武汉理工大学毕业设计（论文）

Linux 内核中 Open-channel SSD 子系统 LightNVM 的搭建与研究

学院（系）：信息工程学院

专业班级：通信 sy1401 班

学生姓名：柏澍晗

指导教师：付　琴

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□，在　　年解密后适用本授权书

2、不保密□

（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

随着存储技术的逐步发展，存储设备对可预测延迟的需求日益增加。然而，类似于固态硬盘的传统服务型块 I/O 设备非但无法满足这一需求，还会存在一些性能上的不足。我们建议使用 Open-Channel SSD 来解决这些问题。Open-Channel SSD 的子系统 LightNVM 优化了它的管理与操作，并通过物理页地址（PPA）I/O 接口与主机共享其内部并行性并将其管理权限交付于主机。

本文在回顾 NAND 闪存局限性的基础上，结合管理传统固态盘所面临的挑战，进一步分析了 Open-Channel SSD 管理的特点并据此确定了 Open-Channel SSD 的基础架构。为实现并优化设备端与主机端的功能交互，我们引入了物理页地址（PPA）I/O 接口来定义分层地址空间、控制命令及向量数据命令，以便准确高效地实现 I/O 操作。为支撑 Open-Channel SSD 复杂的管理工作，我们使用了子系统 LightNVM 并详细介绍了它的结构和功能。最后，基于详尽的理论基础，我们在 Linux 操作系统上，辅助以管理工具及相关库函数，实现了虚拟化 Open- Channel SSD 的成功搭建并在系统平台上验证了检索设备信息、构建物理地址以及执行矢量化 I/O 命令等功能。

关键词：存储技术；存储性能；Open-Channel SSD；LightNVM

Abstract

As storage technologies develop,the demand for predictable delays in storage devices is increasing.However,traditional service block I/O devices similar to SSDs cannot meet this requirement,but there are still some performance deficiencies.We recommend using Open-Channel SSD to solve these problems.The subsystem LightNVM of Open-Channel SSD optimizes its management and operation,and shares its internal parallelism with the host through the physical page address (PPA) I/O interface and delivers its management authority to the host.

This article reviews the limitations of NAND flash memory and combines the challenges of managing traditional SSDs.It further analyzes the characteristics of Open-Channel SSD management and identifies the Open-Channel SSD infrastructure. To implement and optimize the functional interaction between the device and the host, we introduced the physical page address (PPA) I/O interface to define the hierarchical address space,control commands and vector data commands for accurate and efficient I/O operations.To support the complex management of the Open-Channel SSD,we used the subsystem LightNVM and introduced its structure and functions in detail. Finally,based on a detailed theoretical basis,we successfully implemented the virtual Open-Channel SSD on the Linux operating system,assisted with management tools and related library functions,and also verified some functions on the system platform,such as the retrieval of device information， the construction of physical addresses,the performance of vector I/O commands and so on.

Keywords: storage technology;storage performance;Open-Channel SSD;LightNVM

目 录

第 1 章 绪论

1.1 课题研究背景、目的和意义

数十年来，存储技术已成为众多科研机构的研究焦点，它所涉及的研究内容广泛分布于电路及计算机系统等领域，并在各领域取得了极其重大的研究进展。这使得存储技术于日常生活中随处可见，且其应用范围已迅速地由消费类电子产品（如手机、平板电脑、数码相机等）扩展到个人计算机、服务器、数据中心系统、云计算及超级计算等[1]。随着存储技术的逐步发展，固态硬盘（SSD）已然发展成为二级存储的主要形式，且各研究领域对可预测延迟的需求也在日益增加。然而，如果只是简单地将传统磁盘替换为 SSD 进行使用，这种传统的服务型块 I/O 设备非但无法满足这一需求，还会存在一些性能上的不足，如多重日志、严重的长尾延迟效应、不可预测的 I / O 延迟以及资源利用上的不足。这些问题需要利用新的技术结构去解决，以使系统具备最佳的主机开销，且能通过调整实现对读延迟可变性的限制，进一步使得可预测的延迟成为可能。

一类新型固态硬盘——Open-Channel SSD 于存储市场崭露头角，它具有解决这些问题的潜力并能极好地权衡吞吐量、延迟、功耗和容量等相关性能[2]。区别于传统的固态硬盘，Open-Channel SSD 与主机共享其内部并行性并将其管理权限交付于主机。它将某些职责移交给主机并开放供用户使用，使主机能够控制并决定数据的物理布局（SSD 的性能与数据的物理布局紧密相关）及物理 I/O 的调度。Linux 内核中针对 Open-Channel SSD 的子系统是 LightNVM，它极大程度地方便了 Open-Channel SSD 的管理与操作[2]。通过这种方式，存储设备可以使主机适应闪存转换层（FTL）的算法和优化，以匹配它所执行的用户工作负载。用户则可以根据需求及自身数据特点自定义 FTL 层的设计，使其更加高效。这种“由上层定义存储”的方法使 Open-Channel SSD 具备以下三种特性：I/O 隔离、可预测的延迟及软件定义的非易失性存储[3]。

作为 Linux 内核中的 Open-Channel SSD 子系统，LightNVM 通过物理页地址（PPA）I/O 接口将 Open-Channel SSD 开放给主机，并提供分区管理器及可调节的块 I/O 接口。LightNVM 的开销较低，这大大减轻了设备负担。同时，还提高了预测延迟的可能性。上述优势使得“Open-Channel SSD + LightNVM”的实现方式成为克服传统 SSD 性能上缺陷的重要手段[4]。未来，各供应商必将公开更多类型的 Open-Channel SSD 及其管理模型，并进一步研究其核心性能。“Open-Channel SSD + LightNVM”实现方式的定制性也将为关键数据库系统的设计和调优策略提供新的解决方案。

1.2 Open-Channel SSD 的发展历史及研究现状

在计算机系统中，存储器是用于存放计算机内各种有效信息的主要设备，它存储的信

息包括输入的原始数据、计算机程序、中间运行结果和最终运行结果等[5]。控制器发出指令控制存储器在指定的位置写入或读取所需要的信息，赋予计算机记忆功能，并保证其正常工作。

依照存储介质进行分类，存储器有半导体存储器和磁表面存储器两种类型。前者由半导体器件组成，主要包括随机存储器（RAM:Random Access Memory）、只读存储器（ROM:Read Only Memory）和高速缓存（Cache）[6]。后者由磁性材料做成。其中，具有较高性价比的机械式硬盘（HDD:Hard Drive Disk）最为常见，是永久性存储领域的首要选择。然而，近几年来闪存技术持续发展，基于闪存的固态盘走入人们的视线并将逐步取代机械式硬盘的位置。

第一款闪存产品问世于 1989 年，它将新兴的闪存技术带入存储行业研究人员的视野并奠定了未来它在存储市场中的重要地位。闪存是一种永久性存储设备并具有较快的存取和更新速度。闪存通常根据逻辑门类型的不同分为 NOR 闪存和 NAND 闪存[7]。NOR 闪存于发展初期占闪存市场的统治地位，但随着各种手持设备对数据存储量需求的疾速增长，NAND 闪存在存储市场中脱颖而出，已然有逐步赶超 NOR 闪存的势头。1989 年，日本东芝公司发布了 NAND 闪存的结构，此后，NAND 闪存的发展极其迅速。从 1996 年开始，NAND 闪存芯片的存储密度每年都会成倍增长[8][9]。然而，基于闪存的固态盘仍然具有许多限制，比如，写延迟、擦除延迟与读延迟间数量级的差异、存在“写前擦除”的写约束、擦除次数限制闪存寿命等。为进一步提高固态盘存储性能，人们将目光转向了 Open-Channel SSD 的相关研究。

现如今，Open-Channel SSD 已经被一级云提供商使用了一段时间。例如，百度使用 Open-Channel SSD 来精简键值存储堆栈；Fusion-IO 及 Violin Memory 则均实现有一个主机端存储栈来管理 NAND 媒介并提供其匹配的块 I / O 接口[2]。然而，在上述的各种情况下，如何在研究设计领域内将 Open-Channel SSD 有效地整合到基础存储设备的方法已趋于单一化，且其针对各自特定的环境均有一组固定的权衡方案。

2018 年 1 月，Open-Channel SSD 发布了 2.0 版本的标准。目前，Open-Channel SSD 与其子系统 LightNVM 都还处于极其早期的阶段，在市面上很难见到 Open-Channel SSD 且其并不适于直接投入生产。但 Open-Channel SSD 和基于主机的 FTL 所带来的好处却是极其巨大的。对于国内外追求极致存储性能的研究境况，未来极有可能会采用“Open-Channel SSD + LightNVM”的实现方式[4]。

1.3 本论文的主要工作

本文的任务是在深入了解 Open-Channel SSD 及其子系统 LightNVM 的架构及设计原理的基础上，研究在 Linux 内核中构建 LightNVM 的方案，并于 PC 机上成功实现 Open-Channel

SSD + LightNVM 平台的搭建。利用 LightNVM 配套指令进一步理解并研究其工作原理。论文的主要内容和各章的安排如下：

第 1 章 绪论

主要说明了课题的研究目的和意义、Open-Channel SSD 的发展历史及其国际上的研究现状。最后列出了本论文的主要任务和各章节的安排。

第 2 章 Open-Channel SSD 的管理

首先回顾了 NAND 闪存的局限性及管理固态硬盘所需面临的主要挑战。然后，主要介绍了 Open-Channel SSD 管理的特点并确定了开放固态硬盘内部结构的各种约束条件。最后，结合存储行业的相关经验教训给出了 Open-Channel SSD 的基础架构。

第 3 章 物理页地址 I/O 接口

主要介绍了物理页地址（PPA）I/O 接口，该接口专用于 Open-Channel SSD 并定义了分层地址空间、控制命令及向量数据命令。

第 4 章 LightNVM

主要介绍了为 Open-Channel SSD 管理而设计并实现的 Linux 子系统——LightNVM 及其基于主机的闪存转换层（FTL），该闪存转换层名为 pblk 并将 Open-Channel SSD 作为传统块 I/O 设备公开。

