第 11 卷 第 3 期
2022 年 5 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 11 No. 3
May 2022

引文格式：
李迦雳, 刘铎, 陈咸彰, 等. 基于闪存存储的近数据处理技术综述 [J]. 集成技术, 2022, 11(3): 23-41.
Li JL, Liu D, Chen XZ, et al. A survey of flash memory based near-data processing technology [J]. Journal of Integration Technology, 2022, 11(3): 23-41.

基于闪存存储的近数据处理技术综述

李迦雳 刘 铎* 陈咸彰 谭玉娟 曾昭阳

（重庆大学计算机学院 重庆 400044）

摘 要 在冯·诺依曼架构中，存储与计算的分离造成了存储墙问题，导致现有的系统架构难以应对大数据和人工智能时代的数据爆炸。数据的持续增长导致了计算范式的变化，研究者们开始尝试将计算单元移动到存储器中，即近数据处理技术。近数据处理技术是指利用存储控制器的计算能力，执行与数据存取紧密相关的任务，在减少数据迁移的同时，具有低延迟、高可扩展性和低功耗等优点，具有广阔的应用前景。该文首先介绍了近数据处理系统的架构，其次针对特定应用和面向通用场景的相关研究成果进行概述，并总结了软硬件平台和产业进展，最后展望了其未来的发展趋势。

关键词 近数据处理；固态硬盘；计算型存储；闪存存储
中图分类号 TP 391 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20211019001

A Survey of Flash Memory Based Near-Data Processing Technology

LI Jiali LIU Duo* CHEN Xianzhang TAN Yujuan ZENG Zhaoyang

(*College of Computer Science*, *Chongqing University*, *Chongqing* 400044, *China*)
**Corresponding Author*: *liuduo@cqu.edu.cn*

Abstract The isolation of storage and compute units in the Von Neumann architecture leads to the "storage wall" problem, which makes the existing system architecture hard to cope with the challenges of data explosion caused by the wide application of big data and artificial intelligence technologies. The continuous growth of data has led to an evolution in the computing paradigm. Researchers try to move the compute unit to the storage system, that is Near-Data Processing (NDP) technology. NDP technology refers to utilizing the computing power of the storage controller to perform I/O intensive computing tasks, which brings advantages such as low latency, high scalability, and low power consumption while reducing data movement, and has broad application prospects. This article first introduces the near-data computing

收稿日期：2021-10-19 **修回日期**：2021-11-28
基金项目：国家自然科学基金项目(61802038, 62072059)
作者简介：李迦雳，博士研究生，研究方向为智能存储技术；刘铎(通讯作者)，教授，研究方向为计算机系统结构、新型存储体系结构与智能存储系统等，E-mail：liuduo@cqu.edu.cn；陈咸彰，副教授，研究方向为新型存储系统、嵌入式系统等；谭玉娟，教授，研究方向为先进计算机系统结构、智能存储系统与大数据处理等；曾昭阳，博士研究生，研究方向为智能缓存优化。

architecture, subsequently outlines the research results of NDP systems for specific applications and some general scenarios, then summarizes the hardware and software platform and industry progress of NDP, finally looks into the future development trend of NDP technology.

Keywords near-data processing; solid state drive; computational storage; flash storage

Funding This work is supported by National Natural Science Foundation of China (61802038, 62072059)

1 引 言

大数据和人工智能带来的数据爆炸问题，对存储系统和计算系统提出了巨大挑战。快速增长的数据不仅需要大容量的存储器，更需要高效的分析方法来充分挖掘数据的价值。然而，在传统的冯诺依曼架构中，存储部分和计算部分相分离，数据在迁移过程中，需要经过冗长的软件栈以及性能有限的存储接口，这会带来高昂的存储开销，造成系统瓶颈。近数据处理(Near-Data Processing，NDP)技术是解决这一问题的有效手段。

NDP 技术是指利用存储器内部的计算能力进行数据处理，对如数据库、人工智能等具有海量数据的应用进行加速。这种新型架构拥有高性能、减少数据传输量和减轻主机负担的优点。闪存(Flash)由于具有高速、低延迟的特点，被广泛应用于个人电脑、服务器[1]和多种特殊场景[2]中。此外，基于闪存的固态硬盘(Solid State Drive，SSD)因其高带宽和较强的计算能力，成为开展 NDP 架构研究的主要载体之一。目前，NDP 技术在经历了理论探索研究阶段、原型验证阶段后，得到了初步应用，具有广阔的研究和应用前景。本文将介绍基于闪存的 NDP 系统，从传统的计算和存储架构中存在的问题、关键技术、针对特定应用和面向通用场景的研究、产业现状等方面进行介绍，最后总结并展望其未来可能面临的机遇与挑战。

2 近数据处理系统架构

2.1 传统存储架构及其问题

由于机械硬盘带宽低、延迟高，存储器接口和软件栈不会造成其性能瓶颈。当闪存普及后，存储设备性能得到了巨大的提升，但接口和软件栈并没有得到明显的改进。存储栈便成为存储系统的性能瓶颈，阻碍了存储器的性能发挥。传统存储栈的问题主要包含以下方面：

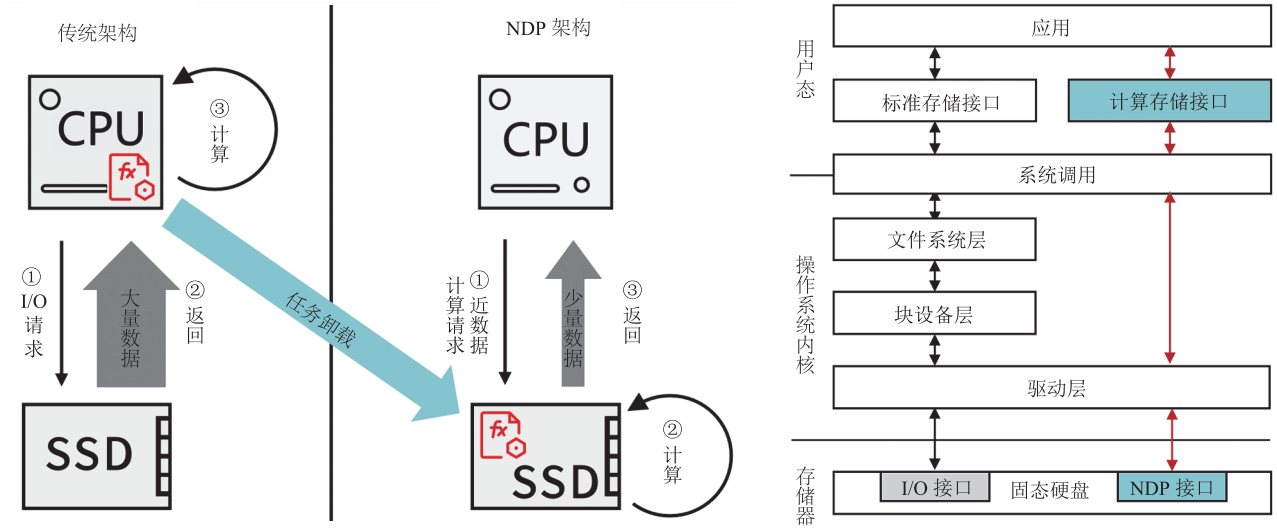
(1)高昂的数据迁移开销。如图 1(a)左侧所示，在传统架构中，存储部分与计算部分相分离，数据读入内存后才能进行计算，在数据库等应用中，这将会导致高昂的数据迁移开销。

(2)冗长的软件栈。以 Linux 系统为例，当应用发出 I/O 请求后，需要依次经过标准存储接口、系统调用、文件系统、块设备层和驱动层等软件层后，才能访问到存储器，如图 1(b)左侧所示。冗长的软件栈降低了访存效率[3]，增加了额外的软件性能开销。

(3)低利用率的存储器性能。由于固态硬盘通常具有多个闪存芯片和读写通道，当并发读写时，固态硬盘内部可以达到很高的带宽，但是由于接口限制，内部带宽不能得到充分利用。

(4)受限的可扩展性。由于处理器和存储器间的传输带宽有限，仅增加存储设备并不能增加存储系统的性能，反而由于可扩展性受限，在某些场景下，可能会造成几十倍的性能浪费[4]。

