数据存储架构与技术

舒继武　编著

人 民 邮 电 出 版 社

北　京

内 容 提 要

本书介绍数据存储架构与技术，涵盖了存储盘与存储介质、存储阵列、存储协议、键值存储、文件系统、网络存储体系结构、分布式存储系统、存储可靠性、存储安全、数据保护等基础内容，对存储维护、存储解决方案、存储技术趋势与发展等主题进行了深入讨论，并提供目前最新的研究成果作为案例，帮助读者加深对数据存储的理解与运用。

本书内容丰富，适合计算机及相关专业高年级本科生或研究生阅读和学习，同时也可供专业技术人员参考。

序一

随着数据量的爆炸式增长，存储技术越来越受到重视。构建高性能存储系统，快速、完整且长久地保存海量数据，关系到国计民生和国家战略安全。

自 20 世纪 90 年代起，我们在国内率先开展网络存储系统关键技术研究。当时存储系统的关键技术主要被欧美机构垄断，但通过整个团队的不懈努力，我们逐一攻破存储系统的可扩展性、可靠性、存储性能等一系列难题，构建了一个个原型系统，并应用和转化了相关关键技术。令我欣慰的是，经过 20 余年的发展，目前中国在存储领域已处于国际领先行列：在存储硬件系统上，华为等中国企业可以提供业界领先、全套、面向不同场景的高性能存储解决方案；在存储软件系统上，依托于大型数据中心和 E 级超算中心，清华存储团队构建了若干超大规模的、可扩展的分布式存储软件系统；在科研成果上，中国高校能够持续在存储领域的国际顶级会议发表学术成果。但我一直比较遗憾的是，我们缺少一本全面介绍阐述存储基本原理和最新研究进展的中文书籍。因此，当舒继武老师和我说想写一本关于数据存储技术的书时，我非常支持。

本书有两个特点我觉得非常好。第一，本书兼顾了存储硬件和存储软件，现如今各种存储硬件层出不穷，比如 NVMe SSD、持久性内存等，它们具有迥异的特性，只有充分了解这些硬件特性，我们才能够构建出高性能的存储软件系统，而本书便对存储软硬件之间的协同进行了深度阐释。第二，本书兼顾了学术界和工业界的视角，一方面，编著者舒继武老师在存储领域有二十多年的科研经验，也是率先在国内进行网络存储技术教学的老师，能够准确把握存储领域的发展轨迹和最新趋势；另一方面，这本书也包含了华为公司同仁们在工业界多年的经验与思考，例如如何进行存储维护，如何为不同应用场景提供高效的存储解决方案等。

希望本书能够帮助读者全面且深入地了解数据存储的关键技术，并能吸引一批优秀的人才从事存储行业，为中国的存储事业添砖加瓦！

中国工程院院士、清华大学教授　郑纬民

序二

呼唤存储专业人才，共筑数据基础设施

人类社会经过了从农业时代到工业时代的发展，目前正在大踏步进入数字化时代。数字化已经深入到金融、电信、制造、交通、能源、教育、医疗等社会、经济、生活的方方面面，我们正在迎来一个万物互联的智能世界。

数据是智能世界的基础性资源和战略性资源。华为预测，到 2030 年，人类将进入 YB（尧字节，尧代表 10^{24}）数据时代。数据的采集、传输、保存、计算、使用，离不开强大的数据基础设施。专业存储承载着千行百业的研发、生产、经营的高价值数据，是数据基础设施的核心组成部分。

存储在数字化时代具有举足轻重的作用。存储容量决定了智能世界的数据总量，企业拥有的数据多，企业的数字化转型就具备了良好的基础。存储性能决定了数据存取的效率，存储、网络、计算三者的性能完全匹配，数据才能真正发挥实时的价值。存储可靠性决定了数据的持久性和可用性，一旦数据丢失或无法访问，将给数字社会、数字经济的运行带来无法挽回的损失。存储安全更是数据安全的最后一道防线，识别、保护关键数据，检测、响应不可预计的攻击并恢复关键数据，构成了完整的数据安全体系架构。

从全球第一台专业存储设备诞生，存储产业一直活跃在创新的前沿。从数据库存储到文件存储，从存储阵列到分布式存储，从机械硬盘存储到全闪存存储，存储的应用场景和系统架构日益丰富多样，有力促进了人类社会数字化水平的提升。伴随着大数据、人工智能、分布式数据库等新兴数据应用的蓬勃发展，新计算、新网络、新介质技术不断涌现，数据中心从枢纽、城市到边缘全面布局，绿色低碳共识不断深入，存储产业正在迎来新一轮的创新浪潮。

数十年来，围绕产、学、研、用的全产业链，一批又一批卓越的存储专业人才成长起来。理论、架构、工程技术、行业场景等领域中学者和专家们的完美结

合，推动存储产业不断向前发展。未来，数据量呈现爆发式增长，技术创新层出不穷，智能世界正在呼唤越来越多的专业人才，认识存储、发展存储。

本书从组件、系统架构到关键技术，从应用场景到设计实现，结合理论知识和工程实践，系统地讲解了存储的专业知识和技术展望，非常适合有志于投身存储产业浪潮的学生、科研人员、企业研发人员、企业 IT 团队以及其他从业人员学习参考。

华为是全球领先的 ICT 基础设施供应商。今天和未来，我们致力于把数字世界，带入每个人、每个家庭、每个组织，构建万物互联的智能世界。华为数据存储始终坚持技术创新，坚持服务全球市场客户。感谢国际权威分析机构高德纳（Gartner）对华为存储市场、产品、销售、客户体验战略和执行的认可，2016 年至今，华为一直是高德纳主存储魔力四象限领导者。华为愿意为学生提供学习、实践、工作的机会，和科研人员、用户联合创新，共同探索未来的存储应用场景和技术，和广大的存储产业从业者营造健康的合作生态，为构建可靠的数据基础设施尽自己最大的努力。

专业存储，海纳数据，释放平凡数据的不凡潜力。让我们携手努力，共创美好数据时代的未来。

华为常务董事、ICT 基础设施业务管理委员会主任 汪涛

前 言

数据是当前信息技术发展的核心资产，需要被高效并可靠地存储起来，以服务众多现实生活中的应用，如互联网、大数据、人工智能及高性能计算等。鉴于数据存储的重要性，从 2003 年开始，我们在清华大学开设了面向计算机专业研究生的“网络存储技术”课程，介绍数据存储领域的基础知识及最新的学术动态。在这么多年的教学过程中，我们发现没有合适的图书可以作为课程参考书：一方面，大部分计算机系统类图书仅将数据存储作为其中的子章节，对数据存储介绍得不够全面系统；另一方面，专门介绍数据存储的图书内容相对陈旧，无法反映当前存储领域在硬件和软件方面的最新进展。因此，我们萌生了编写一本数据存储技术图书的想法。就在这个想法产生不久后，华为数据存储产品线的同事找到了我们，他们也有相似的需求：编写一本数据存储技术的图书，向员工和客户普及数据存储的基本概念和技术。在此契机下，我们与华为的同事一起，编写了这本《数据存储架构与技术》，本书既包含了我们团队在数据存储领域多年的教学和科研经验，也包含了以华为为代表的存储产业界力量在存储维护、存储解决方案等方面的先进思考。

本书共 14 章，内容概述如下。

第 1 章　数据存储的背景

这一章讲述数据存储的背景，包括数据存储在当今信息时代的重要性及一些主要的性能指标。

第 2 章　存储盘与存储介质

在计算机中，数据被存储在不同的存储设备中，这一章主要介绍存储盘与存储介质，包括磁盘、SSD（Solid State Disk，固态硬盘）及主存。针对每类存储盘与存储介质，这一章简述其发展历史、组成结构及性能特征。为了充分发挥硬件的性能，存储盘一般包含复杂的固件设计，比如地址映射、磁盘缓存及磁盘调度等。此外，这一章还简要介绍光存储、磁带等其他存储介质。

第 3 章　存储阵列

单独的存储设备无法满足应用的容量和可靠性诉求，因此需要将多块存储设

备进行集中池化管理，即构成存储阵列。这一章首先介绍存储阵列的硬件架构，包括控制器模块、接口模块等。然后介绍其软件架构并着重介绍 RAID（Redundant Arrays of Independent Disks，独立磁盘冗余阵列）算法。最后，我们介绍如何设计存储阵列，以达到高性能和高可靠的目标。

第 4 章　存储协议

存储协议负责主机与存储设备之间的通信及数据交互。这一章介绍典型的存储协议，包括面向磁盘的 SCSI（Small Computer System Interface，小型计算机系统接口）协议，以及面向 SSD 的 NVMe（Non-Volatile Memory express，非易失性快速存储器）协议。此外，还介绍新型的内存互连协议——CXL（Compute Express Link，缓存一致性互连）协议。

第 5 章　键值存储

现实中的大量数据都可以通过键值对的映射方式来表达，因此键值存储系统应用广泛。这一章介绍键值存储系统常用的索引结构，如散列表、B+树及 LSM 树等，并介绍键值存储系统如何进行数据的布局。此外，这一章还探讨崩溃一致性机制，这类机制使得键值存储系统在崩溃后能恢复到一致的状态。

第 6 章　文件系统

除了键值存储系统，另一种常见的存储软件系统是文件系统。这一章讲述文件系统的基本操作，包括文件操作和目录操作。这一章通过案例逐步剖析文件系统的关键设计模块，如命名空间管理、缓存与一致性等，从而使读者深刻理解文件系统的运作原理。

第 7 章　网络存储体系结构

相比于存储阵列，网络存储系统在扩展性、稳定性及共享访问等方面具有优势。这一章介绍网络存储系统体系结构的发展历程：从最初的直连式存储到集中式网络存储（例如网络附属存储和存储区域网络），再到并行存储、P2P 存储及云存储。此外还介绍存储虚拟化、软件定义存储、超融合架构等前沿技术。

第 8 章　分布式存储系统

分布式存储系统将数据分散存储在多台服务器中，以应对持续增长的数据量。这一章首先介绍分布式存储系统典型架构及其关键衡量指标，然后分别介绍分布式键值存储系统、分布式对象存储系统、分布式块存储系统及分布式文件系统。对于每一类分布式存储系统，我们通过介绍系统实例使读者熟悉其中

的关键技术。

第 9 章　存储可靠性

高可靠性是数据存储的核心诉求之一，这一章首先介绍存储可靠性的基本概念，然后阐述硬盘可靠性和闪存介质可靠性问题，随后分析纠删码技术原理及发展趋势，最后介绍分布式存储系统的可靠性问题。

第 10 章　存储安全

存储安全事故会导致用户隐私数据泄露甚至彻底丢失。这一章首先介绍存储安全的理念和安全体系，然后介绍存储安全的关键技术，包括系统安全、数据安全及安全管理等。此外，这一章还介绍最新的硬件安全技术，比如 TEE（Trusted Execution Environment，可信执行环境）技术。

第 11 章　数据保护

这一章介绍如何对数据进行保护，包括镜像、快照及克隆等技术。此外，我们还介绍了数据保护的三大类场景，即备份、归档和容灾，在这些场景中，数据保护具有不同的作用和定位。

第 12 章　存储维护

存储系统的复杂度日益提高，维护难度也随之增长。这一章介绍存储维护中的预防性维护和纠正性维护，其中前者用于预防存储系统发生故障，而后者用于修复已发生的故障。

第 13 章　存储解决方案

这一章介绍不同场景的存储解决方案，包括运营商、政务融合场景、金融行业、医疗行业及教育行业等，读者能够通过场景实例了解如何根据不同需求设计高效的数据存储方案。

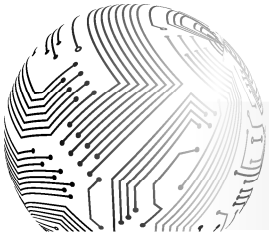
第 14 章　存储技术趋势与发展

这一章介绍存储技术趋势与发展，分别介绍了存内计算、持久性内存、在网存储、智能存储、边缘存储、区块链存储、分离式数据中心架构、高密度新型存储等前沿技术。

致 谢

很荣幸在此感谢一起完成本书的人：清华大学存储系统团队的陆游游副教授，陈游旻博士后、张余豪博士后，汪庆、程卓、吕文豪、颜彬、林家桢、冯杨洋、李俊儒、高健、李泽祺、范如文等博士研究生，厦门大学沈志荣副教授、李乔副教授、高聪明副教授等；以及华为技术有限公司的周跃峰、庞鑫、张福鹏、张国彬、王振、丁志彬、梁佳妮、杨天文、董伟、顾学虎、张南东、王升、闫鹏、李兆男、李国杰、覃国、杨俊涛、廖志坚、晏大洪、潘浩、王鹏、李楚、章鹏、饶成莉、曹长斌、仇幼成、王伟、黎超、李强、张颖、徐旭东、陈卫屏、曾洋洋、刘健、夏冰心、杜翔、曾红丽、陈克云、周希锋、周晓峰、袁琦钊、唐国辉、卢建刚、何杰、戴维、黄蓉、孙林、祁晨睿等。

目 录



第1章 数据存储的背景

数据存储是将信息保存在某种介质上供后续读取的技术。早在远古时代，人类在还没发明文字的时候就通过在绳子上打结来存储数据，即《周易》中所记载的“上古结绳而治”。随着人类社会的进步，数据开始以文字的形式存储在甲骨、竹筒、纸张等介质上。到了当代，信息技术的爆炸式发展催生了各种用于存储数据的硬件设备，如磁带、磁盘等；同时，存储数据的工具也转变为计算机，通过运行在计算机上的存储软件系统对存储硬件设备上的数据进行管理与访问。

如今的人类社会在以前所未有的速度制造数据，预测在 2025 年其总量将达到 181 ZB，需要约 2000 亿块 1 TB 的磁盘进行存储。面对如此海量的数据，存储扮演着至关重要的角色。在这一章，我们将介绍数据存储在信息时代的重要性，以及它的一些主要目标。

1.1 数据存储的重要性

对于个人而言，数据存储的重要性是不言而喻的：我们将具有纪念价值的珍贵照片、视频等数据存储在手机、笔记本计算机等便携设备中，并上传至云盘等高可靠存储系统，用于长期保存数据。此外，数据存储作为信息时代的基石，对经济社会及其他领域的发展具有重要作用。

数据存储促进经济发展。从直接经济效益来看，权威咨询公司高德纳（Gartner）预测 2025 年全球存储市场规模将达到 1500 亿美元。而数据存储产生的间接经济效益更是无法估计，几乎所有的行业都离不开数据存储的支撑：对于电商行业，在“双十一”购物节等重要活动期间，需要存储系统能够快速、可靠、

稳定地处理与交易相关的数据访问，为用户提供良好的使用体验，保证系统在每秒交易数维持在较高水平情况下的稳定性；对于石油勘探行业，需要高性能的存储系统处理模拟油藏产生的海量数据，缩短勘探周期，提高开采效率。

数据存储提高社会治理水平。国务院在 2015 年发布的《促进大数据发展行动纲要》中明确指出："数据已成为国家基础性战略资源，大数据正日益对全球生产、流通、分配、消费活动以及经济运行机制、社会生活方式和国家治理能力产生重要影响。"大数据通过分析数据之间的关系，能有效辅助处理复杂的社会问题，例如根据公共卫生的数据预测瘟疫的传播情况。而大数据的核心就是对数据进行采集、存储和分析，其中存储处于承上启下的关键位置，采集的数据需要及时地保存在存储系统中，并能够高效地服务后续的计算分析。

数据存储助力其他领域的突破。强大的数据存储能力是当今众多学科取得突破的关键，正如李国杰院士指出："海量数据的出现催生了一种新的科研模式，即面对海量数据，科研人员只需要从数据中直接找到或挖掘所需要的信息、知识和智慧，甚至不需要直接接触研究的对象"。例如天文学界在 2019 年发布了人类有史以来第一张黑洞照片，这对验证黑洞相关理论具有重要意义，而这张照片的拍摄离不开数据存储的功劳：射电望远镜捕获与黑洞相关的 5 PB 数据被实时存储在高性能的充氦硬盘上。

1.2 数据存储的目标

正如 1.1 节所述，数据存储具有极高的重要性，因此一代又一代的人对其进行持续的创新设计，以达到高性能、高易用性、高可靠性等目标。本节将简单介绍这些目标以及数据存储的相关软硬件设计，这些内容也将贯穿整本书的其他章节。

1.2.1 高性能

数据存储的性能指标主要包括吞吐率与延迟，其中吞吐率指单位时间内数据存储系统能完成的操作数目，而延迟是指完成单个操作所需要的时间。高性能数据存储追求更高的吞吐率与更低的延迟，这是由持续增长的上层应用需求所驱动的。例如，"双十一"购物节的成交额逐年提升，这就要求存储系统的吞吐率匹配其上升的趋势；在延迟方面，用户越来越关注服务质量，因此存储系统要尽快地完成每一个数据访问操作，最小化用户所能感知的延迟。

为了达到高性能指标，需要对存储硬件和存储软件共同进行设计。如图 1.1

所示，现有的存储硬件种类多样，在性能、容量和价格方面各有优劣，例如 HDD（Hard Disk Drive，硬盘驱动器）的延迟为 10 ms 左右，而 SSD 的延迟仅有 10～100 μs。因此，提升存储系统性能最自然的方法就是采用更快速的存储硬件。但这面临着两个关键问题：首先，存储硬件的性能越好，通常它的成本会越高，导致整体系统价格昂贵，难以普及使用；其次，存储硬件的吞吐率越大，一般其容量会越小，这将限制整体系统能容纳的数据量，难以满足持续增长的海量数据的需求。因此，高性能目标的达成还需要存储软件的设计。

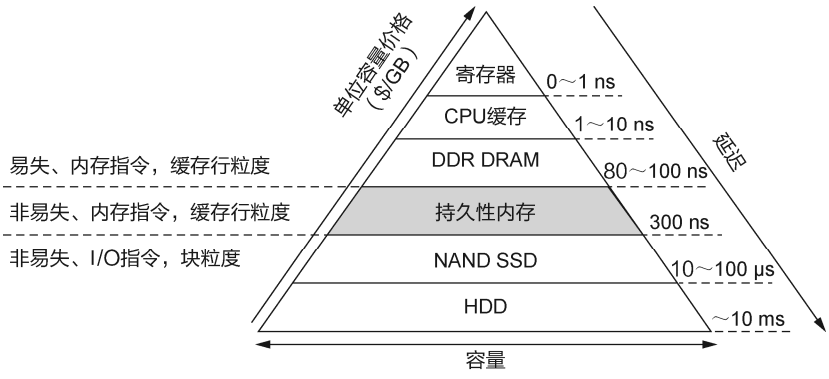


图 1.1　存储金字塔

由于单个存储硬件的性能有限，存储软件经常将大量存储硬件组合起来，以提供更高的吞吐率。最早的技术包括美国加利福尼亚大学伯克利分校提出的 RAID（Redundant Array of Independent Disks，独立磁盘冗余阵列）技术[1]。其中 RAID 0 将数据分散至多块磁盘，同时发挥多块磁盘的性能。除此之外，分布式存储系统也被普遍使用，该存储系统通过网络将多台存储服务器互连，向外部提供极高的聚合吞吐率，例如图 1.2 展示了我国超级计算机“神威・太湖之光”的分布式文件系统架构，它由 80 个 I/O 转发节点、144 个存储节点，以及 72 个磁盘阵列构成，每个磁盘阵列包含 60 块磁盘。

提升性能还可以通过充分利用数据局部性原理设计缓存、预取等机制。数据局部性包括时间局部性和空间局部性：时间局部性是指当某份数据被访问后，在不久后将被再次访问，例如在线购物平台上某些热门商品对应的数据会被频繁地读写；空间局部性是指当某份数据被访问后，其相邻的数据也将在不久后被访问，例如当某个文件被访问后，和它处于同一个文件夹中的其余文件也很有可能被访问。利用时间局部性，存储软件在各个层级设计高效的缓存机制，将经常被访问的数据放置在更快速的系统组件中，由此能同时提升吞吐率和降低延迟。例如，

在单个存储服务器中，Linux 文件系统将文件数据以页缓存（Page Cache）的形式缓存在内存中，并通过 LRU（Least Recently Used，最近最少使用）算法等替换算法提高缓存的命中率；在数据中心中，经常存在由 Memcached 等软件构建的分布式缓存集群，用于作为后端基于磁盘的存储系统的读缓存。利用空间局部性，存储软件也在各个层级设计了预取机制，即将未来可能要访问的数据提前从低速组件调换至高速组件。例如 Linux 文件系统采用了 readahead 预取算法，当文件被顺序访问时，会将后面的内容提前调至页缓存中。

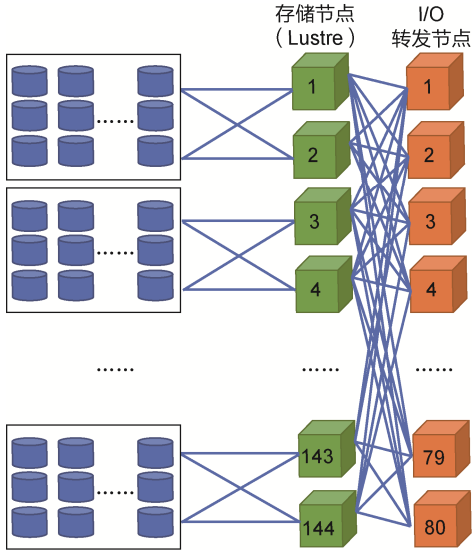


图 1.2 “神威·太湖之光”的分布式文件系统架构[2]

1.2.2 高易用性

数据存储另一大目标就是高易用性，即上层应用应该方便地与存储系统“打交道”，将数据以一种便捷的方式写入存储系统，并支持后续的高效读取。为了提供高易用性，人们设计了不同的数据存储接口，如图 1.3 所示。其中语义最简单的是块接口，即整个存储空间被抽象成大小相等的数据块，每份数据块具有唯一的数字标识符，应用通过数字标识符读写对应的数据块。块接口难以表达大部分应用的数据语义，因此通常作为其他存储接口的底座，而不直接暴露给上层应用。键值接口的易用性高于块接口：它维护键值对的集合，每份键值对包括键和值两部分，均是长度可变的数据块，应用通过键来定位至对应的键值对，进行插入、更新、查询以及删除等操作。由于语义清晰、用法灵活，键值接口被广泛使用，例如亚马逊将在线购物相关的数据存储在键值系统 Dynamo[3]中。文件接口将数据

组织成目录树结构，主要包括目录文件和普通文件，应用可以创建、删除和读写文件。其中普通文件存储着文件数据，目录文件记录着该目录下的普通文件以及其他目录文件的名字，每个文件具有权限等属性，用于访问控制。文件系统这种层次化结构十分适合普通的计算机用户，因此几乎所有的桌面操作系统都自带文件系统，为用户提供管理数据的易用方式。此外，为了进一步方便用户使用，还存在针对键值接口和文件接口的扩展，比如有的键值存储系统支持通过多个键查询同一份数据，即二级索引功能；有的文件系统支持事务语义，即保证多个文件操作的原子性。