第 5 章 LightNVM 平台的搭建与指令验证

主要分析了构建模拟 LightNVM 平台的基本思路与方案，并于搭建成功的 LightNVM 平台上进行了一系列的指令验证。

第 6 章 结论

对本次毕业设计的整个过程进行了一个总结，介绍了设计中遇到的困难及其解决方法，并指出了设计中的不足和改进的建议。

第 2 章 Open-Channel SSD 的管理

固态硬盘通常由数十个闪存芯片封装而成，提供有标准的访问接口和大容量的存储能力。其中，包含多个闪存芯片的闪存存储器通过通道与其控制器进行连接，并由通道来实现命令、地址和数据的传输。Open-Channel SSD 将固态硬盘内部通道和闪存存储器开放由主机控制，并将诸如 I/O 调度与数据放置等 SSD 资源、空间的配置交由主机负责。如何优化 NAND 闪存的管理在现今仍是一个极具挑战性的论题，本章中，我们聚焦于基于 NAND 闪存的 Open-Channel SSD。首先，回顾了 NAND 闪存的局限性及其管理所需面临的主要挑战。然后，确定了开放固态硬盘内部结构的各种约束条件并介绍了 Open-Channel SSD 管理的基本特点。最后，结合存储行业的相关经验教训给出了 Open-Channel SSD 的基础架构。

2.1 NAND 闪存的主要特点

NAND 闪存通过一个类似 I/O 的接口间接地发送指令及地址进行闪存控制，因其内部没有专用的地址线，它无法进行直接寻址操作。相较于 NOR 闪存，对于相同数量的位的存储，NAND 闪存只需较少的逻辑门即可，它体积更小，却具备更大的存储密度，适合海量数据的存储，顺应现今存储行业的发展趋势。另外，NAND 闪存具有更加迅猛的读取速度和写入速度，且其擦除操作以“整块擦除”形式进行，具有简便快捷的特点。

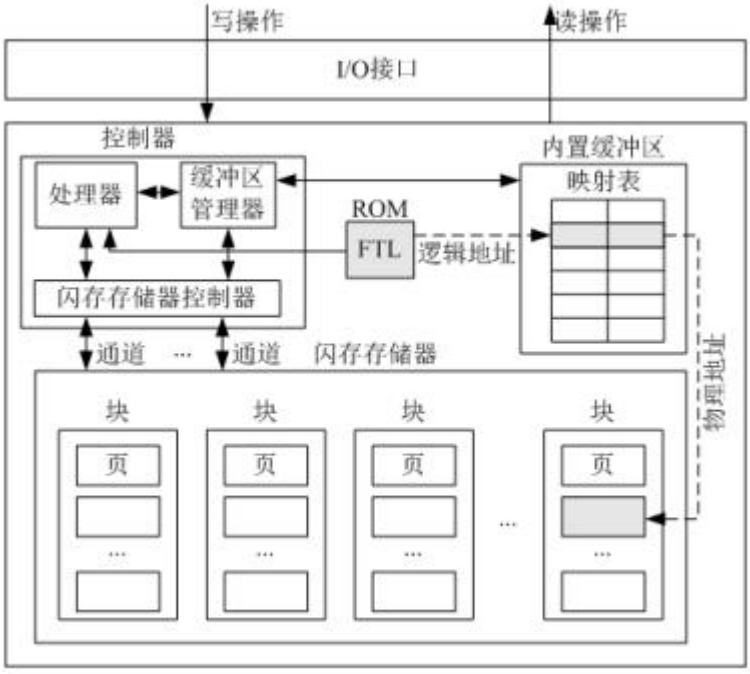

图 2.1 基于 NAND 闪存的固态盘的内部结构示意图

对物理介质的直接研究需要了解其特点并尊重其限制，下面将主要介绍 NAND 闪存的特点和缺陷（由于 NAND 闪存的优势并非本文重点，故不加以赘述）。需要注意的是，由于各类 NAND 闪存都略有差异，一些 NAND 闪存可能会有额外的约束或优化，因其仅在

直接与供应商合作时才会有所应用，在这里不做详细描述。

为更好地理解 NAND 闪存中存在的限制，首先介绍基于 NAND 闪存的固态盘的内部结构，其示意图如图 2.1 所示。固态盘中，闪存存储器采用多级并行结构[10]。为获取较高的数据存取速率以进一步适应现今存储市场对存储设备的期望，闪存中各级别操作都可以并行执行，其内部并行性如图 2.2 所示。

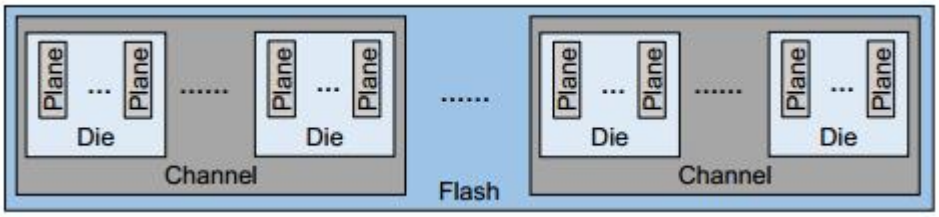

图 2.2 固态盘内部并行性

其中，固态盘被组织成通道、晶粒（die）、面板（plane）、块、页和扇区的层次结构，其中，存储介质（在这里即 NAND 闪存芯片）被分解成块、页面和扇区的组织架构。图 2.2 已清晰地指出，在通道级别、晶粒级别和面板级别，固态盘内部存在并行性，具有并行性的几何体可以被单独选择并独立执行接收到的指令。固态盘的内部并行性大大增加了其吞吐量。

综合固态盘架构及其读写操作特征，它具有如下局限性：

（1）写入操作以闪存页为最小单位：在固态盘中，仅对 NAND 闪存芯片中的闪存页进行部分写入是不可取且无法操作的。写入操作的最小单位是闪存页面，且某一闪存块中的页面被顺序地写入，即当某块中的某一页被写入后，若需对其前面的某一闪存页进行更新，则该更新操作需在该块被擦除后再进行。

（2）存在“写前擦除”的写约束：NAND 闪存不支持对于闪存页面上旧数据的“就地更新”操作，即无法进行对于原始失效数据的直接覆盖。在数据需要更新时，在其他空闲闪存页上进行新数据的写入。将写有已失效旧数据的原始闪存页面置为“无效”状态，待统一执行垃圾回收操作时再一并执行擦除操作。需要注意的是，过多的擦除操作可能会导致固态盘的过快老化，何时执行垃圾回收操作需在设计时多加权衡。

（3）擦除操作以闪存块为最小单元：NAND 闪存无法有选择性地对某个指定的数据项或闪存页进行擦除操作。擦除操作的粒度远远大于写入操作的粒度，必须以整个闪存块为擦除的最小单元，被擦除的块被称为“擦除块”。另外，因过多的擦除操作会导致闪存块的不稳定和过快老化，在设计时，块的擦除次数往往是有一定限制的，这就需要特定的磨损均衡方案对其进行控制。

（4）读取和写入速度存在数量级的差异：NAND 闪存的读写速度是不对称的，其读取速度往往比写入速度迅速一个数量级，而执行擦除操作的时间开销则比执行读取操作的时间开销足足高出两个数量级[11]。甚至，若在写操作或擦除操作之后直接安排读取操作，

读延迟会达到峰值。这种由于读取操作与写入/擦除操作时间开销之间差异所造成的急剧增加的延迟，加剧了闪存延迟的不可预测性。

（5）故障模式：NAND 闪存可能会以如下各种方式发生故障[2]。

① 位错误：存储单元大小的缩减导致了位存储时可能出现的错误的增加。

② 读写干扰：随着位被写入或读取，介质容易干扰到附近的单元。这导致了写入约束，如写操作最小单元是页、块内顺序写入、写前擦除等。

③ 数据保留：随着单元的耗尽，闪存维护有效数据的能力下降。若要将数据保留一段时间，则数据必须被多次重写。

④ 写入/擦除错误：在写入或擦除过程中，由于块级别上某些不可逆的错误，可能会发生故障与错误。在这种情况下，该块上执行的操作应该被取消，并将块中已经写入的数据重写到另一个闪存块中。

⑤ 晶粒故障：因不可抗因素，存储的逻辑单元（即 NAND 闪存芯片上的一个晶粒）可能会随着时间的推移而停止工作。在这种情况下，晶粒中所有数据都将丢失。

2.2 基于 NAND 闪存的固态盘的管理

由于固态盘组织结构及其读写操作的特点，固态盘在管理上具有某些必需的限制。在本节中将介绍与 NAND 闪存管理相关的两个关键挑战：写入缓冲和错误处理。

2.2.1 写入缓冲

当在主机端定义的扇区大小小于 NAND 闪存页大小时，由于写操作最小单元是页，写入缓冲是必要的。为了处理这种不匹配的问题，传统的解决方案是使用缓存：先将扇区的写入临时存放，直到存有足够的数据以填充闪存页面时再将缓存写入存储设备。如果数据必须在高速缓存填满之前保留下来（例如在刷新应用程序的情况下），则会添加填充数据以填满闪存页面。若缓存中积累的数据已能填满闪存页面，则可直接到缓存中读取数据并将其保存到闪存存储器中。

如果将高速缓存设置在主机端，则写入操作可全部由主机发起，从而避免主机和设备之间的干扰。并且，在含写入数据的扇区存入高速缓存时，写入操作即被确认。缺点则是在发生电源故障时缓存中内容可能会丢失。

写入缓存也可以设置在设备端。此时，写入操作有如下两种方法。其一，主机将扇区写入设备，并允许设备对闪存存储器的写入进行管理（当已经积累足够的数据来填充闪存页面时才能写入）。其二，主机明确控制对闪存存储器的写入并让设备保持耐用性。采用前一种方法时，由于写入操作由设备进行管理，则它可能会发起写入且该写入可能会干扰由主机发起的读取操作。这样，设备控制器可能会在工作负载中引入不可预测性。采用后