(5)不灵活的存储接口。由于存储设备的访



(a) 硬件架构对比　　(b) 软件栈对比

图 1　传统计算系统与 NDP 系统的软硬件架构比较

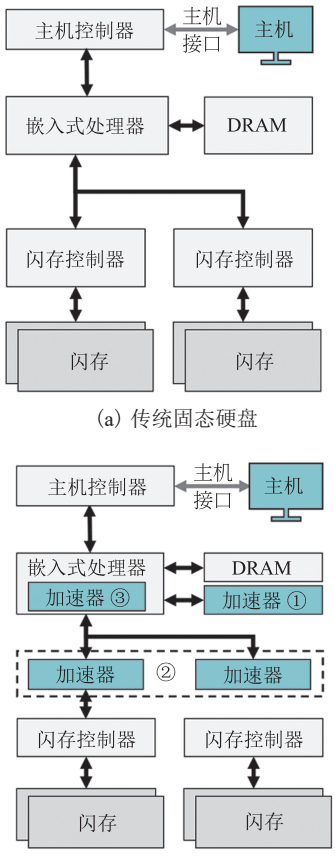
Fig. 1　Comparison of hardware and software architecture between traditional system and NDP system

问接口通常为块接口，所以对数据的请求只能以块为单位。请求只能包含起始块地址、请求长度等信息。然而，不同应用的存储需求各不相同，统一使用块接口缺乏针对性的存储优化能力。

在传统计算机架构的软硬件上，存储系统距离计算系统过远，数据迁移需要高昂的开销。因此，研究者们提出将计算单元置于存储器的新型架构。

2.2　基于闪存的近数据处理系统架构及其特点

在机械硬盘时代，针对传统软件栈存在的问题，有研究者提出用 NDP 的思想[5-6]解决。这种架构的核心思想是利用存储器内部的计算能力，将主机的部分计算任务卸载到存储器中。其中，支持数据处理的存储器称为计算型存储，图 2 为传统固态硬盘与计算型存储架构对比图。传统固态硬盘主要包括主机控制器、嵌入式处理器、闪存、闪存控制器和动态随机存取内存(Dynamic Random Access Memory，DRAM)。与传统固态硬盘相比，计算型存储最大的改进在于存储器具有处理数据的能力。如图 2(b)所示，计算单元的放置通常有 3 种方法：



(a) 传统固态硬盘

(b) 计算型存储

图 2　传统固态硬盘与计算型存储

Fig. 2　Traditional SSD and computational SSD

序号①表示添加在嵌入式处理器之外[7-9]，将其作为协处理器执行计算任务；序号②表示放置在嵌入式处理器与闪存控制器之间[10-11]，此时，每个通道都具有计算能力，具有高并发性的优点；序号③表示放置在嵌入式处理器内，计算单元类型包括硬件加速器[12]以及嵌入式处理器[13-15]。

新型架构与传统架构的工作模式如图 1(a)所示，在传统计算架构中，数据需要从存储器返回到计算单元，而 NDP 架构则将数据处理的一部分或者全部任务卸载至计算型存储中。当 CPU 向存储器发送计算请求后，即可执行其他任务，存储器获取数据后进行计算，然后返回计算结果，减少了大量的数据传输。NDP 系统主要有以下优点：

(1)减少数据传输。存储器只需返回数据处理结果。对于不同的应用，可不同程度地减少数据传输。

(2)减轻主机负担。将计算任务下推到存储器后，主机可以利用原本用于数据传输和计算的系统资源执行其他任务，降低处理器的利用率，提升系统性能。

(3)提高带宽利用率。如图 1(b)所示，NDP 系统提供了绕过包括标准存储接口、文件系统层和块设备层的传统 I/O 栈的方法，通过调用 NDP 接口能够避免高昂的软件栈开销。

(4)提高可扩展性。将处理任务分割、卸载到每个存储器中，可实现性能的线性扩展[5,13]。在 NDP 架构中，只添加存储器即可提升系统性能。

(5)提高安全性。对不同应用授予不同的存储访问权限，只返回数据处理结果而非原始数据，可以降低安全风险。

(6)降低能耗。由于存储器内使用低功耗的嵌入式处理器，功耗远低于主机处理器。此外，对于特定应用可以采用硬件加速器[7,16-18]，进一步提高性能、降低功耗。

早期，对 NDP 系统的研究主要基于机械硬盘，但机械硬盘的内部带宽低，处理器性能弱，在这种架构提出后的一段时间内，后续研究较少。随着闪存时代的到来，固态硬盘得到了普及，其不仅具有很高的内部带宽，并且由于其嵌入式处理器需要进行地址映射、垃圾回收等操作，具有较高的计算性能，因此，NDP 架构再次受到广泛关注，且初步形成了一定的产业规模。

2.3 近数据处理系统中的关键技术

作为一种新型的计算范式，对 NDP 架构的研究和应用涉及软硬件栈的许多方面。如图 3 所示，其在任务分割与卸载、编程框架架构设计和数据处理单元设计等方面有独特的关键技术点：

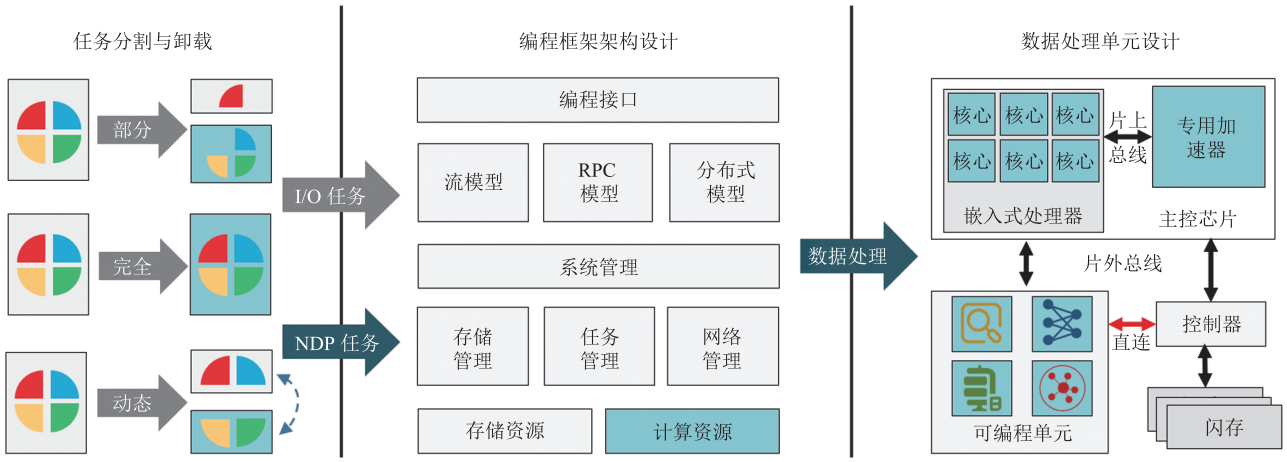


图 3 NDP 系统关键技术

Fig. 3 Key technology of NDP system

(1)任务分割与卸载。NDP 架构的核心思想是将任务卸载到存储器中执行，但存储器的性能有限，所以如何划分任务到存储器中使系统性能最佳成为关键问题之一。主流的做法包括：固定卸载一部分任务的固定分割[12,14,19]、将整个应用完整卸载的完全卸载[9,20]，以及根据系统状态动态卸载任务的动态分割[21-22]。

(2)编程框架的架构设计。由于 NDP 系统的复杂性，在应用开发和系统管理中，编程框架起着十分关键的作用。编程框架的架构设计通常包括：编程接口[23]、编程模型[8,24-25]和系统管理等方面，如何设计高效灵活的计算型存储抽象方法成为 NDP 架构中的关键问题之一。

(3)数据处理单元设计。数据处理方法对 NDP 系统的性能和灵活性有重要的影响。数据处理单元主要分为 3 种类型：专用加速器[3,9]、片外的可编程单元[26]和嵌入式处理器[19,27]。专用加速器具有低功耗和高性能的优点，但无法快速适配新型应用；嵌入式处理器的灵活性高，且开发成本低，但性能较差；可编程单元则相对平衡，在保证灵活性的同时，具有较高的性能。此外，数据处理单元与存储器件间的连接方式也各不相同，如与闪存控制器直连，或利用 DRAM 传输数据等方式。

NDP 是一种全新的系统架构，涉及的关键技术点共同支撑系统的应用，每个方向都值得深入研究，以提高其应用价值。目前，相关研究工作主要围绕通用场景、特定应用、软硬件平台与产业化等方面展开，下文将进行详细阐述。

3　面向通用场景的近数据处理系统研究

本节将从编程框架、文件系统、操作系统等方面对面向通用场景的 NDP 系统的研究工作进行介绍，这些工作降低了开发和应用的成本，提高了系统的性能和易用性，让更多的应用可以从这种新型架构中获益。面向通用场景的 NDP 系统研究总结如表 1 所示。

