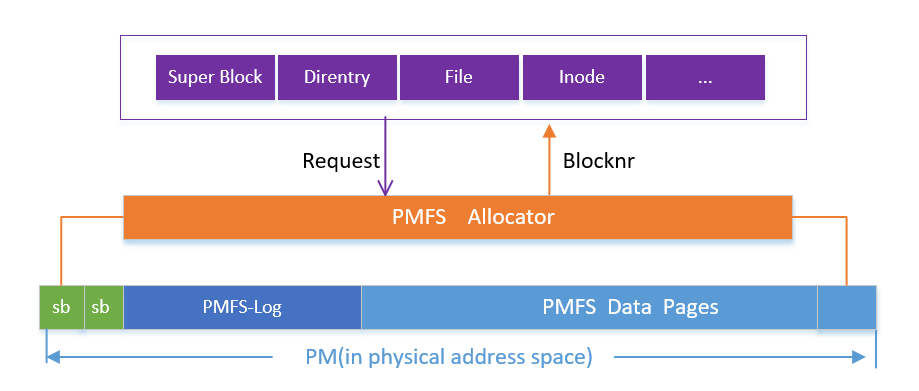
本章介绍了PMFS文件系统，一种针对非易失内存进行优化的、符合POSIX文件系统标准的轻量级文件系统。该文件系统也是本课题研究和优化的对象。本章详细讲述了PMFS的应用背景，系统架构与空间布局，以及PMFS分配器的设计并分析其不足之处，重点研究了分配器的调用过程、组织方式和分配/回收策略，并分析了分配器的效率。最后，针对分配器设计上的不足，提出了几种优化的思路，以期设计出具有良好优化效果新型分配器，提高分配和回收效率，进而从整体上提高PMFS的性能。

# 3 PMFS分配器的优化设计

在上一章中已经详细介绍了PMFS分配器的调用过程、组织方式、分配策略及其分配/回收效率等，并指出分配器的分配/回收效率趋近于线性时间复杂度O（N），存在较严重的性能问题，需要进行优化。在2.4节中已经提到了几种可能的优化思路和方案，但没有进行深入的分析和思考。本章将在前文的基础上，对PMFS分配器的各种可能的优化方案进行深入的分析，提出具有可行性的优化方案，完成优化设计，并分析优化后的分配器的效率。此外，优化的方案也许很多，但要尽量使优化方案针对分配器来进行，不改变原来的调用接口，一旦扩大了优化的对象和范围，将导致更多的不确定性，实现难度也随之增大。

3.1 分配器调用过程优化分析

在2.3.1节已经详细分析了创建文件时的分配器调用流程，并且已经知道对分配器的调用采用的是单元调度策略，即每分配一次块就调用一次分配器，这导致了PMFS分配器很高的调用频率，并产生了相应的调度开销。在文件系统中，超级块、目录项、文件和索引节点等等这些对象需要数据块时都离不开分配器，结合源码，PMFS中分配器一般的调用过程可归纳如下：对象计算出自己所需要的块数量及单元块大小btype，随后递归的向分配器发出分配块的请求，分配器根据所需单元块类型btype，在数据页面中寻找合适的块，找到后返回块的编号blocknr，其示意图如图3-1所示。



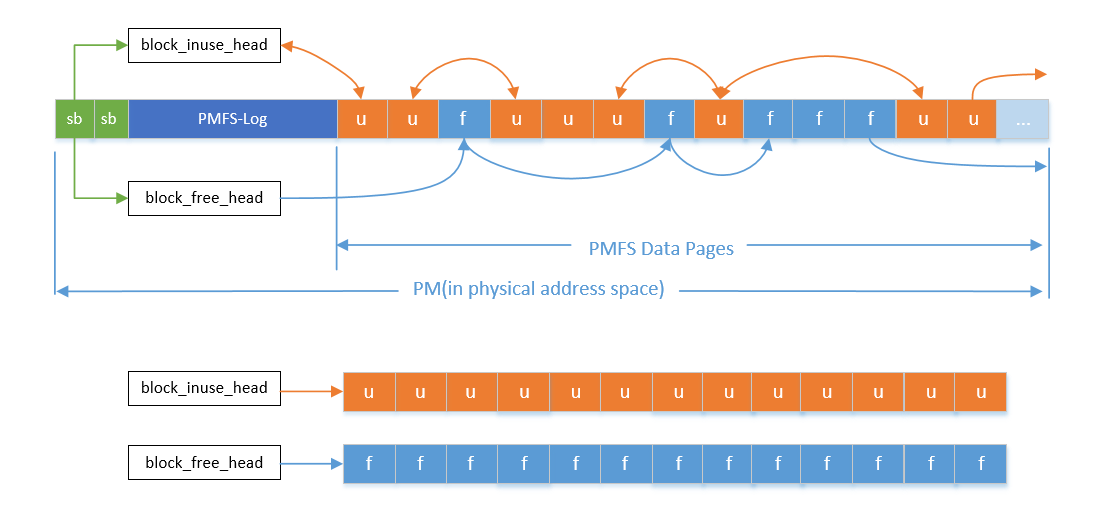
**图3-1 分配器调用示意图**

在2.4节已经提出了一种针对调用过程的优化思路，即将单元分配的方式改为一次性分配。一次性分配方式指的是分配器一次性分配对象所需要的所有块，下面将具体分析这种思路的利弊，再考虑是否采用这种方式。

一次性分配的好处是可以减少分配器的调用次数，同时可以在一定程度上降低频繁调用带来的时间开销。但是一次性分配是要以增加分配器复杂度以及改变对象调用接口为代价的。在当前的实现中，单元分配使得分配器的调用十分简洁，对象只需要传入一个所需块类型参数，而分配器只需返回找到的块起始地址编号blocknr。但在一次性分配方案中，这种简洁性将不复存在。在一次性分配中，出于碎片化问题和利用率的考虑，不可能每次都能进行连续分配，因此在返回找到的若干块时，不能只简单的返回一个起始地址编号，而需要设计复杂的数据结构来保存分配的块的所有信息，并且即使分配器一次性完成了块的分配工作，返回给上层调用者，调用者还需要解析返回的数据结构，获取其中的块信息，最终使用时仍旧是单元方式的使用。综合以上考量，PMFS分配器的调用方式采用单元调度有其合理性，简化了分配器的设计和实现，采用一次性分配方式可能得不偿失。