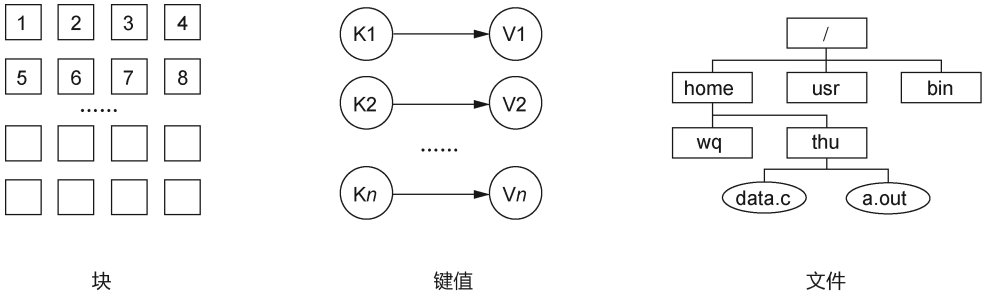


图 1.3　3 种常见的数据存储接口

除了存储接口方面，支持数据共享也是易用性的关键。存储阵列虽然解决了容量和性能的限制，但无法进行数据共享，即位于不同位置的用户无法访问同一份数据。随着网络技术和存储协议的发展，数据存储开始朝网络化共享的方向迈进。例如 NAS（Network Attached Storage，网络附接存储）支持通过局域网共享文件。此外，更大规模的分布式存储系统也在追求极致的共享能力，例如，Facebook 的分布式文件系统 Tectonic[4]能够提供数据中心级别的数据存储功能，同时支持不同应用，达到了极高的资源利用率。

1.2.3　高可靠性

数据存储的高可靠性是指在系统面对异常情况时，数据不丢失且服务不中断。异常情况有很多种，包括磁盘失效、服务器崩溃、网络故障及人为事故等，根据 Facebook 的报告，在一个拥有 3000 台服务器的生产集群中，一天最多会有 110 台服务器崩溃。若数据存储的可靠性没得到保证，会造成严重的后果，例如，2017 年亚马逊的 S3 存储系统发生了持续 5 小时的故障，影响数十万个网站的正常访问，造成约 1 亿 5000 万美元的损失。

实现高可靠性的基本方式是提供数据冗余，主要包括多副本和纠删码两种机

制。如图 1.4 所示，在多副本机制中，每份数据被重复存储在多个不同位置，因此当某个位置的数据由于某种异常事件无法被访问时，其余位置的数据仍然能够提供服务；而在纠删码机制中，系统将多份数据通过某种编码方式计算出若干份校验块，并将这些数据与相关校验块存储在不同位置；当无法访问某份数据时，其内容可通过校验块和其余数据重新计算获得。相比于多副本机制，纠删码机制具有更低的存储空间开销，但计算和恢复开销更高。对于一个成熟的存储系统，在每个层级都具有相应的可靠性保障：在存储设备层，利用 LDPC（Low-Density Parity-check Code，低密度奇偶校验码）等纠错机制检测数据是否出现错误并尝试修复数据；在服务器层，基于 RAID 机制的磁盘阵列机制可以容忍某个存储设备的失效；在集群层，通过分布式多副本机制或分布式纠删码处理服务器的异常事件。此外，数据还有可能被周期性地备份和归档。

图 1.4　多副本与纠删码

尽管上述机制从理论上能保证数据存储的可靠性，但存储软件是由程序员通过代码编写而成的，若代码中存在漏洞，仍然会造成数据内容发生错误。例如存储软件并发访问处理不当会导致数据内容混乱。因此，为了获得高可靠性，存储软件还需要进行大量的正确性测试，甚至采用一些形式化的手段。亚马逊构建新一代对象云存储系统时，运用了轻量级的形式化方法去验证存储软件在接口语义、数据一致性、并发访问正确性等多个方面是否符合预期[5]。

1.2.4　其他目标

除了高性能、高易用性和高可靠性，数据存储还需满足如下的目标。

高性价比。数据存储追求极致的性价比，希望在更低的单位价格下提供更高的性能。为了达到高性价比，常用的做法是进行存储分级，即将热数据保存在高速存储设备上（如 SSD），而将冷数据保存在低速存储设备上（如磁带）。

高安全性。数据存储还需要考虑安全性，以防止用户隐私数据被泄露等严重事件的发生。为了获得高安全性，常用的做法包括数据加密、访问控制、隐私计算等。

高可扩展性。正如前文所述，分布式存储系统通过组合多台服务器，实现高性能数据访问。设计分布式存储系统的重要目标之一就是高可扩展性，即当加入新的服务器时，系统整体性能能够按相应比例提升。为了达到高可扩展性，存储软件需避免某些服务器成为性能瓶颈，常用的做法包括配置负载均衡机制、数据动态迁移机制等。

参考文献

[1] PATTERSON D A, GIBSON G, KATZ R H. A case for redundant arrays of inexpensive disks (RAID)[R/OL]. (1987-12)[2023-04-05].

[2] LIN H, ZHU X, YU B, et al. Shentu: processing multi-trillion edge graphs on millions of cores in seconds[C]//SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2018: 706-716.

[3] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: Amazon's highly available key-value store[J]. ACM SIGOPS operating systems review, 2007, 41(6): 205-220.

[4] PAN S, STAVRINOS T, ZHANG Y, et al. Facebook's tectonic filesystem: Efficiency from exascale[C]//19th USENIX Conference on File and Storage Technologies (FAST 21). 2021: 217-231.

[5] BORNHOLT J, JOSHI R, ASTRAUSKAS V, et al. Using lightweight formal methods to validate a key-value storage node in Amazon S3[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021: 836-850.

参考文献

[1] RINO M, CRIPPA L, MARELLI A. Inside NAND flash memories[M]. Dordrecht: Springer , 2010.

[2] 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.

参考文献

[1] MAIER D, VANCE B. A call to order[C]//BEERI C. In Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '93). New York: Association for Computing Machinery, 1993:1-16.

[2] WILKES J, GOLDING R, STAELIN C, et al. The HP AutoRAID hierarchical storage system[J]. ACM Transactions on Computer Systems. 1996, 14(1):108-136.

[3] CHEN P, LEE E, GIBSON G, et al. RAID: high-performance, reliable secondary storage[J]. ACM Computing Surveys, 1994, 26(2):145-185.

[4] KATZ R, CHEN P. RAID-II: Design and implementation of a large scale disk array controller[R/OL]. (1992-01-01)[2023-04-10].

[5] DENNING P. The locality principle[J]. Communications of the ACM, 2005, 48(7):19-24.

在处理器（主机）可以缓存设备数据的情况下，访问该类设备的流程大致为：① 访存（读或写等）指令未能命中处理各级缓存（Cache）；② 对于读类型指令，需要翻译为读请求，对于写类型指令，仍然需要翻译为读请求，将缓存行读取至 CPU 缓存，在 CPU 内部完成写修改；③ 读请求经由内存管理单元查询，发现请求目的地址处于 CXL 总线地址空间，后翻译读请求并转发至 CXL 总线控制器；④ 由 CXL 总线控制器生成 CXL 访存事务，并由 CXL 总线将事务发送至设备端 CXL 总线控制器；⑤ 在设备端，由设备内部的访存事务翻译、地址映射、DRAM 控制器（若为持久性内存池，则是 PM 控制器）等模块完成数据的读取；⑥设备 CXL 总线控制器返回完成信号读取得到的缓存行数据。写请求的流程大致相同。

4.6　本章小结

存储协议定义了应用程序、主机系统与存储设备之间的交互规则。本章针对存储协议的发展进行详细地阐述，分别介绍了 SCSI 协议、SCSI 链路承载协议、NVMe 协议、NVMe over Fabrics 及最新的内存互连协议。这些关键的存储协议满足了计算机存储系统发展过程中用户对数据访问存储的性能和可靠性等方面的需求，充分发挥了计算机新型硬件的性能特性。随着新兴的存储设备和存储形态的出现，当前的存储协议正在不断发展迭代，以内存互连协议为代表的新型存储协议正在快速发展，如何将传统存储协议与新型存储架构相融合，进一步支持计算机存储系统的快速发展将成为存储协议发展过程中的重要驱动力。

参考文献

[1] WORDEN D J. Storage Networks [M]. Berkeley: Apress, 2004.

[2] KHATTAR R K, MURPHY M S, TARELLA G J, et al. Introduction to Storage Area Network, SAN[R/OL]. (1999-08)[2023-04-10].

[3] ZHANG F. High-speed Serial Buses in Embedded Systems[M]. Singapore: Springer, 2020.

[4] HUFFERD J L. iSCSI: The universal storage connection[M]. Boston: Addison-Wesley Professional, 2003.

[5] HUNG J J, BU K, SUN Z L, et al. PCI express-based NVMe solid state disk[J]. Applied Mechanics and Materials, 2014, (464): 365-368.

[6] XU Q, SIYAMWALA H, GHOSH M, et al. Performance analysis of NVMe SSDs and their implication on real world databases[C]//Proceedings of the 8th ACM International Systems and Storage Conference. Haifa: ACM, 2015: 1-11.

[7] GUZ Z, LI H, SHAYESTEH A, et al. Performance characterization of nvme-over-fabrics storage disaggregation[J]. ACM Transactions on Storage, 2018, 14(4): 1-18.

[8] GUZ Z, LI H, SHAYESTEH A, et al. NVMe-over-fabrics performance characterization and the path to low-overhead flash disaggregation[C]//ACM. 10th ACM International Systems and Storage Conference. Haifa: ACM, 2017: 1-9.

[9] SHAN Y, HUANG Y, CHEN Y, et al. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation[C]//USENIX, 13th USENIX Symposium on Operating Systems Design and Implementation. CARLSBAD: ACM SIGOPS. 2018: 69-87.

[10] LIM K, CHANG J, MUDGE T, et al. Disaggregated memory for expansion and sharing in blade servers[J]. ACM SIGARCH computer architecture news, 2009, 37(3): 267-278.

[11] GU J, LEE Y, ZHANG Y, et al. Efficient memory disaggregation with infiniswap[C]// USENIX, 14th USENIX Symposium on Networked Systems Design and Implementation. Boston: ACM SIGOPS. 2017: 649-667.

[12] AMARO E, BRANNER-AUGMON C, LUO Z, et al. Can far memory improve job throughput? [C]//ACM. 15th European Conference on Computer Systems. New York: ACM. 2020: 1-16.

参考文献

[1] PAGH R, RODLER F F. (2001). Cuckoo Hashing[C]//AUF DER HEIDE FM. Algorithms — ESA 2001. Heidelberg: Springer, 2001, vol 2161.

[2] DONG S, KRYCZKA A, JIN Y, et al. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The {RocksDB} Experience[C]// USENIX Association. 19th USENIX Conference on File and Storage Technologies (FAST 21), 2021:33-49.

[3] O'NEIL P, CHENG E, GAWLICK D, et al. The log-structured merge-tree (LSM-tree)[J]. Acta Informatica, 1996, 33(4):351-385.

[4] GORROD A. Btree vs LSM [EB/OL].(2017-08-24)[2023-04-25].

参考文献

[1] 博韦, 西斯特. 深入理解 Linux 内核(第三版)[M]. 陈莉君, 张琼声, 张宏伟, 译. 北京: 中国电力出版社, 2007.

[2] ARPACI-DUSSEAU R, ARPACI-DUSSEAU A. Operating systems: Three easy pieces[EB/OL]. (2020-08-08)[2023-04-10].

参考文献

[1] SHU J, LI B, ZHENG W. Design and Implementation of a SAN System Based on the Fiber Channel Protocol[J]. IEEE Transactions on Computers, 54(4), 2005: 439-448.

[2] ZHANG G, SHU J, XUE W, et al. Design and Implementation of an Out-of-Band Virtualization System for Large SANs[J]. IEEE Transactions on Computers, 2007, 56(12), 1654-1665.

[3] WEIL S, BRANDT S, MILLER E, et al. Ceph: a scalable, high-performance distributed file system[C]// USENIX Association. Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06). USA: USENIX Association, 2006: 307-320.

[4] REN K, ZHENG Q, PATIL S, et al. IndexFS: scaling file system metadata performance with stateless caching and bulk insertion[C]//IEEE. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14). USA: IEEE Press, 2014: 237-248.

[5] GANESAN P, MANKU G S. Optimal routing in Chord[C]// Society for Industrial and Applied Mathematics. Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '04). USA: Society for Industrial and Applied Mathematics, 2004: 176-185.

[6] YANG Q, HU Y. DCD—Disk Caching Disk: A New Approach for Boosting I/O Performance[C]//ACM. Proceedings of the 23rd annual international symposium on computer architecture. New York: ACM Press, 1996: 169-178.

[7] WILKES J, GOLDING R A, STAELIN C, et al. The HP AutoRAID hierarchical storage system[J]. ACM Transactions on Computer Systems, 1996, 14(1): 108-136.

[8] HSU W W, SMITH A J, YOUNG H C, et al. The automatic improvement of locality in storage systems[J]. ACM Transactions on Computer Systems, 2005, 23(4): 424-473.

[9] BHADKAMKAR M, GUERRA J, USECHE L, et al. BORG: Block-reorGanization and self-optimization in storage systems[R/OL]. (2007-07-01)[2023-04-10].

[10] KHAN O, BURNS R, PLANK J, et al. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads[C]// USENIX Association. Proceedings of the 10th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2012:20.

[11] MENON J, MATTSON D. Distributed sparing in disk arrays[C]//IEEE. Proceedings of the thirty-seventh International Conference on COMPCON. San Francisco: IEEE CS Press, 1992:410-421.

[12] HOLLAND M C. On-line data reconstruction in redundant disk arrays[D]. Pittsburgh: Carnegie Mellon University, 1994.

[13] LEE J Y B, LUI J C S. Automatic recovery from disk failure in continuous-media servers[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(5):499-515.

[14] LUMB C R, SCHINDLER J, GANGER G R, et al. Towards higher disk head utilization: Extracting ”free” bandwidth from busy disk drives[C]// USENIX Association. 4th Symposium on Operating System Design and Implementation. USA: USENIX Association, 2000, 87-102.

[15] TIAN L, FENG D, JIANG H, et al. A popularity- based multi-threaded reconstruction optimization for raid structured storage systems[C]USENIX Association. 5th USENIX Conference on File and Storage Technologies, FAST 2007. San Jose: USENIX Association, 2007:277-290.

[16] WAN J, WANG J, XIE C, et al. S2-RAID: Parallel RAID architecture for fast data recovery[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25 (6), 1638-1647.

[17] WU S, JIANG H, FENG D, et al. Workout: I/O workload outsourcing for boosting RAID reconstruction performance[C]//USENIX Conference. 7th USENIX Conference on File and Storage Technologies. USA: USENIX Association, 2009: 239-252.

[18] GOEL A, SHAHABI C, YUEN DIDI YAO, et al, Scaddar: An efficient randomized technique to reorganize continuous media blocks[C]//IEEE. Proceedings of the 18th International Conference on Data Engineering (ICDE). San Jose: IEEE, 2002: 473-482.

[19] GONZALEZ J, AND CORTES T. Increasing the capacity of raid5 by online gradual assimilation[C]//SNAPI. Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os. New York: Association for Computing Machinery, 2004:17-24.

[20] ZHANG G, SHU J, XUE W, et al. SLAS: an efficient approach to scaling round-robin striped volumes[J]. Transactions on Storage, 2007, 3(1):1-39.

[21] ZHANG G, ZHENG W, SHU J. ALV: A New Data Redistribution Approach to RAID-5 Scaling[J]. IEEE Transactions on Computers, 2010, 59(3): 345-357.

[22] 舒继武, 李思阳, 张广艳. 存储虚拟化研究综述[J]. 中国计算机学会通讯, 2017, 13(6): 14-24.

引文件，如 BetrFS、Giraffa 和 CalvinFS。BetrFS 使用全路径名字符串来索引本地文件系统中的文件元数据。Giraffa 使用全路径名字符串作为文件元数据的主键，将元数据存储到数据库中。CalvinFS 通过对全路径名字符串进行计算定位文件元数据。因此它们在路径解析阶段可以并行发送网络请求来检查每一个中间目录的权限。然而它们使文件系统的层次目录树语义语义难以实现。例如，目录重命名操作的开销巨大，因为它改变了所有后代的全路径名，导致所有后代的元数据必须迁移到新的位置。

8.7　本章小结

分布式存储系统将数据分散存储至大量服务器，通过聚合它们的计算、存储和网络资源，以提供大容量、高性能的存储服务。与本地存储系统不同，分布式存储系统在可扩展性、一致性、可用性方面具有更严格的要求。分布式存储系统的种类繁多，包括分布式键值存储系统、分布式对象存储系统、分布式块存储系统及分布式文件系统等；它们向上层应用暴露不同的访问接口，这些接口的语义也极大影响了对应分布式存储系统的设计，例如分布式文件系统需要考虑如何将元数据划分到不同的服务器，并提供全局统一的目录树抽象。未来，随着应用的发展，新的分布式存储系统将层出不穷，例如辅助人工智能任务的分布式向量存储系统。

参考文献

[1] HERLIHY M P, WING J M. Linearizability: A correctness condition for concurrent objects[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12(3): 463-492.

[2] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: Amazon's highly available key-value store[J]. ACM SIGOPS operating systems review, 2007, 41(6): 205-220.

[3] OUSTERHOUT J, GOPALAN A, GUPTA A, et al. The RAMCloud storage system[J]. ACM Transactions on Computer Systems (TOCS), 2015, 33(3): 1-55.

[4] LAMPORT L. Paxos made simple[J]. ACM SIGACT News (Distributed Computing Column) 32, 2001, 4 (121): 51-58.

[5] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C]// USENIX. 2014 USENIX Annual Technical Conference (Usenix ATC 14). USA: USENIX Association, 2014: 305-319.

[6] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system[J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40.

[7] KARGER D, LEHMAN E, LEIGHTON T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web[C]//ACM. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. New York: Association for Computing Machinery, 1997: 654-663.

[8] WEIL S A, BRANDT S A, MILLER E L, et al. CRUSH: Controlled, scalable, decentralized placement of replicated data[C]//IEEE. SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. New York: Association for Computing Machinery, 2006: 31.

[9] WANG L, ZHANG Y, XU J, et al. {MAPX}: Controlled Data Migration in the Expansion of Decentralized {Object-Based} Storage Systems[C]// USENIX. 18th USENIX Conference on File and Storage Technologies (FAST 20). USA: USENIX Association, 2020: 1-11.

[10] MICKENS J, NIGHTINGALE E B, ELSON J, et al. Blizzard: Fast, cloud-scale block storage for cloud-oblivious applications[C]//USENIX. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). USA: USENIX Association, 2014: 257-273.

[11] NIGHTINGALE E B, ELSON J, FAN J, et al. Flat datacenter storage[C]//USENIX. 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). USA: USENIX Association, 2012: 1-15.

[12] TAK B, TANG C, CHANG R N, et al. Block-level storage caching for hypervisor-based cloud nodes[J]. IEEE Access, 2021, 9: 88724-88736.

[13] LI H, ZHANG Y, LI D, et al. Ursa: Hybrid block storage for cloud-scale virtual disks[C]//ACM. Proceedings of the Fourteenth EuroSys Conference 2019. 2019: 1-17.

[14] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[C]//Proceedings of the nineteenth ACM symposium on Operating systems principles. New York: Association for Computing Machinery, 2003: 29-43.

[15] LV W, LU Y, ZHANG Y, et al. {InfiniFS}: An Efficient Metadata Service for {Large-Scale} Distributed Filesystems[C]//USENIX. 20th USENIX Conference on File and Storage Technologies (FAST 22). USA: USENIX Association, 2022: 313-328.

[16] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[17] LI S, LIU F, SHU J, et al., A Flattened Metadata Service for Distributed File Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(12):2641-2657.

[18] LI S, LU Y, SHU J, et al. LocoFS: A Loosely-Coupled Metadata Service for Distributed File System[C]//ACM. The International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver: Association for Computing Machinery, 2017: 1-12.

数据迁移、数据缓存、数据复制及数据拆分等技术保障系统运行时的负载均衡。

9.6　本章小结

可靠性是存储系统的一个重要指标，它定义了存储系统可以无故障持续运行的概率。存储可靠性问题存在于存储系统的各个层级结构中，包括底层介质、存储设备、单机系统及分布式系统等。不同的存储层次和应用场景对可靠性的需求存在一定的差异，因此，在存储可靠性保障技术设计过程中，需要充分考虑软硬件结构特征和应用访问特征，实现面向存储介质、设备、系统以及用户友好的可靠性保障技术。

本章首先从存储可靠性的基本概念出发，介绍了关于可靠性的关键指标和计算方面，并阐述了可靠性在不同层次的设计理念以及可靠性和可用性之间的区别。随后，针对不同层次的存储可靠性问题，本章详细分析了错误及故障来源，并介绍了关键可靠性优化技术。其中，针对硬盘设备级别的可靠性，介绍了硬盘的基本错误原因、故障预警和检测技术，以及面向不同环境因素的硬盘可靠性设计方法。针对闪存介质可靠性，分析了闪存介质的错误来源和可靠性优化关键技术。针对系统级可靠性，介绍了多副本、纠删码等关键技术，并详细分析了典型的纠删码技术和发展趋势，最后总结了分布式存储系统可靠性的关键优化技术。

可靠性保障技术是存储系统发展过程中的一个重要研究方向，随着新型存储设备和存储架构的发展，存储系统可靠性问题也将面临新的问题和挑战，同时也将为存储系统可靠性保障技术带来新的发展机遇。

参考文献

[1] COUGHLIN T. 175 Zettabytes by 2025[EB/OL]. (2018-11-27)[2023-04-11].

[2] RAZA M. Reliability vs availability: What's the difference?[EB/OL]. (2020-05-13)[2023-04-11].

[3] KADEKODI S, RASHMI K V, GANGER G R. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity[C]//Proceedings of 17th USENIX Conference on File and Storage Technologies (FAST 19). USA: USENIX Association, 2019: 345-358.

[4] WU S, LAN S, ZHOU J, et al. BitFlip: A bit-flipping scheme for reducing read latency and improving reliability of flash memory[C]//Proceedings of International Conference on Massive Storage Systems and Technology (MSST). IEEE, 2020.

[5] MEZA J, WU Q, KUMAR S, et al. 2015. A large-scale study of flash memory failures in the field[J]. ACM SIGMETRICS Performance Evaluation Review. New York: ACM, 43(1): 177-190.

[6] CAI Y, GHOSE S, HARATSCH E F, et al. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives[J]. Proceedings of the IEEE. IEEE, 2017, 105(9): 1666-1704.

[7] MOHAN V, SIDDIQUA T, GURUMURTHI S, et al. How I learned to stop worrying and love flash endurance[C]//Proceedings of the 2nd Workshop on Hot Topics in Storage and File Systems (HotStorage). USA: USENIX Association, 2010, 10: 3-3.

[8] PARK J, JEONG J, LEE S, et al. Improving performance and lifetime of NAND storage systems using relaxed program sequence[C]//Proceedings of the 53rd Annual Design Automation Conference (DAC). IEEE, 2016: 1-6.

[9] CAI Y, YALCIN G, MUTLU O, et al. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime[C]//Proceedings of the 30th International Conference on Computer Design (ICCD). IEEE, 2012: 94-101.

[10] MOHAN V, SANKAR S, GURUMURTHI S, et al. reFresh SSDs: Enabling high endurance, low cost flash in datacenters[R]. Univ. of Virginia, Tech. Rep. CS-2012-05, 2012.

[11] CAI Y, HARATSCH E F, MUTLU O, et al. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling[C]//Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2013: 1285-1290.