3.1　近数据处理系统编程框架研究

由于 NDP 应用的复杂性，编程框架一直是研究重点之一。编写一个完整应用存在的问题包括以下几个方面：

(1)开发任务复杂。目前，在 NDP 领域没有固定标准，且缺乏通用的编程模型、协议和库等，应用的开发工作包括：修改应用以卸载部分功能，基于现有通信协议进行修改以满足应用需

表 1　面向通用场景的 NDP 系统研究总结

Table 1　Research on NDP system for general scenarios

工作名称		应用领域	编程模型	系统平台	是否可公开获取
编程框架	Willow[24]	KV 数据库、文件操作等	远程过程调用(RPC)	使用 FPGA 模拟	否
	Biscuit[25]	指针追踪、数据库扫描等	基于流的模型	商用 SSD ＋ 每个通道一个硬件加速器	否
	INSIDER[8]	近邻检索、位图解压等	基于流的模型	使用亚马逊 FPGA 云模拟实现	完全开源
	blockNDP[28]	数据库等	基于流的模型	使用 DRAM 模拟 NAND Flash	部分开源
文件系统	DevFS[29]	存储器内文件系统	标准存储接口	在驱动层模拟实现(基于 PMFS)	否
	CompoundFS[30]	合并常见 I/O 操作	复合文件操作接口	使用 NVDIMM 模拟 Flash	否
操作系统	Newport[13]	支持 Linux、以太网网络栈	分布式模型	专用 ARM 核心用于 OS 和执行计算	可购买
	Shadowgraphy OS[31]	对 NDP 系统进行管理	计划支持透明加速	未实现	否
其他支持	CIDR[32]	数据压缩、数据去重	无(对用户透明)	FPGA 加速卡 ＋ SSD 阵列	否
	FCSV-Engine[33]	计算型存储虚拟化	用户自定义模型	FPGA 硬件辅助虚拟化	否
	Summarizer[22]	动态任务卸载	一组可动态卸载 API	ARM(执行 FTL) ＋ FPGA(闪存控制器)	否

求，以及嵌入式处理器代码开发以实现盘内数据处理。因此，NDP 应用的开发任务复杂。

(2) 运行环境差异。在开发过程中，通常会包含应用、驱动和盘内（嵌入式、FPGA）等不同部分的开发，因此需要考虑不同的开发、测试和运行环境。

(3) 任务卸载问题。一般情况下，固态硬盘控制器的计算能力远低于主机的 CPU，因此，不能简单地将整个应用都迁移到固态硬盘中执行。在开发过程中要考虑，如何分割以及卸载任务使得整个 NDP 系统的性能最大化。

(4) 调试困难。由于缺乏统一的开发环境，通常需要通过不同的方法查看各个组件的运行情况，从而对系统各个部分进行调试，增大了调试复杂性。

对于编程框架存在的问题，Active Disks[5]给出的解决方案是基于流的编程模型 disklet。韩国三星电子提出的 Smart SSD[27]在编程框架方面提供了指令回调、线程管理、内存管理和获取数据的 API；在通信方法上，设计了基于 SATA 接口的会话通信协议，包含 Open、Get、Close 3 个指令。在 Open 接口被调用时，调用用户注册的 Open 回调函数并返回会话 ID，然后用户可以通过 Get 接口监视程序执行状态，以及获取执行结果，最后使用 Close 销毁会话。

2014 年，加州大学圣迭戈分校提出一种高安全性、多并发的可编程固态硬盘 Willow[24]。Willow 内部包含多个存储处理单元（Storage Processor Units，SPU），每个 SPU 包含内存、存储资源的同时，还运行着一个超小型操作系统 SPU-OS，SPU 间通过盘内总线相连，支持同时运行多种 NDP 应用。主机与 Willow 通过远程过程调用进行通信，主机创建并管理着一组主机远程调用端点用于通信。一个完整的 SSD App 包含应用、内核模块和 SSD 内应用 3 个部分。针对安全性问题，Willow 会追踪远程过程调用的所属进程，对于远程过程调用的任务执行、数据访问等操作执行权限检查。

2016 年，针对以前的系统多注重概念验证，没有系统性应用的问题，韩国三星电子提出的编程框架 Biscuit[25]分为软件和硬件两个方面的工作：在软件方面，提出了基于流的编程模型，一个任务流由多个 SSDlet 组成，每个 SSDlet 包含数据的输入、输出和处理方法，主机端程序负责调度与组合 SSDlet 以形成任务流，任务流可支持单生产者多消费者、多生产者单消费者等模式，可实现 Map-Reduce 等数据处理模型；在硬件方面，在每个闪存通道上设置一个模式匹配加速器，可对每次读请求进行并行的数据筛选，然后基于 NVMe 协议，对提出的可编程固态硬盘进行硬件实现，声称具有产品级强度。然而，Biscuit 由于硬件限制，存在性能较低、没有缓存一致性和缺乏内存管理器（Memory Management Unit，MMU）等问题，降低了该工作的可用性。

2019 年，加州大学洛杉矶分校提出了一种完整的 NDP 计算框架 INSIDER[8]。该工作采用基于 FPGA 的数据流处理方法，将用户定义的数据处理模块加载到 FPGA 的应用槽（App Slot）中。当应用需要使用某个数据处理模块进行数据处理时，就需要将文件与该模块所在的 App Slot 绑定，INSIDER 读取该文件后会将文件数据经 App Slot 处理后再返回。一个 FPGA 中有多个 App Slot，可同时支持多种 NDP 应用。为了降低开发难度，INSIDER 还提供了包括应用编译器（基于 LLVM）、加速器编译器（基于 Vivado HLS）、运行时库和驱动程序在内的完整软件栈。

此外，在降低开发难度方面，也进行了许多研究工作：华为慕尼黑开放实验室设计了一种编程框架 blockNDP[28]，用户可通过框架提供的工具将盘内部分的代码编译为中间码（blocklet），然后传输到固态硬盘中，数据流经 blocklet 处理后返回，减少了数据传输。美国微软公司和

NGD System 推出的计算型存储 Newport[13]通过以太网构建了主机和固态硬盘的通信机制，利用 OpenMPI 进行任务分发，并借助分布式文件系统提供文件访问服务和一致性保证。一致的操作系统环境虽然降低了开发复杂度，但是硬件成本过高，此外，完整的操作系统带来了高昂的软件开销。美国英特尔公司提出的基于虚拟对象的编程模型[34]将数据处理所需要的块地址、操作码和操作参数打包为虚拟对象(virtual objects)后发送给固态硬盘进行处理，处理结果通过虚拟对象的方式返回，该模型对软件栈进行少量修改即可支持 NDP 架构。

3.2　近数据处理技术对文件系统的支持

文件系统可利用计算型存储进行优化，如提高性能、增强安全性等。2018 年，威斯康辛大学与华为技术有限公司提出的设备内文件系统 DevFS[29]在存储设备内部实现文件系统，可通过日志、超级块、索引节点(inode)等类似于 Linux 文件系统的方式进行管理。当断电后，固态硬盘可通过内置电容写回内存中的数据，减少了日志开销。此外，DevFS 通过为应用提供 POSIX 兼容接口以绕过内核存储软件栈，减少了软件开销。针对存储器内部内存受限的问题，DevFS 提出了反向缓存机制，将索引节点进行分解后，仅在盘内保留频繁访问的部分，将其他部分传输到主机的内存中，节约了存储器内存开销。DevFS 在驱动层基于 PMFS 进行了模拟实验，与内核文件系统相比，该系统的读写性能分别提高了 2 倍和 1.6 倍。

在 DevFS 的基础上，美国罗格斯大学提出的 CompoundFS[30]针对广泛存在的组合 I/O 操作提供了额外的接口。如在写后计算校验码的操作中，当写入数据后，直接利用存储器计算出校验码和后写入。减少了多次 I/O 操作带来的多次系统调用和数据拷贝的开销。

此外，利用计算型存储的可编程特性可为操作系统和应用提供新的存储接口和抽象模型。一方面，可将固态硬盘内部的结构暴露给主机，使主机能够获取固态硬盘的内部信息，文件系统能够控制固态硬盘的擦除、垃圾回收等操作，使得 I/O 时延可控[35]，提升存储系统性能。这方面的代表工作包括开放通道 SSD[17,35]，以及在最新的 NVMe 协议中加入分区命名空间 SSD[36]，这两种 SSD 都提供了让应用控制 SSD 内部读、写和擦除行为的能力。另一方面，可提供新型的存储接口，赋予存储器更大的数据管理权限，提升存储器的性能优化空间，降低主机负担。这方面的代表工作有提供键值存取接口的 KVSSD[37-38]。Do 等[4]提出了计算型存储灵活可编程的接口，在未来的云计算中可实现存储系统的快速迭代。