3.2 分配器组织方式优化分析及设计

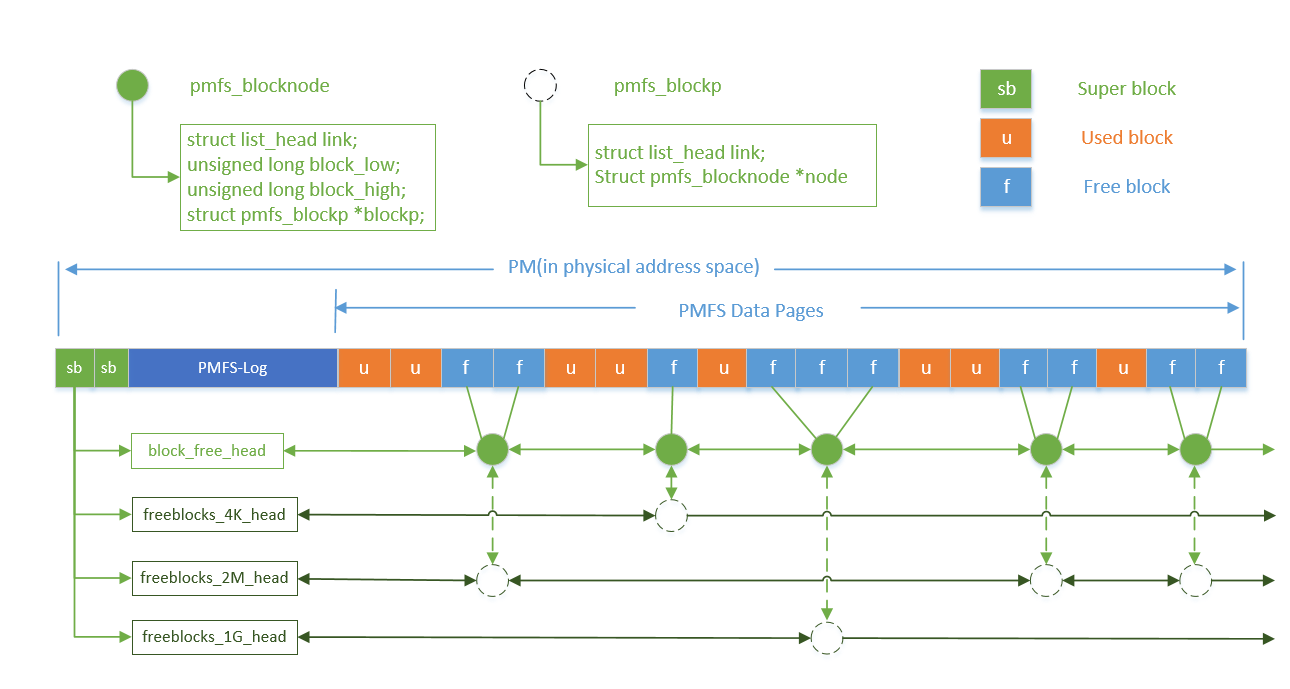
PMFS分配器在数据页面上的块组织方式采用的是线性链表结构，并在已分配块上使用了成组链接索引法，在上一章中已经进行了详细的分析，认为这种组织方式对于分配器的频繁调用来说效率不够高，因此必须要加以优化。在2.4节已经提出了两种针对组织方式的优化方案，一种是结合操作系统虚拟内存管理器的内存块管理策略而提出的[20]，将空闲块与已分配块分开置于不同的链表中（方案一）；另一种是从并行性的角度考虑，将单链表改成多链表结构，提高并行度以取得更高的效率（方案二），此外还有压缩的位示图索引[25]，双成组链接索引等可能的思路。下面将针对这几种优化思路作出具体分析。



**图3-2 空闲-分配双链表法**

方案一的示意图如图3-2所示。结合示意图可以看出，方案一的好处是，在进行单个块的分配时，无需进行遍历查找，只需在空闲链表上取下表头即可，是常数时间复杂度的分配方案，但具体分析会发现这个方案的缺陷也很明显。第一点，这种方案只能适用于单个块的分配，当要进行多块分配（PMFS支持多种数据页面大小，如2M，1G等），则在分配时仍然需要进行遍历；第二点，由于分配器的频繁调用，维护双链表的开销也不可忽视，相较于原始方案，无论是分配还是回收都会增加若干链表操作；第三点，双链表结构如果不根据块地址高低进行排序，则遍历查找的复杂度降为O(N)，如果对双链表进行排序，则会带来排序开销。综合上述几点考量，方案一的虽然能带来一定的优化效果，但相应的开销也不可忽视，有很大的不确定性，从目前的分析来看，很有可能开销大于优化效果。方案二的主要思想是将单链表拆为多链表，实现并行查找已提高效率，但如果在分配器中引入多线程或多进程等并行策略，将极大的增加实现难度和不确定性；位示图法是很多文件系统分配器采用的组织方式（如Ext4），但若采用位示图法需要对分配器及上层调用对象做较大的改动；而双索引法，相较于原始方案虽然能在一定程度上提高分配的效率，但随之也带来了维护双索引的开销。

上述提到的各种优化思路，通过详细的分析考虑，发现都是各有利弊，作为一般化的文件系统分配器组织方式，应用在PMFS上都难以取得令人满意的优化效果。但是综合上述几种思路的优点，再结合PMFS本身的一些特性，下面将提出一种以成组链接法为基础的综合性组织方式优化方案——分类多级成组链接索引法，优化方案的示意图如图3-3所示。



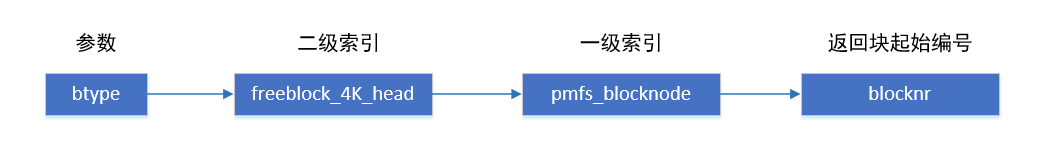
**图3-3 组织方式综合优化方案示意图**

结合内存管理的空闲链表法，将在已分配块上建立的成组链接索引改为在空闲块上建立相应的成组链接索引，即一级索引，实现空闲块与已分配块的分开管理。其次，考虑到PMFS分配器只分配3种固定类型的单元块（4K，2M，1G），因此可以据此对所有的连续空闲块进行分类：1）4K<=size<2M；2）2M<=size<1G；3）size>=1G。按照这个分类，将空闲链拆分为3条链表，即二级索引，当分配不同类型的块时，可以直接在对应的链表表头取出块即可，而无需遍历，并且可以实现一定程度上的并行查找。这一方案综合了空闲链表法、成组链接法、多链并行以及PMFS本身的特性（大页面支持），虽然在增加了数据结构以及空间复杂度，但是取得的优化效果也是明显的，维护的开销与原始方案相比也没有增加太多，可以通过测试进行检验。