一种方法，主机可以完全控制对于设备端缓存的访问。此方法避免了设备控制器生成的写入（避免两种写入的混淆），并使主机完全控制闪存存储器的操作。需要注意的是，两种方法都要求设备固件具有电源失效技术，以便在掉电时将写入缓冲器中的内容存储到闪存存储器中。

2.2.2 错误处理

对于 NAND 闪存的读取操作、写入操作和擦除操作，错误处理相关机制的引入是必需的。数据的恢复是错误处理机制解决的主要问题。对于扇区级数据的恢复，若其所有方法都已尝试且无效，则会发生读取操作失败的现象，这是需要尽力去避免的。对于这种情况，我们可以通过引入纠错码（ECC）、调整阈值或采用基于奇偶校验的保护机制（RAID/RAIN）来进行数据的恢复[12]。

纠错码（ECC）的引入有效地补偿了误码，常见的纠错码有 BCH 或 LDPC[13]。一般而言，ECC 编码以扇区为单位进行存储，扇区的大小小于一个页面。我们常通过奇偶处理将 ECC 作为与页面关联的元数据存储在闪存页面的带外区域。

调整阈值也可以有效地恢复被破坏的数据，其缺点是每一次的阈值调整操作需手动进行且会触发多次闪存页的读取操作，这会带来许多额外的时间开销。进而，我们开始采用 RAID 技术进行错误处理以更迅速地恢复损毁数据，同时，RAID 技术还能处理由于晶粒故障而造成的损失。

对于写入故障，块级别数据的恢复也是必要的。某些情况下，部分块可能已被写入且在执行数据恢复时应该被读取。在这里，需要给已写入的块提供足够的缓冲区空间以进行数据恢复。

如果在擦除时发生故障，则不会有指令重试操作或数据恢复操作。该块将被简单地标记为坏块。

综合考虑各方面因素，负责错误处理的功能模块设置于设备端更佳。

2.3 Open-Channel SSD 管理的设计指标

Open-Channel SSD 为固态硬盘的管理开辟了一个巨大的设计空间。基于行业趋势和早期 LightNVM 采用者的反馈，此设计空间受到一些限制。

（1）磨损均衡机制不可与应用相关联。

闪存的寿命受限于其存储单元的写入/擦除操作的次数。PE 周期是衡量闪存寿命的单位，它表示闪存完全擦写一次的时间周期。在闪存的生命周期中，必须有足够优质的闪存介质来执行写入操作。因此，有效的磨损均衡方案是必要的。NAND 闪存随擦除次数的增加而老化，通常反映为访问时间的延长以及读取或写入数据在时间开销上的倍增。理想状

况下，为获得最佳的存储性能，我们希望 I/O 延迟随访问块老化程度的波动较小或尽量无波动，则以引入一定开销为代价的、独立于应用程序的磨损均衡方案是必需的。

常用的磨损均衡方案有动态磨损均衡和静态磨损均衡[14]。动态磨损均衡以 PE 循环次数为衡量标准。数据更新时闪存控制器依据该衡量标准选择空闲块并将新数据写入其中。需要注意的是，被选中的空闲块不能包含任何数据。此时，产生了一个新的映射条目。闪存控制器发起映射更新指令并指出新块的位置，同时将旧数据设置为“无效”状态。垃圾回收操作收集无效数据时，含有“无效”状态数据的闪存块被选为擦除块，统一进行垃圾回收操作后即可重新变为可被利用的空闲闪存块。概括地说，通过科学地选择更新块避免对于同一闪存块的反复擦写操作，动态磨损均衡解决了活跃块上反复写的顽疾，避免了固态盘的过快老化和磨损不均。然而，它仅均衡了有数据写入的块的磨损，对于未被写入的数据块，动态磨损均衡算法将无法顾及它的磨损。对此我们提出了静态磨损均衡算法，希望通过该算法实现全部数据块（包含那些未被写入的块）磨损的均衡。静态磨损均衡算法中，我们依据 PE 计数将闪存块分为静态数据块区域和空闲数据块区域，然后将具有最高 PE 计数的静态数据块区域与具有最低 PE 计数的空闲数据块区域进行交换。

对主机或控制器而言，在执行数据更新时，对于如何在晶粒中选定空闲块的问题，必须能够综合考虑以下三个方面。其一是对于坏块的隐藏。其二是要能在分配块时以 PE 循环计数为衡量指标实现动态磨损均衡。最后是能通过将冷数据转移至热块来实现静态磨损均衡。此类决策应基于设备上收集的元数据——闪存块的 PE 循环计数、闪存页的读取计数和坏块位置等。

PE 周期取决于器件的特定电路设计和制造工艺，若其在主机上进行管理，则无法对 Open-Channel SSD 进行担保。实际上，SSD 供应商并不具备评估存储设备能否继续有效工作的资格。为提供可靠的保证，磨损均衡及块元数据的管理必须在设备上进行。

（2）向主机暴露介质特征的效率低下且限制介质抽象。

在设计时，SSD 供应商和 NAND 供应商要使固态盘的嵌入式闪存转换层能够贴合对应存储芯片的特性。对此，供应商们对存储介质进行了评估。对于诸如阈值调整和纠错码等 NAND 闪存芯片所独有的内部细节，若是交由应用程序或者开发人员负责，极大可能会导致性能不佳的后果并加重人工负担，这是极其不理智且应该被避免的。因此，在设备上控制、管理闪存介质是可取且必须的，这极大地简化了主机中的逻辑。

（3）根据实际使用情况，在主机端或设备端处理写入缓冲。

由 2.2.1 小节分析可知，写入缓冲可根据实际情况设置在主机端或设备端。若将写入缓冲设置在主机端，因为用于执行磨损均衡方案及维护物理介质的信息所占空间可忽略不计，将其存储在设备 SRAM 或持久性介质中即可。设备上无需 DRAM，进而大幅降低了设备功耗。若将写入缓冲设置在设备端，通过控制器内存缓冲器（CMB）即可简单便捷地

管理写入缓冲区[15]。

2.4 Open-Channel SSD 的体系结构

根据对固态盘管理责任在主机和设备间的划分，可以定义不同类型的 Open-Channel SSD。图 2.3 比较了三种 Open-Channel SSD 的核心固态盘管理模块。图（a）是传统块 I/O SSD；综合本文 2.2 及 2.3 小节的讨论，给出了如图（c）所示的理想化 Open-Channel SSD，目前该方案正处于研究阶段且有望在未来得以实现；图（b）是本文中考虑实现的 Open-Channel SSD，其中 PE 周期和写入缓冲在主机端进行管理，它将提供保证，从而支持设备上的 PE 循环管理及磨损均衡方案的正常工作。观察图可发现，传统块 I/O SSD 主机和设备间接口通过逻辑地址进行读写操作，而 Open-Channel SSD 的主机和设备间接口则通过物理地址进行读取、写入和擦除操作，这个与传统 I/O 接口大不相同的标准接口，即是下一章将要详细介绍的物理页地址（PPA）I/O 接口。

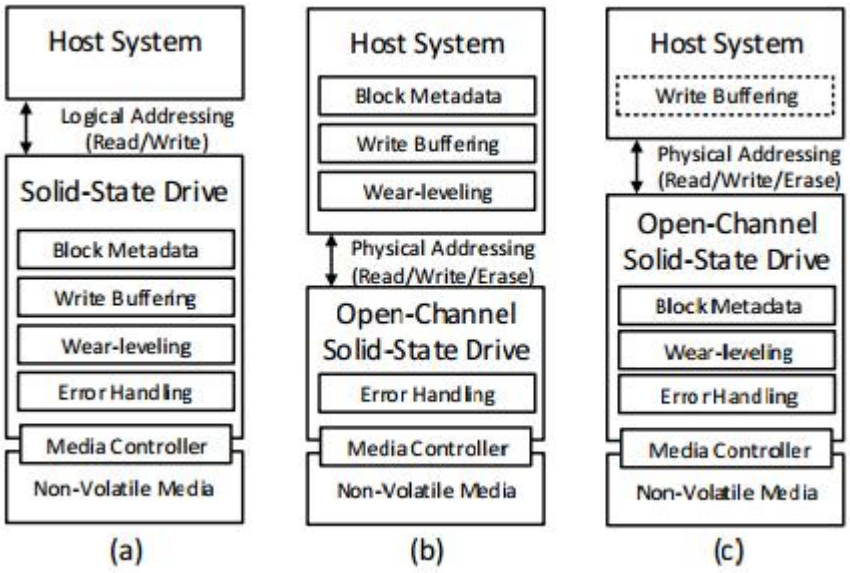

图 2.3 各类别 Open-Channel SSD 的核心 SSD 管理模块

（a）传统块 I/O SSD （b）本文中 Open-Channel SSD （c）未来 Open-Channel SSD

第 3 章 物理页地址 I/O 接口

为方便主机与设备的交流沟通，我们提出了一个基于分层地址空间的 Open-Channel SSD 接口——物理页地址（PPA）I / O 接口。它通过定义管理命令公开设备几何，同时定义了控制命令和向量数据命令使得主机能够有效控制 SSD 的管理，并通过数据命令高效地存储和检索数据。该接口独立于嵌入在 Open-Channel SSD 上的非易失性介质芯片的类型。

本文中，PPA I/O 接口是作为针对 NVM Express 1.2.1 规范的厂商特定扩展来实现的，该规范定义了 PCIe 连接的 SSD 的优化接口[15]。

3.1 地址空间

固态盘架构和其中闪存芯片存储介质的架构决定了物理页地址的地址空间：

（1）SSD 架构

Open-Channel SSD 向主机提供一组通道，每个通道包含一组并行单元（PU），该并行单元也称为 LUN。我们将 PU 定义为设备上的并行单元。PU 可以覆盖一个或多个物理晶粒，且单个晶粒只能是一个 PU 的成员。每个 PU 每次仅能处理一个 I/O 请求[2]。

（2）介质架构

无论介质如何，每个 PU 上的存储空间都是量化的。NAND 闪存芯片被分解成块、页面（写入操作的最小单位）和扇区（纠错码的存储单位）的层次结构。

固态盘控制器可以自主定义 PU 的物理意义。通过这种方式，控制器可以在 PU 级展示反映底层存储介质性能的性能模型。在本文中，为减轻闪存控制器负担，要求使介质性能模型尽可能的简单易懂，我们假设 PU 仅覆盖单个物理晶粒。