NDP 系统由于需要直接读写存储器中的数据，因此，理解数据的格式非常重要。现有的解决方案通常采用将一个文件的所有块地址传输给固态硬盘的方式，但这种方法会带来额外的传输开销，还不利于盘内和主机数据一致性的维护。如何将文件系统与 NDP 技术结合，提高数据的检索、读取与处理效率是未来值得研究的问题。

3.3　面向近数据处理系统的操作系统研究

在计算型存储中，操作系统负责提供程序运行环境、调度多种任务高效运行。Active Disks[5]中提出了 DiskOS，该系统可提供内存管理、流间通信和 disklet 调度的功能。Willow[24]中提出的 SPU-OS 运行在每个 SPU 上，并通过互联网络形成分布式网络。SPU-OS 与主机中的驱动程序协同工作，提供简单的多任务管理和安全性保证。在 blockNDP[28]中，拓展 FTL 实现了类似于操作系统的功能，可支持多核、多线程执行、动态内存分配以及内存保护，提供了运行时环境用于执行中间码以及保证安全性。美国 NGD System 公司提出的一种计算型存储 Newport CSD[13]包含 4 个 64 位的 ARM 处理核心，其运行系统为 Linux 系统。该系统通过基于 NVMe 协议构建的以太

网络实现了完整的网络栈，将固态硬盘作为块设备接入系统，因此，具有与主机端几乎相同的编程方法与运行环境。

华为慕尼黑开放实验室[31]指出了引入 NDP 架构专用操作系统的必要性，以及其应当具备的功能。该实验室认为 NDP 系统存在异构性、一致性和并发性问题，开发者难以解决，因此，需要操作系统的协助。该操作系统的挑战包括局部性问题、安全性问题、主机与存储的任务调度问题、多任务调度问题、盘间调度问题、编程性问题和延迟问题。该工作还提出了 Shadowgraphy OS，目标是支持应用的透明加速、最小化所需单元数量、最小化数据拷贝和去中心化的控制。

目前，在操作系统方面，针对 NDP 技术开展的操作系统工作较少，现有工作多是基于裸机开发。但随着任务复杂度和数量的提升，出于安全性需求和对编程难度的考虑，针对 NDP 架构的操作系统将会成为研究热点。

3.4 其他面向通用场景的近数据处理系统研究

针对 NDP 架构的通用性和性能问题，有工作从其他方面进行了研究：

数据压缩：美国 ScaleFlux 公司提出的 KallaxDB[39]通过固态硬盘内的 FPGA 实现了数据的透明压缩，支持 MySQL、Hadoop 等应用，当压缩率为 50% 时，连续读写性能仍不受影响。美国东北大学提出的 Active Flash[40]使用嵌入式处理器的闲置资源进行数据压缩，并利用无损压缩方法压缩二进制数据和文本数据，可平均减少 49% 的存储量。韩国浦项科技大学提出了 CIDR 技术[32]，并基于 FPGA 设计了用于固态硬盘阵列的新型数据缩减系统——将数据压缩和数据去重工作卸载到计算型存储中执行，写负载性能可提高 2.47 倍。伊利诺伊大学香槟分校提出的 Almanac[41]利用计算型存储实现了历史状态查询和数据回滚。韩国浦项工科大学[42]针对固态硬盘内部进行数据重删导致重删率降低和由 CPU 执行重删导致的 CPU 瓶颈问题，将数据签名和表管理卸载到 FPGA 执行，降低了 CPU 开销，并可检测所有重复数据。美国 Dropbox 公司提出的 Lepton[43]利用 NDP 架构的思想，将图片压缩功能部署在分布式文件服务后端，该系统被应用于生产已有超过一年的时间。

虚拟化：为了解决存储任务处理时间占据处理器运行时间高达 10%～20% 的问题，芝加哥大学提出的 LeapIO[44]利用 ARM 协处理器实现了存储栈的虚拟化，将整个存储服务卸载到协处理器中完成，并将虚拟 NVMe 存储器透明地提供给虚拟机使用，在保证存储性能的同时，降低了 CPU 使用率。韩国首尔大学[33]提出的 FCSV-Engine 利用一种快速、灵活、经济的机制来虚拟化可计算存储设备。FCSV-Engine 基于 FPGA 实现，通过硬件辅助虚拟化和资源编排，实现了高虚拟化性能，其通过动态构造多个虚拟计算存储设备，并在硬件层次上调度虚拟计算存储设备的 NDP 能力，实现了较高的成本效益。

任务分割与调度：由于盘内资源受限，如何合理地分割以及卸载任务以提高系统性能是一个重要的问题。芝加哥大学[21]通过分析 Spark SQL 的工作负载，将可卸载的功能分为 3 类，提出应该首先关注既容易加速又能减少数据量的可卸载功能，最典型的例子是卸载正则表达式匹配操作。针对存储器计算能力弱的问题，南加州大学提出的 Summarizer[22]设计了一组 API，应用程序可利用 API 卸载数据密集型任务。在卸载执行过程中，系统根据存储器内部带宽和外部带宽、主机和存储器计算能力，以及固态硬盘中的任务数量，动态决定该计算是否可以卸载到存储器中执行。弗吉尼亚理工大学提出的工作流感知的存储系统 AnalyzeThis[45]，可通过数据放置和工作流编排，最小化数据移动开销并优化工作流性能，用户可通过提供的接口读写数据、提交分析工作流等。卡耐基梅隆大学提出的 DeltaFS[46]通过收

集计算节点上的空闲计算、内存和网络资源来进行数据计算，隐藏服务器的瓶颈，并动态地重新组织程序的写入，加快后续的查询。

利用计算型存储进行数据压缩是一种主流应用，包括数据库、文本数据和图像数据等。在不更改上层软件的情况下，许多工作能做到透明压缩。但是，存储器内部缺乏对数据语义的感知，压缩率受限，当对数据压缩率和性能有较高要求时，可以利用计算型存储的可编程特性针对不同数据类型提供不同的存储接口，在存储器内部进行针对性的存储优化。目前，对计算型存储虚拟化的研究较少，当计算型存储大规模部署以及需要执行多种应用时，虚拟化是必不可少的技术之一，值得相关学者对其进行进一步的研究。在现有 NDP 架构下，任务分割方法通常是固定的，可能造成某一端的性能瓶颈，未来可从动态分割任务方面进行研究，以提高全系统性能。

3.5　小结

目前，面向通用场景的 NDP 系统的研究主要包括：(1) 编程框架和操作系统等框架类研究，其旨在简化系统开发，提高系统可用性和性能；(2) 文件系统和数据压缩等支持类研究，其可对现有应用提供支持。

在框架类研究方面，已有许多关于 NDP 编程框架的研究不同程度地减少了开发工作量，降低了开发难度。但目前的编程框架仍有很大的局限性：(1) 未形成统一的编程环境，需要在多种开发环境下切换，固态硬盘内的开发环境相对原始，且缺乏库的支持；(2) 对应用不透明，需要对应用进行大量的定制化开发工作；(3) 研究工作大多未进行开源，研究者很难在此基础上进行进一步的研究和使用。针对 NDP 系统的操作系统是解决以上问题的一个有效途径，但目前相关研究几乎处于空白的状态。

在支持类研究方面，对于优化文件系统的性能和安全性，NDP 技术具有广阔的应用前景，且在数据存取阶段，文件系统对 NDP 的数据存取阶段也可提供良好的支持，但文件系统开销、数据一致性等问题还需进一步研究。盘内数据压缩实现了以较低的成本让现有应用获益，是 NDP 系统具有前景的应用之一，如何保证 I/O 性能、多盘间协同压缩提高压缩率是下一步的研究方向。

NDP 系统框架为应用提供支持，同时应用的需求也指导着框架的研究。现有的框架仍存在不足，如不能做到统一开发环境、对多应用的良好支持等。此外，由于缺乏文件系统和操作系统的支持，导致存储器对数据的获取、解析以及资源管理较为困难，阻碍了 NDP 系统的大规模应用。

4　面向特定应用的近数据处理系统研究

本节将对特定应用的 NDP 系统研究工作进行介绍。目前，针对特定应用，普遍使用 NDP 系统进行加速，基于这种新型架构的应用通常具有数据量大、计算简单且可以并行处理的优点。其中，较为典型的代表有数据库应用，以及其他神经网络、图计算等应用。面向特定应用的 NDP 工作总结如表 2 所示。