3.3 分配/回收策略优化分析及设计

分配/回收策略的是基于分配器的组织方式的，在原始方案中分配和回收都是基于已分配块上的成组链接索引来进行，通过成组链接索引加速块的查找过程，并且还要使用对应的对齐策略和合并策略，以减少碎片化。对组织方式进行优化后，分配和回收策略必将随之变化，下面分析分配和回收策略的优化方案。

分配策略的优化如下：首先根据所需的块的类别，找到对应块类型的链表表头，此时找到的是二级索引，但实际块的取用都要在一级索引上进行，通过二级索引结点与一级索引结点之间的双向指针，可以定位出一级索引上的块，从而完成分配。实际分配时，优化的分配策略无需考虑对齐问题，可全部采用左对齐策略，因为通过二级索引定位的连续块必定能满足需求；但需要维护二级分类索引，即如果连续块经过分配后，其长度降低到不满足当前分类索引的要求，则需要将该结点移动到下一级索引链中。优化后分配策略的示意图如图3-4所示。



**图3-4 分配策略示意图**

回收策略的调整略微复杂一些，这是由于回收块时传入的参数是已分配的块编号和类别，无法根据参数信息直接定位到相邻的空闲块结点索引，因此回收策略与原始方案类似，通过空闲块上的成组链接索引来加速查找过程，找到所要释放的块后，根据其所在位置采取相应的合并策略，或者生成新的结点。由于合并之后连续块长度增加，因此还要注意维护二级索引。

3.4 优化方案整合及效率分析

上文已经对PMFS分配器的调用过程、组织方式及分配/回收策略的各种优化思路作出了详细的分析，并最终确定了合理的优化方案。优化后分配器的调用过程仍然采用单元分配的方式，保持分配器的简洁性并且不用改动上层的调用接口。组织方式的优化是整个优化方案的核心所在，结合了管理文件系统空闲块常用的几种方法（如空闲链表法、成组链接法、拆分多链）以及PMFS本身的特性（支持多种页面），将原始方案中在已分配块上建立的索引改为在空闲块上建立索引，同时增加数据结构，根据PMFS分配器支持的三种数据页面大小，在一级索引之上建立分类别的二级索引。优化组织方式后，分配和回收策略也随之调整，分配策略无需遍历查找，而回收策略与原始方案相似。下面分析整体的优化效率。

首先，由于建立二级索引，优化方案增加了分配器组织方式的空间占用，增加的结点数目与一级索引上结点数目相同（N个），但是二级索引结点中只存储了一个指向一级索引的指针，结点体积较小，整体上空间复杂度并未增加，仍然是O(N)。优化方案的优势主要体现在时间复杂度上，原始方案中分配块的过程时间复杂度趋近于O(N)，优化后通过两级索引分配块的时间复杂度降为常数时间O(1)，但同时二级索引带来了额外的维护开销，增加了一些指针操作，总体上来说，优化后的分配器效率应当会有所提升。

3.5 本章小结

本章主要探讨了PMFS分配器的优化设计方案，从分配器的调用过程、组织方式及分配/回收策略等方面进行了详细的优化分析，阐述了各种优化思路的优劣对比，并最终确定了一套具有综合性、可行性的优化方案。在本章最后，对分配器的各种优化进行了整合，并进行了效率分析，从理论上认定该方案能取得一定的优化效果。

# 4 PMFS分配器的优化实现

基于第三章提出的设计方案，本章将对PMFS分配器的组织方式和分配/回收处策略实现优化。在组织方式中将先实现空闲块上的一级成组链接索引，然后在此基础上根据PMFS支持的三种处理器数据页面大小实现二级分类索引，完成对组织方式的优化。在分配/回收策略上，实现常数时间复杂度的分配方案，优化合并及维护索引的机制，提高分配器整体效率。

4.1 组织方式的优化实现

**4.1.1 一级成组链接索引实现**

在第二章已经知道PMFS文件系统的空间布局，在超级块及其冗余备份、PMFS日志项之后便是文件系统的数据页面（图2-3），虽然PMFS支持三种处理器页面大小，但在物理地址空间的划分上，采用的是4K大小的数据块，并且从起始地址开始给所有数据块编号，编号从0开始。前面的优化方案中已经提出，要在空闲块上建立成组链接索引，即将连续的空闲块合为一“组”，并将这些空闲块“组”链接在一起。为便于管理这些空闲块组，沿用原始方案中的数据结构pmfs\_blocknode，将每个空闲块组视为一个结点，每个结点需要记录该组空闲块的低地址编号及高地址编号，同时为便于查找和维护，管理结点的链表采用linux内核实现的双向循环链表结构struct list\_head（位于头文件list.h中），此外，每个结点还有一个指向二级索引结点的指针，结点数据结构如表4-1所示。

表4-1 pmfs\_blocknode数据结构

|  |  |  |
| --- | --- | --- |
| 成员名称 | 成员类型 | 成员含义 |
| link | list\_head | 双向循环链表结点 |
| block\_low | unsigned long | 低（起始）地址块编号 |
| block\_high | unsigned long | 高（结束）地址块编号 |
| blockp | pmfs\_blockp | 二级索引结点指针 |

一级成组链接索引表的表头存放于pmfs\_sbi\_info结构中的block\_free\_head字段中，同时该结构中还存放了指向超级块的指针等许多重要信息，后面将实现的二级分类索引的表头也同样存放于该结构体中，便于查找和取用。在pmfs\_blocknode这一数据结构中，需要说明的是成员link，这是一个list\_head类型的结构体，是linux内核实现的双向循环链表结构，包含next和prev两个指针，通过这两个指针可以很方便的进行链表操作，此外在linux/list.h中还通过宏的方式实现了许多常用操作，如list\_for\_each\_from（遍历），list\_empty（判空），list\_entry（取结点）等，大大减少了实现双向循环链表的复杂度，方便操作。

**4.1.2 二级分类索引实现**

根据第三章可以知道，分配器优化方案的核心便是基于一级成组链接索引而实现的二级分类索引，每一个一级索引结点都对应一个二级分类索引结点。其中的“分类”源自于PMFS对三种处理器页面（4K，2M，1G）的支持。在一级成组索引链表中包含所有的空闲块组，而二级索引链表根据空闲块组的大小进行分类：1）当空闲块组的大小大于等于4K而小于2M时，将该结点置于标记为4K的二级索引链上；2）当空闲块组的大小大于等于2M而小于1G时，将该结点置于标记为2M的二级索引链上；3）当空闲块组大小大于1G时，置于标记为1G的二级索引链上，二级索引链表的分类如表4-2所示。