物理页地址被组织为反映固态盘及其存储介质体系结构的分解层次结构。例如，NAND 闪存可以被组织为面板、块、页面和扇区的层次结构，而诸如 PCM 的字节可寻址存储器则是扇区的集合[2]。尽管 SSD 架构的组件——通道和 PU 仅存在于物理页地址中，我们也可以将介质架构组件抽象化，具体如图 3.1 所示。

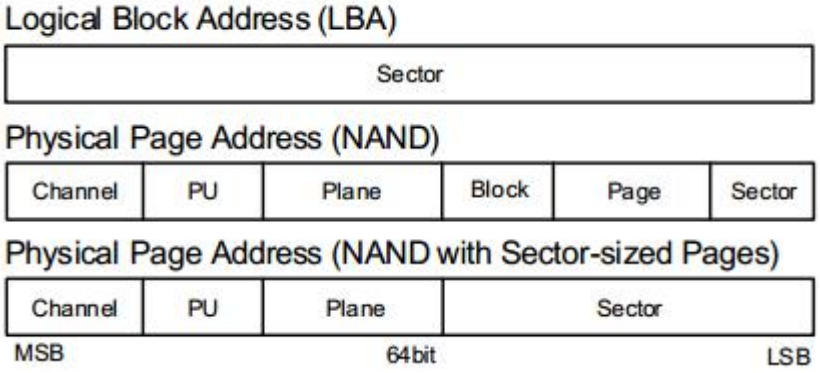

图 3.1 与 NAND 闪存的物理页地址相比较的逻辑块地址

物理页地址具有灵活性，其格式不会限制每个设备的最大通道数或每个 PU 的最大块

数。上述限制由设备定义且能忽略某些介质特定的组件。

将 SSD 和介质体系结构编码为物理地址即可组成物理页地址，该地址能够实现嵌入在 Open-Channel SSD 中的硬件单元的定义，该硬件单元将输入 I/O 映射到适当的物理位置。物理页地址的格式是用十六进制表示的独立变量，简单明了且便于表示。这样，对名称空间的操作（例如，对某一通道上某闪存页进行写入操作，或读取某闪存块上某一页中的数据项），通过各层次结构编号的简单变换即可有效地实现。

我们通过举例说明物理页地址的具体使用方法。例如，NAND 闪存以“块、页面和扇区”的形式组织，在此层次上可以逻辑地组织物理页地址空间代替传统的逻辑块地址（LBA）。这使得 PPA 地址空间可以通过传统的读取/写入/trim 命令暴露出来，而无需进行过多更改。与传统的块 I/O 接口相比，PPA I/O 接口具有一些额外的限制。其中，写入操作必须在一个块内顺序地向闪存页发出且应向整个块发起 trim 命令，以便设备将该命令解释为擦除操作。

3.2 信息公开

为使主机有效地控制固态盘的管理，Open-Channel SSD 需公开以下特征：

（1）设备几何，即物理页地址空间的尺寸参数。

Open-Channel SSD 包含多少个通道？通道中有多少个 PU？每个 PU 中有多少个面板？每个面板中有多少个块？每个块中有多少页？每页中有多少个扇区？每个页面的带外区域又有多大？这些都是设备几何所涉及到的问题。我们假设给定地址空间的 PPA 尺寸是统一的。若固态盘内部闪存芯片阵列由不同类型的存储芯片组成，则 SSD 必须将存储器以独立的地址空间开放且每个地址空间都基于其对应的闪存芯片。

（2）设备性能。

根据当前衡量 Open-Channel SSD 性能的参考指标及研究期望，要求能够较准确地获取在闪存页上执行读取、写入操作与在闪存块上执行擦除命令的典型延迟及最大延迟，同时还需返回发送到通道内独立 PU 中的进行中命令的最大数量。

（3）基于介质特性的元数据。

基于 NAND 闪存介质特性的元数据包括设备上闪存类型、用户可访问的带外区域的大小、闪存块 PE 计数、闪存页读取计数及坏块位置等。随着介质的发展和复杂化，最佳做法是由存储设备管理介质元数据，仅向主机公开相关信息。

（4）控制器功能。

由 2.2 节论述可知，闪存控制器能够支持写入缓冲、故障处理等配置。向主机公开控制器功能，可在特定时刻维护设备性能。如在刷新命令发起时，主机能够强制命令控制器将其缓冲区的内容写入存储介质留存，以防止数据丢失。

3.3 Open-Channel SSD 的 I/O 操作

物理页地址 I/O 接口定义有向量数据命令，该类指令直接反映 NAND 闪存单元的读取、写入和擦除操作。传统的逻辑块地址（LBA）访问模式与起始 LBA、被访问扇区和数据缓冲区高度关联。物理页地址 I/O 接口定义的数据命令对传统的 LBA 访问模式进行改进，将设备内物理地址以向量形式武装，使得执行读取或写入操作时能够有效地运用 SSD 的内部并行性。例如，对于尺寸大小为 64KB 的应用程序数据，若假设页面大小为 4KB，则可以将写入命令同时应用于 16 个页面的扇区来分割该数据，从而有效地支持分散访问。

具体而言，每个 I/O 指令都能以 NVMe I/O 的读取/写入形式进行表征。我们用单个物理页地址（矢量）或一个指向地址列表的指针替代 LBA 访问模式中的起始逻辑块地址（SLBA）字段，其中，指针指向的地址列表被称作为 PPA 列表，它包含有即将被访问的每个扇区的 LBA。当数据命令完成时，物理页地址 I/O 接口单独为每个地址返回一个独立的完成状态。这样，主机可以避免对并行执行的数据命令的混淆并从对应于不同地址的故障中完成恢复操作。

为能在 Open-Channel SSD 上高效准确地执行读取、写入和擦除等 I/O 操作，我们考虑使用 PPA 列表这一替代方案。为充分论述这一选择的正确性，我们对现有的三种 I/O 访问模式进行了对比和评估，从中，能够清楚地观察出矢量 I/O 方案的优势。

（1）NVMe I/O 指令：NVMe I/O 指令串行发布。当缓冲区内数据足以填充一个完整的闪存页空间时，指令将控制数据于介质中的存储。此时，每个指令都会触发其执行状态的改变并向控制器返回新的提交。

（2）分组 I/O：通过分组 I/O，多个页面构成一个提交，仅向控制器通知一次，但控制器仍需维护每个提交的状态。

（3）向量 I/O：借助矢量 I/O，每个数据命令完成时，物理页地址 I/O 接口为每个地址返回其完成状态，该操作无需借助控制器，进一步简化了闪存控制器的设计与管理。

另外，对于不同的存储介质，I/O 命令可根据介质特性提供额外的限制与警示，包括对面板操作模式（单面板、双面板或四面板操作模式）及擦除/编程暂停操作的设置，同时，还能够限制重试操作的次数。面板操作模式定义一次编程需要多少个面板参与。因固态盘默认按顺序访问 PU，闪存控制器可以使用面板操作模式的相关提示来有效地并行编程面板上的存储空间。类似地，擦除/编程暂停操作允许读取操作暂停有效的写入或编程操作，从而改进其访问延迟，但需要付出的代价是更长的写入和擦除时间。对重试操作次数的限制允许主机通知控制器避免耗尽全部选择进行某数据的读取或写入。例如，某数据已经在其他地方可用，若对重试操作次数加以限制，则该操作会在较短时间内失败，该限制大大降低了时间开销并使得固态盘设备能够提供更好的服务质量。

第 4 章 LightNVM

LightNVM 是 Linux 内核中 Open-Channel SSD 的子系统。在本章中，我们将概述其体系结构，并详细介绍其物理块设备 pblk。

4.1 LightNVM 的体系结构

如图 4.1 所示，LightNVM 可细分为三层，其中各个层次结构都为 Open-Channel SSD 提供了结构抽象并标注有与用户空间交互的方式。在本节中，我们将详细介绍 LightNVM 层次结构中的 NVMe 设备驱动程序、LightNVM 子系统以及 LightNVM 目标与硬件交互所使用的高级 I/O 接口，LightNVM 层次结构中的物理块设备 pblk 将在 4.2 小节中详细介绍。

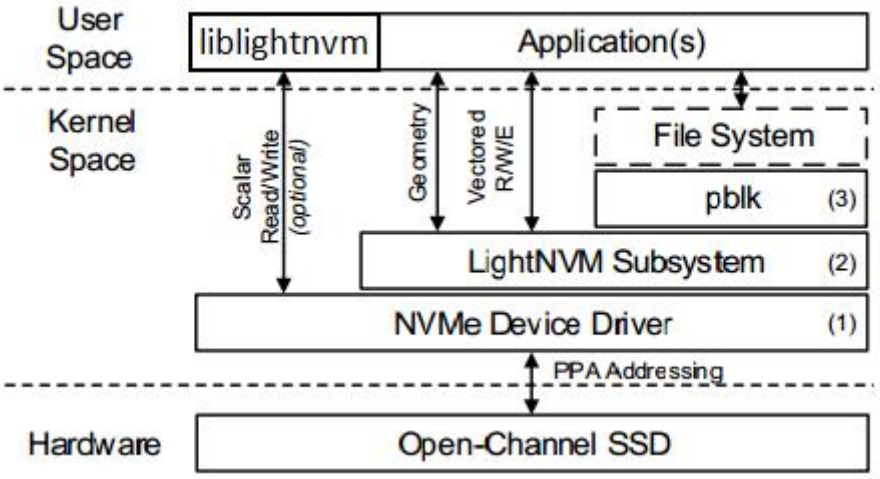

图 4.1 LightNVM 子系统结构

（1）NVMe 设备驱动程序

支持 LightNVM 子系统的 NVMe 设备驱动程序使内核模块能够通过物理页地址 I/O 接口访问 Open-Channel SSD。设备驱动程序将硬件设备视为传统的 Linux 设备并通过标量读写操作将其开放给用户空间，从而使得应用程序能够通过 ioctl 函数与设备进行交互，其中，ioctl 是设备驱动程序中对设备 I/O 通道进行管理的函数[16]。在这种情况下，如果 PPA 接口中指令以 LBA 访问模式发布，NVMe 设备驱动程序也能相应地发出 I/O 命令。需要注意的是，图 4.1 中已表明此类 I/O 属标量读写操作，该操作是可选的。