[12] LI Q, SHI L, XUE C J, et al. Improving LDPC performance via asymmetric sensing level placement on flash memory[C]//Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2017: 560-565.

[13] CAI Y, LUO Y, HARATSCH E F, et al. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery[C]//Proceedings of the 21st International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2015: 551-563.

[14] JEONG J, HAHN S S, LEE S, et al. Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling[C]//Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2014: 61-74.

[15] PAN Y, DONG G, WU Q, et al. Quasi-nonvolatile SSD: Trading flash memory nonvolatility to improve storage system performance for enterprise applications[C]// Proceedings of the 18th International Symposium on High-Performance Comp Architecture (HPCA). IEEE, 2012: 1-10.

[16] CAI Y, YALCIN G, MUTLU O, et al. Error analysis and retention-aware error management for NAND flash memory[J]. Intel Technology Journal, 2013, 17(1).

[17] CHEN Z, HARATSCH E F, SANKARANARAYANAN S, et al. Estimating read reference voltage based on disparity and derivative metrics: U.S. Patent 9,417,797[P]. 2016-8-16.

[18] WU Y, COHEN E T. Optimization of read thresholds for non-volatile memory: U.S. Patent 9,595,320[P]. 2017-3-14.

[19] LUO Y, GHOSE S, CAI Y, et al. Enabling accurate and practical online flash channel modeling for modern MLC NAND flash memory[J]. IEEE Journal on Selected Areas in Communications. IEEE, 2016, 34(9): 2294-2311.

[20] HUANG P, SUBEDI P, HE X, et al. FlexECC: partially relaxing ecc of mlc ssd for better cache performance[C]//Proceedings of the 2014 USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2014: 489-500.

[21] GAO C, SHI L, WU K, et al. Exploit asymmetric error rates of cell states to improve the performance of flash memory storage systems[C]//Proceedings of the 32nd International Conference on Computer Design (ICCD). IEEE, 2014: 202-207.

[22] WILSON E H, JUNG M, KANDEMIR M T. ZombieNAND: Resurrecting dead NAND flash for improved SSD longevity[C]//Proceedings of the 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 2014: 229-238.

[23] GAO C, YE M, LI Q, et al. Constructing large, durable and fast SSD system via reprogramming 3D TLC flash memory[C]//Proceedings of the 52nd Annual International Symposium on Microarchitecture (MICRO). IEEE, 2019: 493-505.

[24] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[C]//Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP). New York: ACM, 2003.

[25] REED I, SOLOMON G. Polynomial Codes over Certain Finite Fields[J]. Journal of the Society for Industrial & Applied Mathematics, 1960, 8(2): 300-304.

[26] 李明强. 磁盘阵列的纠删码技术研究[D] . 北京: 清华大学, 2011.

[27] 傅颖勋. 存储系统中的若干纠删码机制研究[D] . 北京: 清华大学, 2014.

[28] 沈志荣. 纠删码存储系统性能优化研究[D] . 北京: 清华大学, 2015.

[29] The Apache Software Foundation. Apache Hadoop 3.0.0[EB/OL]. (2017-12-08) [2023-04-11].

[30] Ceph. WELCOME TO CEPH[EB/OL]. (2016)[2023-04-11].

[31] MURALIDHAR S, LLOYD W, ROY S, et al. f4: Facebook's warm blob storage system[C]// Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2014.

[32] HUANG C, SIMITCI H, XU Y, et al. Erasure coding in windows azure storage[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC). USA: USENIX Association, 2012.

[33] RASHMI K V, NIHAR B S, GU D, et al. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster[C]// Proceedings of 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage). USA: USENIX Association, 2013.

[34] DIMAKIS A G, GODFREY P B, WU Y, et al. Network coding for distributed storage systems[J]. IEEE transactions on information theory. IEEE, 2010, 56(9):4539-4551.

[35] RASHMI K V, CHOWDHURY M, KOSAIAN J, et al. EC-Cache: load-balanced, low-latency cluster caching with online erasure coding[C]// Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2016:401-417.

[36] BLAUM M, BRADY J, BRUCK J, et al. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures [J]. IEEE Transactions on Computers. IEEE, 1995, 44(2): 192-202.

[37] CORBETT P, ENGLISH B, GOEL A, et al. Row-diagonal parity for double disk failure correction[C]//Proceedings of the 3rd USENIX conference on File and storage technologies. USA: USENIX Association, 2004: 1.

[38] XU L, BRUCK J. X-code: Mds array codes with optimal encoding[J]. IEEE Transactions on Information Theory. IEEE, 1999, 45(1): 272-276.

[39] FU Y, SHU J. D-Code: An efficient RAID-6 code to optimize I/O loads and read performance[C]// Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2015: 603-612.

[40] PAPAILIOPOULOS D S, DIMAKIS A G. Locally repairable codes[J]. IEEE Transactions on Information Theory, 2014, 60(10): 5843-5855.

[41] SATHIAMOORTHY M, ASTERIS M, PAPAILIOPOULOS D, et al., XORing elephants:

Novel erasure codes for big data[J]. Proceedings of the VLDB Endowment, 2013, 6(5):325-336.

[42] KHAN O, BURNS R, PLANK J, et al. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads[C]// Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST). USA: USENIX Association, 2012.

[43] DIMAKIS A G, GODFREY P B, WU Y, et al. Network coding for distributed storage systems[J]. IEEE Transactions On Information Theory, 2010 56(9):4539-4551.

[44] RASHMI K V, SHAH N B, KUMAR P V. Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction[J]. IEEE Transactions On Information Theory, 2011, 57(8):5227-5239.

[45] PAMIES-JUAREZ L, BLAGOJEVIC F, MATEESCU R, et al. Opening the chrysalis: On the real repair performance of msr codes[C]// Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2016:81-94.

[46] VAJHA M, RAMKUMAR V, PURANIK B, et al., Clay Codes: Moulding MDS codes to yield an MSR code[C]// Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2018:139-154.

[47] RASHMI K V, SHAH N B, GU D, et al. A 'Hitchhiker's' guide to fast and efficient data reconstruction in erasure-coded data centers[C]// Proceedings of the ACM conference on SIGCOMM (SIGCOMM). New York: ACM, 2014:331-342.

[48] LI R, LEE P P C, HU Y. Degraded-first scheduling for mapreduce in erasure-coded storage clusters[C]// Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 14). IEEE, 2014:419-430.

[49] HUANG J, LIANG X, QIN X, et al. PUSH: a pipelined reconstruction i/o for erasure-coded storage clusters[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(2):516-526.

[50] MITRA S, PANTA R, RA M-R, et al. Partial-Parallel-Repair (PPR): A distributed technique for repairing erasure coded storage[C]// Proceedings of the 11th European Conference on Computer Systems (EuroSys). 2016:1-16.

[51] LI R, LI X, LEE P P C, et al. Repair pipelining for erasure-coded storage[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC 17). USA: USENIX Association, 2017:567-579.

[52] SHEN Z, SHU J, LEE P P C. Reconsidering single failure recovery in clustered file systems[C]// Proceedings of the 46th Annual IEEE/IFIP International Conference on

Dependable Systems and Networks (DSN 16). IEEE, 2016:323-334.

[53] SHEN Z, SHU J, HUANG Z, et al. ClusterSR: Cluster-aware scattered repair in erasure-coded storage[C]// Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2020:42-51.

[54] HU Y, CHENG L, YAO Q, et al. Exploiting combined locality for wide-stripe erasure coding in distributed storage[C]// Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association. 2021:233-248.

[55] LIN S, GONG G, SHEN Z, et al. Boosting full-node repair in erasure-coded storage[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC). USA: USENIX Association, 2021:641-655.

[56] HAN S, LEE P P C, SHEN Z, et al. Toward adaptive disk failure prediction via stream mining[C]// Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS). IEEE, 2020:628-638.

[57] BOTEZATU M M, GIURGIU I, BOGOJESKA J, et al. Predicting disk replacement towards reliable data centers[C]// Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). New York: ACM, 2016: 39-48.

[58] LI J, JI X, JIA Y, et al. Hard drive failure prediction using classification and regression trees[C]// Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2014:383-394.

[59] MA A, DOUGLIS F, LU G, et al. RAIDShield: Characterizing, monitoring, and proactively protecting against disk failures[C]// Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2015:241-256.

[60] ZHU B, WANG G, LIU X, et al. Proactive drive failure prediction for large scale storage systems[C]// Proceedings of the 29th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2013:1-5.

[61] XU Y, SUI K, YAO R, et al. Improving service availability of cloud systems by predicting disk error[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC). USA: USENIX Association, 2018:481-494.

[62] LU S, LUO B, PATEL T, et al. Making disk failure predictions smarter![C]// Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2020:151-167.

[63] ZHANG J, HUANG P, ZHOU K, et al. HDDse: Enabling high-dimensional disk state

embedding for generic failure detection system of heterogeneous disks in large data centers[C]// Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC). USA: USENIX Association, 2020:111-126.

[64] SHEN Z, LI X, LEE P P C. Fast predictive repair in erasure-coded storage[C]// Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2019:556-567.

[65] HU Y, WANG Y, LIU B, et al. Latency reduction and load balancing in coded storage systems[C]// Proceedings of Symposium on Cloud Computing (SoCC). IEEE, 2017: 365-377.

[66] XIA M, SAXENA M, BLAUM M, et al. A Tale of two erasure codes in hdfs[C]// Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2015:213-226.

[67] WANG Z, WANG H, SHAO A, et al. An adaptive erasure-coded storage scheme with an efficient code-switching algorithm[C]// Proceedings of the 49th International Conference on Parallel Processing, (ICPP). New York, NY, USA: 2020:1-11.

[68] ZHOU P, HUANG J, QIN X, et al. PaRS: A popularity-aware redundancy scheme for in-memory stores[J]. IEEE Transactions on Computers. 2019, 68(4):556-569.

[69] KADEKODI S, MATURANA F, SUBRAMANYA S J, et al. PACEMAKER: Avoiding HeART attacks in storage clusters with disk-adaptive redundancy[C]// Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2020:369-385.

[70] TARANOV K, ALONSO G, HOEFLER T. Fast and strongly-consistent per-item resilience in key-value stores[C]// Proceedings of the 13th EuroSys Conference (EuroSys). 2018: 1-14.

[71] REED I S, SOLOMON G, Polynomial codes over certain finite fields[J], Journal of the Society for Industrial and Applied Mathematics, 1960, (8)2: 300-304.

[72] WU S, SHEN Z, LEE P P C. Enabling I/O-efficient redundancy transitioning in erasure-coded kv stores via elastic reed-solomon codes[C]// Proceedings of International Symposium on Reliable Distributed Systems (SRDS). 2020: 246-255.

[73] WU S, DU Q, LEE P P C, et al. Optimal data placement for stripe merging in locally repairable codes[C]// Proceedings of the IEEE Infocom 2022 - IEEE Conference on Computer Communications (INFOCOM). 2022:1669-1678.

[74] YAO Q, HU Y, CHENG L, et al. StripeMerge: Efficient wide-stripe generation for

large-scale erasure-coded storage[C]// Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS). IEEE, 2021:483-493.

[75] MATURANA F, MUKKA V S C, RASHMI K V. Access-optimal linear MDS convertible codes for all parameters[C]// Proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020:577-582.

[76] MATURANA F, RASHMI K V. Convertible Codes: New class of codes for efficient conversion of coded data in distributed storage[C]// Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS). 2020:1-26.

[77] ZHANG G, SHU J, XUE W, et al. SLAS: An efficient approach to scaling round-robin striped volumes[J], ACM Transactions on Storage, 2007, 3(1):3-es.

[78] ZHENG W, ZHANG G. FastScale: Accelerate RAID scaling by minimizing data migration[C]// Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2011.

[79] ZHANG G, ZHENG W, SHU J. ALV: A new data redistribution approach to RAID-5 scaling[J], IEEE Transactions on Computers, 2010, 59(3):345-357.

[80] WU C, HE X, HAN J, et al. SDM: A stripe-based data migration scheme to improve the scalability of RAID-6 [C]// Proceedings of the IEEE International Conference on Cluster Computing (CoCC). IEEE, 2012:284-292.

[81] WAN J, XU P, HE X, et al. H-Scale: A fast approach to scale disk arrays via hybrid stripe deployment[J], ACM Transactions on Storage, 2016, 12(3): 1-30.

[82] HUANG J, LIANG X, QIN X, et al. Scale-RS: An efficient scaling scheme for rs-coded storage clusters[J], IEEE Transactions on Parallel and Distributed Systems, 2015, 26(6):1704-1717.

[83] ZHANG X, HU Y, LEE P P C, et al. Toward optimal storage scaling via network coding: from theory to practice[C]// Proceedings of the IEEE Infocom 2018 - IEEE Conference on Computer Communications (INFOCOM). 2018:1808-1816.

[84] CHENG L, HU Y, LEE P P C. Coupling decentralized key-value stores with erasure coding[C]// Proceedings of the ACM Symposium on Cloud Computing. New York: ACM, 2019:377-389.

[85] YUAN D, LUO Y, ZHUANG X, et al. Simple testing can prevent most critical failures: an analysis of production failures in distributed data-intensive systems[C]//Proceedings of the 11th Symposium on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2014: 48-68.

[86] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[C]// Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2010: 1-10.

[87] ABADI D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story[J]. IEEE Computer. 2012, 45(2): 37-42.

[88] LAMPORT L. Paxos made simple[J]. ACM SIGACT News, 2001, 4 (121): 51-58.

[89] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C]// Proceedings of the USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2014: 305-319.

[90] WANG Z, ZHAO C, MU S, et al. On the Parallels between Paxos and Raft, and how to Port Optimizations[C]//Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC). New York: ACM, 2019: 445-454.

[91] CHOU T C K, ABRAHAM J A. Load balancing in distributed systems[J]. IEEE Transactions on Software Engineering. IEEE, 1982 (4): 401-412.

[92] ALAKEEL A M. A guide to dynamic load balancing in distributed computer systems[J]. International Journal of Computer Science and Information Security, 2010, 10(6): 153-160.

[93] KARGER D, LEHMAN E, LEIGHTON T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web[C]// Proceedings of the 29th Annual Symposium on Theory of Computing (STOC). New York: ACM, 1997: 654-663.

乃至国家层面的重要关切。本章对存储安全进行了系统的分析介绍。其首先从存储安全的理念及体系进行概述，并分别从系统安全、数据安全和安全管理等 3 个层面进行细致剖析；其次，本章介绍了当前存储安全的一些前沿应用及其所适合的应用场景，包括加密搜索、远程数据销毁及同态加密等。当前存储与网络正在进行深度融合，存储安全将是实现数据共享和隐私保护兼得的重要之匙。相信随着存储技术的进一步发展，未来也将出现更多的安全需求场景，驱动着新的存储安全技术不断涌现。

参考文献

[1] BRENNER M. Biggest risks of cloud computing and how to mitigate them [EB/OL]. (2021-12-13)[2023-05-30].

[2] SHU J, SHEN Z, XUE W. Shield: A stackable secure storage system for file sharing in public storage[J]. Journal of Parallel and Distributed Computing, 2014:74(9), 2872-2883.

[3] ARNAUTOV S, TRACH B, GREGOR F, et al. Scone: Secure linux containers with intel SGX[C]// Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2016: 689-703.

[4] KATZ J, LINDELL Y. Introduction to modern cryptography[M]. Boca Raton:CRC Press, 2020.

[5] HELLMAN M. New directions in cryptography[J]. IEEE transactions on Information Theory, 1976, 22(6): 644-654.

[6] RIVEST R L, SHAMIR A, ADLEMAN L. A method for obtaining digital signatures and public-key cryptosystems[J]. Communications of the ACM. New York: ACM, 1978, 21(2): 120-126.

[7] GAMAL T E. A public key cryptosystem and a signature scheme based on discrete logarithms[J]. IEEE Transactions on Information Theory, 1985, 31:469-472.

[8] CATTANEO G, CATUOGNO L, DEL SORBO A, et al. The design and implementation of a transparent cryptographic file system for UNIX[C]//Proceedings of the 2001 USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2001.

[9] 肖达. 存储系统中的数据安全方法与技术[D]. 北京: 清华大学, 2008.

[10] 肖达, 舒继武, 薛巍, 等. 基于组密钥服务器的加密文件系统的设计和实现[J]. 计算机学报, 2008, 31(4):600-610.

[11] BLAZE M. A cryptographic file system for UNIX[C]//Proceedings of the 1st ACM Conference on Computer and Communications Security. New York: ACM, 1993: 9-16.

[12] WRIGHT C P, MARTINO M C, ZADOK E. NCryptfs: A secure and convenient cryptographic file system[C]//Proceedings of the 2003 USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2003: 197-210.

[13] BINDEL D, CHEW M, WELLS C. Extended cryptographic file system[EB/OL]. (2001-01)[2023-05-30].

[14] FU K E. Group sharing and random access in cryptographic storage file systems[D]. Cambridge: Massachusetts Institute of Technology, 1999.

[15] KUMAR U K, UMASHANKAR B S. Improved hamming code for error detection and correction[C]//Proceedings of the 2nd International Symposium on Wireless Pervasive Computing (ISWPC). IEEE, 2007:498-450.

[16] GUPTA P, KUMAR S. A comparative analysis of SHA and MD5 algorithm[J]. International Journal of Computer Science and Information Technologies, 2014, 5(3):4492-4495.

[17] STALLINGS W. Cryptography and network security: principles and practice[M]. London: Pearson Education, 2020.

[18] ATENIESE G, BURNS R, CURTMOLA R, et al. Provable data possession at untrusted stores[C]//Proceedings of the 14th ACM conference on Computer and Communications Security (CCS). New York: ACM, 2007: 598-609.

[19] KOO D, SHIN Y, YUN J, et al. Improving security and reliability in merkle tree-based online data authentication with leakage resilience[J]. Applied Sciences, 2018, 8(12): 2532.

[20] TIAN H, CHEN Y, JIANG H, et al. Public auditing for trusted cloud storage services[J]. IEEE Security & Privacy, 2019, 17(1):10-22.

[21] 薛矛. 一种共享存储环境下的安全存储系统[D]. 北京:清华大学, 2011.

[22] SONG D X, WAGNER D, PERRIG A. Practical techniques for searches on encrypted data[C]// Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P). IEEE, 2000: 44-55.

[23] BONEH D, DI CRESCENZO G, OSTROVSKY R, et al. Public key encryption with keyword search[C]//Proceedings of the Lecture Notes in Computer Science. Springer, 2004: 506-522.

[24] BONEH D, FRANKLIN M. Identity-based encryption from the weil pairing[J]. Siam Journal on Computing, 2003, 32(3): 586-615.

[25] SAHAI A, WATERS B. Fuzzy identity-based encryption[C]// Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, 2005: 457-473.

[26] WEI M, GRUPP L M, SPADA F E, et al. Reliably erasing data from flash-based solid state drives[C]//Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2011:105–117.

[27] BONEH D, LIPTON R J. A revocable backup system[C]//Proceedings of the USENIX Security Symposium (Security). USA: USENIX Association, 1996: 91-96.

[28] XUE L, YU Y, LI Y, et al. Efficient attribute-based encryption with attribute revocation for assured data deletion[J]. Information Sciences, 2019, 479: 640-650.

[29] LIU C, RANJAN R, YANG C, et al. MuR-DPA: Top-down leveled multi-replica Merkle hash tree based secure public auditing for dynamic big data storage on cloud[J]. IEEE Transactions on Computers, 2014, 64(9): 2609-2622.

[30] YANG C, TAO X, ZHAO F, et al. A new outsourced data deletion scheme with public verifiability[C]//Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications (WASA). Springer, 2019: 631-638.

[31] BAUMANN A, PEINADO M, HUNT G. Shielding applications from an untrusted cloud with haven[C]//Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation (OSDI). USA: USENIX Association, 2014: 267-283.

[32] SHINDE S, LE TIEN D, TOPLE S, et al. Panoply: Low-tcb linux applications with SGX enclaves[C]// Proceedings of the Network and Distributed System Security Symposium (NDSS). Internet Society, 2017:1-15.

[33] TSAI C C, PORTER D E, VIJ M. Graphene-SGX: A practical library os for unmodified applications on SGX[C]// Proceedings of the 2017 USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2017: 645-658.

[34] Large-scale data systems group. SGX-LKL-OE (open enclave edition) [EB/OL]. (2022)[2022-03-07].

[35] ORENBACH M, LIFSHITS P, MINKIN M, et al. Eleos: Exitless os services for SGX enclaves[C]//Proceedings of the 12th European Conference on Computer Systems (EuroSys). New York: ACM, 2017: 238-253.

[36] WEISSE O, BERTACCO V, AUSTIN T M. Regaining lost cycles with hotcalls: A fast interface for SGX secure enclaves[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA). New York: ACM, 2017: 81-93.

[37] BAILLEU M, THALHEIM J, BHATOTIA P, et al. Speicher: Securing LSM-based key-value stores using shielded execution[C]//Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST). USA: USENIX Association, 2019: 173-190.

[38] BAILLEU M, GIANTSIDI D, GAVRIELATOS V, et al. Avocado: A secure in-memory distributed storage system[C]//Proceedings of the 2021 USENIX Annual Technical Conference (ATC). USA: USENIX Association, 2021: 65-79.

[39] DPDK Project. DPDK[EB/OL]. (2022-01-01)[2022-03-07].

[40] KALIA A, KAMINSKY M, ANDERSEN D. Datacenter RPCS can be general and fast[C]// Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation. USA: USENIX Association, 2019: 1-16.

[41] Intel Corporation. Intel SGX developer SDK for Linux [EB/OL]. (2021-01-01) [2022-03-07].

[42] KIM T, PARK J, WOO J, et al. Shieldstore: Shielded in-memory key-value storage with SGX[C]//Proceedings of the 14th European Conference on Computer Systems (EuroSys). New York: ACM, 2019: 1-15.

[43] ZHOU W, CAI Y, PENG Y, et al. VeriDB: An sgx-based verifiable database[C]// Proceedings of the 2021 International Conference on Management of Data (SIGMOD). New York: ACM, 2021: 2182-2194.

[44] BLUM M, EVANS W, GEMMELL P, et al. Checking the correctness of memories[C]// Proceedings 32nd Annual Symposium of Foundations of Computer Science. IEEE, 1991: 90-99.

[45] TRAMER F, BONEH D. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware[EB/OL]. (2019-02-27)[2023-05-30].

[46] KIM K, KIM C H, RHEE J J, et al. Vessels: Efficient and scalable deep learning prediction on trusted processors[C]//Proceedings of the 11th ACM Symposium on Cloud Computing. New York: ACM:2020: 462-476.

[47] SUN Y, WANG S, LI H, et al. Building enclave-native storage engines for practical encrypted databases[C]//Proceedings of the VLDB Endowment. New York: ACM, 2021, 14(6): 1019-1032.

[48] YANG F, CHEN Y, LU Y, et al. Aria: Tolerating skewed workloads in secure in-memory key-value stores[C]//Proceedings of the 37th International Conference on Data Engineering (ICDE). IEEE, 2021: 1020-1031.

[49] 杨帆. 可信执行环境下的高效内存存储关键技术研究[D]. 北京: 清华大学, 2022.

[50] X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks[S/OL]. [2023-05-30].