表4-2 二级分类索引表

|  |  |  |
| --- | --- | --- |
| 类型 | 表头 | 对应一级结点大小范围 |
| 4K | freeblocks\_4K\_head | 4K<=size<2M |
| 2M | freeblocks\_2M\_head | 2M<=size<1G |
| 1G | freeblocks\_1G\_head | size>=1G |

由于二级索引表中每条链表都有其大小范围的限制，因此当知道了所需分配的块的类别时，对应的二级索引链表中每个结点都可以满足需求，因此只需取其表头便可，而无需遍历一级索引表。二级索引链表也采用双向循环链表结构，其结点pmfs\_blockp中只存放对应的一级索引结点指针，结点的数据结构如表4-3所示。

表4-3 pmfs\_blockp数据结构

|  |  |  |
| --- | --- | --- |
| 成员名称 | 成员类型 | 成员含义 |
| link | list\_head | 双向循环链表指针 |
| blocknode | pmfs\_blocknode \* | 对应一级索引结点指针 |

二级分类索引表的表头都存放在pmfs\_sb\_info这一结构体中，取用较方便，同时还可以注意到，在一级结点和二级结点之间存在双向指针，这样做的好处在于，在执行分配时通过二级索引结点找到一级索引点，而在执行回收时，可以通过一级索引结点找到对应的二级索引结点，方便维护索引结构。

4.2 分配/回收策略优化实现

**4.2.1 分配策略**

由于二级索引表是针对PMFS支持多种处理器页面进行的优化实现，其好处最主要就体现在分配块的过程上。在原始方案中，分配某种类型的块时需要遍历数据页面以找到合适的空闲块组，并且根据空闲块组的大小采取不同的对齐策略，然而在优化后的方案中，由于每条二级索引链表中的每个结点都能满足特定类型的块要求，分配过程完全避免了遍历开销。

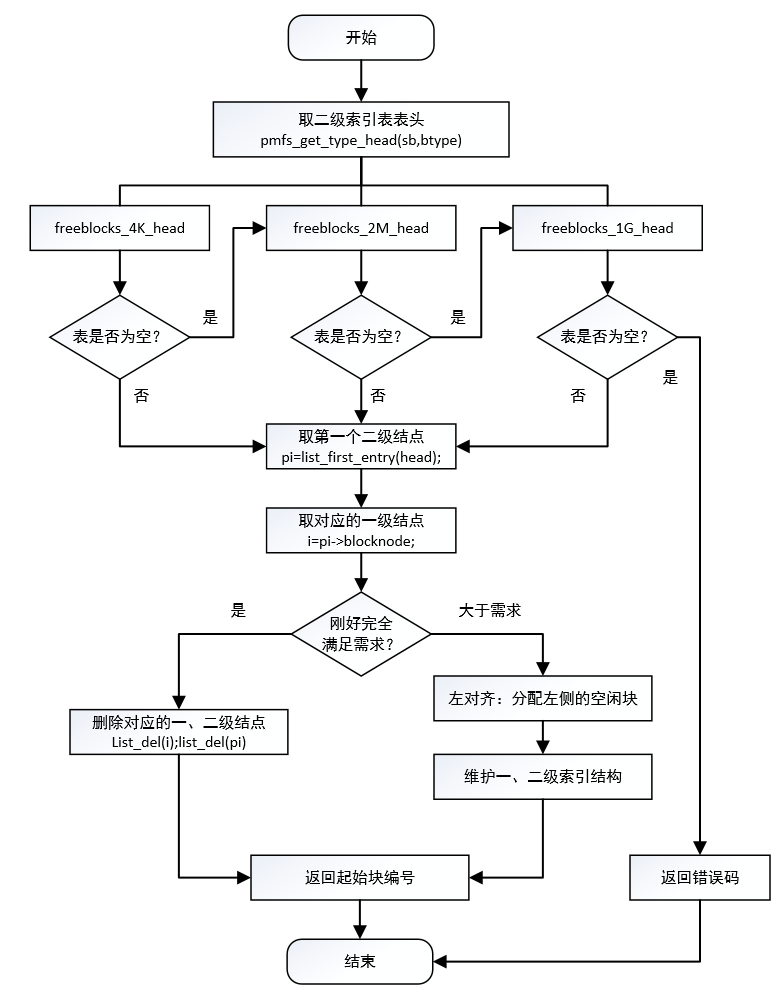


图4-1 优化后分配空闲块流程图

以完成一个2M类型块的分配过程为例，上层对象发起请求并传入所需块的类型（2M），据此取得2M类型的二级索引表表头freeblocks\_2M\_head，取出二级索引表的第一个结点，再通过指针取得一级索引结点，并从中获取起始块编号，返回给调用者，完成分配。分配策略的流程图如图4-1所示。

**4.2.2 回收及合并策略**

相比于分配策略，回收策略显得更为复杂，这是由于回收块的过程中会面临更多的情形，需要采取不同的合并策略来维护一级成组链接索引的结构，并减少碎片化。在进行块回收时，传入的参数只有块的类型和编号，不能直接定位出块的位置，因此需要借助一级索引来加速查找块的过程，查找的思路是：遍历一级索引链表，当遍历指针i的高地址块编号小于待回收块编号时，遍历下一个结点；当大于待回收块编号时，意味着已经找到了待回收块邻近的结点，后面需要做的就是将待回收块加入到一级索引之中，在特定情形下还需要建立对应的二级索引。