（2）LightNVM 子系统

子系统的实例在支持 PPA I/O 接口的块设备中进行初始化。该实例允许内核通过内部 nvm_dev 数据结构和 sysfs 文件系统向应用程序公开设备几何，其中，sysfs 是一个向用户空间导出内核数据结构、对象和性质的文件系统[17]。同时，它还使用 blk-mq 设备驱动独有的 I/O 接口作为设备矢量接口，使矢量 I/O 指令可以通过设备驱动程序高效发布[18]。

（3）高级 I/O 接口

LightNVM 目标通过高级 I/O 接口或通过一个目标自定义提供的特定应用程序接口使内核空间模块或用户空间应用程序能够顺利地访问 Open-Channel SSD，其中高级 I/O 接口可以是诸如 pblk 提供的块 I/O 接口等标准接口（详情参阅下述 4.2 节）。

4.2 物理块设备 pblk

作为子系统 LightNVM 的目标，物理块设备 pblk 实现了一个完全关联且基于主机的闪存转换层（FTL），该 FTL 公开了传统的块 I/O 接口。pblk 的主要职责如下：

（1）处理闪存控制器和介质特定的约束。例如，写入操作的最小单位是闪存页，缓冲区内数据量必须足以填充整个闪存页空间才能进行编程操作。

（2）将逻辑地址映射到物理地址，保证相关映射表（L2P）的完整性并采取相对机制以应对崩溃时的最终恢复。

（3）错误处理。

（4）垃圾回收（GC）的实施。

（5）刷新操作的处理。由于典型的闪存页大小通常大于 4KB（4KB 是一个扇区的大小），则在系统掉电或发生故障时，pblk 应采取刷新操作强制缓冲区中正在传输的数据在完成前（即未填满一个写入页面之前）能够存储在设备上。

下面对物理块设备 pblk 的部分职能机制进行说明。

4.2.1 写入缓冲

基于第 2 章综合得出的 Open-Channel SSD 的基础体系结构，本文中使用主机端应用程序对物理块设备 pblk 中写入缓冲进行管理并执行相关操作。写入缓冲区内部由两个独立的缓冲区组成：一个是存储 4KB 用户数据条目的数据缓冲区（4KB 对应一个扇区的大小），另一个是存储各条目元数据的上下文缓冲区。缓冲区的大小是闪存页大小（FPSZ）、被写入的闪存页页数（PP）及 PU 数量（N）的乘积[2]。举例说明，若 FPSZ = 64KB，PP = 8，N = 128，则写入缓冲区大小为 64MB。

写入缓冲区由多个生产者和单个消费者访问[19]：

（1）生产者：物理块设备用户及其垃圾回收操作的执行者均会产生 I/O 指令，在指令执行前，它们将 I/O 指令作为指令行条目插入写入缓冲区队列中进行排队等候并进行缓冲留存。待 I/O 指令发起新数据的写入操作时，映射表会使用输入行进行更新，并根据返回的完成状态确认该写入操作。若在写入缓冲区满时写入新数据，写入缓冲区将溢出并将重新安排对于该条新数据的写入操作。若新数据对应的逻辑地址已建立与物理地址的映射，则在将原来写入的旧数据置为“无效状态”后，根据映射关系写入新数据。

（2）消费者：当数据由写入缓冲区内写入存储介质中时，单个线程将消耗缓冲的条目以完成写入操作，这一过程反映了消费者的作用。一般来说，当写入缓冲区中内容足够支撑整个闪存页或者是在有刷新命令被发起时会引起上述情况的发生。若设备访问模式设置为多面板状态，则在编程时还需额外考虑面板的数量。例如，在四面板编程状态下写入16KB 大小的数据量，应需同时对设备几何体中的四个面板进行写入，需要消耗写入缓冲区中的 64KB 的空间用于单次写入。在执行写入操作的准备阶段，需维持逻辑地址到物理地址的映射关系。确认映射关系状态后（是否已建立映射或该映射关系是否已失效等），矢量 I/O 指令形成并被发送至设备。需要注意的是，设备刷新的情况下，若写入缓冲区内数据量未达到写入操作的最小写入单元，物理块设备 pblk 会在写命令发送到设备之前在其中添加填充数据（即未映射的数据）以满足相应写约束。填充数据不影响数据的写入。也就是说，当应用程序或文件系统刷新时，pblk 确保将所有未完成的数据写入介质以避免有效数据丢失造成的性能下降。完成写入后，消费者线程清空写入缓冲区，并在必要时使用填充数据填满最后一个闪存页。为使数据持久化，最后的写入命令会标注有一个额外的注释，该注释指示它在刷新成功之前必须完成上述步骤。

4.2.2 映射表的恢复

对于逻辑地址到物理地址映射正确性的维护工作大大影响了块设备中数据的一致性。对此，为保证数据存储的准确，我们以三种形式维护映射表的冗余版本，以期进一步提高设备的存储性能。

（1）作为快照，在关机时映射表的冗余版本以 L2P 的完整副本形式进行存储，或者是为维持块操作（闪存块的分配和擦除操作等），按期将映射表的冗余版本作为日志格式闪存转换层的检查点进行存储。

（2）作为块级元数据，映射表的冗余版本存储在各闪存块的首页和尾页。其中，首页被用来存储该块的序列号及其对前一个块的引用。当一个块被完全写入时，最后一页用于存储与该块中第一页相同的序列号（为避免恢复期间的额外读取）、该块指向下一个块的指针和一个闪存转换层日志，该日志由 L2P 映射表中对应于块中数据的相关部分组成。采用此策略，我们得以有序地实现闪存转换层的功能，避免“就地更新”带来的坏影响，防止旧映射对于新映射的直接覆盖。

（3）将映射表的部分映射关系留存在写入设备各个闪存页的带外区域内备用。在这里，对于设备几何体层次中的闪存页而言，它们都具有必备的两个参数，其一是根据 L2P 映射关系得出的与其物理地址对应的逻辑地址，另一个则是表示其页面状态的状态位。通常情况下，各闪存页带外区域将上述两个参数共同保存。

设备重新启动的情况十分常见，然而，为维持闪存中数据的一致性，对于设备的每一

次初始化操作都将触发映射表的完全恢复。在正常关机的情况下，映射表在重新启动时仍是可用的，此时可以直接从磁盘检索映射表并将其加载到内存中。但是，在非正常关机的情况下，逻辑地址到几何体物理地址的映射关系有一定损毁，对于映射表的恢复是必需的。我们为此设计了一个两阶段的恢复过程。首先，检索所有可用块的尾页，分析其状态并将它们分为空闲块、部分写入块和完全写入块三类。为减轻检索负担，我们可以查看序列号，并有选择性的恢复被写入的闪存块。已写入的块包括部分写入块和完全写入块两种，对此，正式进入两阶段恢复过程。在第一阶段，使用序列号对完全写入的块进行排序。然后用存储在每个闪存块尾页上的映射部分更新 L2P 映射表。类似地，在第二阶段中，部分写入的块被排序并恢复该部分的 L2P 映射表。之后，对闪存块进行线性扫描，当检索到带外区域上具有无效位的页面时即可停止扫描。

4.2.3 错误处理

传统的闪存转换层负责读取、写入和擦除失败的处理工作，与此不同的是，物理块设备 pblk 仅处理写入错误和擦除错误。如 2.2.2 小节所述，本文中 Open-Channel SSD 在设备端完成 ECC 和阈值调整的相关配置。对于读取失败，从设备出发，损毁数据是无法恢复的，其恢复操作必须由高于 pblk 的系统上层来管理。

发生写入失败时，为保障设备性能，物理块设备 pblk 设计有两个恢复机制。首先，对于发生写入失败的扇区，使用各扇区的完成位（该位在命令完成条目中编码）来识别与它们相对应的块。同时，对于可用块中这些有写入故障的扇区，控制器重新组织其映射关系并将 I/O 指令重新提交给设备。若识别所得的对应块已被标记为坏块，第二种机制在此时被启动。在这里，坏块中剩余的页被填充并被垃圾回收处理。

发生擦除失败时，对应擦除块将被直接标记为坏块。由于此时没有写入数据，因此没有要恢复的数据。

第 5 章 LightNVM 平台的搭建与指令验证

Open-Channel SSD 对于改善存储设备性能的意义重大。本章主要分析构建模拟 LightNVM 平台的基本思路与方案，并于搭建成功的 LightNVM 平台上进行基本的指令验证，具体介绍“Open-Channel SSD + LightNVM”平台是如何检索设备信息、构建物理地址并发布矢量化 I/O 命令的。

5.1 LightNVM 平台的搭建

5.1.1 硬件支持

根据要求可提供多种 Open-Channel SSD。公共的 Open-Channel SSD 有[20]：

（1）CNEX 实验室的 LightNVM SDK；

（2）EMC Dragon Fire Board Open-Channel OX 控制器；

（3）Radian 内存系统 RMS325.