4.1　近数据处理系统在数据库中的应用

在 NDP 系统研究初期，加州大学提出的 Active Disks[5] 基于 NDP 系统对数据库加速进行了研究。在多盘并行下，Select 操作和 Sort 操作能够获得数倍的性能提升。进入闪存时代后，2011 年，韩国全北大学提出了基于闪存的 NDP 架构[47]，并对数据库的扫描操作进行了加速。2013 年，威斯康星大学、韩国三星电子与美国微软公司联合提出了 Smart SSD[27]，基于商用固态硬盘实现了 NDP 系统，并对 Microsoft SQL Server 中的 Select 和 Aggregate 操作进行加速，实现了最高 2.7 倍的性能提升和降低了超过 2/3 的能耗。

表 2 面向特定应用的 NDP 系统工作总结

Table 2 Summary of NDP system for specific applications

工作名称		应用领域	用户接口	系统平台	是否可公开获取
数据库	Smart SSD[27]	关系型数据库	基于会话的模型	使用嵌入式 ARM(第一代) 内置 FPGA 加速器(第三代)	否
	YourSQL[19]	关系型数据库	对用户透明	NDP 专用 ARM 核心	否
	POLARDB[12]	关系型数据库	对用户透明	内置 FPGA 负责计算与控制	否
	nKV[38]	键值数据库	键值操作接口	使用 SSD 内 FPGA 以及 ARM 实现 NDP 操作	否
人工智能	Cognitive SSD[7]	近似图像检索	自定义图像检索 API	使用 SSD 内 FPGA 硬件加速	部分开源
	RecSSD[14]	个性推荐网络	自定义 I/O、计算接口	使用嵌入式 ARM 加速	完全开源
	Stannis[48]	网络训练加速	任何 Keras、TF 网络	使用嵌入式 ARM 加速	否
	DeepStore[49]	全连接网络、卷积网络等	自定义读写、查询 API	基于模拟器(SSDSIM)添加硬件加速器	否
其他应用	GraphSSD[20]	图计算	自定义图操作 API	使用 SSD 内 FPGA 硬件加速	否
	GLIST[9]	图计算(社交网络)	自定义图操作 API	使用 SSD 内 FPGA 硬件加速	否
	REACT[50]	模式匹配	自定义匹配 API	使用模拟器(Gem5)实现	否

数据库作为 NDP 架构最具优势的应用场景，目前对其研究已经较为成熟。2016 年，韩国三星公司发布的 YourSQL[19]通过将查询的数据扫描阶段卸载到计算型存储中，实现了早期的数据过滤，在 TPC-H 基准测试中，缩短了 72% 的执行时间，性能提升最高可达 15 倍。值得注意的是，该工作基于编程框架 Biscuit[25]，声称具有产品级强度。2020 年，中国阿里巴巴公司和美国 ScaleFlux 公司[12]将计算型存储设备部署到中国阿里巴巴公司的云原生数据库 POLARDB 中，将数据库的表扫描操作卸载到计算型存储中，在 12 种 TPC-H 查询中，时延缩短超过 30%。自有公开文献记录以来，在真实数据库服务中，这是首次大规模地部署计算型存储设备。

除关系型数据库外，对计算型存储在键值数据库中的应用也有一定研究。2017 年，瑞士联邦理工学院提出的 Caribou[51]通过网络提供对分布式 DRAM/NVRAM 存储的键-值访问接口，每个存储节点都具备高带宽的数据处理能力，并通过复制提供容错能力。2018 年，韩国三星公司提出的 KVSSD[37]在固态硬盘内部实现了键值存储，将 LSM-Tree 由主机内存移入 SSD 内部，将 FTL 中的逻辑地址-页映射修改为键-页映射，提高了数据检索效率。德国罗伊特林根应用技术大学提出的 nKV[38]在 KVSSD 概念的基础上加入了硬件检索加速器，研发了盘内检索接口 Scan，实现了 2 倍的性能提升。

4.2 近数据处理系统对 AI 应用的优化

随着 AI 时代的到来，具有数据量大、计算量大特点的神经网络已深入到各个领域。有研究尝试将 NDP 技术应用到神经网络中，对推理和训练进行加速。

中国科学院计算技术研究所提出的 Cognitive SSD[7]通过在固态硬盘内置基于 FPGA 的加速器 DLG-x，实现了近似图像检索加速。DLG-x 加速器包括卷积神经网络加速器和近邻检索加速器，服务器在离线训练完成后，将神经网络权重等参数写入 DLG-x 加速器。当用户检索时，首先将图片发送给卷积神经网络加速器，经神经网络推理后生成该图片的哈希特征值，然后近邻检索加速器根据哈希特征值检索出与该图片最相似的 K 张图片返回给用户。与传统近似图像检索系统相比，该加速器平均减少了 69.9% 的延迟。

美国 NGD System[52]公司将推出的计算型存

储 Newport 应用到神经网络中，并在与美国微软公司合作的工作中[13]，利用 Newport 实现了近似图片检索和目标追踪。NGD System 在与加州大学尔湾分校合作的工作中[48]，利用 Newport 实现了分布式近数据神经网络训练，利用基准测试获取性能参数，计算出每个固态硬盘分配的最佳批处理大小，并尽可能地使所有计算同时完成。

为解决个性推荐模型中嵌入表参数量大的问题，哈佛大学提出 RecSSD[14]将嵌入表操作中的聚合操作卸载到固态硬盘中执行，利用 UNVMe 驱动库实现了用户态的驱动器访问，降低了端到端的时延。为解决非结构化数据查询的 I/O 瓶颈问题，伊利诺伊大学香槟分校提出的 DeepStore[49]设计了一种支持全连接网络、卷积网络、元素(element-wise)和排序 4 种智能查询操作的近存储计算加速器，与使用 GPU 的查询系统相比，性能提高了 17.7 倍。

4.3　近数据处理系统在其他领域中的应用

除数据库与神经网络外，NDP 技术还应用于图计算、大数据分析、模式匹配等大数据场景中。

图计算：在社交网络、金融分析和推荐系统等应用中，图计算发挥着关键作用，但图的大小可能超过内存容量，导致 I/O 操作成为图计算应用的瓶颈。南加州大学提出的 GraphSSD[20]在固态硬盘中执行图分析操作和图更新操作的同时，在图存储时考虑图结构，采用顶点-页面映射方案取代固态硬盘中传统的逻辑到物理页面映射机制，实现了 1.85 倍的性能提升。中国科学院计算技术研究所提出的 GLIST[9]在存储器中添加了图学习加速器，包含图采样器、缓冲区和 PE 阵列，支持图更新、模型注册等操作，与 CPU 和 GPU 的方法相比，GLIST 分别实现了 13.2 倍和 10.1 倍的性能提升。美国麻省理工学院提出的 GraFBoost[53]通过 NDP 技术，利用少量内存即可实现高性能的图分析。

大数据分析：大数据的超大数据量对存储和分析系统提出了新的挑战。韩国汉阳大学提出的 Intelligent SSD[10-11]通过在每个闪存通道上设置加速器的方式加速数据挖掘过程，将 Map-Reduce 模型应用到 NDP 系统中，由每个加速器执行 Map 操作后利用嵌入式处理器执行 Reduce 操作，带来了 2～4 倍的性能提升。美国 ScaleFlux 公司[39]的 CSS1000 和韩国三星公司[54-55]都曾尝试将 NDP 技术应用到 Hadoop 框架中。

模式匹配：模式匹配是许多应用的基础，其具有数据量大但有效数据少的特点，适合利用 NDP 技术进行加速。韩国延世大学提出的 REACT[50]设计了一种适用于 NDP 架构的正则匹配加速器，利用并行处理架构，可同时处理多个输入流，通过隐藏延迟提高性能，基于 NVMe SSD 可提高 22.6% 的吞吐量。INSIDER[8]利用 FPGA 实现了 grep 操作的加速，实现了超过 8 倍的性能提升。

4.4　小结

目前，针对特定应用的 NDP 系统包括数据库、人工智能、图计算等。数据库具有原始数据量大、有效数据少、数据操作较简单和易于分布式处理的特点，是 NDP 系统的主要应用场景。人工智能需要大量的数据集，其利用 NDP 技术进行神经网络训练和推理，除了可以减少大量数据传输、实现并行处理外，还可以带来安全性能的提升，未来可与联邦学习[56]技术相结合，解决在高隐私要求和分布场景下的神经网络训练问题。但固态硬盘的资源受限，NDP 系统如何应对神经网络所需的高算力需求是亟待解决的问题。图计算中的图数据具有一定的结构性，现有工作一般是通过更改存储接口，以适应图存储的特点，并在存储器中更改映射结构，以获得更高的存取效率，此外，还可利用硬件加速器对图计算相关操作进行加速。