齐日志（或快照）同步起始位置。

② 异步复制通过后台任务将主 LUN12 的 T1 时间点快照数据同步至从 LUN2 上，并在同步任务完成后将主从 LUN 的快照删除。通知 A 站点处于 Standby 状态的异步复制将同步前对齐的日志（或快照）位置前的所有差异丢弃。

当 A—B 站点的双活因站点 B、链路等故障发生业务切换后，或者 B—C 站点间的复制链路发生故障导致 B—C 站点处于工作状态的异步复制无法继续数据同步时，A—C 站点的异步复制会自动切换为工作状态，接管 B—C 站点的异步复制任务，并从最后一次 B—C 站点的异步复制同步完成时刻通知的差异日志位置开始继续增量同步。该过程不需要人工介入，即可保证异地灾备中心 C 站点的数据可用性，缩短 RPO。

11.4　本章小结

数据是当今大数据和人工智能时代的关键生产资料，关键数据的丢失会带来严重的后果。数据面临的威胁包括系统软硬件故障、人为差错、网络攻击、自然灾害等多个方面，因而无法避免数据丢失的发生。因此，数据保护技术的基础在于增加数据的复制和副本，以提高对数据部分丢失的容忍度。基于此，本章从数据保护标准、技术和场景 3 个方面进行了介绍。随着全球企业数字化转型，全球数据总量海量增长，给数据的存储和保护带来新的挑战。

参考文献

[1] 王德军，王丽娜. 容灾系统研究[J]. 计算机工程, 2005, 31(6):4.

[2] 国务院信息化工作办公室. 重要信息系统灾难恢复指南[Z]. 北京：国务院信息化办公室, 2005.

[3] 全国信息安全标准化技术委员会.信息安全技术 信息系统灾难恢复规范: GB/T 20988—2007[S]. 北京：中国标准出版社, 2007.

[4] GARCIA-MOLINA H, POLYZOIS C A. Issues in disaster recovery[C]//IEEE, 35th IEEE Computer Society International Conference on Intellectual Leverage. IEEE Computer Society, San Francisco, 1990: 573-577.

[5] 向小佳. 数据保护若干关键技术的研究[D]. 北京：清华大学, 2019.

[6] YAO J, SHU J, ZHENG W. Distributed storage cluster design for remote mirroring based on storage area network[J]. Journal of Computer Science and Technology, 2007, 22(4): 521-526.

[7] ZHENG X F, OUYANG B, ZHANG D N, et al. Technical system construction of Data Backup Centre for China Seismograph Network and the data support to researches on the Wenchuan earthquake[J]. Chinese Journal of Geophysics, 2009, 52(5): 1412-1417.

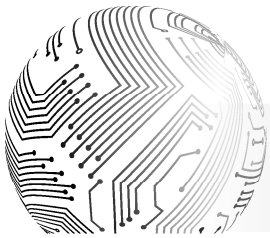
[8] WHITLOCK M C, MCPEEK M A, RAUSHER M D, et al. Data archiving[J]. The American Naturalist, 2010, 175(2): 145-146.

[9] PIWOWAR H A, VISION T J, WHITLOCK M C. Data archiving is a good investment[J]. Nature, 2011, 473(7347): 285-285.

[10] HAMMERSLEY M. Qualitative data archiving: some reflections on its prospects and problems[J]. Sociology, 1997, 31(1): 131-142.

[11] TOIGO J. Disaster recovery planning: Preparing for the unthinkable[M]. ACM Digital Library, 2002.

[12] FALLARA P. Disaster recovery planning[J]. IEEE Potentials, 2004, 23(5): 42-44.



第14章 存储技术趋势与发展

随着大数据、云计算、人工智能和区块链等技术的快速发展，用户对高性能、高容量、高可靠的数据存储的需求愈发强烈，需要探索新的存储技术来应对上述技术的发展。本章主要关注存储技术的发展趋势，从新型存储模式，非易失性存储系统及应用存储优化等各方面分别介绍了闪存存储系统、存内计算、持久性内存、在网存储、智能存储、边缘存储、区块链存储、分离式数据中心架构、高密度新型存储等前沿技术。

14.1 闪存存储系统

随着闪存密度提升、价格下降，闪存逐步被应用到桌面机、服务器和数据中心。特别是闪存具有不同于磁盘的硬件特性，相比于传统磁盘，闪存具有性能高、能耗低和体积小等优势，这些都为构建高效的存储系统带来了巨大的机遇，同时也面临新的挑战。闪存在实际应用中的形态，由早期的 SSD、闪存卡，逐渐发展到闪存阵列和闪存集群系统，研究人员致力于对已有的存储结构、系统软件及分布式协议进行革新[1]。例如，研究移除闪存设备上的 FTL，设计基于裸闪存的存储系统；优化闪存集群上的数据迁移策略，延长闪存寿命；研究闪存与特定应用负载特征相结合的软件定义存储等。

针对闪存集群系统的研究，两个代表性工作是 FAWN[2]和 Gordon[3]。FAWN 是美国卡内基-梅隆大学基于闪存介质构建的可扩展、低能耗、高性能的集群系统。与闪存卡和闪存阵列仅关注子系统的性能及可靠性的设计不同，FAWN 从集群整体设计的角度考虑闪存与处理器的匹配，以降低整体能耗。FAWN 采用低频低能耗的 CPU 与闪存存储相匹配，提高系统各组成部分在数据密集型计算中的利用

率。FAWN 实现的键值存储系统，消耗 1 J 能量可进行 364 次查询，相比普通桌面系统消耗 1 J 能量进行 1.96 次查询，能耗降低 99.46%。Gordon[3]是美国加利福尼亚大学圣地亚哥分校设计的，与 FAWN 关注的高性能、低能耗不同，Gordon 主要工作是设计 FTL 来匹配处理器和内存芯片的性能，发挥闪存芯片间的并发特性，设计包括地址动态映射、合成大物理页等，同时采用并发与流水机制。Gordon 在单板上集成 256 GB 的闪存和 2.5 GB 的 DRAM，目前已经应用到美国圣地亚哥超算中心，用于天体物理、基因组测序等数据密集型计算领域。

由 SSD、闪存卡，进而发展到闪存阵列和闪存集群系统，能构建更大规模的闪存存储系统，是一种横向扩展方式。但闪存设备的 FTL 影响着闪存硬件特性的发挥，进而也影响闪存存储系统的构建，由此诞生了一些新型的闪存结构，例如 OC SSD（Open-Channel SSD，开放通道 SSD）和 ZNS SSD（Zone Namespace SSD，区域命名空间 SSD），这是闪存系统的一种纵向扩展方式。

14.1.1　OC SSD

早期，闪存多用于嵌入式系统，在嵌入式系统中，闪存以裸闪存形式存在，嵌入式闪存文件系统在文件系统中实现 FTL 的相关功能，包括地址映射、垃圾回收和磨损均衡等。

在 SSD 发展的初期，为了兼容磁盘的接口，采用 FTL 将闪存的读写擦除接口转换为传统的读写接口[4]，如图 14.1（a）所示。SSD 提供与传统磁盘相同的读写接口，并采用 FTL 转换为闪存命令。由于 SSD 与传统磁盘的读写接口兼容，传统文件系统可无缝地部署在 SSD 上，这样的做法在兼容性上取得了很大的优势，使 SSD 可以很容易替换传统的磁盘。

现有不少研究在该使用模式下针对 SSD 特性优化文件系统。F2FS[5]是三星公司针对 SSD 进行定向优化的文件系统。F2FS 是针对闪存的特性，改良传统的日志型文件系统的一个例子。文件系统日志的每个段与底层闪存的块保持一致大小，并对数据进行冷热分组，以此来提高 SSD 垃圾回收的效率。同时利用地址映射表，以避免元数据更新时一直到根的滚雪球式的更新方式，减少元数据写入量，提高 SSD 寿命。

但是，FTL 掩盖了内部细节，软件难以利用硬件的特性。SSD 内部 FTL 要求 SSD 具备较强的嵌入式处理器和设备内缓存，尤其是在盘内闪存容量较大的情况下。后来，Fusion-io 公司提出了主机端 FTL 的设计架构，如图 14.1（b）所示，以充分利用主机的处理器与内存资源。美国普林斯顿大学与 Fusion-io 公司基于主机端 FTL 实现了新型闪存文件系统 DFS[6]，将文件系统的块分配操作下移至 FTL，

从而避免了文件系统空间管理与闪存设备 FTL 管理的冗余管理开销。尽管主机端 FTL 的设计有效利用了主机端资源，但文件系统与 FTL 之间仍采用简单的读写接口，限制了语义信息的使用。

进一步，如图 14.1（c）所示，清华大学存储团队提出了软件直管闪存（Software Managed Flash）SSD 架构[7]，这种架构去除了 SSD 中的 FTL，由主机侧软件直接对闪存介质进行管理，可以进一步去掉冗余管理，并可充分利用闪存内部并发特性[8]等，这种架构也衍生出 OC SSD。基于此架构，提出了可重构的闪存文件系统元数据管理方法[8]、内部并发感知的闪存文件系统数据管理方法[9]、基于开放通道架构的键值存储加速方法[10]、分布式文件系统元数据设计技术[11]、闪存硬件支持的高效事务处理方法[12-13]等一系列的关键技术和方法，构建出了软硬件协同的闪存存储系统 TH-SSS (Tsinghua-Solid Storage System)。OC SSD 通过暴露设备内部的细节给主机端，将 FTL 的功能交由主机端来实现，主机负责数据的放置和管理。然而，OC SSD 存在一些问题。首先，OC SSD 的生态不完善，需要对存储栈进行特定的修改，软件端会有比较大的改动。其次，不同的 SSD 厂商的内部管理也不同，所以在兼容性方面存在较大的问题[14]。因此，后来也衍生出 ZNS SSD 等形态。

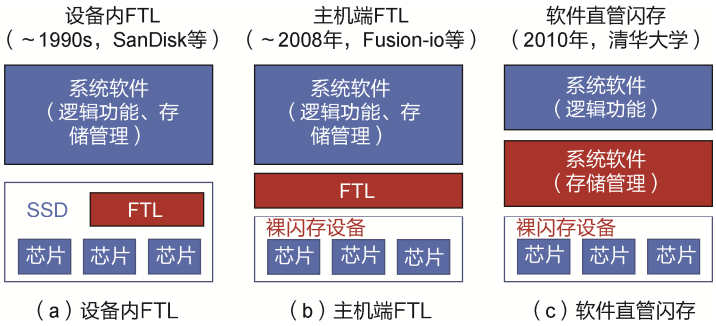


图 14.1　SSD 架构的演进[4]

14.1.2　ZNS SSD

ZNS SSD 则是在 OC SSD 的基础上进行了改进，主要的思想仍然是向主机端暴露闪存内部的细节，但是通过引入分区存储的方式实现了一个统一的存储接口，并且实现了 NVMe 的标准化[15]。通过这些改进，ZNS SSD 提供了更简化、统一和标准化的存储解决方案，降低了对主机软件和驱动程序的修改要求，提高了软件开发和系统集成的便利性。

ZNS SSD 去掉了 FTL 中的大部分功能，包括地址转换和映射等。这些功能被转移到了主机端进行处理。这种设计使得 ZNS SSD 更加简化和透明，允许主机操

作系统或存储管理软件直接管理存储设备。在传统 SSD 中，FTL 采用细粒度映射，需要大尺寸的映射表来支持地址转换功能；但在 ZNS SSD 中，映射可以通过块粒度来实现，大大降低了映射表所需要的开销。由于映射表需要部分存储在 SSD 设备的 DRAM 中，块粒度映射的 ZNS SSD 对于 DRAM 的需求也有显著的降低。此外，因为将数据管理的功能转移到主机上，所以消除了 ZNS 设备内部垃圾回收的需要，不存在写放大的问题，也就消除了对于预留空间的需求，提高了存储设备的有效容量。

ZNS SSD 的基础是分区存储，分区存储在此之前就已经存在，其概念更多应用在 SMR HDD 中，将逻辑地址分为具有不同于常规写入约束的相同大小区域，ZNS SSD 则是在分区存储的基础上，增加了一些其特有的概念。如图 14.2 所示，ZNS SSD 将存储空间划分为多个区域，这些区域必须按顺序写入，每个区域都由一个写指针来跟踪下一次写入的位置，区域不能进行覆盖写，只有当区域进行重置之后，才能重新使用写指针之前的 LBA。区域容量将区域大小分为可写和不可写的部分，区域容量小于或等于区域大小。区域容量的引入是为了能够将区域与硬件介质的擦除块对齐。

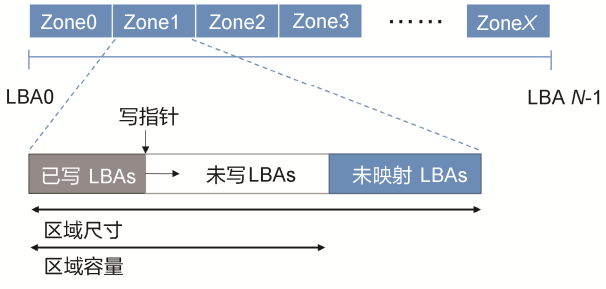


图 14.2　ZNS SSD 分区示意图

ZNS SSD 的每个区域都有一个状态，区域可以在各个状态之间进行转换，区域的状态决定了该区域的操作限制及资源分配的情况，主要包括以下 6 个状态[16]。

Empty：区域内部为空，代表没有数据，但是不能对该状态区域进行写操作。

Full：区域容量全都被写满。

Open：区域处于可以写数据的状态，主机为该状态区域分配资源。

Close：区域仍然处于活动状态，但是被主机收回了资源。

Read Only：区域只能进行读取。

Offline：区域不能进行读和写，代表该区域已经被损坏且不能再使用。

通过定义和管理这些区域状态，ZNS SSD 可以实现更高效的写入操作和均衡的擦除，并简化数据管理和维护。

由于主机和设备的资源有限，必须对使用资源的区域进行限定，在 ZNS SSD 中同时可以操作的区域有数量限制，包括开放区域限制（Open Zones Limit）和活跃区域限制（Active Zones Limit）。开放区域代表可以进行写操作区域的数量，这个限制由主机的内部资源（写缓冲）和硬件介质资源（通道数量）决定；活动区域包括处于 Open 和 Close 状态的所有区域，活动区域代表可以进行存储数据的区域，通过在区域的 Open 和 Close 之间转换来决定对哪些区域进行写操作。

分区存储设备无法像传统的设备可以即插即用，使用 ZNS SSD 设备必须遵守分区存储的约束和规则。为了将分区存储设备集成到系统中，需要特定的软件栈的支持。目前，Linux 分区存储生态系统已经支持了多种分区存储。第一种方式是通过文件系统实现集成，对于符合 ZBD（Zoned Block Device，区域块设备）接口的文件系统来说，其已经能够支持分区设备顺序写的约束，所以能够适配 ZNS SSD，一个典型的例子是 F2FS；而对于一般的文件系统来说，则需要设备映射器的支持来处理设备顺序写的约束。第二种方式是直接通过原始块接口进行访问，应用程序可以通过原始的块接口直接对分区存储设备进行数据的存取，这种方式也需要使用设备映射器来将随机写转化为顺序写。第三种方式是通过文件访问接口来实现集成，对于特定的应用程序设计符合其特性的小型文件系统插件，例如，RocksDB 可以通过 ZenFS 文件系统来访问 ZNS SSD。

ZNS 产品的出现促进了 ZNS 的相关研究工作，研究者们针对 ZNS 的特性进行了研究，并提出了适配 ZNS 的冗余阵列机制、文件系统及键值存储系统等，目前基于 ZNS 的存储技术主要有以下几种。

ZNS 设备特性[16]：ZNS 在读写一个区域之前需要先将区域激活，ZNS 内部缓冲区的大小对于激活区域的数量有限制。不同供应商的 ZNS 对于激活区域的上限不同，部分种类的 ZNS 的激活区域上限能达到 4096 个。然而，高激活区域上限的 ZNS 并不能可靠地保证不同区域之间的性能隔离，实验显示部分区域之间共享带宽，这会造成应用之间的性能干扰。为此，已有研究准确描述了 ZNS 的性能冲突区域组，并针对性地设计了请求重映射机制，从而平衡不同冲突组之间的请求，以避免性能干扰。

键值存储系统：西部数据公司的研究者们全面地描述了 ZNS 设备的特性[15]，并且为键值数据库 RocksDB 设计了专用的 ZenFS 以支持其在 ZNS 上运行。该研究指出，相比传统 SSD，ZNS SSD 的性能更加稳定，对设备内部的内存空间消耗

也更少。ZenFS 设计了专用的数据区域和日志区域，日志区域存放超级块的修改日志及 RocksDB 的写前日志。ZenFS 还设计了区域分配机制，将 RocksDB 中生存周期相近的分段文件尽可能地写入同一个区域中，便于未来统一对该区域进行擦除。

面向 ZNS 设备的文件系统：已有研究面向 ZNS 设备设计了文件系统 ZNS+[17]。ZNS+针对已有闪存文件系统 F2FS 无法直接运行在 ZNS 上的特点，修改了 F2FS 内部的穿插写入的设计，提出了适应 ZNS 内部数据组织的穿插写入方案。此外，ZNS+还为 ZNS 设备引入了新的复制原语，支持主机端发出命令后 ZNS 内部直接执行数据复制，避免数据迁移对 ZNS 带宽的浪费。ZNS+对于 ZNS 的原语扩充为 ZNS 未来的研究者提供了新的思路，主机执行垃圾回收时避免将数据读到内存中整合，成为未来面向 ZNS 的存储软件的研究方向之一。

面向 ZNS 的冗余阵列机制：冗余阵列机制原本是基于块设备设计的，为了充分利用磁盘和固态硬盘顺序写入性能高的特点，研究者们提出了 LogRAID[18-19] 的方式，以追加写入的方式向阵列中追加数据。具体来说，LogRAID 会将写入阵列的数据转化为顺序写入，LogRAID 需要维护原地写转化为异地写的映射关系。但 LogRAID 可能需要读取设备上校验区的数据，并再次写入原位置以完成修改。然而，这个过程难以通过现有 ZNS 接口得到支持。因此，卡内基-梅隆大学的研究者们提出了面向 ZNS 的冗余阵列机制 RAIZN[20]，在多块 ZNS 设备上提供支持容错的区域抽象。具体来说，RAIZN 为了解决部分校验更新的问题，在每块 ZNS 上开辟出一个单独的区域用于存放临时的部分校验更新及其他元数据。当校验块所在条带的数据完全被填充后，RAIZN 会将单独区域的校验信息写回原地址，以避免在冗余阵列层维护映射表产生额外开销。

面向 ZNS 的内存交换技术 ZNSwap：当系统遭受内存压力的时候，系统会将内存中的一些页存放到底层的存储设备中来缓解内存的压力，随着存储技术的发展，现在内存交换技术不仅在内存快满的时候使用，也被用于内存扩展来优化系统的性能。但在传统的系统交换中，由于 SSD 对上层不透明，存在垃圾回收时对无效页的迁移，尤其是当设备利用率高的时候，严重影响 swap 区域的换入换出效率。而 ZNS 设备对主机透明，没有复杂的 FTL，并且消除垃圾回收，主机对设备有更高程度的控制。ZNSwap[21]实现了操作系统交换技术和 ZNS SSD 的协同设计，通过对交换空间进行细粒度的空间管理，实现自定义的数据管理和放置。同时，通过设计主机端的垃圾回收操作来替代 TRIMs 指令，避免在交换时对无效数据的复制，消除了 TRIMs 指令带来的性能开销。

目前，ZNS SSD 已经在 NVMe 2.0 规范中实现了标准化，这个规范是在原有

分区设备规范的基础上，引入了 ZNS 设备的特性，例如区域容量、最大活动区域限制及区域附加功能等[22-23]。ZNS SSD 未来的发展还有许多可以探索的空间，例如，区域隔离性的进一步探究，包括在区域级别实现更严格的隔离，以避免不同应用或用户之间的干扰，并提供更好的数据保护和隐私性；区域大小的灵活性，当前的 ZNS SSD 将所有区域设定为相同的大小，但未来可以考虑引入灵活的区域大小。根据应用的需求和存储容量的分配进行更精细的调整，提供更高的灵活性和性能优化。此外，一个重要的问题是如何完善 ZNS SSD 的生态系统，以促进其广泛应用。如何在最小限度修改软件栈的前提下，使大多数应用程序能够方便地使用 ZNS SSD，这决定着以后主流存储设备能否从传统的 SSD 顺利过渡到 ZNS SSD。

闪存最终是什么样的架构，仍然是个开放的问题。但是，软件直管闪存所指出的核心问题，即软件与硬件如何高效协同管理闪存以发挥闪存效率，仍将是架构探索中的关键。

14.2 存内计算

处理器和存储器是现代计算机系统的重要组成部件。近年来，处理器的性能增长迅速，存储器的访问速度提升缓慢，两者的不平衡性造成“存储墙”问题。在基于冯·诺依曼架构的计算机系统中，处理单元和存储单元相互分离，应用程序运行时需要频繁地在处理器和存储器之间搬运数据，消耗大量时间。另外，海量数据传输产生的能耗远高于计算能耗，数据传输成为系统能耗的主要“贡献者”，由此产生“功耗墙”问题。PIM（Processing-in-Memory，存内计算）技术的出现，为以上问题的解决提供了变革契机。该技术将处理器和存储器紧密集成在一起，实现数据的就近计算，从根本上减少了数据的移动距离，有望在提升计算机系统性能的同时降低能耗。

14.2.1 近存计算

在现代计算机体系结构中，处理器与内存和外存彼此分离，它们之间的功能相互独立，计算机处理数据时，必须通过片外总线将数据传输到处理器中。然而，片外总线距离长且带宽有限，使数据传输成为处理器的性能瓶颈。近存计算在存储器附近放置了部分逻辑计算单元，从而可以在接近数据的位置进行运算，减少了代价高昂的数据移动开销。近存计算按照存储层次可分为近存储计算和近内存计算。

1. 近存储计算

近存储计算将对数据敏感的运算卸载到接近外存存储介质的运算单元中，可以降低计算机系统 CPU 和内存的负荷，提升应用的响应速度。目前 SSD 已经得到了广泛应用，因此基于 SSD 的近存储计算方案也吸引了研究学者的关注。例如，CSSD[24]采用软硬件协同的方法实现了可编程的计算 SSD 来加速图深度学习算法。它在 SSD 旁放置了一块具备计算能力的 FPGA（Field Programmable Gate Array，现场可编程门阵列），使用 FPGA 进行图神经网络算法中数据的预处理操作，而且为相应的硬件设计了软件栈，负责将用户编写的代码编译为可在 FPGA 上运行的代码。阿里巴巴公司的 AliFlash V5 SSD 采用了近存储计算架构，在其关系型数据库场景中进行卸载加速，保持低延迟的同时提升了带宽。

2. 近内存计算

相较于 SSD 等外部存储介质，内存到处理器的距离更近，二者间的数据迁移开销也明显较低。但是，在一些带宽敏感的应用场景中仍然需要近内存计算以提升其性能。DRAM 是目前应用最广泛的内存单元，常见的基于 DRAM 的近内存计算方案主要采用 2.5D/3D 堆叠技术或 HBM（High Bandwidth Memory，高带宽内存）封装技术，在 DRAM 内部集成额外的计算单元。例如，学术界提出的 Max-PIM[25]通过在 DRAM 内部集成同或（XNOR）电路支持完整的逐位布尔逻辑计算，并基于此以并行的方式实现了在 DRAM 中搜索最大值和最小值的功能，可在大数据排序和图计算等应用领域发挥作用；Spacea[26]则利用 3D 堆叠技术将计算逻辑集成到 DRAM 中，从硬件设计和数据映射两个角度面向稀疏矩阵-向量乘法设计了存内计算架构；TRiM[27]发现了 DRAM 数据路径具有分层的树状结构，在 DDR4/5 的各层级分别加入计算单元来扩充 DRAM 数据路径，可用于优化个性化推荐系统。在工业界，三星公司于 2021 年公布了在 HBM 存储中集成浮点数计算阵列、流水线解码控制单元及本地寄存器文件单元的可编程方案，大幅度地提升了计算性能并降低了功耗[28]。