其中 CNEX 实验室的 LightNVM SDK 能够被借用于学术目的。

若无实际硬件可用，也可以使用支持 Open-Channel SSD 的 qemu-nvme 来实现虚拟化 Open-Channel SSD。

5.1.2 软件支持

要使用 Open-Channel SSD，主机中存储栈必须支持 Open-Channel SSD 的相关规范。该支持可以通过以下三个方面实现[3]：

（1）操作系统：Linux 内核，其版本应达到 4.12 以上。4.12 版本以上的 Linux 内核能够实现 Open-Channel SSD 规范，它通过/dev/block/nvme*/lightnvm/实现 sysfs 条目以枚举 SSD 的设备几何。在内核启动时即可访问这些设备信息。

（2）管理工具：nvme-cli，其版本应达到 0.8 以上。nvme-cli 工具通过 lnvm 扩展支持 Open-Channel SSD。可以使用 nvme lnvm 的相关指令来查看它所支持的参数。LightNVM 目标的初始化是通过 nvme-cli 完成的，nvme-cli 也实现了目标的管理。

（3）相关库：liblightnvm。为成功安装 liblightnvm，要求 Linux 内核版本达到 4.14 以上。liblightnvm 使得用户能够使用与 Open-Channel SSD 相关的应用程序编程接口（API），以便实现用户对于 Open-Channel SSD 的访问。

综合上述限制于要求，搭建“Open-Channel SSD + LightNVM”平台的基本架构如图 5.1 所示。其中，底层 Linux（Ubuntu）版本应为 4.12 及以上，qemu-nvme 上 Linux 版本应为 4.14 及以上并安装 nvme-cli 和 liblightnvm 以支持 Open-Channel SSD 的管理和访问。

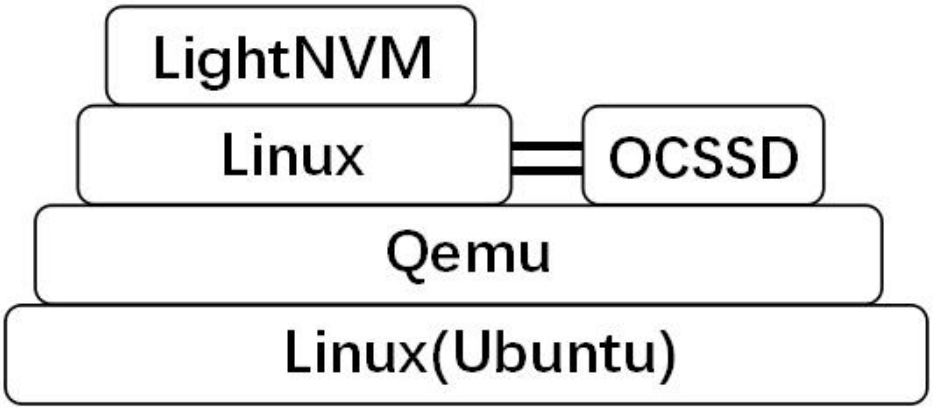

图 5.1 “Open-Channel SSD + LightNVM”平台的基本架构

成功搭建的系统启动后即可检测到 Open-Channel SSD，此时可将 LightNVM 目标初始化为顶层设备并直接使用传统的块 I/O 接口进行交互，另外，也可以使用 liblightnvm 进行访问。

5.2 LightNVM 的指令验证

5.2.1 获取设备信息

使用下列语句启动 qemu-nvme:

```
qemu-system-x86_64 -hda ./ubuntu.img --enable-kvm -m 2G -smp 12 \
-drive file=./blknvme,if=none,id=mynvme \
-device
nvme,drive=mynvme,serial=deadbeef,namespaces=1,lver=1,lmetasize=16,ll2pmode=0,nlbaf=5,lba_index=3,mdts=10,\
lnum_lun=8,lnum_pln=1,lsec_size=4096,lsecs_per_pg=4,lpgs_per_blk=512,ldebug=0,lstrict=1 \
-net user,hostfwd=tcp::7777-:22 -net nic
```

其中，设置的设备几何参数有：LUN（即 PU）数量为 8；面板数量为 1；扇区大小为 4096 字节；每个闪存页包含 4 个扇区；每个闪存块包含 512 个闪存页。上述参数设置详细如指令行中标有下划线注明的部分所示。

成功安装 nvme-cli 后，其具体命令可以通过“nvme lnvm --help”指令列出，如图 5.2 所示。

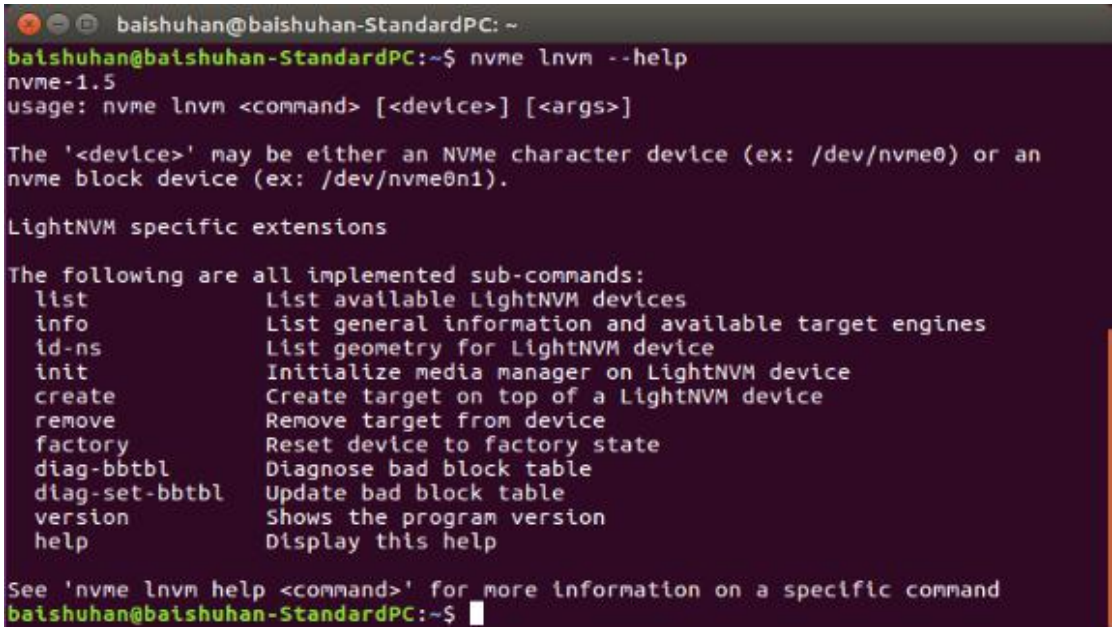

图 5.2 nvme lnvm 指令列表

指令“nvme lnvm list”能够列出已注册的设备并标明该设备的版本和块管理器名称，如图 5.3 所示，本系统中仅有一个名为 nvme0n1 的设备。

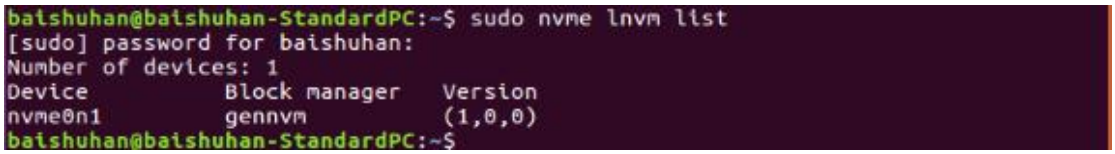

图 5.3 设备列表

指令“nvme lnvm info”能够列出可用的目标类型及其版本，如图 5.4 所示，本系统中有两个目标类型，分别为 pblk 和 rrpc。

图 5.4 目标列表

了解设备的物理几何结构对于物理寻址的工作是至关重要的。设备信息通过调用“nvm_dev info /dev/$DEVICE”指令获取，其中$DEVICE 指代系统中设备名称，结果如图 5.5 所示。由图中 dev_geo 部分所表示的设备几何参数可知，创建所得设备 nvme0n1 的设备几何与初始化时设置参数一致。

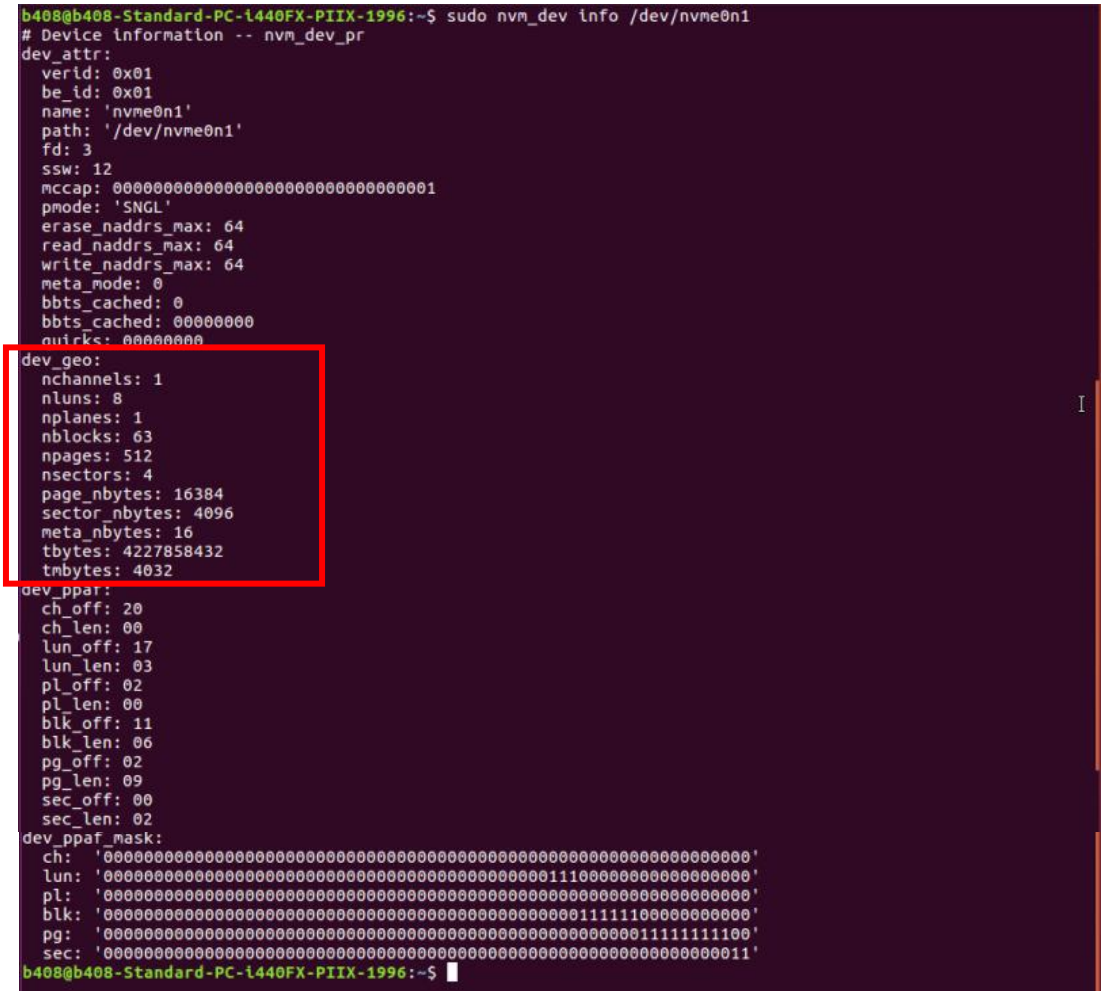

图 5.5 设备信息

5.2.2 目标的创建和移除

LightNVM 可以通过 nvme-cli 工具进行管理和交互，它允许用户创建、移除和管理 LightNVM 的目标和设备。

首次使用设备 nvme0n1 时，需先使用“nvme lnvm init -d nvme0n1”指令为其注册一个块管理器并进行初始化。之后便可在使用 gennvm 块管理器注册的设备上添加目标，对应指令为“nvme lnvm create -d $ DEVICE -n $ TARGET_NAME -t $ TARGET_TYPE -b $ LUN_BEGIN -e $ LUN_END”，其中有以下几点需要注意：