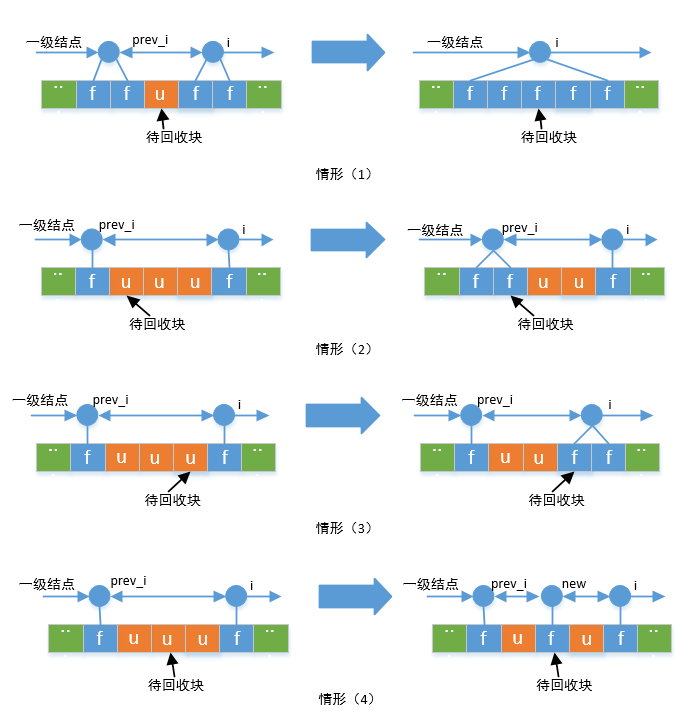


图4-2 回收过程的合并策略示意图

回收块时可能会面临以下四种情形：1）待回收块刚好填满前后两个结点之间的“缝隙”；2）待回收块刚好“贴近”前一个结点；3）待回收块刚好“贴近”后一个结点；4）待回收块位于两结点之间，并且不“贴近”任一个结点。回收过程中的合并策略示意图如图4-2所示。下面将分别阐述各种情形下的回收和合并策略。对于情形1，待回收块变为空闲块后，在一级索引链表中从前一个结点的起始块到当前结点的结束块的整个地址空间组成一个连续的空闲块组，采取的策略是删除前一个一级结点及对应的二级结点，保留当前结点，并扩展其块地址编号；对于情形2，直接将待回收块合并到左侧的结点中，并扩展结点的地址索引；对于情形3，则将待回收块合并到右侧的结点中，并扩展结点的地址索引；对于情形4，则需要建立新的一级结点和对应的二级结点。此外，还有一种可包含在上述几种情形之中的特殊情形，主要是由于没有前置结点prev\_i引起的，例如在情形1、2中，若没有前置结点，则无法进行向左合并，需要建立新的结点

**4.2.3 索引结构的维护**

由于二级分类索引表对所管理的一级结点有大小范围的要求，因此在回收和分配的过程中都需要对索引结构进行维护。对于分配过程，若没有完全分配，则可能由于分配后结点的长度（高地址与低地址之间的差值）变小了，导致一级结点的长度不再复合当前二级索引链的范围要求，则需要将其二级索引结点移动到更“低”的二级索引链当中，如从2M链移动到4K链上；对于回收过程，经过回收合并后一级结点的长度必然增大，此时同样要判断一级结点长度是否满足原先的二级索引链的范围要求，如果不满足，则需要将二级索引结点移动到更“高”的链中。

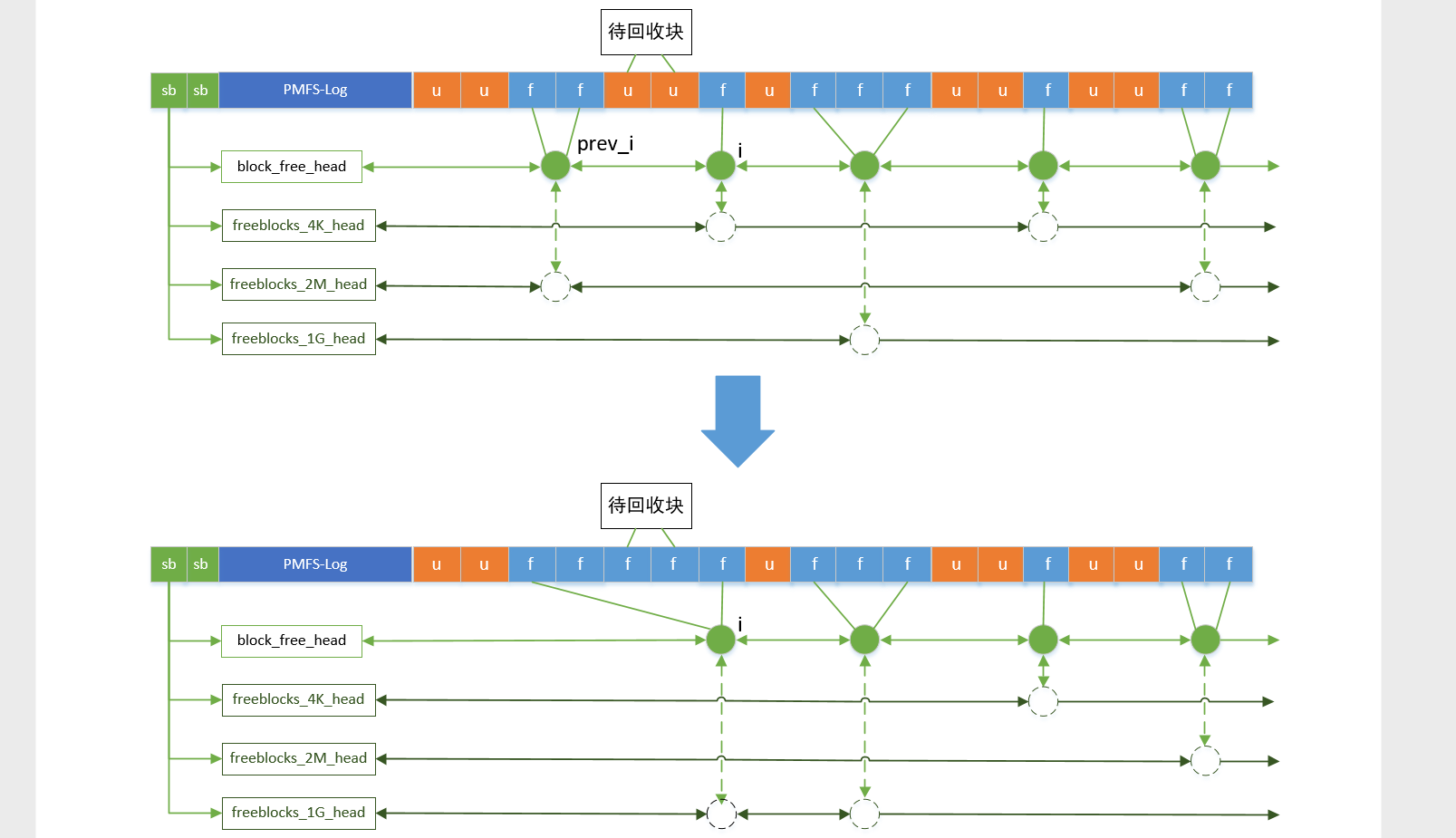


图4-3 索引结构维护示意图

以4.2.2小节中的情形1为例，在完成一级索引结构上的合并操作之后，删除了前置结点，以及对应的二级结点。假设原先结点i的长度size<2M，即为4K链上的结点，但合并后结点i的长度变大了许多，是的size>=1G，这时就需要将结点i对应的二级索引结点移动到1G类型的二级索引链上，如图4-3所示。