14.2.2　存算一体化

除了在存储器附近放置计算单元实现近存计算之外，某些存储介质在已有的外围电路的支持下可以同时完成计算和存储的功能，基于这些存储介质设计的计算架构被称为存算一体化架构。根据存算能力的差异及存储介质的不同特点，存算一体化技术可以分为 3 个层次，如图 14.3 所示。由上到下，存算设备的计算延时逐渐增长，读写速度依次减慢，而存储容量不断增大。

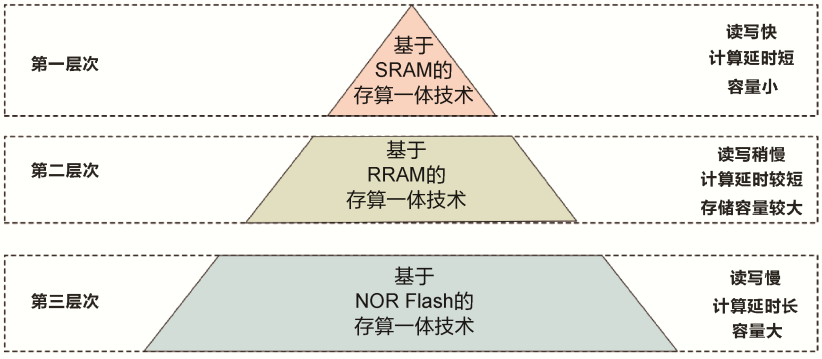


图 14.3　存算一体化技术分类

基于 SRAM 的存算一体技术：SRAM 通过切换晶体管的状态来存储数据，在通电的情况下，数据一直保持不变。由于 SRAM 的访问速度非常快，通常被用来作为计算机存储系统中的高速缓存。基于 SRAM 的存算一体技术可以直接在高速缓存中进行数据的计算，具有可靠性强、可扩展性高等优点。目前，该技术主要通过基于电压域和基于时域的模拟计算实现。在电压域方面，该技术一般会先使用 DAC（Digital-to-Analog Converter，数模转换器）将数字量转换为电压，然后通过电荷共享的方式实现计算，最后用 ADC（Analogto-Digital Converter，模数转换器）将计算结果的模拟量重新转换为数字量。研究者以拥有 6 个晶体管的 SRAM 和局部计算单元为基础，通过输入电压的方式实现了多比特数据乘法运算[29]。在时域方面，该技术通常会使用线性的路径延迟或者脉冲带宽来表示多位数字。有研究使用脉冲宽度调制的方式设计了一款基于 8 个晶体管的 SRAM 三明治结构的存算一体设备，可以实现 DNN（Deep Neural Network，深度神经网络）模型的运算[30]。另外，国内创新性企业苹芯科技发布了自研的可商用 SRAM 存算一体单元 PIMCHIP-S200。九天睿芯基于 SRAM 推出了可广泛应用于视觉领域的感存算一体架构芯片 ADA20X。

基于 RRAM 的存算一体技术：RRAM 是一种通过改变单元的电阻来存储数据的非易失性存储器，可以由一系列字线和位线连接 RRAM 单元组成交叉阵列结构。当向每条字线施加外部电压时，电流将依据基尔霍夫定律传递并汇集到位线。通过检测每条位线末端的电流，可以得到相应列的模拟电流总和。通过利用位线电流求和的特性并附加一些外围电路（如数模转换器、模数转换器等），RRAM 交叉阵列可获得以模拟量的形式执行本地向量-矩阵乘法运算的能力。近年来，基于 RRAM 的 DNN 存算一体化设计受到学术界和工业界的广泛关注。例如，PRIME[31]为 DNN 设计了基于 RRAM 的微体系结构和电路，用来支持模型不同层的运算，同时展示了如何将不同规模的模型参数映射到该架构的计算单元中。

ISAAC[32]设计了一个流水线架构，实现了 DNN 模型不同层之间的并行执行，而且定义了新的数据编码技术，减少了模数转换器的开销。在工业界，台积电正在积极推进基于 RRAM 的存算一体架构落地应用，昕原半导体也以 RRAM 介质为核心投入大量经费来研发存算一体芯片，它们都可以作为人工智能应用新的基础支撑。

基于 NOR Flash 的存算一体技术：NOR Flash 是一种传统的非易失性存储器，其基本的存储单元主要为浮栅晶体管，通过引入电荷实现数据的存储。由于 NOR Flash 的制造工艺已经非常成熟且成本低廉，基于 NOR Flash 的存算一体技术有广阔的应用空间，引起了许多研究学者的兴趣。例如，一种多电阻等级的低功耗双端浮栅晶体管，可用于模拟神经计算[33]；一种带有数字输入/输出接口和可配置精度功能的低功耗模拟电路，对感性电路、数模转换器和模数转换器进行了优化，可支持高能效的向量-矩阵乘法运算[34]。除学术界外，许多企业也致力于基于 NOR Flash 的存算一体技术的研发应用。美国的 Mythic 公司推出了模拟 AI 芯片 M1108AMP，可用于视频分析，视觉检测等领域；国内知存科技发布了基于 NOR Flash 的 WTM1001 智能语音芯片。

14.3　持久性内存

新型持久性内存以其高集成度、低静态功耗、数据掉电不丢失、性能接近 DRAM 等特性，为存储系统的发展带来了巨大机遇。然而，相比于磁盘、闪存等传统外存存储介质，持久性内存硬件具有明显的差异，如何构建持久性内存系统仍面临诸多挑战，这主要表现在以下方面：第一，软件栈开销高，持久性内存将持久性数据读写访问延迟从毫秒级降至纳秒级，而传统存储架构是针对外存设计的，持久化路径上的软件栈开销较高，难以发挥新型存储器件的性能优势；第二，一致性开销高，持久性内存提供主存层次的数据持久性，而处理器的片上缓存系统依然是易失性的，系统故障可能导致持久性内存上的持久性数据处于不一致的中间状态，而传统的一致性技术往往会引入过高的持久化延迟，严重降低持久性内存系统的性能；第三，空间利用率低，持久性内存的价格明显高于传统外存，传统主存空间管理机制容易引入主存碎片，主存碎片问题将显著降低持久性内存的空间利用率，增加系统的成本。围绕上述关键问题，相关研究机构及企业基于持久性内存重构了各类存储系统，例如文件系统、键值存储系统、分布式存储系统等。

14.3.1　文件系统

文件系统是操作系统中最基础的模块，它将设备存储空间以文件的形式组织

为可索引的文件目录树，从而方便用户存取数据。为兼容现有的应用程序，将非易失性内存组织成文件系统是一种重要的技术途径。一种简单的方法是直接使用现有的外存文件系统管理非易失性内存空间，该方案能够快速实现性能提升，但是，其缺陷是软件开销大，难以充分利用持久性内存的硬件优势。具体原因体现在以下两个方面。一方面，操作系统的统一抽象带来的开销将掩盖持久性内存的高性能特性。操作系统在对文件系统进行统一抽象的时候，会屏蔽不同介质上的差异性，以此提供统一的接口。但是传统的外存延迟高、带宽低，这与持久性内存的特点相悖，因此，传统的文件系统的抽象并不能充分发挥持久性内存的性能。另一方面，持久性内存的字节寻址特性不能被充分利用。传统的文件系统是基于外存存储设备设计的，这些设备的访问粒度均为块；而持久性内存的访问粒度为字节，因此直接接入会导致严重的数据写放大问题，同时还会引入一致性管理难的问题。为了解决上述问题，现有研究主要从一致性保障、移除缓存和用户态文件系统 3 个方面开展了不同尝试，进而实现了文件系统性能的显著提升。

1. 一致性保障机制

为了解决持久性内存字节访问粒度和现有文件系统基于块设备设计不匹配的问题，微软研究院在 2009 年提出了字节寻址的持久性内存文件系统 BPFS[35]。BPFS 使用树状结构作为文件系统的基本数据结构。为了减少系统树状结构中级联更新所带来的额外开销，BPFS 充分利用持久性内存字节可寻址的特性，提出了短路影子页（Short-Circuit Shadow Paging）方式原子更新数据。同时，BPFS 还将顺序性和持久性解耦，减少了刷新缓存的开销。

英特尔公司在 2014 年提出的 PMFS[36]基于持久性内存的 8 字节数据原子更新的特性，重新设计了元数据更新的策略，如图 14.4 所示。对于小尺寸数据更新，PMFS 使用原地原子更新和细粒度日志追加机制；对于大尺寸数据更新，PMFS 采用了 Undo 日志和 CoW 混合的方式保证数据的一致性。

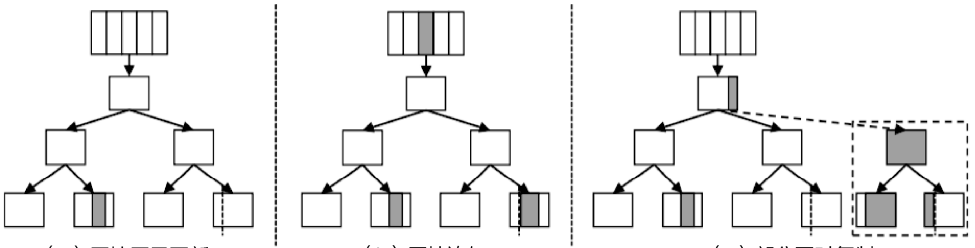


图 14.4　PMFS 的不同更新策略

美国加利福尼亚大学圣地亚哥分校提出的NOVA[37]使用日志的方式来组织元数据。对于元数据的修改，NOVA 采用追加修改内容和原子更新指针的方式保证元数据的一致性。对于数据部分，NOVA 采用 CoW 机制。对于 rename 等涉及对多个日志结构进行修改的复杂操作，NOVA 采用日志机制保证这类操作的崩溃一致性。

2. 移除缓存

由于持久性内存和 DRAM 的访问性能类似，操作系统针对外存访问设计的 DRAM 缓存不再高效，其引入的冗余的数据复制还会影响持久性内存的性能。相关研究从移除页缓存和移除元数据缓存两方面来解决这一问题。

Ext4、BtrFS 等传统文件系统增加了兼容持久性内存的直接访问模式，该模式允许用户直接访问持久性内存中的数据来移除页缓存。PMFS、NOVA、BPFS 等文件系统则使用内存映射的方式来移除页缓存。美国得克萨斯农工大学提出的SCMFS[38]对数据组织进行优化，通过页表映射的方式，使文件系统中的文件有连续的地址空间，从而提高程序访问的性能。

byVFS 则是直接在物理文件系统上对元数据进行操作，这样移除了元数据缓存，发挥持久性内存的优势以提升性能。

3. 用户态文件系统

移除缓存能够一定程度上缓解 VFS 对于持久性内存的性能限制，但是 VFS 仍然会带来很多不必要的开销，如复杂的软件执行逻辑、粗粒度的锁管理等。Aerie[39]是首个基于用户态的持久性内存文件系统，它绕过了 VFS，充分发挥持久性内存的性能。Strata[40]也是在用户态实现的一种混合介质文件系统，它能够同时管理了多种不同的存储设备，如持久性内存、SSD、HDD 等。清华大学提出的KucoFS[41]可以在用户态访问持久性内存文件系统的文件数据，但仍将复杂的元数据处理逻辑卸载至内核完成。

14.3.2　键值存储系统

键值存储系统提供了针对单个键值的查询、插入、更新、删除等操作，因其良好的扩展性和实时性被广泛应用于网页检索、电子商务、社交网络等领域。持久性内存为构建大容量、高性能、低时延的键值存储系统提供了强有力的硬件支撑。目前，围绕持久性内存构建键值存储系统主要从索引结构、空间管理等方面展开。

1. 持久性索引结构

索引结构是键值存储系统中的重要模块，通过键快速查询对应的数据项辅助键值存储系统。索引结构主要分为两类，一类是散列表，其特点是扩展性好、查

询开销低，但仅支持单点查询；另一类是树状索引（例如 B+树等），其特点是将键值对进行有序组织，支持高效的范围查找，但是查询速度慢，树状结构维护开销大。目前，基于持久性内存构建数据结构主要需要考虑以下几个问题。

（1）读写不对称

持久性内存的读写延迟及带宽具有显著的不对称特征，并且存在耐久性方面的问题。标准的 B+树存在频繁的排序、平衡等操作，引入了大量的写开销，进而导致了严重的性能问题及磨损问题。

（2）一致性优化机制

在突然断电或者系统崩溃时，持久性内存中的数据结构可能出现不一致的状态，导致数据结构丧失查询功能，甚至丢失数据。因此，持久性内存索引结构必须提供高效的一致性保障措施。

针对索引结构的特点，研究人员通过系统提供的持久化原语，设计了更加精细的一致性更新策略。例如，CDDS-Tree[42]是惠普实验室针对非易失性主存设计的一致性 B+树。它为每个数据项分配了一个版本号区间。更新操作为每个更新的数据项生成一个新的版本，并将其插入到树节点的合适位置。针对删除操作，CDDS-Tree 通过版本号的设置便可轻松完成，且整个过程不影响旧数据项。在适当的时机，CDDS-Tree 才会回收这些旧数据项，从而保证它在发生系统错误时能找到正确的数据版本。然而，基于版本号的一致性更新机制会引发严重的写放大问题。因此，后续很多工作还进一步尝试了从不同方面减少 B+树的一致性开销，例如引入间接查询层、允许中间不一致状态等。

2. 空间管理

非易失性主存的空间管理主要包含分配和释放操作。在系统执行主存分配/释放操作的过程中，系统错误可能导致上述操作处于不一致的非法状态，从而导致系统重启后出现主存泄露或者野指针访问等问题。此外，非易失性主存的数据在系统关机后会被保留下来，同理，持久性内存碎片也同样被保留下来。如果系统缺乏一种有效的主存碎片处理机制，主存碎片会持续积累，严重降低非易失性主存的空间利用率。

为了减少主存碎片，英特尔的 PMDK 针对不同大小的主存分配操作采用了不同的分配策略。对于小于 256 KB 的分配操作，它采用分离适配策略，通过使用 35 种不同尺寸的分配类，将每个 256 KB 的超级块切割成多个更小的尺寸为 8 字节倍数的主存块，满足一定区间的主存分配操作。虽然细粒度的分离适配策略在一定程度上减少了小于 256 KB 的分配操作所产生的主存碎片，但是大于 256 KB 的分配操作依然易于引入较高的主存碎片。

为了消除更细粒度的主存碎片，清华大学提出的 LSNVMM[43]将整个非易失性主存组织成一个日志结构。对于所有分配操作，它将新数据直接添加到日志末尾，而不是将主存超级块切割成固定大小的主存块，从而消除了大部分的内部碎片。此外，它通过迁移合法数据，回收未被使用的主存空间，将其组织成更大的连续区域，达到了消除外部碎片的目的。并且，LSNVMM 的碎片清理过程不需要中断整个系统的正常运行，对整个系统的性能影响较低。

14.3.3　分布式存储系统

随着持久性内存和高速网络技术（例如 RDMA 等）的发展，分布式场景下的网络和存储的硬件性能都得到了大幅度提升。但是直接将持久性内存和 RDMA 整合到现有的分布式存储系统中并不能发挥两者的性能。这是由于现有的分布式软件栈和分布式协议都是基于传统的网络和存储设备设计的，存在冗余和低效等问题。现有的研究对软件栈、分布式协议等进行重新设计，以充分发挥持久性内存和 RDMA 在分布式场景下的性能。

1. 软件栈

虽然传统的软件栈的延迟开销对于外存访问而言并不明显，但随着持久性内存和 RDMA 的出现，传统软件栈的冗余设计所带来的开销变得不可忽视。清华大学提出的 Octopus[44]（图 14.5）通过 RDMA 直接访问统一的分布式持久性共享内存池，以减少数据的冗余复制，并充分利用硬件的读写带宽；提出客户端主动式的数据 I/O 以减少了服务器的 CPU 和网络负载；设计自识别远程过程调用协议实现低延迟的元数据访问。

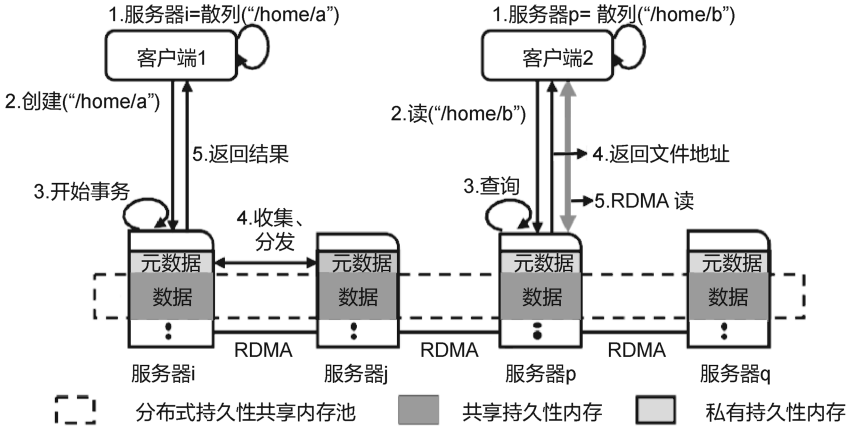


图 14.5　Octopus 的架构

Orion[45]使用 RDMA 网络将 NOVA 单机文件系统扩展到分布式场景下。Orion 通过维护多个副本的元数据和数据以提高容灾能力，并在读取远端的日志时使用单边的 RDMA 原语，降低了服务器的处理压力。

2. 分布式协议

分布式协议是一组通信协议，用于在众多计算系统之间实现开放的、基于标准的互操作性，主要包括副本协议、缓存一致性协议和事务协议。

（1）副本协议

Mojim[46]基于 RDMA 实现了持久性内存系统的数据容错。主节点上的数据通过 RDMA 传输到镜像节点中的持久性内存中，减少了主节点的 CPU 刷写开销。同时镜像节点在后台将数据异步地备份至多个备份节点，以提供更高的可靠性。

（2）缓存一致性协议

Hotpot[47]基于分布式持久性共享内存的抽象为应用提供了简易的编程接口，使单节点应用可以充分利用分布式存储资源。Hotpot 采用本地缓存进行数据访问加速，并基于 RDMA 提出了两种分布式缓存一致性提交协议：第一种通过多阶段提交，支持不同节点对同一个缓存页进行并发的写操作；第二种通过集中式的锁服务，保证对于一个页，同一时刻只有一位写者。

（3）事务协议

微软提出的 FaRM[48-49]针对 RDMA 重新设计了基于乐观并发控制的分布式事务协议，其核心思想包括 3 个方面：首先，FaRM 采用无副本的协调者，这既消除了协调者状态的复制开销，又简化了系统恢复；其次，FaRM 将副本和事务合并到一层，协调者直接与所有的主副本和从副本进行通信，减少了系统软件开销；最后，FaRM 通过 RDMA 单边写原语将数据推入从副本，由此降低延迟。

14.4 在网存储

新型可编程网络设备支持软件定义网络包的处理过程，为存储系统设计带来了巨大机遇，其中最典型的代表是可编程交换机和智能网卡[50]。

可编程交换机的核心是专门定制的可编程网络处理芯片。该芯片基于可重配置的匹配表架构（Reconfigurable Match Table Architecture），包括多个高速硬件流水线。用户可以编程如下的 3 个部件：解析器（Parser），规定网络包的协议格式；寄存器数组（Register Array），高速 SRAM，用于存储数据，一般只有 10～20 MB 空间；动作-匹配表（Match-Action Table），规定当流经交换机的网络包包头元素满足特定条件时（如 UDP 端口号为 11）时，交换机对该网络包进行修改和路由，

并读写寄存器数组。现有的可编程交换机能够以线速转发网络包，聚合带宽可以达到 10 Tbit/s 以上。

智能网卡由网卡芯片和可编程硬件组成，其中可编程硬件主要有 3 种类型：ARM CPU、NPU（Network Processing Unit，网络处理单元）和 FPGA。这些可编程硬件可以处理网卡芯片接收/发送的网络包，其中 ARM CPU 的处理能力最弱而编程难度最低，FPGA 与之相反。

利用可编程交换机和智能网卡，研究人员设计了高性能的在网存储系统，在网络路径上执行存储系统的核心任务，例如数据协调、数据调度、数据缓存等[50]。

14.4.1　在网数据协调

可编程交换机是服务器之间通信的中枢，适合进行分布式存储系统中的数据协调，其中典型的代表有 Concordia[51-52]和 SwitchTx[53]。

Concordia 是清华大学提出的分布式共享内存系统。在分布式共享内存系统中，为了减少数据的远程访问，每台服务器具有本地缓存；如何保证不同服务器缓存之间的一致性是经典问题。现有的缓存一致性协议需要服务器之间进行昂贵的分布式协调，引入额外的网络往返和 CPU 开销，极大地降低了系统在数据共享时的性能。针对该问题，Concordia 利用可编程交换机，提出在网缓存一致性协议，如图 14.6 所示。具体来说，Concordia 在可编程交换机中记录缓存块的元数据，包括缓存块状态、持有缓存块的服务器列表等；当收到缓存一致性请求时，交换机根据对应元数据，准确地将请求路由至目的服务器集合。此外，Concordia 在可编程交换机内设计了高效的读写锁，用于序列化并发冲突的请求。由于可编程交换机内存容量有限，Concordia 设计了一种所有权转移机制，只让交换机处理活跃缓存块的一致性；对于不活跃的缓存块，它们的一致性由服务器维护。

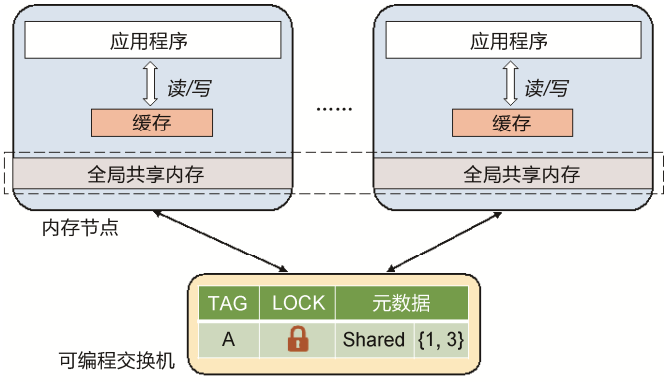


图 14.6　Concordia 的架构

SwitchTx 是清华大学提出的分布式事务系统。分布式事务系统将数据划分在不同服务器中，并通过分布式并发控制和提交协议向应用提供事务语义。这些协议存在很高的协调开销，包括网络通信、CPU 排队等。这些开销在事务提交的关键路径上，会导致事务的高延迟和高冲突，严重降低系统性能。针对上述问题，SwitchTx 设计了可扩展的在网协调机制，将分布式事务协调过程抽象为多次“收集-分发”操作的组合，并将这些操作卸载到集群中的多个可编程交换机中，由此减少事务执行的网络跳数以及 CPU 开销。此外，SwitchTx 还将事务语义与网络流控结合，重新设计了事务的准入控制机制。SwitchTx 能有效降低分布式事务处理的网络开销，提升系统吞吐率并降低事务延迟。