（1）$ DEVICE：即“nvme lnvm list”指令列出的可用设备。

（2）$ TARGET_NAME：要公开的目标的名称。

（3）$ TARGET_TYPE：目标类型。目标需要在其实例化之前于运行时单独进行编译。目前，rrpc 和 pblk 是可用的实现。

（4）$ LUN_BEGIN：分配给目标的 LUN（即 PU）范围的下限。

（5）$ LUN_END：分配给目标的 LUN（即 PU）范围的上限。

举例说明，综合考虑图 5.5 中返回的设备信息，假设 NVMe 设备 nvme0n1 将 LUN 0 至 7 分配给一个名为 mydevice 的 pblk 实例，对应指令为“nvme lnvm create -d nvme0n1 -n mydevice -t pblk -b 0 -e 7”，结果如图 5.6 所示。

```
b408@b408-Standard-PC-i440FX-PIIX-1996:~$ sudo nvme lnvm init -d nvme0n1
b408@b408-Standard-PC-i440FX-PIIX-1996:~$ sudo nvme lnvm create -d nvme0n1 -n mydevice -t pblk -b 0 -e 7
b408@b408-Standard-PC-i440FX-PIIX-1996:~$
```

图 5.6 创建目标

成功创建目标后，用户可向/dev/$ TARGET_NAME 发出读取和写入操作。

若要移除某目标，指令为“nvme lnvm remove -n $ TARGET_NAME”。

5.2.3 构建物理地址

地址指定设备通道、LUN（即 PU）、面板、块、页和扇区的相对位置。其中，并非库中所有部分都需要使用全部的位置信息。最常见的途径有：

（1）LUN 地址：在通道中指定通道和 LUN 的相对位置。

（2）块地址：指定通道、通道内 LUN、LUN 内面板以及面板内块的相对位置。

（3）扇区地址：指定设备中所有几何体的相对位置。本系统即采用扇区地址构建物理地址并以此实现一系列的 I/O 操作。

大多数库都将一个或多个物理地址作为其位置参数以实现物理寻址操作，物理地址有通用格式和设备格式两种。

物理地址的通用格式由结构体 nvm_addr 表示。可以通过指定设备几何内直到扇区的相对位置（即扇区地址）来构造地址。举例说明：构建几何体依次为通道 0、LUN 7、面板 0、块 32、页 200 及扇区 3 的物理地址的通用格式，其指令为“nvm_addr from_geo /dev/nvme0n1 0 7 0 32 200 3”。需要注意的是，相对地址是零索引的，所以通道 0 即是第一个通道。结果如图 5.7 所示。

```
b408@b408-Standard-PC-i440FX-PIIX-1996:~$ sudo nvm_addr from_geo /dev/nvme0n1 0 7 0 32 200 3
addr: {ppa: 0x0007000300c80020, ch: 00, lun: 07, pl: 0, blk: 0032, pg: 200, sec: 3}
b408@b408-Standard-PC-i440FX-PIIX-1996:~$
```

图 5.7 物理地址的通用格式

如图所示，其物理地址的通用格式为 0x00 07 00 03 00c8 0020，是一个用十六进制表示的独立变量。为方便观察，将这个 16 位数按需以空格符分开，依次表示通道、LUN、面板、扇区、页和块的编号。请注意，物理地址的通用格式中各几何体的排列顺序并非是简单地按包含关系来排列的。

因为除了低层命令接口 nvm_cmd 外，接口的各个部分都由库处理设备进行格式转换，故而库用户不需关心设备格式。但是，在需要 nvm_cmd 设备格式或 nvme-cli 等其他工具

的地址的情况下，可以使用以下格式将通用格式转换为设备格式，仍以上述例子说明，对应指令为“nvm_addr gen2dev /dev/nvme0n1 0x0007000300c80020”，结果如图 5.8 所示，转换所得设备格式为 0x00000000000f0323。

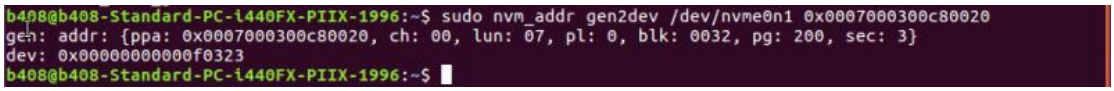

图 5.8 物理地址的设备格式

5.2.4 执行矢量 I/O 操作

在获取设备信息及构建物理地址的知识基础上，我们可以深入构建并执行矢量化 I/O 命令。正如 2.1 小节所言，为成功地处理并执行矢量 I/O 操作，我们面临着一些限制。

（1）写入前擦除

要处理的第一个约束是块在写入之前必须擦除。通过物理地址，我们可以构造一个带有物理地址的单个命令并发送擦除指令：

NVM_CLI_PMODE="0" nvm_addr erase /dev/nvme0n1 0x0000000000000020

其中，通过设置环境变量 NVM_CLI_PMODE="0"来将系统置为单面板模式。结果如下图 5.9 所示。若擦除失败则说明擦除块为坏块。

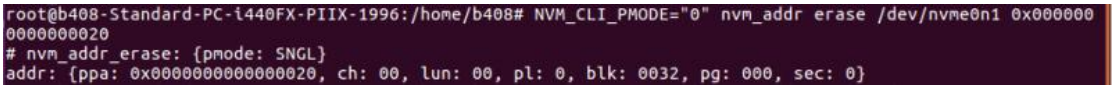

图 5.9 擦除操作

（2）写入的最小单位是闪存页

写入操作的两个主要约束条件是它们必须于完整闪存页的粒度进行，且在块内连续。在实际执行写入操作时需要注意的是，矢量化 I/O 执行写入的 C API 使用 nvm_addr_write 进行寻址，其有效负载必须与扇区大小对齐。

对于如图 5.5 所示的设备几何，完整闪存页是由四个扇区组成的，其中各扇区大小均为 4096 字节，则满足最小写入约束的命令包含四个地址且其有效载荷为 16384 个字节的数据。该命令可以通过 CLI 构建，其指令为“nvm_addr write /dev/nvme0n1 0x0000000000000020 0x0000000100000020 0x0000000200000020 0x0000000300000020”，结果如图 5.10 所示。

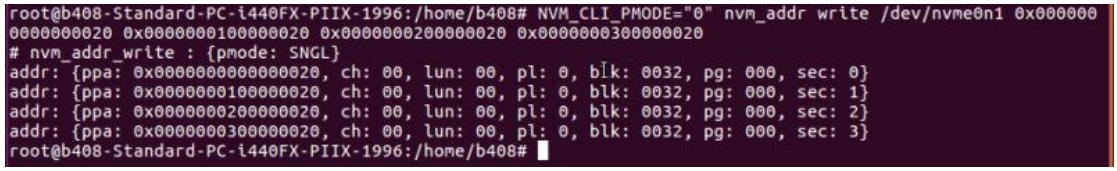

图 5.10 写入操作

写入操作引入了一个额外的限制，它们必须对所有面板上的块执行写操作。 执行时可以通过设置环境变量 NVM_CLI_PMODE="0"选择禁用多面板模式，或通过构建满足面

板模式约束的命令来完成，即于单个命令内完成系统全部面板上块的写入。本系统设置面板数量为 1，故不存在上述问题。

（3）读取操作

我们可以通过指令“nvm_addr read/dev/nvme0n1 0x0000000000000020”读取单个扇区，结果如图 5.11 所示。

图 5.11 单个扇区的读取操作

使用-o FILE 选项将从设备读取的数据写入文件系统，对应指令为“nvm_addr read /dev/nvme0n1 0x0000000000000020 -o /tmp/dump.bin”，其中写入的有效载荷可以通过 hexdump 检查获得，对应指令为“hexdump /tmp/dump.bin -C -n 128”，结果如图 5.12 所示。

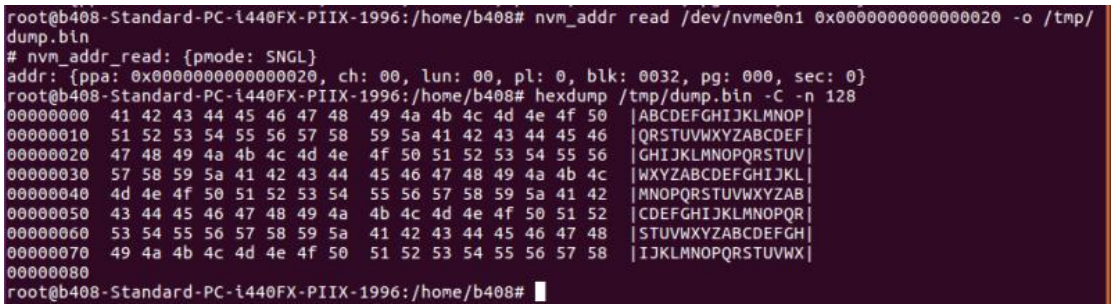

图 5.12 单个扇区的读取与显示

指令“hexdump /tmp/dump.bin -C -n 128”中-C 表示输出十六进制及其对应的 ASCII 码，-n 128 表示仅格式化输出文件的前 128 个字节，上述显示条件与图 5.12 中输出一致。