4.3 本章小结

本章描述了对PMFS分配器的组织方式和分配/回收策略优化方案的具体实现过程，详细阐述了一级成组链接索引和二级分类索引的实现，其中二级分类索引是基于PMFS支持多种处理器页面大小来建立的，是组织方式优化的核心所在。此外还详细描述了优化的分配策略、回收和合并策略以及对多级索引结构的维护方法。

# 5 性能测试与分析

5.1 测试环境

通过详细的分析和论证，本文提出了一种针对PMFS分配器的组织方式和分配/回收策略的综合优化方案，并且完成了优化实现。为了验证本文提出的优化方法的有效性，对优化前后的PMFS文件系统在相同的系统平台上进行了测试与对比分析。

PMFS文件系统本身虽然是一种针对非易失内存进行优化的文件系统，但是目前非易失内存技术并不十分成熟，实际产品也不普遍，在现有条件下，一般都选择用DRAM内存来模拟非易失内存[26,27]。在Linux操作系统中，通过memmap选项从常规的内存区域中划分出一个新的内存区域来，和已有的内存划分区域共存，而划分出来的内存区域则作为仿真非易失内存的设备，由PMFS进行管理。本文中实验测试机器是一台服务器，服务器的硬件与软件测试环境如表5-1所示。

表5-1 测试环境

|  |  |
| --- | --- |
| CPU | Intel Xeon CPU E5-2643 v3 @ 3.40GHz，6核 |
| 内存 | 8GB |
| 机械硬盘 | WDC WD10EZEX-08WN4A0，1T |
| 主板 | Supermicro X10DRi |
| 操作系统 | CentOS 6.5(x86\_64) |
| 内核版本 | Linux 3.11.0 |
| PMFS源码 | https://github.com/linux-pmfs/pmfs |
| 测试软件 | Filebench 1.4.9.1 |

5.2 测试方法

**5.2.1 环境搭建**

本文的研究对象是针对非易失内存进行优化的、符合POSIX文件系统标准的PMFS文件系统，其源码由Intel开源发布在github网站上，与Linux 3.11内核绑定。为了保证PMFS的可用性，测试时也使用Linux 3.11.0版本内核。对于操作系统的选择是比较自由的，可以选用ubuntu或者CentOS，但是根据网上查到的一些些资料显示，使用CentOS具有更好的可用性，并且操作系统版本不宜太高，于是选用了CentOS 6.5。在安装CentOS 6.5操作系统时，推荐使用光盘安装方式，如果采用U盘镜像安装，可能会由于U盘文件系统格式不支持大于4G的单个镜像文件导致安装失败。编译Linux 3.11.0内核时，要注意用低版本内核编译高版本内核，同时为了后续PMFS安装调试的方便，在配置内核时（make menuconfig）最好将PMFS配置为内核模块方式，这样可以避免每次改动PMFS源码后需要编译整个内核，提高工作效率，以上这些都是在环境搭建过程中的一些经验之谈，仅供参考之用。

**5.2.2 PMFS的编译安装**

PMFS的源码可从github网站上下载（见表5-1），配置好内核后，PMFS以内核模块的方式进行编译安装。具体步骤是：先通过make命令将PMFS编译为内核模块pmfs.ko文件，然后通过insmod命令安装内核模块pmfs，最后通过mount命令安装PMFS文件系统。需要注意的是，在安装PMFS之前需要先设置内核启动参数memmap，在操作系统内存区域中划分一段区域来模拟非易失内存，其具体操作是修改/boot/grub/grub.conf文件中Linux 3.11.0 内核对应的启动项参数。

**5.2.3 Filebench的使用**

本文采用一款文件系统性能的自动化测试工具——Filebench[31]，来测试优化前后的PMFS性能。该工具通过快速模拟真实应用服务器的负载来测试文件系统的性能。它不仅可以方针文件系统的微操作（如copyfiles，createfiles，randomread，randomwrite），而且可以仿真复杂的应用程序（如varmail，fileserver，oltp，dss）。Filebench的源码可在<https://github.com/filebench/filebench>进行下载，解压源码包后执行1)./config; 2)make; 3)make install三个命令即可完成安装。安装Filebench后，在/usr/local/share/filebench/workloads目录下有很多已经定义好的标准工作负载，其实质是用与Filebench配套的WML（Workload Model Language）[5]语言写的测试配置文件，通过该配置文件可以设置文件或文件集的各种属性，便于模拟各种负载。在进行测试时，通过filebench命令即可进入交互模式，通过load命令加载各种配置文件，既可以是标准负载，也可以是自定义的负载，通过run命令启动测试。

5.3 测试结果及分析

根据前面的优化方案和实现部分可以知道，本文主要是对PMFS文件系统的分配器进行了重点优化，其优化效果将集中体现于元数据的创建等性能上，同时对于文件系统整体的读写性能也应当有一定的提升，下面将使用文件系统性能自动化测试工具Filebench集中测试优化前后文件系统的性能并进行分析。

在优化方案中，分配策略的优化是核心之一，其效果应当集中体现在元数据的创建上，因此利用Filebench内置的文件创建工作负载进行测试分析。在进行对比分析时，本项测试以文件集的平均文件大小为控制变量，分别测试平均文件大小为16k，32k，64k，128k，256k，512k的文件集创建文件的速度，创建文件的个数为10000个，其中文件大小的遵从伽马分布，优化前后测试结果对比如图5-1所示。

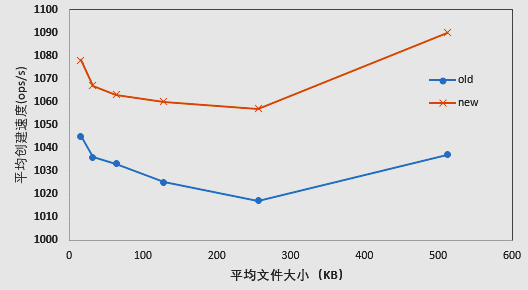


图5-1 文件创建速度对比

在文件元数据创建速度对比图中，可以明显看到优化后PMFS创建文件的速度得到了明显的提升，在平均文件大小从16k到256k的数据对比中，优化方案平均取得了3%左右的优化率。值得注意的是，在256k到512k之间，可以明显看到一个转折，创建速度突然骤升，这是因为当平均文件大小达到0.5M时，将有一部分文件大小将超过2M，从而导致PMFS分配器改用2M大小的结点来进行分配，从而加速了分配的过程，提高了创建速度，在512k这一点处，优化后的方案取得了约5%左右的优化。由于内存大小的限制，当继续增大平均文件大小时，会导致从内存中分配出来仿真非易失内存的区域空间不够用，仅从当前所测得的数据分析来看，优化方案的确使得文件元数据的创建得到了提升，取得了优化效果。