在系统设计时，针对特定应用的 NDP 系统

充分考虑了应用的特性，具有较好的性能，得到了广泛的应用。但由于专用性强，需要进行全系统设计，开发代价较高，若需要对系统进行更改，成本很高。如何提高性能、利用有限的性能进行系统优化，以及增强系统通用性和灵活性是未来的研究重点。

5 近数据处理系统软硬件平台与产业现状

NDP 架构从最早的基于模拟器的理论研究阶段[5-6,47]到硬件验证阶段[7,9,13]，再到目前正在进行的真实系统部署阶段[12]，从基于机械硬盘的研究[5]到基于 SATA SSD 的研究[27]，再到基于 NVMe SSD 的研究[7]，实验方法、平台以及工业界的产品与标准互相促进，共同进步。本节主要针对支持相关研究的模拟器、硬件平台、相关产品和标准进行介绍。NDP 系统软硬件平台与产业现状总结如表 3 所示。

5.1 支持近数据处理研究的软硬件平台

除在商业硬件上实现 NDP 架构[19,25]的工作外，由于成本和平台的限制，许多研究都采用模拟的方法。芝加哥大学设计的 FEMU[57]，基于 QEMU 实现了多通道的固态硬盘模拟器，可模拟黑盒(块设备)、白盒(开放通道)和 ZNS 固态硬盘，FEMU 包含整个计算和存储系统的软硬件栈模拟，可支持 NDP 架构的相关研究。韩国延世大学提出的 SimpleSSD[58-59]基于 Gem5 实现了全系统模拟器，可对嵌入式处理器、内存和闪存等整个软硬件栈进行建模，并开展相关研究[50]。INSIDER[8]基于亚马逊的 FPGA 云实现了闪存控制器和近数据计算单元。韩国科技院[66]使用英特尔推出的众核加速卡 MIC 模拟多核嵌入式处理器。伊利诺伊大学香槟分校和华中科技大学使用 SSDSim[49,60]模拟可编程固态硬盘。此外，多种对现代多队列固态硬盘进行全系统模拟的模拟器[67-68]也具有进行 NDP 系统研究的潜力。

随着固态硬盘的普及和应用，涌现出了许多支持 NDP 系统研究的开放硬件平台，相关研究工作不再局限于软件模拟。韩国汉阳大学推出的开放固态硬盘平台 OpenSSD[61]通过内置的 ZYNQ 芯片实现了包括 PCIe 控制器、闪存控制器和 FTL 等功能，开放二次开发方法，具有很高的灵活性，可实现开放通道固态硬盘[35]和多种 NDP 系统[9,14,28]。戴尔公司推出的可编程固态硬盘 DFC[62-63]具有 8 核 ARMv8 处理器、16 GB 内存和多种硬件加速器(压缩解压缩、加密等)，通过可替换的存储模块实现了对多种存储介质的支持。可同时支持 RDMA 访问，支持通过 PCIe 安

表 3 NDP 系统软硬件平台与产业现状总结

Table 3 Summary of software and hardware platform and industry status of NDP system

工作名称			应用领域	系统特点	系统平台
软硬件平台	模拟器	FEMU[57]	块设备、ZNS 等模拟	全系统模拟，动态性能分析	QEMU
		SimpleSSD[58-59]	模拟多种闪存设备	全系统模拟，动态性能分析	Gem5
		SSDSim[60]	模拟单通道固态硬盘	离线 trace 分析，模拟开销低	独立应用
	硬件	OpenSSD[61]	支持 SSD 相关研究	全系统可编程，支持硬件加速	Xilinx ZYNQ
		Dell DFC[62-63]	支持 FTL、编解码等研究	模块化设计，可支持多种介质	SoC(基于 ARM)
产业现状	产品	NGD System[13,52]	目标跟踪、网络训练等	支持分布式计算和存储框架	SoC(基于 ARM)
		ScaleFlux[39]	数据压缩、数据库等	对用户透明	FPGA
		深圳大普[15]	加解密、智能数据处理等	支持操作系统	SoC(基于 ARM)
	标准	ARM NDP 白皮书[64]	数据中心、智慧城市等	通过 OS、容器等方式支持 NDP	ARM R82
		SNIA 计算型存储草案[65]	对计算型存储管理、安全、操作等方面进行了定义	采用分布式模型	适用于块设备、KVSSD、NVM 等

装闪存或其他形式的非易失性存储器[4]。

5.2　近数据处理系统产业现状

目前，NDP 架构已经在某些固态硬盘产品中实现，除前文提到的 ScaleFlux[39,69]支持数据压缩功能和定制化的数据处理功能，NGD System[52]发布的包含 NDP 专用的 4 核 ARM A53 的企业级计算型存储外，还有中国深圳大普[15]发布的嵘神 5 系列 PCIe SSD。该产品包含其自研的固态硬盘控制器 DPU600，支持数据实时加解密存储和智能数据处理与分析等功能。三星 Smart SSD 已经发布的第 3 代产品[26,70]，该产品通过 FPGA＋独立固态硬盘控制器的架构实现计算型存储。ARM 发布的计算型存储专用核心 R82[71]，可以运行 Linux 系统，最高支持 1 TB DRAM，结合 ARM Neon 等技术可满足高端计算型存储的需求。

在产业标准方面，2019 年，中国阿里巴巴公司提出了计算型存储 AliFlash V5[72]，并与韩国三星公司、美国 ScaleFlux 公司合作构建生态。2020 年，SNIA 成立了计算型存储工作组，发布了计算型存储体系结构和编程模型草案[65]，目的是制定行业标准，实现设备和系统之间的互操作，该草案定义了存储器与主机或存储器间的接口和功能，包含管理、安全和操作等方面；同年，ARM 发布了计算型存储白皮书[64]，讨论了计算型存储的优势、应用方法与应用领域。

5.3　小结

多种模拟器和开放硬件平台为 NDP 架构的研究带来了便利，但这些模拟器和硬件平台通常不是针对 NDP 技术研究开发的，缺乏相关资料和库的支持，导致基于模拟器和开放硬件平台的研究和开发门槛较高，如何降低开发难度是一个尚未解决的问题。目前，NDP 架构并未被大规模应用，与产品成本较高、生态不成熟等因素有关。随着产品性能和易用性的提升，以及标准的成熟，NDP 技术将会在实际场景中得到越来越多的应用。

6　总结与展望

目前，基于闪存的 NDP 技术作为解决存储墙问题的有效手段得到了一定程度的研究与应用。相关研究与应用包括编程框架、文件系统、数据压缩等面向通用的场景，以及数据库、人工智能等面向专用应用的场景，均具有一定的性能、成本和功耗的优势。但是，目前 NDP 架构仍不够成熟，在通用性和性能等方面还需进一步完善。目前的解决方案还存在以下问题：

(1) 编程模型与开发框架。目前，虽然在 NDP 领域已有一定数量的研究成果和上市产品，但并没有形成固定的编程模型，有基于会话的模型[27]、数据流模型[8]和分布式模型[13]等各种模型。针对特定应用的加速通常是基于 NVMe 协议扩展的自定义指令，由于目前缺乏相关标准，缺乏通用性，为后续开发、应用和研究带来了困难。随着行业的成熟以及标准的建立，相信这些问题会逐渐得到解决。

(2) 数据访问问题。与传统的以计算为中心的架构相比，NDP 系统需要从存储器内部直接获取数据，带来了新的挑战，例如：①数据安全问题，如何保证 NDP 应用无法访问到不属于当前用户的数据；②访问效率问题，现有方案通常采用传输数据块地址的方法告知 SSD 所需数据位置，带来了额外的数据访问开销，且加重了 CPU 的负担；③一致性问题，如何保证主机访问到的数据与 NDP 单元访问数据的一致性问题。

(3) 可扩展性问题。当进行大规模部署时，NDP 系统能够充分发挥其高并发性以及高可扩展性的优势，但现有工作通常在单存储器场景下展开，当部署数量提高时，会带来性能隔离、多副本数据一致性、异构数据的处理等问题和挑战。此外，由于闪存的寿命较短，当存储单元寿命耗尽时，计算单元通常仍可正常工作。对于失去存储功能但仍具有计算功能的节点，如何妥善处理