14.4.2 在网数据调度

部分研究工作利用智能网卡调度数据请求，其中典型代表是 AlNiCo[54]。

AlNiCo 是清华大学提出的事务调度系统。近些年，事务处理系统的发展有两个趋势：第一，网络带宽有了明显的改善，单机系统有能力承载大量网络请求；第二，现代服务器的 CPU 核心数量越来越多，导致多核之间事务处理存在资源争用。这两个趋势共同构成了一个关键问题：如何将每个事务请求调度到最合适的 CPU 核芯上执行，以提高多核服务器的并行事务处理能力？然而调度是把双刃剑，在享受调度带来的收益的同时需要付出额外的计算开销。现有基于 CPU 的调度方法存在较大开销，难以满足事务处理的低延迟要求。而新兴的智能网卡为事务调度提供了机会：智能网卡位于请求处理的关键路径并且具有计算加速能力。AlNiCo 利用基于 FPGA 的网卡将事务请求智能地调度到不同 CPU 核心，以降低事务处理过程中的冲突，如图 14.7 所示。具体来说，AlNiCo 将事务请求、CPU 核心状态以及负载特征抽象为适合 FPGA 处理的向量，利用 FPGA 快速做出调度决策。AlNiCo 能够支持多种并发控制协议，以极低的延迟开销完成请求调度，降低事务处理时的冲突，提升系统吞吐率。

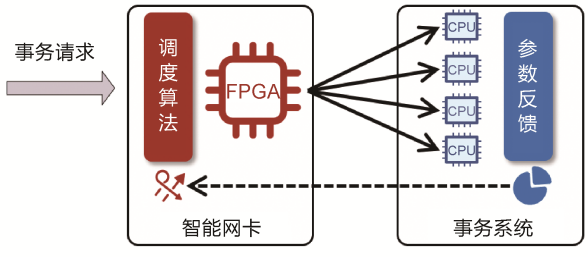


图 14.7　AlNiCo 事务调度系统

14.4.3　在网数据缓存

可编程交换机和智能网卡的内存可以用于缓存存储系统的数据，由此减少网络往返开销，并提供高吞吐率服务。其中的典型代表是 NetCache[55]。

NetCache 是美国约翰斯·霍普金斯大学提出的分布式键值存储系统负载均衡方案。分布式键值存储系统将数据分散至多台服务器中，但由于数据的访问经常具有倾斜性，即部分热点数据会被经常访问，分布式键值存储系统会出现服务器之间负载不均衡的情况：某些服务器过载，无法及时处理用户的请求；而某些服务器接收的请求过少，资源空闲。针对该问题，NetCache 提出使用可编程交换机缓存分布式键值存储系统中的热点键值对，以缓解负载不均衡问题，具体架构如图 14.8 所示。NetCache 在可编程交换机中实现了热点检测模块，能够快速地判断热点键值对，并将其存储在寄存器数组中。当交换机收到用户的读请求时，若命中，则将交换机中的键值对返回；否则，该读请求会被路由至对应服务器；当交换机收到用户写请求时，若命中，则把交换机中的键值对标为无效，最后将写请求路由至对应服务器。由于交换机的吞吐率极高，NetCache 能够高效处理大量热点读请求，以保证服务器之间达到负载均衡的状态。

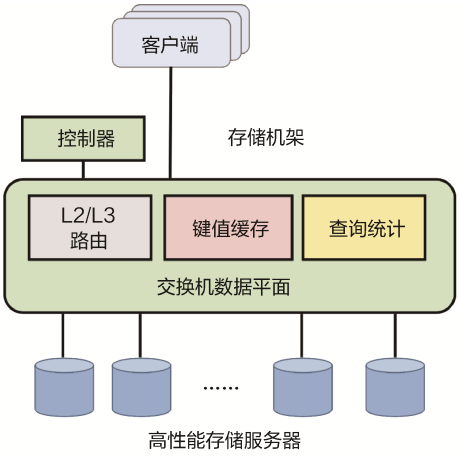


图 14.8　NetCache 架构

14.5　智能存储

近些年，以深度学习为代表的 AI 技术在诸多领域取得了令人振奋的成果：DeepMind 公司的 AlphaGo 在 2017 年击败了当时世界排名第一的围棋冠军柯洁；OpenAI 公司于 2022 年 11 月推出了 AI 聊天机器人 ChatGPT，其在自动问答、文

本生成等方面表现出媲美真人的能力。AI 的发展也为数据存储带来了新的机遇和挑战：一方面，运用 AI 技术可加速存储系统，即 AI for Storage；另一方面，AI 也需要高性能的存储系统进行支撑，即 Storage for AI。

14.5.1 AI for Storage

现有的研究工作利用 AI 技术，极大地提升了存储系统的适应能力和性能，主要包括学习索引、参数自动调优、启发式算法优化。

1. 学习索引

索引是存储系统的核心组件，用于维护数据的位置信息，常用的索引结构（例如 B+树）空间占用大、内存访问多。为了解决以上问题，美国麻省理工学院的研究人员于 2018 年提出了学习索引（Learned Index）的概念[56]，使用简单的模型替换原来的索引节点，其核心思想是让模型学习键的累积分布函数，利用模型预测键的分布位置，以计算代替在索引节点中搜索的过程。图 14.9 展示了学习索引的示例，当搜索键为 510 的数据时，直接计算 $H(x)$函数，获得目的数据的位置下标 1。相比于传统索引，学习索引的空间开销小，且节省了多次内存访问，可达到极低的访问延迟。原始的学习索引不支持插入、删除操作，面临着重新训练代价高的问题。为此，研究人员提出了多种方法，其中较常用的是引入临时缓冲区[57]：该方法将新插入的数据项写入临时缓冲区，然后周期性地将缓冲区中的数据通过重新训练合并到原有索引中。

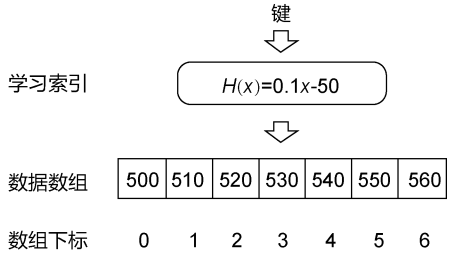


图 14.9 学习索引的示例

研究人员尝试将存储系统中的索引模块替换成学习索引，其中具有代表性的研究工作包括 Bourbon[58]和 XStore[59]。

Bourbon 针对的是 LSM 键值存储系统。如第 5 章所述，LSM 系统在外存设备上被组织成多个 SSTable，每个 SSTable 中包含了一定数目的有序数据项。为了减少 I/O 操作次数，LSM 在 SSTable 加入索引块（Index Block）记录索引信息。Bourbon 将 SSTable 的索引块替换成学习索引，以加速数据的查询。Bourbon 提出

了若干高效使用学习索引的指导原则，例如倾向于对底层的 SSTable 进行学习，因为它们生命周期更长，不会造成频繁的模型失效和重建。

XStore 针对的是基于 RDMA 的有序键值存储系统。该类系统通常在服务器通过树状结构维护数据，客户端采用 RDMA 单边原语直接查询数据。为了减少网络往返次数，键值存储系统在客户端构建索引缓存（Index Cache），缓存键到数据远程地址的映射。传统的索引缓存由 B+树等结构构成，会导致两个问题：首先是会占据客户端的大量内存空间；其次是引入了多次内存访问，延迟高，尤其是考虑到 RDMA 的网络往返时间极低。为此，XStore 利用学习索引构建了高性能索引缓存，达到了极佳的性能与空间占用的权衡：与传统方式相比，XStore 可以以 20%的性能代价来降低 99%的内存占用。

2. 参数自动调优

存储系统包含大量可配置的参数（例如 Ceph 分布式文件系统的参数数目超过 1500 个），这些参数会极大影响存储系统在不同负载和场景下的性能。传统的人工调优方案十分耗时，且需要相关人员具有丰富的经验；此外，调优结果很难移植到其他硬件平台上。为此，一些研究工作采用机器学习的方式对存储系统进行参数自动调优。这里主要介绍清华大学提出的 Sapphire 系统[60]。

Sapphire 为分布式存储系统推荐配置参数，其架构如图 14.10 所示，主要组成部分包括控制器和机器学习模型。控制器接受用户的设置，包括集群配置、最大迭代次数等。控制器管理分布式存储集群，通过执行控制命令使得配置参数值的更改在系统中生效。此外，控制器通过测试工具对存储系统进行测试，并将结果存入数据库中。机器学习模型由排名模型和优化模型组成。排名模型处理所有的测试结果，并根据它们对系统性能的影响生成参数排名列表。基于参数排名，优化模型使用高影响力的配置参数生成搜索域。在该搜索域中，优化模型找到最优的参数配置。Sapphire 采用了一种基于模拟的方法，通过小规模的测试集群来学习和建立优化模型，然后根据结果为大规模在线集群推荐最优参数配置。

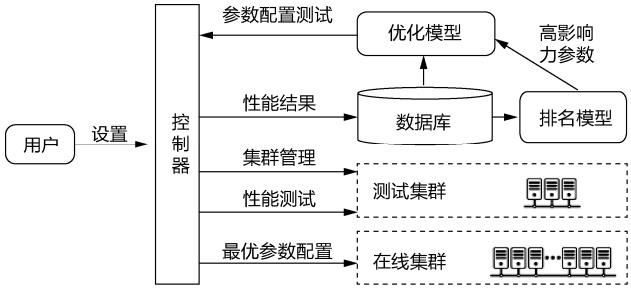


图 14.10　Sapphire 的架构

3. 启发式算法优化

传统存储系统采用各类启发式算法，以提升系统效率，例如缓存替换算法、冷热分离算法等。这些启发式算法无法在所有场景下都获得较佳的效果。由于这些启发式算法的核心都是进行预测，例如预测数据冷热情况，因此十分适合运用AI技术。这里主要介绍谷歌提出的Llama[61]及芝加哥大学提出的LinnOS[62]。

Llama是一款针对C++程序的内存管理系统。在现有服务器应用中，C++对象的内存分配十分频繁，且生命周期短暂。因此，当采用大页（Huge Page）优化内存访问性能时，内存碎片化极其严重。Llama在冷启动时对内存分配进行采样（包括上下文、用户层的一些数据等），预测对象的驻留时间，并利用系统运行时信息进行模型的持续训练。Llama基于预测结果组织数据堆，以减小碎片化。为了解决推理时间过长的问题，Llama将预测结果短暂地缓存在散列表中。

LinnOS旨在提升SSD存储系统的性能可预测性。SSD内部存在诸多后台操作，例如垃圾回收、读修复等，会严重影响读写请求的访问延迟。LinnOS通过神经网络对SSD的访问延迟进行预测，并当某次I/O的预测延迟较高时，撤销此I/O，并将它重定向至其他SSD，以此避免产生延迟尖峰。LinnOS采用二元分类以提高预测的准确性，同时使用当前I/O队列长度、历史I/O队列长度、历史I/O延迟作为预测的输入，以减少神经网络的参数数目和计算量。但是由于LinnOS采用离线预测，因此无法很好地适应工作负载的变化。

14.5.2 Storage for AI

训练数据集和模型参数的规模急剧增长，给存储系统带来了挑战。一方面，加速器（如GPU、TPU）的存储容量增长缓慢，难以满足机器学习任务日益增长的存储容量需求。另一方面，传统存储系统的读写性能远低于加速器的处理性能，使得存储成为性能瓶颈。因此，研究人员提出了多种专用存储系统，以高效支持机器学习任务中的各个阶段，包括数据加载、数据预处理及模型训练等。

1. 数据加载阶段

训练所需的数据集通常位于本地外存或远端存储系统中。在训练过程中，数据集被分批加载到加速器的内存中。现有系统常采用预取和缓存等方式提高数据加载的效率，其中典型的研究工作有NoPFS[63]和SHADE[64]。

NoPFS是一款用于机器学习任务的数据加载框架，由苏黎世联邦理工学院提出。在分布式训练过程中，频繁的随机小数据样本访问会导致共享文件系统的阻塞，降低训练数据集加载性能。现有的数据加载框架通过修改访问模式或双缓冲等方式来提高数据加载效率，但是这些方式破坏了数据集训练的随机性或带来了

额外硬件开销。针对该问题，NoPFS 利用伪随机数生成器的伪随机性，生成近似最优的预取和缓存策略。具体来说，NoPFS 使用给定的随机种子，预测数据集样本的访问时刻，并据此进行访问模式分析和性能建模，得到近似最优的样本预取顺序和路径，最后生成与之对应的缓存策略。

SHADE 是一款面向机器学习任务的数据缓存系统，由弗吉尼亚理工学院等机构提出。机器学习任务的数据访问模式对现有缓存策略不友好：机器学习任务采用随机采样策略进行训练，导致数据的局部性差。针对此问题，SHADE 基于数据样本重要性设计了新的数据集采样算法，在单轮中多次使用重要的数据样本，以提高数据的局部性。此外，SHADE 计算训练数据的重要性程度，缓存重要的数据样本，提高缓存命中率。

2. 预处理阶段

预处理速率需不低于模型训练速率，才能减少对模型训练的影响，提高加速器的硬件利用率。预处理系统的典型代表是 tf.data[65]。

tf.data 是谷歌提出的针对机器学习任务的数据预处理系统，其架构如图 14.11 所示。机器学习任务使用 CPU 进行数据预处理，再将预处理后的数据传输到训练专用的加速器（如 GPU、TPU）中。但 CPU 的处理速度远低于加速器，导致预处理成为瓶颈。为解决此问题，tf.data 针对数据预处理设计了专用的 API 和运行时。具体来说，在 tf.data 中，中心化的任务分发器将预处理任务指派到多个节点执行，训练节点直接从多个预处理节点上获取预处理后的数据。此外，tf.data 还提供缓存操作，用户可以将预处理的结果缓存到本地存储设备中，由此降低重复预处理的开销。由于不同模型和数据集的预处理模式存在差异，tf.data 在运行时自动调优预处理的并行度和内存缓冲区大小，以最大化预处理性能。

3. 模型训练阶段

模型参数规模的增长速度远高于加速器存储容量的增长速度。为解决该问题，研究人员提出了在加速器间以分布式形式存储模型数据、使用异构存储介质扩充容量等方法。典型工作有针对稀疏模型的 Fleche[66]和 PetPS[67]，以及针对稠密模型的 ZeRO[68]和 Mobius[69]。

Fleche 是由清华大学提出的向量化（Embedding）表缓存方案。基于深度学习的推荐模型含有多张包含海量稀疏参数的向量化表，其带来的大量不规则稀疏 DRAM 访问已成为推荐模型的主要性能瓶颈。现有系统在 GPU 内存上缓存热点参数以减少 DRAM 访问。然而，现有的缓存方案空间利用率低、GPU Kernel 维护开销高。为解决上述问题，Fleche 使用基于霍夫曼编码的方式重编码特征 ID，并将所有的 embedding 表进行统一管理，以捕捉全局热点、提升命中率。此外，

Fleche 提出一种自识别的 Kernel 融合技术，减少 Kernel 的维护开销。

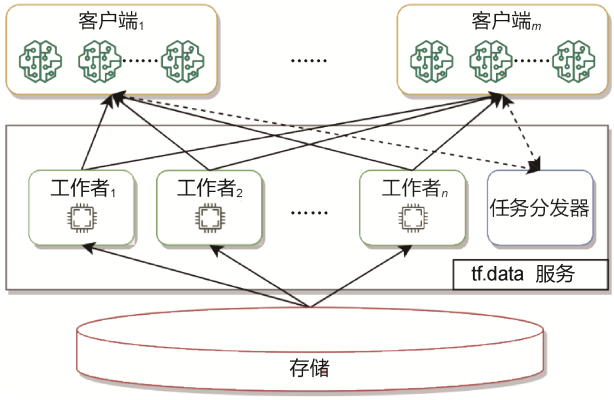


图 14.11　tf.data 的架构

PetPS 是由清华大学提出的基于持久性内存的参数服务器系统。现代工业级稀疏大模型的参数量已达万亿级。为提供实时的参数访问，现有方案将这些稀疏大模型存储于多台参数服务器的 DRAM 之中。然而，随着模型参数量的不断膨胀，DRAM 带来的存储成本高、崩溃恢复时间长等问题日益严重。PetPS 使用高性价比的持久性内存存储稀疏大模型的参数。为了克服持久性内存读延迟高的问题，PetPS 设计了一个专用散列索引，通过预取等机制最少化持久性内存的读取次数。此外，PetPS 将参数序列化任务卸载至网卡，从而提升 CPU 效率。

ZeRO 是由微软提出的大模型训练系统。在传统的数据并行训练模式中，每张 GPU 中存储着完整的一份模型，数据被切分到不同的 GPU 中用于训练。但随着模型规模的不断增加，单张 GPU 无法存储完整的模型。为解决此问题，ZeRO 将单个模型参数分片存储到多张 GPU 中。在模型某一分片训练前，含有该分片的 GPU 将模型参数广播到所有 GPU。一张 GPU 完成训练后，梯度被传输到负责存储该分片的 GPU 中进行梯度聚合，最后进行参数更新。

Mobius 是由清华大学提出的针对消费级 GPU 的大模型训练框架，其架构如图 14.12 所示。在消费级 GPU 服务器中，通信链路带宽远低于 GPU 内存带宽，且多张 GPU 共享通信链路带宽。因此，ZeRO 频繁的集合通信会带来巨大开销，影响训练效率。为解决该问题，Mobius 利用服务器中的异构存储资源以满足大模型训练的存储需求，并使用流水线训练的方式降低训练的通信开销。此外，Mobius 对计算和通信进行建模，使用混合线性规划的方法得到最优的模型切分方案。

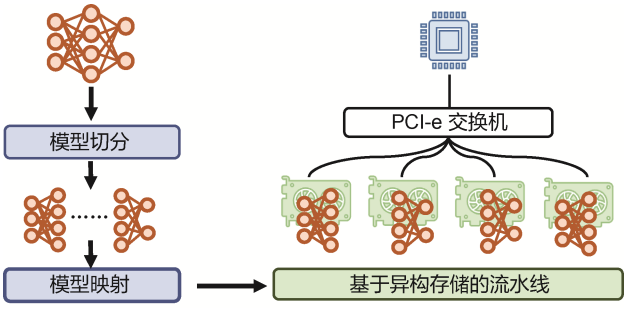


图 14.12　Mobius 的架构

14.6　边缘存储

随着物联网及 5G 网络等技术的成熟与普及，边缘设备急剧增加，所产生的边缘数据呈现爆炸式增长。为了高效存储这些数据，边缘存储这一新型范式正发挥重要作用。边缘存储将数据分散保存在邻近的边缘存储设备，可以大幅度缩短产生数据的终端、计算设备、存储设备之间的物理距离，提供高速低延迟的边缘数据访问能力，为海量物联网数据存储带来了新的机遇。然而，边缘存储也面临供电、空间、算力、通信等方面的限制，实时存储和处理边缘数据面临严峻的考验。

为克服上述挑战，工业界和学术界对边缘存储系统进行了广泛的设计和研究。在工业界，腾讯设计了适用于边缘场景的存储一体机 TStor；三星推出了具备计算能力的存储设备 Smart SSD；华为采用超融合设备 FusionCube 作为边缘数据存储的基础设施；阿里巴巴研发了 OpenYurt 软件平台支持边缘存储管理。在学术界，清华大学、加利福尼亚大学洛杉矶分校及哥伦比亚大学等研究机构在边缘存储设备[70]、边缘存储软件和协议[71-73]，以及边缘数据的组织和检索[74]等方面对边缘存储系统展开了相关研究。

14.6.1　边缘存储设备

边缘存储设备是保存数据的物理载体，其固有的硬件资源数量是影响数据存储性能的关键因素。然而，边缘存储设备的存储和计算资源有限。另外，数据感知端、存储单元和计算单元之间较长的数据路径进一步降低了数据实时存储和访问的能力。为了解决上述问题，研究人员面向边缘场景提出“感存算融合”的概念，将感知接口、存储单元和计算器集成在一个设备上，通过减少数据物理传输路径的方式降低数据的存储和访问延迟。

研究工作者对感存算融合设备进行了深入的探究，其中具有代表性的工作是

TH-iSSD[75]。TH-iSSD 是清华大学设计的感存算融合设备，其架构如图 14.13 所示。首先，该设备将数据感知器、存储单元和计算加速器的控制逻辑集成在一个硬件控制器中，使得数据移动的成本降至最低。同时，TH-iSSD 具有高度可重构性，在给定部署需求下，TH-iSSD 的传感元件和计算加速器可以被替换，以满足电源和应用逻辑的要求。其次，高速闪存存储器具备读写性能不平衡、及写前需擦除等特征，使得存储器的性能得不到充分利用。为了应对上述挑战，TH-iSSD 引入了优先级感知的并行 I/O 调度机制，请求调度器以细粒度的方式动态地对 I/O 请求重新排序，充分利用了存储器的内部带宽。最后，现有的感存算融合设备通常将存储器作为没有文件系统的原始块来管理数据，需要修改大量的主机代码来发挥设备的能力。为了改变现状，TH-iSSD 提供了易于使用的文件抽象，使用户不用考虑数据放置而只关注应用的计算逻辑。

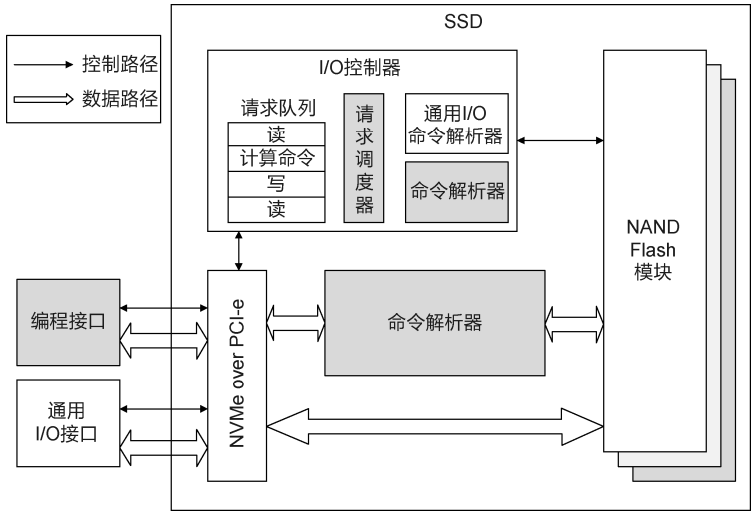


图 14.13　TH-iSSD 设备的架构

14.6.2　边缘存储 I/O 栈

边缘存储 I/O 栈负责处理边缘设备中数据的 I/O 请求，其由具有层次结构的软硬件组件共同组成，包括用户空间、文件系统、页缓存、通用块层、设备驱动与块设备等，其主要目的是为应用程序提供与存储设备（如硬盘驱动器、固态硬盘等）的数据交互功能。

对边缘存储 I/O 栈进行高效的抽象化设计与管理极具挑战性，主要原因在于边缘设备的异构性。边缘设备通常具有不同的硬件配置和操作系统，因此，边缘存储 I/O 栈需要为不同的设备设计匹配的驱动程序与应用接口，使应用程序可以