在验证完分配器的优化效果后，下一步需要测试文件系统整体性能是否得到了一定的优化。在本项测试中，采用标准工作负载中的文件服务器负载测试（fileserver.f），该标准工作负载测试了创建文件、读/写文件、打开/关闭文件、添加内容、删除及统计等多项操作，每项操作有单独的数据统计，并且有最终的总计数据，下面首先对比优化前后文件系统整体的I/O读写性能，如图5-2所示。据图可知，优化后整体I/O性能得到了一定的提升，根据数据统计计算，在读写性能方面，优化方案取得了平均3%~5%的优化效果。

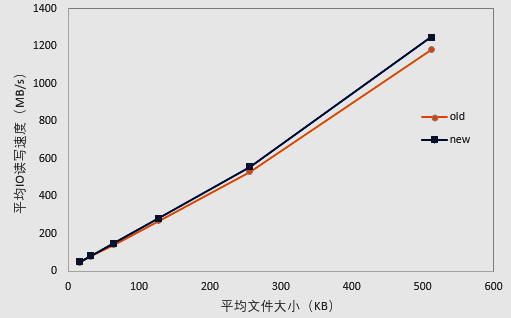


图5-2 I/O读写性能对比

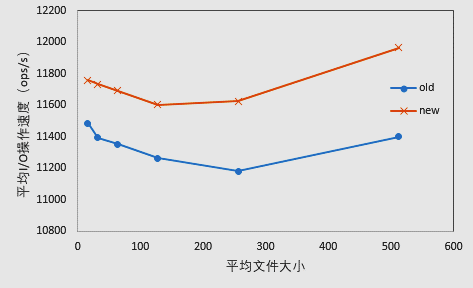


图5-3 整体操作速度对比

根据测试结果中总体统计数据，优化前后文件系统整体操作速度对比如图5-3所示。整体操作速度也取得了约3%左右的提升，并且随着平均文件大小的提高，这一比例有所上升。总体而言，本次测试达到了预期的目的，验证了优化方案的有效性，但是测试过程仍然有许多不足，例如由于测试过程中是用部分内存模拟仿真非易失内存，但这并不足以模拟真实的情况，其次由于内存大小的限制，本次并没有测试超大文件的性能（超过1G大小的单个文件），此外优化方案的优势在长期使用及碎片化较严重的文件系统中将更加明显，这是由于原始方案在碎片化严重时会导致更长的遍历的时间，而优化方案则完全避免了遍历操作，在这种情况下优化方案能取得更高的优化率。

5.4 本章小结

本章通过文件系统性能自动化测试工具Filebench测试了优化前后的PMFS文件系统的性能，通过两者测试数据的对比，表明优化后PMFS的文件操作等各项性能相较于原生系统取得了一定程度的提高，尤其是元数据的创建和删除等，文件系统整体的读写速度也有了一定的提高。本章主要验证了本文实现的PMFS分配器的组织方式和分配/回收策略的优化可以有效的改善分配器的效率，提高文件系统整体性能。

# 6 总结和展望

经过对非易失内存技术和相关文件系统的调研，本文研究和优化基于非易失内存的文件系统PMFS，针对PMFS分配器提出了一些优化方法来改进其在性能和效率方面存在的问题。本文以PMFS为基础，从分配器的组织方式和分配/回收策略等两个方面来进行研究，设计更为高效的二级分类成组链接索引结构来替代原系统中的单链成组链接结构，同时改进了分配和回收策略，最后通过文件系统性能自动化测试工具Filebench测试验证了本文提出的优化方案的有效性。本文的主要工作包括：

（1）对非易失内存相关的技术进行了调研，重点分析了非易失内存在现有存储体系结构中的地位及其在操作系统中的管理归属问题，同时研究了国内外关于非易失内存技术的发展趋势，为后续对PMFS文件系统开展优化研究提供了理论和实践基础。

（2）优化了PMFS分配器的组织方式。分配器管理文件系统在物理内存地址空间上的所有数据页面，同时PMFS支持多种处理器页面大小格式，根据PMFS的特性，设计了二级分类成组链接索引，将数据页面中的连续空闲块组根据大小分类，满足不同类型的块分配需求。二级索引可以直接定位所需的块，将分配块的时间复杂度降为常数时间，提高了元数据操作的性能。

（3）优化了分配/回收策略，提出索引结构的维护方案。根据优化后的组织方式，提出了新的回收和分配策略，在分配过程中无需进行遍历查找，并且充分简化了对齐和合并策略，同时减少了空闲块组的碎片化情况。同时，当空闲块组的长度发生变化时，分配器将根据其长度对其进行再归类，以维护多级索引结构，保证其有效性。

本文实现的针对PMFS分配器的优化方案只是PMFS文件系统优化的一个部分，在PMFS文件系统中还有很多待改进的地方，在后续的研究里还需要对PMFS进一步优化和完善。例如可以考虑将文件系统中对空闲块的管理方式改为位示图的方式，可以更为高效的查询和管理，此外还可以考虑对PMFS进行并行性研究和优化，从而进一步提高文件系统的性能。

非易失内存技术还不是非常成熟，相关应用以及设备也不是非常普遍，因此还没有可应用的非易失内存设备作为开发的工具供研究使用。模拟设备并不能真实反映出非易失内存设备的特性。希望在不久的将来能够使用真实的非易失内存设备进行研究。

# 致 谢

时光荏苒，岁月如梭。四年华中大本科生的学习生活即将结束，这是我人生中一段珍贵而难忘的时光。2014年9月，当我步入这所被绿荫覆盖的1037号森林时，就深深喜爱上了这优美的校园环境，随后的时间里我更是被这里浓郁的学术氛围、儒雅的人文气息、朴实的校园风气所吸引、折服。在这里的四年时光，我不仅增长了知识，开拓了视野，提高了自己的思考和学习能力，还认识了许多可敬、可亲又可爱的老师和同学，这些都将是我宝贵的人生财富。在论文即将付梓之际，借此文抒发心中的感恩之情，聊表心意。

首先要感谢我的导师冯丹老师。虽然冯老师公务繁忙，经常要离校出差，但是冯老师依旧通过邮件等方式关注着我的学习情况，并根据我自身的兴趣，给我指明了毕设课题的研究方向，鼓励我学习文件系统的知识。