这些节点是大规模部署下亟待解决的问题。

除上文中提及的当前解决方案中存在的问题外，未来的研究还可以从以下几个方面进行：

(1)与 AI 技术结合。与 AI 技术的结合可以通过两方面进行：①设计针对 AI 应用进行优化的近数据存储系统。如通过 NDP 技术优化 AI 网络的数据处理部分[14]，针对 AI 算法的特性(如容错率高、非常结构化的访问模式)进行存储优化，设计表达神经网络数据访问意图的 API[73]，通过缓存中间结果[74]、检查点[75]、数据追踪[76]等技术优化 AI 网络的各个阶段。②使用 AI 技术对存储系统进行优化。如利用神经网络预测数据访问模式，有利于优化数据缓存[77]、预取[78]和存储资源配置[79]，还可检测存储系统的故障和数据损坏[80]等。

(2)研发针对 NDP 架构的操作系统。由于存储器内的计算资源越来越丰富，需要管理的资源和工作负载类型也越来越多，传统的裸机开发方法已经难以应对这些挑战。若引入完整的桌面级操作系统(如 Linux)，会导致性能开销过高，给性能受限的嵌入式处理器带来过大的负担；而嵌入式操作系统(如 FreeRTOS)又难以支撑计算型存储对任务动态加载、提供运行环境的需求，因此，需要研究面向 NDP 架构的操作系统，在满足资源管理需求的同时，充分利用存储器性能。

(3)实现对更多应用的加速。由于相关标准的缺乏和生态的不成熟，针对特定应用进行优化在未来的一段时间内仍将是 NDP 技术的主要应用方法。但 NDP 系统具有性能相对较低，所面向的应用访存特性差异大等特点。即使是具有相对通用的编程框架，也需要针对不同的应用进行针对性的优化，才能充分发挥 NDP 系统的优势，规避短板。因此，一方面可针对更多类型的应用(如自动驾驶、金融分析和药物研发等)进行优化，另一方面还可研究具有一定通用性的领域专用软件接口，以及对相关软件库进行研究。通过软件接口和库的深度优化，可简化具有相似特性的 NDP 应用的开发工作，便于应用的部署，提升应用性能。

在大数据和人工智能时代，NDP 架构减少了数据迁移，提高了扩展性和安全性，降低了功耗和成本，是解决存储墙问题的有效途径。本文总结了 NDP 架构的概念、优势、相关研究、相关产品和行业标准，提出了目前存在的问题以及未来的研究方向。NDP 技术现已得到了学术界和工业界越来越多的关注，具有广阔的应用前景。然而，该技术仍然面临着许多问题以及尚未探索的领域，值得相关学者对其进一步深入探索。

参 考 文 献

[1] 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.
Lu YY, Shu JW. Survey on flash-based storage systems [J]. Journal of Computer Research and Development, 2013, 50(1): 49-59.

[2] 高怡祯. 基于闪存的星载大容量存储器的研究和实现 [J]. 电子技术应用, 2003, 29(8): 75-78.
Gao YZ. Research and implementation of spaceborne mass memory based on flash memory [J]. Application of Electronic Technique, 2003, 29(8): 75-78.

[3] Ren YJ, Min C, Kannan S. CrossFS: a cross-layered direct-access file system [C] // Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, 2020: 137-154.

[4] Do J, Sengupta S, Swanson S. Programmable solid-state storage in future cloud datacenters [J]. Communications of the ACM, 2019, 62(6): 54-62.

[5] Acharya A, Uysal M, Saltz J. Active Disks: programming model, algorithms and evaluation [J]. ACM SIGOPS Operating Systems Review, 1998, 32(5): 81-91.

[6] Riedel E, Gibson G, Faloutsos C. Active storage for large-scale data mining and multimedia applications [C] // Proceedings of the 24th Conference on Very Large Databases, 1998: 62-73.

[7] Liang SW, Wang Y, Lu YY, et al. Cognitive SSD: a deep learning engine for in-storage data retrieval [C] // Proceedings of the 2019 USENIX Annual Technical Conference, 2019: 395-410.

[8] Ruan ZY, He T, Cong J. INSIDER: designing in-storage computing system for emerging high-performance drive [C] // Proceedings of the 2019 USENIX Annual Technical Conference, 2019: 379-394.

[9] Li CY, Wang Y, Liu C, et al. GLIST: towards in-storage graph learning [C] // Proceedings of the 2021 USENIX Annual Technical Conference, 2021: 225-238.

[10] Bae DH, Kim JH, Kim SW, et al. Intelligent SSD: a turbo for big data mining [C] // Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, 2013: 1573-1576.

[11] Cho S, Park C, Oh H, et al. Active disk meets flash: a case for intelligent SSDs [C] // Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, 2013: 91-102.

[12] Cao W, Liu Y, Cheng ZS, et al. POLARDB meets computational storage: efficiently support analytical workloads in cloud-native relational database [C] // Proceedings of the 18th USENIX Conference on File and Storage Technologies, 2020: 29-41.

[13] Do J, Ferreira VC, Bobarshad H, et al. Cost-effective, energy-efficient, and scalable storage computing for large-scale AI applications [J]. ACM Transactions on Storage, 2020, 16(4): 1-37.

[14] Wilkening M, Gupta U, Hsia S, et al. RecSSD: near data processing for solid state drive based recommendation inference [C] // Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021: 717-729.

[15] DapuStor. Roealsen5 [EB/OL]. (2022-03-04)[2022-03-30]. http://www.dapustor.com/.

[16] Woods L, István Z, Alonso G. Ibex——an intelligent storage engine with support for advanced SQL off-loading [J]. Proceedings of the VLDB Endowment, 2014, 7(11): 963-974.

[17] Jun SW, Liu M, Lee S, et al. BlueDBM: an appliance for Big Data analytics [C] // Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture, 2015: 1-13.

[18] Schmid R, Plauth M, Wenzel L, et al. Accessible near-storage computing with FPGAs [C] // Proceedings of the 15th European Conference on Computer Systems, 2020: 1-12.

[19] Jo I, Bae DH, Yoon AS, et al. YourSQL: a high-performance database system leveraging in-storage computing [J]. Proceedings of the VLDB Endowment, 2016, 9(12): 924-935.

[20] Matam KK, Koo G, Zha HP, et al. GraphSSD: graph semantics aware SSD [C] // Proceedings of the 46th International Symposium on Computer Architecture, 2019: 116-128.

[21] Zou C, Chien AA. Empowering architects and designers: a classification of what functions to accelerate in storage [R]. TR-2020-02, UChicago CS, 2020.

[22] Koo G, Matam KK, Te I, et al. Summarizer: trading communication with computing near storage [C] // Proceedings of the 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture, 2017: 219-231.

[23] Do J, Kee YS, Patel JM, et al. Query processing on Smart SSDs: opportunities and challenges [C] // Proceedings of the 2013 ACM SIGMOD

International Conference on Management of Data, 2013: 1221-1230.

[24] Seshadri S, Gahagan M, Bhaskaran S, et al. Willow: a user-programmable SSD [C] // Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, 2014: 67-80.

[25] Gu B, Yoon AS, Bae DH, et al. Biscuit: a framework for near-data processing of big data workloads [J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 153-165.

[26] Samsung. Samsung SmartSSD computational storage drive [EB/OL]. (2018-10-18)[2022-03-30]. https://samsungsemiconductor-us.com/smartssd/.

[27] Do J, Kee YS, Patel JM, et al. Query processing on Smart SSDs: opportunities and challenges [C] // Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 2013: 1221-1230.

[28] Barbalace A, Decky M, Picorel J, et al. blockNDP: block-storage near data processing [C] // Proceedings of the 21st International Middleware Conference Industrial Track, 2020: 8-15.

[29] Kannan S, Arpaci-Dusseau AC, Arpaci-Dusseau RH, et al. Designing a true direct-access file system with DevFS [C] // Proceedings of the 16th USENIX Conference on File and Storage Technologies, 2018: 241-256.

[30] Ren Y, Zhang J, Kannan S. CompoundFS: compounding I/O operations in firmware file systems [C] // Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems, 2020.

[31] Barbalace A, Iliopoulos A, Rauchfuss H, et al. It's time to think about an operating system for near data processing architectures [C] // Proceedings of the 16th Workshop on Hot Topics in Operating Systems, 2017: 56-61.

[32] Ajdari M, Park P, Kim J, et al. CIDR: a cost-effective in-line data reduction system for terabit-per-second scale SSD arrays [C] // Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture, 2019: 28-41.

[33] Kwon D, Kim D, Boo J, et al. A fast and flexible hardware-based virtualization mechanism for computational storage devices [C] // Proceedings of the 2021 USENIX Annual Technical Conference, 2021: 729-743.

[34] Adams IF, Keys J, Mesnier MP. Respecting the block interface-computational storage using virtual objects [C] // Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems, 2019: 10-16.

[35] Lerner A, Bonnet P. Not your grandpa's SSD: the era of co-designed storage devices [C] // Proceedings of the 2021 International Conference on Management of Data, 2021: 2852-2858.