第 6 章 结论

6.1 论文工作总结

针对现今存储行业对存储设备性能要求的不断提高，尤其是对可预测延迟的需求正在不断地增长，我们选择研究 Open-Channel SSD 及其子系统 LightNVM 以克服传统固态硬盘在性能上的缺陷和限制。本设计主要研究学习了 Open-Channel SSD 的基础架构及其各部分功能、物理页地址 I/O 接口的工作原理及 LightNVM 的结构和功能。论文的主要研究内容如下：

1. 研究并分析了 Open-Channel SSD 的基础架构。通过回顾 NAND 闪存局限性及管理固态硬盘所需面临的主要挑战，我们明确了 Open-Channel SSD 管理的特点并确定了开放固态硬盘内部结构的各种约束条件。最后，结合存储行业的相关经验教训给出了 Open-Channel SSD 的基础架构。

2. 研究并学习了物理页地址 I/O 接口的工作原理及 LightNVM 的结构和功能。我们选择利用物理页地址（PPA）I/O 接口来定义分层地址空间、控制命令及向量数据命令。同时，为 Open-Channel SSD 管理而设计并实现了 Linux 子系统——LightNVM 及其物理块设备 pblk。

3. 利用虚拟机成功搭建了“Open-Channel SSD+LightNVM”平台，并通过指令验证具体介绍了设备信息的检索、物理地址的构建及矢量 I/O 的执行是如何实现的。

6.2 后续工作展望

基于目前国内外对 Open-Channel SSD 的研究现状，本论文只对较基础的 Open-Channel SSD 架构和功能进行了研究和分析。在未来，关于 Open-Channel SSD 的内容还有很大的研究空间。下一步在本论文的基础上，将会对以下问题进行深入的研究：

1. 对于理想化 Open-Channel SSD 架构的研究。后续的工作中，可将重点聚焦于如何将块元数据模块与磨损均衡模块移至设备端工作，以期进一步提高存储设备性能并减轻主机负担。

2. 对于“Open-Channel SSD+LightNVM”平台复杂功能的验证，如对存储设备中坏块表的管理和维护等。因复杂功能的验证所需机器损耗及运行内存较大，虚拟 Open-Channel SSD 无法支持，后续或可订购公共 Open-Channel SSD 以实际验证。

参考文献

[1] Graefe G. The five-minute rule twenty years later, and how flash memory changes the rules[C]// International Workshop on Data Management on New Hardware. ACM, 2007:6.

[2] Bjørling M, Gonzalez J, Bonnet P. LightNVM: The Linux Open-Channel SSD Subsystem[C]// File and Storage Technologies. 2017.

[3] Open-Channel SSD.https://openchannelssd.readthedocs.io/en/latest/, 2014.

[4] 2018 存储技术热点与趋势总结.https://zhuanlan.zhihu.com/p/34455548,2018.

[5] 胡腾云. NASP 海量网络存储系统的研究与实现[D]. 东南大学, 2015.

[6] 佚名. DDR 内存的继任者 DDR2 内存——未来内存市场的新主宰[J]. 电脑采购周刊, 2005(7):3-4.

[7] 黄滨. 大容量 NAND 型闪存数据管理系统研究[D]. 南京大学, 2008.

[8] 章从福. 东芝 NAND 闪存市场传捷报[J]. 半导体信息, 2006(4):93-94.

[9] 江兴. NAND 闪存容量达到 4GB 水平[J]. 半导体信息, 2004, 41(5):27-27.

[10] 刘作龙, 王乐, 刘帅. 多级并行结构的 NAND Flash 存储盘设计[J]. 航空计算技术, 2014(3):113-117.

[11] 林子雨.闪存数据库概念与技术.厦门.厦门大学数据库实验室,2015.

[12] FUSION-IO. Introduces "Flashback" Protection Bring Protective RAID Technology to Solid State Storage and Ensuring Unrivaled Reliability with Redundancy. Businesswire, 2008.

[13] 彭福来, 于治楼, 陈乃阔,等. 面向 NAND Flash 存储的纠错编码技术概述[J]. 计算机与现代化, 2017(11):35-40.

[14] 高立森. 固态硬盘控制器磨损均衡算法研究[D]. 上海交通大学, 2011.

[15] NVMHCI WORK GROUP. NVM Express 1.2.1,2016.

[16] 孙艳. 硕士论文-Linux 环境下数据获取系统及设备驱动的研究实现[J]. 2000.

[17] Michael Hennerich, Robin Getz. ADI 公司如何看待自由和开源软件[J].

[18] Bjørling M, Axboe J, Nellans D, et al. Linux block IO: introducing multi-queue SSD access on multi-core systems[J]. 2013:1-10.

[19] 李晓宇. 操作系统中并发进程的生产者—消费者问题的研究[J]. 许昌学院学报, 2013, 32(2):52-56.

[20] liblightnvm.http://lightnvm.io/liblightnvm/prereqs/ocssd.html,2016.

附录 A

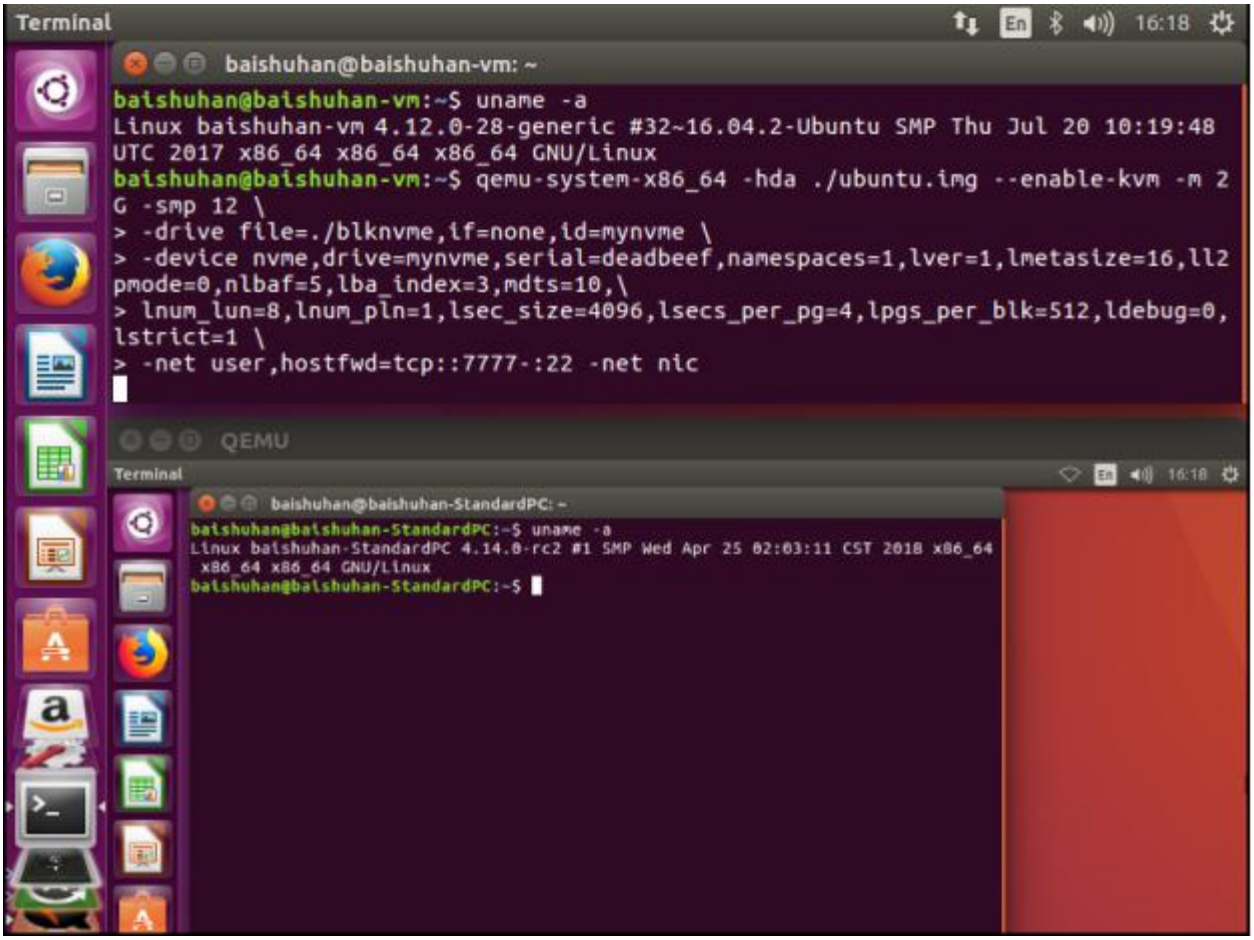

图 A1 LightNVM 平台各层 Linux 内核版本

致 谢

本文是在付琴副教授的悉心指导下完成的。付琴老师博学的知识、严谨的工作作风、平易近人的人格态度都让我获益匪浅。付琴老师不仅在学术上指导我们，帮助我们解答毕业设计中的难题，同时还在思想和生活上给予了热心的帮助。感谢付琴老师！

感谢吴秋霖、朱玥同学在毕业设计上对我的帮助。几位同学对学术怀有热枕的态度，他们实事求是，工作勤恳，在论文完成的过程中给予了深切关怀。感谢与他们一同度过的快乐时光，在最美好的青春里遇到你们是我的幸运。

感谢学校给了我们一次宝贵的机会，使我们能够接触到最先进的存储技术，让我们切身经历了从理论到学术研究的转变。在这个过程中，我在思想上和心理上都有了很大的改变，为我下一步进入研究生学习阶段打下了坚实的基础。

最后要感谢我的家人，他们在我求学路上付出了大量的心血和汗水，特别是我的母亲，一直在背后默默的支持着我。谨以此文献给我的家人，感谢二十多年的养育之恩。