运行在不同的设备上。

研究人员尝试为异构的设备设计统一的存储 I/O 栈，其中具有代表性的研究工作是 λ-IO[76]。λ-IO 是清华大学为边缘存储设备设计的 I/O 栈，其架构如图 14.14 所示。λ-IO 从接口、运行时与调度 3 个方面展开设计，实现了高效管理计算和存储资源的目标。在接口方面，λ-IO 在主机和设备上扩展了 I/O 栈，为应用程序扩展了额外的编程接口。除了支持原本的 I/O 操作以外，应用程序还可以在读取和写入数据期间提交 λ 请求来定制计算逻辑、加载和调用计算逻辑。通过扩展接口，开发人员仍然可以使用他们熟悉的编程风格来访问和处理文件数据，从而隐藏计算任务的执行和调度细节。在运行时方面，λ-IO 在传统的 I/O 栈基础上，通过对 eBPF（Extended Berkeley Packet Filter，扩展的伯克利包过滤器）进行改进，使其跨越了主机和设备的边界，支持指针访问与动态长度循环，并引入额外的信息以支持动态验证。在调度方面，λ-IO 采用动态请求调度，它针对内核与存储设备的请求执行时间进行建模，快速地将请求发送到更快的一侧以实现高效调度的目标。

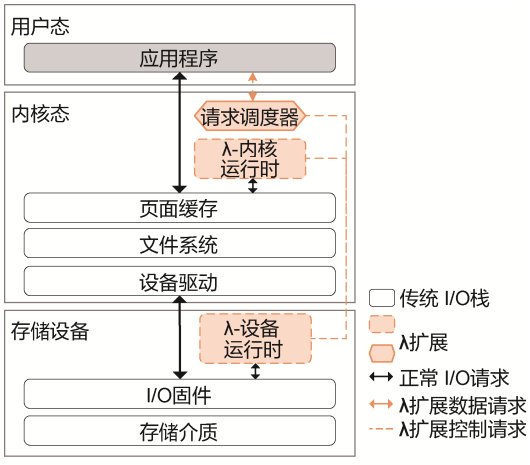


图 14.14　λ-IO 的架构

14.6.3　边缘数据组织与检索

边缘数据的组织和检索是指在边缘存储架构中对存储在边缘设备或边缘服务器上的数据按一定的结构进行存储，并依据某种方法从大量数据中查找的过程。分布式键值存储是一种以键值对形式组织数据，且其可以通过键来查询所需数据的典型存储系统。由于其低延迟的存储和访问特性，非常适合用于边缘数据存储系统。现有的分布式键值存储常采用随机数据放置策略，如一致性散列和散列槽分片。然而，随机数据放置策略忽略了数据存储客户端及其发出的键值访问请求

具有随时间变化的特征，导致使用该策略的系统平均请求延迟较长。

为了缓解上述问题，许多研究工作针对数据放置策略开展优化，其中具有代表性的研究工作是 PortKey[77]。PortKey 是一款根据客户端时变移动性和数据访问模式执行动态自适应数据放置的分布式键值存储，它解决了现有分布式键值存储设计无法适配边缘存储随环境高动态变化的问题。PortKey 明确了边缘应用程序的时变移动性和延迟模式，将数据放置形式化为一个在线优化问题，采用贪心算法以实现与最优决策相近的快速放置。PortKey 主要包括数据收集和数据放置决策两个过程。在数据收集时，PortKey 采用一系列轻量级技术生成简洁的延迟草图。具体来说，PortKey 首先探测客户端/服务器之间的端到端延迟，在随后的时间窗中，当客户端移动时通过位置感知技术触发重新收集延迟信息的功能。在数据放置决策过程中，PortKey 采用自适应求解器单独地对键进行操作，并通过贪心策略分配，以此解决主机存储约束。该分配优先考虑对整体数据存储性能影响最大的键值对，即在存储需求和访问频率中做出权衡。例如，一些被频繁访问但仅由单个客户端访问的键，直接跳过自适应求解器。这种贪婪的启发式算法放弃了最优放置，以换取快速放置。

图 14.15 展示了 PortKey 的工作流。PortKey 作为一个软件模块被集成在现有数据存储系统之上。其中，客户端数据存储用于跟踪每个键值访问请求，并支持智能地监控客户端/服务器延迟。这些数据访问和延迟信息将以应用程序定义的窗口大小上传到自适应放置引擎。在接收到所有客户端的信息之后，自适应放置引擎首先计算全局网络距离矩阵。放置求解器将该矩阵与客户端访问键的集合相结合，执行快速近似全局最佳的键值放置，并向适当的数据服务器发出迁移指令。

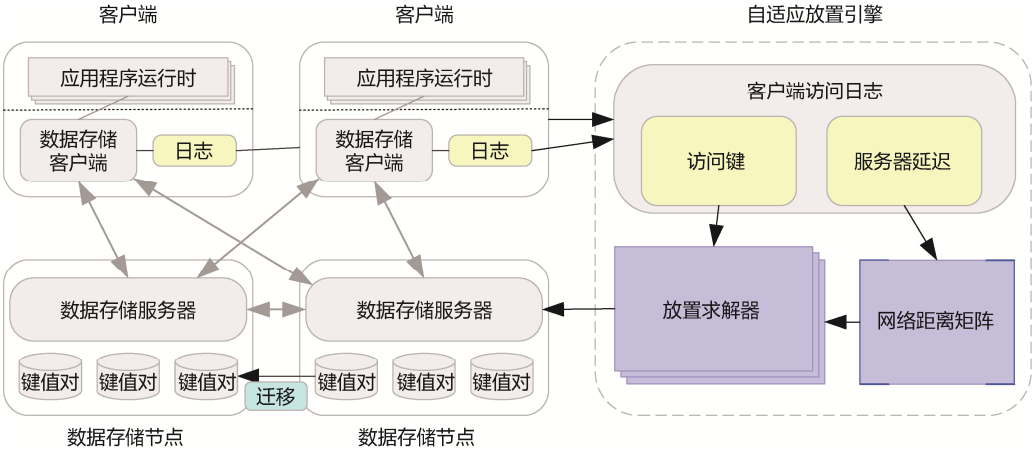


图 14.15　PortKey 的工作流

14.7　区块链存储

区块链涉及密码学、P2P 网络、共识算法及智能合约等多种技术，具有去中心化、防篡改、可追溯等特性。近几年区块链技术发展迅速，其应用也由最初的数字货币、金融服务，扩展到医疗、教育、政务、供应链、版权保护、物联网安全等诸多领域，展现出其巨大的价值。目前，基于区块链的应用多是采用分布式存储来实现安全的可信存证及查询溯源等功能。然而，区块链存储系统仍存在着诸多不足，如存储效率低、存储开销巨大、查询速度慢等问题，这些问题一直制约着区块链的发展，成为区块链应用落地的瓶颈。

14.7.1　区块链存储系统简介

典型的区块链使用文件系统、键值数据库及关系型数据库等存储系统保存数据。不同的区块链根据其设计特点、数据用途和访问频次情况选择不同的存储系统，表 14.1 总结了经典区块链使用的数据存储系统及特征。

表 14.1　典型区块链存储系统

存储系统	关系型语义	存储数据内容	读写速度均衡性	典型区块链的使用
文件系统	最弱	区块数据	写密集	比特币等
LevelDB	弱	区块数据、索引数据、状态数据	写密集	比特币、以太坊等
RocksDB	弱	区块数据、索引数据、状态数据	读写较均衡	FISCO-BCOS 等
CouchDB	较强	状态数据	读密集	Hyperledger Fabric 等
MySQL	强	区块数据、索引数据、状态数据	读写均衡	FISCO-BCOS 等

文件系统主要存储区块数据，一般将区块数据进行二进制编码后存储在文件中，按照文件编号查找数据，其写性能尚可，但读性能比较差。例如，比特币[78]使用文件系统存储区块数据。

键值数据库基于键值结构设计，常见的键值数据库包括 LevelDB、RocksDB 及 CouchDB。LevelDB 和 RocksDB 通常被用来存储区块链上的区块数据、索引数据和状态数据。LevelDB 通过 LSM 树将数据分批顺序存储在非易失性存储器中，具有较强的写性能，尤其适用于早期区块链系统写入密集型应用。区块链以太坊[79]在其实现中采用了 LevelDB 数据库。RocksDB 在 LevelDB 的基础上进一步完善，其整体性能更高，并且能够根据情况灵活调节读写性能。区块链 FISCO-BCOS[80]

采用了 RocksDB 数据库。CouchDB 具有较好的读性能并且支持复杂查询，被 Hyperledger Fabric[81]区块链用来存储状态数据。

MySQL 是关系型数据库，读写性能均衡但弱于其他存储系统。FISCO-BCOS 因处理复杂查询场景而使用 MySQL 作为存储引擎。

14.7.2 区块链存储系统优化

目前，区块链在行业中的应用主要包含溯源和存证两类，二者关系到链上数据的存储及查询。在存储方面，完全去中心化的区块链采用节点账本的全备份来保持数据一致性。即便这种多节点高冗余机制给区块链带来了全网数据一致、及时更新和共享的优点，也带来了节点存储数据量大的问题。在查询方面，绝大多数区块链系统使用块链式结构，该结构的优点是通过散列链可以有效防止篡改，但也带来了倒查追溯效率低的问题。针对以上两个关键问题，目前主要采用的优化方案如下。

1. 链上内容裁剪

位于区块链账本中的部分数据的存储价值低，主要原因在于这些数据存储时间久、使用频次低。为了缓解节点存储数据量大的问题，可以删除一些久远的、很少使用的数据，并将删除操作作为一条交易记录保存在链上。例如，在比特币钱包中，开发者设计了一种区块数据的修剪策略来应对存储容量大的挑战。该策略通过先构建完整的 UTXO（Unspent Transaction Output，未花费的交易输出）集合，然后丢弃历史交易数据（即删除旧数据），达到节省本地存储空间的目的。

2. 区块链分片

为了缓解数据处理和存储压力大的问题，分片技术应运而生。该技术将大量的数据分别保存在不同的服务器中，每个服务器负责计算本地数据。区块链借鉴上述思想，将跨多个节点的区块链划分为多个小组，每个小组内的节点形成单独的规模较小的区块链，即一个区块链分片。区块链系统中产生的交易按策略分配到各个分片中，由片内的节点处理。因此，采用区块链分片技术拥有诸多的优势：第一，单一节点不需要存储和处理全部的交易数据；第二，整体的性能并不受限于单一节点的能力；第三，区块链系统可以并行处理多笔不同的交易。

3. 区块存储结构优化

区块链使用的存储系统会显著影响数据的存储性能。链上分布式数据存储系统 UStore[82]综合了众多数据库和分布式系统的优势进行改进，通过构建类似 Git（一种流行的源代码版本控制系统）的数据结构，实现了比一般键值存储系统更好

的性能和更丰富的查询功能。此外，在 UStore 存储系统的基础上改良而设计的 Forkbase[83]存储引擎，可以存储数据的多个版本，每个版本有唯一的标识数据内容。另外，ForkBase 在存储结构中引入了面向模式的分裂树结构，能高效确定并消除重复数据，以此降低系统中的数据冗余。

4. 链下存储支持

辅助存储通过结合链下存储系统来优化区块链。其核心思想是区块链上只存储有限的关键数据，大部分数据转存到链下存储系统。为了方便查询链下数据，需要在链上建立这些数据的索引信息。访问数据时，首先在链上查询链下数据的索引，然后到链下存储系统读出相应的数据。该方法利用了辅助存储系统容量充足的优点缓解了链上数据存储的问题。

14.8　分离式数据中心架构

随着全球数据的指数级激增，数据中心在存储和管理数据方面正面临空前挑战，基于服务器架构的传统数据中心在资源利用率、扩展性、性能等方面的缺陷日益显著，已经愈发难以满足业务需求。近年来，一种分离式数据中心架构得到了学术界和工业界的广泛关注：该架构下，硬件资源被拆分为不同的硬件资源池（例如处理器池、内存池、存储池等），并通过高速网络互连；管理员可以按需扩展特定的硬件资源池，且各类硬件资源可以在不同应用间灵活共享。然而，分离式数据中心架构在访存模式、存储层级、容错模型、软件开销等方面呈现出显著差异，这为构建分离式架构友好的系统软件带来了新的挑战[84]。

14.8.1　背景

为存储大规模数据，传统数据中心将服务器节点通过网络互连以支持动态扩展。然而，不同大数据业务对各类硬件资源的需求具有显著差异，导致以服务器为最小单元的扩展方式出现了资源利用率、扩展性、性能等诸多方面的问题[85]。

（1）数据保存周期与服务器更新周期不匹配

人工智能、大数据等重要应用产生了海量数据，这些数据需按照其生命周期策略（例如 8～10 年）进行保存。而在传统的数据中心中，服务器的换代周期由处理器的升级周期（例如 3～5 年）决定。这种数据生命周期与服务器更新周期之间巨大的差异导致系统资源被大量浪费，服务器中的存储资源可能会随 CPU 升级而淘汰，为此需要进行相应的数据迁移等。

（2）内存资源在时空维度的不均衡

内存资源占用了服务器较大部分（可达 50%）的成本，但由于存在时空维度的不均衡现象，其资源利用率极低：谷歌公司集群的内存利用率平均仅为 45%；阿里巴巴集群的内存利用率也不足 65%。具体而言，从时间维度上看，一台服务器的应用进程对内存的需求会随时间变化，而通常服务器会按照峰值需求配置内存容量，因此大多数时间会存在内存闲置的情况；从空间维度上看，同一个时刻，不同服务器的内存使用量差异很大，这表明了整个数据中心层次的内存浪费。

（3）云原生应用对计算和存储的弹性诉求

随着云原生应用（如云原生数据库、Serverless 应用）的发展，其对弹性资源分配的诉求日益增多。具体而言，对于存储资源，云原生应用希望其能够根据数据量无穷地扩展；而对计算资源，则希望能够按照请求的密度进行细粒度分配。例如，在云原生数据库中，数据被存储在后端对象系统中，而执行事务操作的虚拟机根据 SQL 请求的流量被动态地添加或移除；在 Serverless 应用中，每个函数请求会创建独立的容器。

（4）昂贵的数据中心税

云数据中心通过虚拟化技术将物理资源出售给租户。然而为了灵活地提供各类新的需求（如数据加密和压缩），每台服务器会耗费大量的 CPU 资源用于网络和存储的虚拟化。例如，谷歌公司数据中心运行基础设施软件所缴纳的“数据中心税”高达 30%。这带来了 3 个方面的问题：首先，这些被消耗的 CPU 资源无法出售给租户，影响云服务商的盈利；其次，由于与前台的任务共享缓存等资源，这些基础设施软件会影响正常应用的性能；最后，通用 CPU 的性能增长远慢于 I/O 外设，难以持续提供高性能虚拟化。消除数据中心税的一个主要思路是使用专用处理器卸载基础设施任务。然而，在传统服务器架构中，CPU 是一切的中心，专用处理器作为外设难以高效地访问服务器中的其他资源。

14.8.2 架构特点及关键技术

为了应对传统数据中心架构在资源利用率低下、灵活性不足等诸多方面的问题，分离式数据中心应运而生，其架构如图 14.16 所示[85]。分离式数据中心架构将硬件资源按类别拆分为不同的硬件资源池，并通过高速网络将这些资源池互连。其中，内存池主要包含 DRAM 资源，用于为应用提供低延迟的临时数据保存（例如进程空间的数据）。存储池包含低速的 HDD 资源和高速的 SSD 资源（NAND 介

质或者 Optane 介质）。考虑到持久性内存同时具有字节寻址和持久化的能力，它可用于扩展内存池的容量或提升存储池的性能[86-90]。计算资源池主要包括 CPU 池、GPU 池、FPGA 池及其他异构计算资源。在网络互连方面，RDMA 支持计算资源池直访内存池和持久性内存池；NVMe-oF 支持计算资源池直访 HDD 池和 SSD 池；而 CXL 网络支持所有资源之间的互相访问，但其扩展性低于其他两种网络技术，因此更适合于小规模的数据中心。

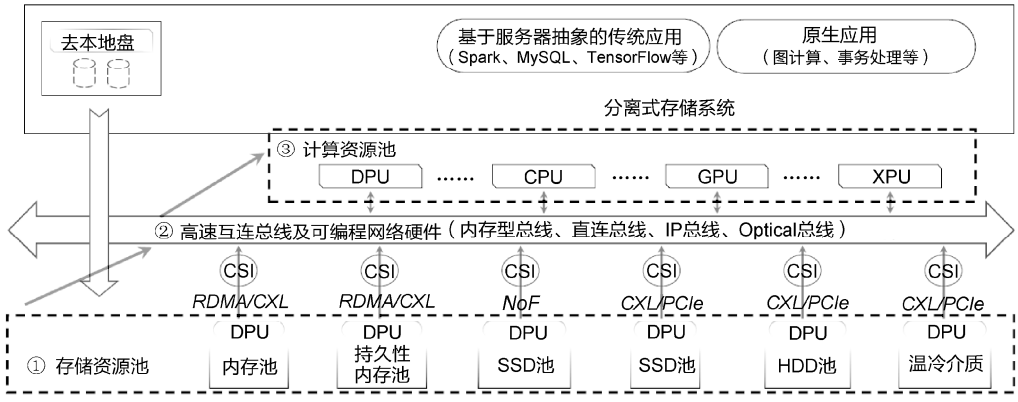


图 14.16　分离式数据中心架构

为了避免 CPU 每次读写内存都触发网络访问，在分离式数据中心架构中，计算资源池中的处理器会配备少量的本地内存用于缓存，以减少网络访问。与之类似的是内存和存储池通常会配备少量的计算资源（如智能网卡中的 ARM CPU 核心），用于执行一些管理任务（如空间分配、垃圾回收）。在分离式数据中心架构下，由于硬件资源的分离，网络扮演了至关重要的角色：数据在网络路径上流动，并在不同硬件资源之间进行交换。为了加速网络路径，分离式数据中心配备了智能网卡、可编程交换机等可编程网络设备。其中智能网卡可作为各种资源池的控制平面，执行资源初始化、异常处理等任务，并对虚拟化等基础设施进行卸载，降低 CPU 资源消耗。而可编程交换机具有线速的处理能力和中心化的位置，可加速分布式硬件资源之间的协调，以减少分离式数据中心的软件开销。

相较于传统架构，新型存算分离架构最为显著的区别在于更为彻底的存算解耦，该架构不再局限于将 CPU 和外存解耦，而是彻底打破各类存算硬件资源的边界，将其组建为彼此独立的硬件资源池（例如计算资源池、内存池、HDD/SSD 池等），从真正意义上实现各类硬件的独立扩展及灵活共享；更为细粒度的处理分工，即打破了传统以通用 CPU 为中心的处理逻辑，使数据处理、聚合等原本 CPU

不擅长的任务被专用加速器、DPU 等执行，从全局角度实现硬件资源的最优组合，进而提供极致的能效比。相比于传统数据中心，分离式数据中心架构具有如下的优势。

（1）高资源利用率

通过将硬件资源分离，构成共享的资源池，不同的应用可以分时复用所有的内存和存储资源，因此此时只需按整个数据中心（而不是传统的服务器粒度）的使用峰值来配备各类资源；同时，不同种类的资源可以独立扩展：当扩充某类资源（例如内存）时，无须像服务器架构一样添加其他资源（例如 CPU），以最小化资源的浪费。

（2）高灵活性

在应用侧，单个应用在执行过程中对各类资源的需求量会不断变化，分离式数据中心架构可以快速地将数据中心的空闲资源分配出来，提供给需要的应用，由此快速应对负载的变化（例如突发流量），以达到应用的极致弹性；在硬件侧，随着摩尔定律逐渐失效，数据中心开始采用各类异构的加速器（例如 TPU、FPGA）。在传统服务器架构下，服务器主板上的插槽是固定且数量有限的，难以应对未来新型异构算力的不断增加，而分离式数据中心架构抛弃了服务器的概念，当需要引入某类新设备时，只需构建对应的资源池，并将其连入网络。

（3）低数据中心税

分离式数据中心打破了以 CPU 为中心的传统架构，各类算力均能同等地访问网络、内存、存储等资源。因此，在分离式数据中心架构下，虚拟化等基础设施可以很容易地卸载至 FPGA、智能网卡等中，由此大幅度释放 CPU 资源，显著降低数据中心税。

分离式存储系统对存储资源池进行统一管理，并将其提供给计算资源池中的处理器使用。构建分离式存储系统涉及如下具有挑战的关键技术。

（1）接口抽象

内存资源池需要提供某种接口抽象，将内存空间暴露给远端的应用、运行时、操作系统和处理器硬件。其中最关键的设计准则是如何在性能和兼容性之间权衡。现有的分离式内存接口抽象主要有基于操作系统内存交换机制、基于内存访问自动均衡（AutoNUMA）、基于专用用户库、基于 JAVA 运行时等途径，它们在互连技术、兼容性、页最小管理粒度、性能等方面各具差异，表 14.2 详细对比了其差异。

表 14.2　接口抽象的差异

接口抽象	互连技术	兼容性	最小管理粒度	性能	额外开销
内存交换机制	RDMA	高	内存页（4 KB）	低	缺页中断处理软件栈开销
AutoNUMA	内存语义总线（CXL）	高	内存页（4 KB）	中	热点页扫描开销
专用用户库	RDMA	低	任意粒度	高	无
JAVA 运行时	RDMA	高	任意粒度	中	JAVA 运行时软件开销

（2）数据交换

为了减少网络访问，需要充分利用计算节点的小容量本地内存。此时，数据交换技术显得尤为重要。内存交换的主要目标是将热点数据存放在计算节点的本地内存，而将访问不频繁的数据存放在远端内存空间，这主要涉及精准的热点内存信息采集、高效的数据迁移机制、及时的数据预取策略等关键环节。大部分应用通过 CPU 指令访问分离式内存，此过程系统软件无法直接介入，因此缺乏相应的时机统计相关的访存信息。目前主要包括软件插桩、页表标记位、CPU 硬件计数等几类热点内存数据的追踪机制。当准确获得了当前应用程序访问页的冷热分布情况后，需要根据页的实际存储位置执行数据迁移，将访问频繁的页从远端内存读取至本地，同时将本地访问不频繁的页逐出至远端。现有工作主要关注迁移时机、通路选择、前后台协调等问题。数据迁移机制是根据现有的热点信息调整内存页在近远端的摆放，而数据预取策略则是根据当前的访问特征预测未来可能发生的数据访问操作，并提前将对应的页存放至本地内存。在分离式内存中，预取页需要跨网数据传输，预取出错代价很高，因此预取的准确性极为重要。

（3）分离式内存管理

当内存资源被多个计算节点共享使用时，需要高效的数据管理技术保证并发读写的正确性，包括如何设计并发索引、分布式事务协议及数据分区策略。与传统内存不同，内存池的计算能力有限，分离式内存中的索引及协议一般使用 RDMA 单边原语来完成。如何设计 RDMA 友好的数据结构、如何协调并发操作是设计针对分离式内存管理的主要挑战。

（4）分离式文件及对象管理

与内存资源管理不同，管理外存资源时需要提供给上层应用的语义更丰富，包括对象接口及具有目录树结构的文件接口。此外，作为存储系统，还需要考虑可靠性、持久性等重要指标。由于存储池的计算能力有限，如何做到轻量而高效的对象及文件管理十分关键。