其次我要感谢张峥博士，在毕设工作期间切实的为我解决了各种疑难问题，给我指出了明确的学习和研究方向，并且经常督促我注意毕设的工作进度，极大的支持和帮助了我的学习和毕设工作，在这里我向他表示真挚的感谢和敬意。同时也要感谢余亚博士，在实验室为我解决了许多问题，对我的学习进行了帮助和指导。

然后要感谢本科期间我所有课程的任课老师，感谢他们在课堂上的辛勤付出和谆谆教诲。也感谢本科期间14级计科10班的所有同学，因为有你们我的大学生活才更加精彩和丰富，我很高兴也很幸运能和你们在这里一起学习生活，特别要感谢宿舍的马忠臣，刘志文，盛涛涛等舍友，感谢他们平时对我的帮助和鼓励。

在这里还要感谢我的辅导员付洁老师，还有校团委的姚坦老师、王晓月老师，以及戴建政、杜江凯、刘静等朋友，谢谢他们在我的学习和学生工作方面对我的帮助和照顾。也要感谢蔡春芳、田冲等同学在思想上帮助我进步，使我成为光荣的中共预备党员。还要感谢汪子奇、韦珠林、张智均等同学经常与我一起去篮球场，使我有一个健康的好身体。

最后要感谢的是我的父母和家人，感谢他们在我求学路上无私的付出，始终默默站在背后给予我支持和鼓励，我衷心的祝愿他们永远平安健康。

# 参考文献

[1] Persistent Memory. <https://en.wikipedia.org/wiki/Persistent_memory>.

[2] PMFS source code. <https://github.com/linux-pmfs/pmfs>.

[3] GNU Core Utilities. http://www.gnu.org/ software/coreutils/.

[4] Filebench. <http://sourceforge.net/apps/mediawiki/filebench>.

[5] Workload Model Language. <https://github.com/filebench/filebench/wiki/Workload-model-language>

[6] Subramanya R Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, Jeff Jackson. System Software for Persistent Memory. In EuroSys’14 Proceedings of the Ninth European Conference on Computer Systems,Article No.15, Amsterdam, The Netherlands – April 14-16,2014.

[7] 孙广宇,舒继武,王鹏.面向非易失内存的结构和系统级设计与优化综述.《华东师范大学学报（自然科学版）》,2014,5:72-81.

[8] BAI Tongxin, DOND Zhenjiang, Cai Manyi, FAN Xiaopeng, XU Chengzhong, LIU Lixia.A Survey of System Software Techniques for Emerging NVMs.《ZTE Communications》,2017,1:35-42.

[9] Sorin, Daniel J. Persistent Memory. Computer(00189162), 2017,50(3):12.

[10] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K.Gupta, R. Jhala, and S. Swanson. NV-Heaps: Making Persistent Objects Fast and Safe with Next-generation, Non-volatile Memories. In Proceedings of the Sixteenth International Conferenceon ArchitecturalSupportfor Programming Languages and Operating Systems, ASPLOS XVI, pages 105–118, 2011.

[11] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better I/O Through Byte-addressable, Persistent Memory. In Proceedings of the ACM

SIGOPS 22nd Symposium on Operating Systems Principles,SOSP ’09, pages 133–146, 2009.

[12] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In Proceedings of the 36th

Annual International Symposium on Computer Architecture,ISCA ’09, pages 24–33, 2009.

[13] H. Volos, A. J. Tack, and M. M. Swift. Mnemosyne:Lightweight Persistent Memory. In Proceedings of the Six-teenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVI, pages 91–104, 2011.

[14] FREITAS R F, WILCKE W W. Storage-class memory: The next storage system technology[J]. IBM Journal of Research and Development, 2008, 52(4.5): 439-447.

[15] Wu X, Reddy A L. SCMFS: a file system for storage class memory. in: Proceedings of 2011 International Conference for High Performance Computing, Networking,Storage andAnalysis. Seatle,WA:IEEE, 2011.39~50

[16] Dalinger I. Formal Verification of a Processor with Memory Management Units.

Vdm Verlag Dr Müller, 2008,1(1):33~35

[17] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable,

persistent memory. Proceedings of Sosp, 2009,2(3):133~146

[18] Kang Y, Yang J, Miller E L. Object-based SCM: An efficient interface for Storage Class Memories. in:Proceedings of the 27th Symposium on Mass Storage Systems and Technologies. Denver, CO:IEEE, 2011.1~12

[19] D. Nagle M. E. Factor, S. Iren, et al. The ANSI T10 object-based storage standard

and current implementations. IBM Journal of Research and Development, 2008,52(4):401~411

[20] 邱骞.基于持久内存文件系统的可扩展性优化研究[D].武汉：华中科技大学，2015.

[21] Mauerer.Wolfgang.深入理解Linux内核架构.第三版.北京：人民邮电出版社,2010.413-511

[22] 何先波，李明东，王锦，李宝林，张刚园. LINUX操作系统中通用双向循环链表的实现分析.《西华师范大学学报（自然科学版）》，2012，2:213-217

[23] Duy Le, Haining Wang. An Effective Memory Optimization for Virtual Machine-Based Systems. IEEE Transactions on Parallel and Distributed Systems, 2011,22(10):1705-1713

[24] Badam A. How Persistent Memory Will Change Software Systems. Computer, 2013,

46(8):45 ~ 51

[25] Kesheng Wu, Ekow J. Otoo, Arie Shoshani. Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems(TODS), 2016,31(1):1-38

[26] Dalinger I. Formal Verification of a Processor with Memory Management Units. Vdm Verlag Dr Müller, 2008,1(1):33~35

[27] Lee Eunji, Hyokyung Bahn,Sam H. Noh. Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory.in:Proceedings of the 11th USENIX Conference on File and Storage Technologies. San Jose, CA:FAST,2013.1~8

[28] [Charles P. Wright , Nikolai Joukov , Devaki Kulkarni , Yevgeniy Miretskiy , Erez Zadok, Auto-pilot: a platform for system software benchmarking, Proceedings of the annual conference on USENIX Annual Technical Conference, p.53-53, April 10-15, 2005, Anaheim, CA](https://dl.acm.org/citation.cfm?id=1247413)

[29] 庞丽萍，阳富民.计算机操作系统.第二版.北京：人民邮电出版社，2014.170-183

[30] Remzi H, Andrea C. Operating Systems: Three Easy Pieces. Arpaci-Dusseau Books,2015,35-51

[31] A. Wilson, "The new and improved filebench," in Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST), 2008.

# 附录

×××××××××××××××××××××××××××

( 宋体小4号)

……

……

……