[36] Bjørling M, Aghayev A, Holmberg H, et al. ZNS: avoiding the block interface tax for flash-based SSDs [C] // Proceedings of the 2021 USENIX Annual Technical Conference, 2021: 689-703.

[37] Wu SM, Lin KH, Chang LP. KVSSD: close integration of LSM trees and flash translation layer for write-efficient KV store [C] // Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition, 2018: 563-568.

[38] Vinçon T, Bernhardt A, Petrov I, et al. nKV: near-data processing with KV-stores on native computational storage [C] // Proceedings of the 16th International Workshop on Data Management on New Hardware, 2020: 1-11.

[39] Chen XB, Zheng N, Xu SK, et al. KallaxDB: a table-less hash-based key-value store on storage hardware with built-in transparent compression [C] // Proceedings of the 17th International Workshop on Data Management on New Hardware, 2021: 1-10.

[40] Tiwari D, Boboila S, Vazhkudai S, et al. Active Flash: towards energy-efficient, in-situ data analytics on extreme-scale machines [C] // Proceedings of the 11th USENIX Conference on File and Storage Technologies, 2013: 119-132.

[41] Wang XH, Yuan YF, Zhou Y, et al. Project Almanac: a time-traveling solid-state drive [C] // Proceedings of the 14th EuroSys Conference 2019, 2019: 1-16.

[42] Ajdari M, Park P, Kwon D, et al. A scalable HW-based inline deduplication for SSD arrays [J]. IEEE Computer Architecture Letters, 2017, 17(1): 47-50.

[43] Bellare M, Garay JA, Hauser R, et al. Design, implementation, and deployment of the *i*KP secure electronic payment system [J]. IEEE Journal on Selected Areas in Communications, 2000, 18(4): 611-627.

[44] Li HC, Hao MZ, Novakovic S, et al. LeapIO: efficient and portable virtual NVMe storage on ARM SoCs [C] // Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, 2020: 591-605.

[45] Sim H, Kim Y, Vazhkudai SS, et al. AnalyzeThis: an analysis workflow-aware storage system [C] // SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015: 1-12.

[46] Zheng Q, Cranor CD, Jain A, et al. Streaming data reorganization at scale with DeltaFS indexed massive directories [J]. ACM Transactions on Storage, 2020, 16(4): 1-31.

[47] Kim S, Oh H, Park C, et al. Fast, energy efficient scan inside flash memory SSDs [C] // Proceeedings of the International Workshop on Accelerating Data Management Systems, 2011: 36-43.

[48] HeydariGorji A, Torabzadehkashi M, Rezaei S, et al. Stannis: low-power acceleration of DNN training using computational storage devices [C] // Proceedings of the 2020 57th ACM/IEEE Design Automation Conference, 2020: 1-6.

[49] Mailthody VS, Qureshi Z, Liang WX, et al. DeepStore: in-storage acceleration for intelligent queries [C] // Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019: 224-238.

[50] Jeong WS, Lee C, Kim K, et al. REACT: scalable and high-performance regular expression pattern matching accelerator for in-storage processing [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(5): 1137-1151.

[51] István Z, Sidler D, Alonso G. Caribou: intelligent distributed storage [J]. Proceedings of the VLDB Endowment, 2017, 10(11): 1202-1213.

[52] HeydariGorji A, Rezaei S, Torabzadehkashi M, et al. Hypertune: dynamic hyperparameter tuning for efficient distribution of DNN training over heterogeneous systems [C] // Proceedings of the 2020 IEEE/ACM International Conference On Computer Aided Design, 2020: 1-8.

[53] Jun S W, Wright A, Zhang SZ, et al. GraFBoost: using accelerated flash storage for external graph analytics [C] // Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, 2018: 411-424.

[54] Park D, Wang JG, Kee YS. In-storage computing for Hadoop MapReduce framework: challenges and possibilities [J]. IEEE Transactions on Computers, 2016, 3(1): 1-14.

[55] Kang Y, Kee Y, Miller EL, et al. Enabling cost-effective data processing with Smart SSD [C] // Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies, 2013: 1-12.

[56] Duan M, Liu D, Chen XZ, et al. Self-balancing federated learning with global imbalanced data in

mobile systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(1): 59-71.

[57] Li HC, Hao MZ, Tong MH, et al. The CASE of FEMU: cheap, accurate, scalable and extensible flash emulator [C] // Proceedings of the 16th USENIX Conference on File and Storage Technologies, 2018: 83-90.

[58] Gouk D, Kwon M, Zhang J, et al. Amber*: enabling precise full-system simulation with detailed modeling of all SSD resources [C] // Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, 2018: 469-481.

[59] Jung M, Zhang J, Abulila A, et al. SimpleSSD: modeling solid state drives for holistic system simulation [J]. IEEE Computer Architecture Letters, 2017, 17(1): 37-41.

[60] Hu Y, Jiang H, Feng D, et al. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance [J]. IEEE Transactions on Computers, 2012, 62(6): 1141-1155.

[61] Kwak J, Lee S, Park K, et al. Cosmos＋ OpenSSD: rapid prototype for flash storage systems [J]. ACM Transactions on Storage, 2020, 16(3): 1-35.

[62] Picoli IL, Pasco CV, Jónsson BÞ, et al. uFLIP-OC: understanding flash I/O patterns on open-channel solid-state drives [C] // Proceedings of the 8th Asia-Pacific Workshop on Systems, 2017: 1-7.

[63] Do J, Lomet D, Picoli IL. Improving CPU I/O performance via SSD controller FTL support for batched writes [C] // Proceedings of the 15th International Workshop on Data Management on New Hardware, 2019: 1-8.

[64] Arm Limited. A guide to computational storage on arm [R]. Arm Ltd, 2020. https://www.arm.com/solutions/storage/computational-storage.

[65] Molgaard J, lzhar A, Bates S, et al. Computational storage architecture and programming model [R]. Working Draft Version 0.8 rev 0, SNIA, 2021.

[66] Zhang J, Kwon M, Swift M, et al. Scalable parallel flash firmware for many-core architectures [C] // Proceedings of the 18th USENIX Conference on File and Storage Technologies, 2020: 121-136.

[67] Tavakkol A, Gómez-Luna J, Sadrosadati M, et al. MQSim: a framework for enabling realistic studies of modern multi-queue SSD devices [C] // Proceedings of the 16th USENIX Conference on File and Storage Technologies, 2018: 49-66.

[68] Jung M. OpenExpress: fully hardware automated open research framework for future fast NVMe devices [C] // Proceedings of the 2020 USENIX Annual Technical Conference, 2020: 649-656.

[69] Zheng N, Chen XB, Li JP, et al. Re-think data management software design upon the arrival of storage hardware with built-in transparent compression [C] // Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems, 2020: 20-26.

[70] Salamat S, Aboutalebi AH, Khaleghi B, et al. NASCENT: near-storage acceleration of database sort on SmartSSD [C] // Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021: 262-272.

[71] Arm Limited. Powerful real-time processor for smart storage applications [EB/OL]. (2020-09-04)[2022-03-30]. https://www.arm.com/products/silicon-ip-cpu/cortex-r/cortex-r82.

[72] Zhu F, Li L, Sun H, et al. In-storage computing SSD specifications and applications [C] // Proceedings of the Flash Memory Summit, 2019.

[73] Klimovic A, Wang YW, Stuedi P, et al. Pocket: elastic ephemeral storage for serverless analytics [C] // Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, 2018: 427-444.

[74] Miao H, Li A, Davis LS, et al. Towards unified data

and lifecycle management for deep learning [C] // Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering, 2017: 571-582.

[75] Yoon J, Jeong WS, Ro WW. Check-in: in-storage checkpointing for key-value store system leveraging flash-based SSDs [C] // Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020: 693-706.

[76] Schelter S, Boese JH, Kirschnick J, et al. Automatically tracking metadata and provenance of machine learning experiments [C] // Proceedings of the Machine Learning Systems Workshop at NIPS, 2017: 27-29.

[77] Vietri G, Rodriguez LV, Martinez WA, et al. Driving cache replacement with ML-based LeCaR [C] // Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems, 2018: 928-936.

[78] Liao SW, Hung TH, Nguyen D, et al. Machine learning-based prefetch optimization for data center applications [C] // Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009: 1-10.

[79] Liu RP, Liu D, Chen XZ, et al. Self-adapting channel allocation for multiple tenants sharing SSD devices [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 41: 294-305.

[80] Garcia CE, Camana MR, Koo I. Machine learning-based scheme for multi-class fault detection in turbine engine disks [J]. ICT Express, 2021, 7(1): 15-22.