（5）智能硬件卸载

分离式数据中心配备有可编程交换机、智能网卡等可编程网络硬件。分离式存储系统可利用这些硬件进行数据管理、分布式协议等的卸载，由此减少软件开销并提升性能。在分离式架构中，内存节点的算力较弱，导致计算节点大多通过单边 RDMA 访问远端内存。然而，RDMA 的语义有限，只支持读写和原子指令，所以在一些复杂场景会导致大量网络往返，降低系统性能。为此，一些研究者利用存储资源池配备的智能网卡、可编程交换机、DPU 等扩展分离式架构下的 RDMA 语义及存储协议。

14.8.3 未来趋势

分离式内存架构还具有以下几个研究趋势。

（1）分离式内存的进程容错

在传统数据中心内，对于一个进程而言，其内存空间对应的物理资源仅保存在本地服务器中；然而，在分离式数据中心架构下，一个进程的内存会保存在内存池中的多个内存节点中，这不可避免地扩大了进程的故障域（Failure Domain）：当某个内存节点崩溃，对应进程就会由于丢失内存数据而无法运行。因此，进程容错是分离式数据中心架构中的关键且极具挑战的问题。传统的副本机制需要成倍的内存使用量，这与分离式数据中心架构的一大初衷——提高资源利用率背道而驰，因此，目前有少量研究工作利用纠删码机制支持进程的容错。这些研究工作激进地将所有位于内存池的数据进行容错，并未考虑到某些数据本身是可恢复的，例如在存储池中存在检查点或快照的数据。因此，未来的研究需要对内存数据进行选择性容错，在保证系统高可靠的同时最小化容错开销。对于纠删码等容错机制，可以卸载至智能网卡等可编程网络设备上进行加速。此外，需研究当计算节点失效后如何快速将其上的进程迁移至其余正常的计算节点。

（2）异构网络下的系统设计

在未来的分离式数据中心架构中，资源池之间互连的网络必定是异构的，例如在机架层，机架内部的服务器通过 CXL 网络共享访问 CXL 内存设备；在集群层，通过 RDMA 网络，不同机架内的服务器可以互相访问内存。而不同的网络在性能、接口方面的特性具有较大差异，例如 CXL 网络延迟低、操作同步且支持原生的 Load/Store 指令，而 RDMA 网络延迟较高、操作异步且以传统外设 I/O 的方式进行远程读写。因此，研究者需要思考如何根据异构网络拓扑，将内存资源和存储资源分散至不同网络层级。考虑到存储资源本身的异构性（持久性内存、高速固态硬盘、慢速磁盘等），如何协同异构网络和异构存储也是个重要问题。在异

构网络之下，应用程序依赖的编程模型也需要重新考量：是让操作系统进行统一管理，还是将网络属性直接暴露给上层应用。

（3）异构算力下的系统设计

现有关于分离式数据中心架构的研究，主要涉及内存资源和存储资源的管理，而较少关注计算资源。随着云计算、人工智能的普及，数据中心存在大量的异构计算资源，如 CPU、GPU、FPGA 和 AI 加速器等。在分离式数据中心架构下，这些资源被聚集成对应的异构计算池，如何充分发挥出它们的最大性能是关键的研究问题，主要研究挑战包括两方面：首先，对于某个计算任务，如何将不同异构计算池的算力进行封装以供使用，同时支持算力的动态调配；其次，异构资源池需要通过高效的方式与内存池和存储池进行数据交换，即如何抽象远程的内存和存储资源，让 GPU、FPGA 和 AI 加速器等异构计算设备能够快速的定位、检索、读写数据。

14.9　高密度新型存储

随着数据量和数据保存需求的急剧增长，目前的主流存储介质（如硬盘、磁带等）面临严重的“容量墙”问题。这个问题可以展开为以下两个方面的问题：介质存储密度的增长速度远低于数据量的增长速度，以及介质寿命远低于人们希望的数据存储寿命。

从介质的存储密度方面来说，主流存储介质的数据存储能力每年只提高 20% 左右，远远跟不上数据增长的速度。例如，希捷公司于 2020 年发布的基于 IDC 调研数据的《数据新视界》报告显示，2020—2022 年，企业数据将以年均 42.2% 的速度增长，这一速度约为存储密度增长速度的两倍。从介质的存储寿命方面来说，主流存储介质依赖磁信号（如机械硬盘和磁带）或电信号（如固态硬盘）存储信息，无法长期保存数据，使用寿命一般在 5～10 年。然而，许多数据的寿命远超设备的使用寿命。因此，亟需更高密度、更长寿命的新型存储技术（如叠瓦式磁性存储、光存储、DNA 存储等）来解决容量墙问题。

14.9.1　叠瓦式磁性存储

传统 HDD 的存储密度已接近物理极限（约 1 TB/in^2），难以满足未来的大容量存储需求，业界主流硬盘厂商（如西部数据、希捷）转向研发更高存储密度的新型 HDD 存储技术——SMR（Shringle Magnetic Recording，叠瓦式磁记录）。SMR 利用现有磁头和盘片介质技术，对制造工艺进行微小改动，实现了比传统 HDD

更高的存储密度，单位面积存储容量提高 10%～25%，其主要原理是在盘片上重叠部分磁道，实现在相同面积下容纳更多的磁道，如图 14.17 所示。

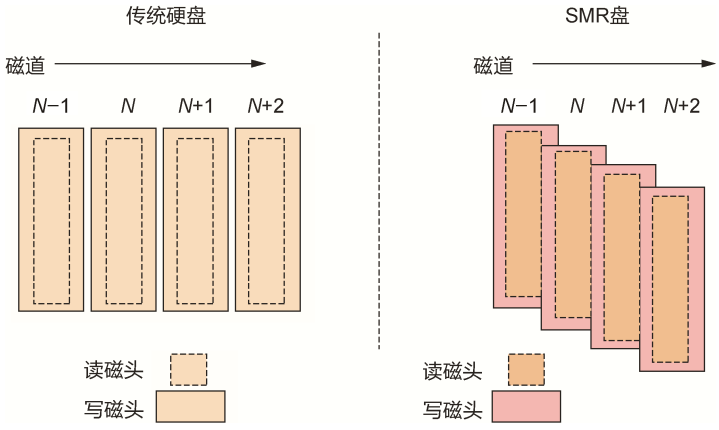


图 14.17　叠瓦式磁记录和传统硬盘的对比

SMR 的写磁头和相邻磁道重叠，这一结构的变化带来数据写入的挑战：其一是不支持随机写，其二是不支持原地更新写。为解决上述挑战有如下关键技术。

RoW 技术：将所有的写入进行聚合，以类似日志结构的方式执行写入，解决 SMR 盘不支持随机写入和原地更新的写入问题。

垃圾回收技术：RoW 技术的写入会带来空间碎片化问题，导致 SMR 盘的写入单元无法重新写入数据，进而造成空间利用率下降。因此，需要通过垃圾回收将写入单元中有效数据块迁移到另一个写入单元。

冷热数据分流技术：在垃圾回收过程中会执行多次的数据迁移，导致写入次数增加，即“写放大”问题。可利用冷热数据技术，结合对数据生命周期的识别，实现冷热数据分配到不同的写入单元，进而降低写放大。

14.9.2　高密光存储

不管是高密度的 HDD，还是磁带，其本质都是基于磁技术的存储。随着保存时间的增长，磁存储设备的保存能力将逐渐降低，最终可能影响数据的正确保存，例如，基于 HDD 的存储系统，一般每 3～5 年就要进行一次数据迁移，而基于磁带的存储系统，一般每 10 年左右就要进行一次数据迁移。突破大容量数据存储和数据长期的保存问题正是新型光存储技术的热点方向。

蓝光存储已被业界的大型互联网企业采用来保存冷数据。但是蓝光存储有一个物理极限，已证明单张蓝光光盘的层数不会超过 40 层，容量不会超过 1 TB[6]。

因此，业界开始在全息光存储、超分辨率光存储、玻璃存储等方向上进行探索，实现更高密度的光存储。

传统光盘的技术路径是基于平面坑、槽的打点记录和检测，这些坑、槽的尺寸直接影响盘片的容量。为了增加光盘的容量，业界将原本局限于平面的光存储扩展到立体空间中，即全息光存储，实现更高的存储密度。当前基于全息光存储的单盘容量已达到 2.5 TB，随着全息材料技术的突破，未来单盘容量可达 4 TB，甚至更大。

传统光盘，如 CD、DVD 和蓝光光盘，其读写都是基于 1/0 调制，不同的是所采用的激光波长不同。然而，即使继续降低激光波长，形成记录点的可分辨光斑尺寸受衍射极限所制约，容量密度难以继续扩展。超分辨率光存储通过双束光记录方式，突破衍射极限，进一步缩小记录点尺寸，提升容量密度。从理论上来说，基于超分辨率的光存储单盘容量有望达到 1 PB。

玻璃存储是将光的多个特性维度（如介质本身的三维空间、偏振、波长、光强等）应用到玻璃，实现单个记录点的多种表达方式，从而实现容量密度的提升。当前基于玻璃存储的单盘容量已达到 360 TB。

14.9.3　DNA 存储

DNA 存储技术是利用人工合成的脱氧核糖核酸（DNA）作为存储介质，即将二进制数据编码成 DNA 的碱基（A/C/G/T）信息并进行存储，具有存储密度高（1 克 DNA 可存储 2 PB 数据）、保存时间长（数据保存时间可长达数千年）的优点。

DNA 存储包括如下 6 个步骤，如图 14.18 所示，前 3 步对应数据的写入，后 3 步对应数据的读取：第一步编码，将二进制数据映射为 DNA 序列；第二步，合成，按照 DNA 序列合成 DNA 链；第三步存储，将 DNA 链存储到载体中；第四步检索，通过碱基对的异或检验方式提取 DNA 分子；第五步测序，将 DNA 分子组合成 DNA 序列；第六步解码，将 DNA 序列中的信息还原成二进制数据。

2020 年 11 月在闪存峰会上微软、西部数据、DNA 数据存储公司 TWIST 联合 DNA 测序公司 illumina 建立 DNA 存储联盟，以发展 DNA 归档存储的商业生态系统。DNA 存储尚处于起步阶段，为更好地发展 DNA 存储，我们还需要解决如下几个关键问题：如何实现 DNA 存储的覆盖写？如何实现 DNA 存储的随机写？如何实现 DNA 存储中数据的高可靠性？这些在传统存储系统中能轻松完成的任务，在 DNA 存储中还需要进一步探索和研究。

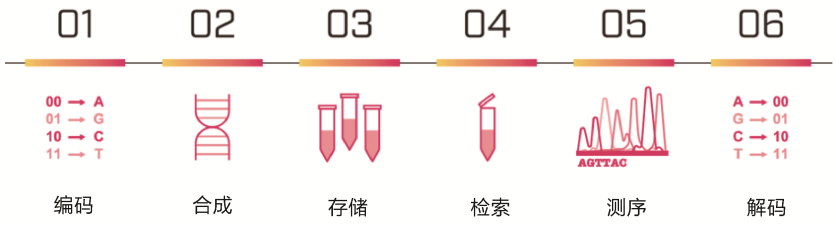


图 14.18　DNA 存储的流程

14.10　本章小结

数据存储系统作为信息技术的基础设施之一在数字经济时代扮演着举足轻重的角色。为应对新时代海量数据存储的挑战，满足高性能、高容量、高可靠及高扩展性的数据存储需求，新的存储技术层出不穷。本章从新型存储模式，非易失性存储系统及应用存储优化等方面介绍了存储技术的趋势与发展，重点突出了学术前沿的代表性工作。在新型存储模式方面，阐述了存内计算、在网存储系统、分离式存储、边缘存储及高密度新型存储；在非易失性存储系统方面，对闪存存储和持久性内存存储系统展开了分析；在应用存储优化方面，总结了智能存储和区块链存储。这些前沿存储技术的创新与突破将为信息产业带来新的机遇，对追求高质量发展的数字经济时代具有重大的意义。

参考文献

[1] 舒继武, 陆游游, 张佳程, 等. 基于非易失性存储器的存储系统技术研究进展[J]. 科技导报, 2016, 34(14):86-94.

[2] ANDERSEN D G, FRANKLIN J, KAMINSKY M, et al. FAWN: A fast array of wimpy nodes[C]// Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP). Big Sky, Montana, USA: ACM, 2009: 1-14.

[3] CAUFIELD A M, GRUPP L M, GORDON S S. Using flash memory to build fast, power-efficient clusters for data-intensive applications[C]//Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York, NY, USA: ACM, 2009: 217-228.

[4] 陆游游. 闪存文件系统的关键技术研究[D]. 北京: 清华大学, 2014.

[5] LEE C, SIM D, HWANG J, et al. F2FS: A new file system for flash storage[C]//Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST). Santa Clara, CA, USA: USENIX, 2015: 273-286.

[6] JOSEPHSON W K, BONGO L A, FLYNN D, et al. DFS: A file system for virtualized flash storage[C]// Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST). Berkeley, CA, USA: USENIX, 2010: 85-99.

[7] LU Y, SHU J, ZHENG W. Extending the lifetime of flash-based storage through reducing write amplification from file systems[C]//Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST). Berkeley, CA: USENIX, 2013: 257-270.

[8] ZHANG J, SHU J, LU Y. ParaFS: A Log-Structured File System to Exploit the Internal Parallelism of Flash Devices[C]//2016 USENIX Annual Technical Conference (USENIX ATC). Denver, Colorado, USA: USENIX, 2016:87-100.

[9] LU Y, SHU J, WANG W. ReconFS: A Reconstructable File System on Flash Storage[C]//the 12th USENIX Conference on File and Storage Technologies (FAST). San Jose, CA, USA:USENIX, 2014:75-88.

[10] ZHANG J, LU Y, SHU J, et al. FlashKV: Accelerating KV Performance with Open-Channel SSDs[J]. ACM Transactions on Embedded Computing Systems, 2017, 16(5): 1-19.

[11] LI S, LU Y, SHU J, et al. LocoFS: A Loosely-Coupled Metadata Service for Distributed File System[C]//The International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver: ACM, 2017:1-12.

[12] LU Y, SHU J, GUO J, et al. LightTx: A Lightweight Transactional Design in Flash-based SSDs to Support Flexible Transact[C]//31st IEEE International Conference on Computer Design (ICCD), Asheville NC, USA: IEEE, 2013:115-122.

[13] LU Y, SHU J, GUO J, et al. High-Performance and Lightweight Transaction Support in Flash-based SSDs[J]. IEEE Transactions on Computers, 2015, 64(10):2819-2832.

[14] BAE H, KIM J, KWON M, et al. What you can't forget: exploiting parallelism for zoned namespaces[C]//Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems. New York: ACM, 2022:79-85.

[15] BJØRLING M, AGHAYEV A, HOLMBERG H, et al. ZNS: Avoiding the block interface tax for flash-based[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21), USENIX, 2021:689-703.

[16] Zoned Storage. NVMe Zoned Namespaces[EB/OL]. (2020-10-09)[2023-06-08].

[17] HAN K, GWAK H, SHIN D, et al. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction.[C]//OSDI 21. USENIX, 2021:147-162.

[18] KIM J, LIM K, JUNG Y, et al. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays.[C]//USENIX Annual Technical Conference. USENIX, 2019: 799-812.

[19] COLGROVE J, DAVIS J D, HAYES J, et al. Purity: Building fast, highly-available enterprise flash storage from commodity components[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015: 1683-1694.

[20] KIM T, JEON J, ARORA N, et al. RAIZN: Redundant Array of Independent Zoned Namespaces [C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2023: 660-673.

[21] BERGMAN S, CASSEL N, BJØRLING M, et al. ZNSwap: Un-block your swap[C]// 2022 USENIX Annual Technical Conference. Carlsbad: USENIX, 2022:1-25.

[22] SHIN H, OH M, CHOI G, et al. Exploring performance characteristics of ZNS SSDs: Observation and implication[C]//2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA). IEEE, 2020:1-5.

[23] NICK T, TRIVEDI A. Understanding NVMe Zoned Namespace (ZNS) Flash SSD Storage Devices[EB/OL]. (2022-01-03)[2023-06-08]. arXiv:2206.01547.

[24] KWON M, GOUK D, LEE S, et al. Hardware/Software Co-Programmable Framework for Computational SSDs to Accelerate Deep Learning Service on Large-Scale Graphs[C]// Proceedings of the USENIX Conference on File and Storage Technologies (FAST). USENIX，2022: 147-164.

[25] ZHANG F, ANGIZI S, FAN D. Max-PIM: fast and efficient Max/Min searching in DRAM[C]//Proceedings of the Design Automation Conference (DAC). IEEE, 2021: 211-216.

[26] XIE X, LIANG Z, GU P, et al. Spacea: sparse matrix vector multiplication on processing-inmemory accelerator[C]//Proceedings of the International Symposium on HighPerformance Computer Architecture (HPCA). IEEE, 2021: 570-583.

[27] PARK J, KIM B, YUN S, et al. TRiM: enhancing processor-memory interfaces with scalable tensor reduction in memory[C]//Proceedings of the International Symposium on Microarchitecture (MICRO). ACM, 2021: 268-281.

[28] LEE S, KANG S, LEE J, et al. Hardware architecture and software stack for PIM based on commercial DRAM technology: industrial product[C]//Proceedings of the Annual International Symposium on Computer Architecture (ISCA). IEEE, 2021: 43-56.

[29] SI X, TU Y N, HUANG W H, et al. 15.5 A 28nm 64Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips[C]//Proceedings of the International Solid-State Circuits Conference (ISSCC). IEEE, 2020: 246-248.

[30] YANG J, KONG Y, WANG Z, et al. 24.4 sandwich-RAM: an energy-efficient in-memory BWN architecture with pulse-width modulation[C]//Proceedings of the International Solid-State Circuits Conference (ISSCC). IEEE, 2019: 394-396.

[31] CHI P, LI S, XU C, et al. Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 27-39.

[32] SHAFIEE A, NAG A, MURALIMANOHAR N, et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 14-26.

[33] DANIAL L, PIKHAY E, HERBELIN E, et al. Two-terminal floating-gate transistors with a lowpower memristive operation mode for analogue neuromorphic computing[J]. Nature Electronics, 2019, 2(12): 596-605.

[34] MAHMOODI M R, STRUKOV D. An ultra-low energy internally analog, externally digital vector-matrix multiplier based on NOR flash memory technology[C]//Proceedings of the Design Automation Conference (DAC). IEEE, 2018: 1-6.

[35] CONDIT J, NIGHTINGALE E B, FROST C, et al. Better I/O through byte-addressable, persistent mem-ory[C]//Matthews J. SOSP ’09: Proceedings of the 22nd Symposium on Operating Systems Principles. New York, NY, USA: ACM, 2009:133-146.

[36] DULLOOR S R, KUMAR S, KESHAVAMURTHY A, et al. System software for persistent memory[C]//Bultermann D, Bos H. EuroSys ’14: Proceedings of the 9th European Conference on Computer Systems New York, NY, USA: ACM, 2014:1-15.

[37] XU J, SWANSON S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories[C]//Brown A, Popovici F. FAST ’16: Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2016:323-338.

[38] WU X, REDDY A L N. SCMFS: A file system for storage class memory[C]//Lathrop S. SC ’11: Proceedings of 24th International Conference for High Performance Computing,

Networking, Storage and Analysis. New York, NY, USA: ACM, 2011:1-23.

[39] VOLOS H, NALLI S, PANNEERSELVAM S, et al. Aerie: Flexible file-system interfaces to storage-class memory[C]//Bultermann D, Bos H. EuroSys '14: Proceedings of the 9th European Conference on Computer Systems. New York, NY, USA: ACM, 2014:1-14.

[40] KWON Y, FINGLER H, HUNT T, et al. Strata: A cross media file system[C]//Chen H, Zhou L. SOSP '17: Proceedings of the 26th Symposium on Operating Systems Principles. New York, NY, USA: ACM, 2017:460-477.

[41] CHEN Y, LU Y, ZHU B, et al. Scalable Persistent Memory File System with Kernel-Userspace Collaboration[C]// Proceedings of the USENIX Conference on File and Storage Technologies (FAST). USENIX, 2021, 21: 81-95.

[42] VENKATARAMAN S, TOLIA N, RANGANATHAN P, et al. Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory[C]// Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST), USENIX, 2011.

[43] HU Q, REN J, BADAM A, et al. Log-Structured Non-Volatile Main Memory[C]// Proceedings of the USENIX Annual Technical Conference (ATC). USENIX, 2017: 703-717.

[44] LU Y, SHU J, CHEN Y, et al. Octopus: An RDMA-enabled distributed persistent memory file sys-tem[C]//Silva D, Ford B. USENIX ATC '17: Proceedings of the 23rd Conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX, 2017:773-785.

[45] YANG J, IZRAELEVITZ J, SWANSON S. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks[C]//Merchant A, Weatherspoon H. FAST '19: Proceedings of the 17th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX, 2019:221-234.

[46] ZHANG Y, YANG J, MEMARIPOUR A, et al. Mojim: A reliable and highly-available non-volatile memory system[C]//Ozturk O, Ebcioglu K. ASPLOS '15: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: ACM, 2015:3-18.

[47] SHAN Y, TSAI S Y, ZHANG Y. Distributed shared persistent memory[C]//Curino C. SoCC '17: Proceedings of the 8th Symposium on Cloud Computing. New York, NY, USA: ACM, 2017: 323-337.

[48] DRAGOJEVIĆ A, NARAYANAN D, HODSON O, et al. FaRM: Fast remote memory[C]//Mahajan R, Stoica I. NSDI'14: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. Berkeley, CA, USA: USENIX, 2014:401-414.

[49] DRAGOJEVIĆ A, NARAYANAN D, NIGHTINGALE E B, et al. No compromises: Distributed transactions with consistency, availability, and performance[C]//Miller E. SOSP '15: Proceedings of the 25th Symposium on Operating Systems Principles. New York, NY, USA: ACM, 2015:54-70.

[50] 汪庆, 李俊儒, 舒继武. 在网存储系统研究综述[J/OL]. 计算机研究与发展. (2023-08-02)[2023-08-18].

[51] 汪庆. 分布式内存存储系统的网存协同关键技术研究[D]. 北京：清华大学，2023.

[52] WANG Q, LU Y, XU E, et al. Concordia: Distributed Shared Memory with {In-Network} Cache Coherence[C]//19th USENIX Conference on File and Storage Technologies (FAST 21). USENIX, 2021: 277-292.

[53] LI J, LU Y, ZHANG Y, et al. SwitchTx: Scalable In-Network Coordination for Distributed Transaction Processing[J]. Proceedings of the VLDB Endowment, 2022, 15(11):2881-2894.

[54] LI J, LU Y, WANG Q, et al. AlNiCo:SmartNIC-accelerated Contention-aware Request Scheduling for Transaction Processing[C]//2022 USENIX Annual Technical Conference (USENIX ATC 22). USENIX, 2022: 951-966.

[55] JIN X, LI X, ZHANG H, et al. Netcache: Balancing key-value stores with fast in-network caching[C]//Proceedings of the 26th Symposium on Operating Systems Principles. New York: ACM, 2017:121-136.

[56] KRASKA T, BEUTEL A, CHI E H, et al. The case for learned index structures[C]// Proceedings of the 2018 international conference on management of data. New York: ACM, 2018: 489-504.

[57] TANG C, WANG Y, DONG Z, et al. XIndex: a scalable learned index for multicore data storage[C]//Proceedings of the 25th ACM SIGPLAN symposium on principles and practice of parallel programming. New York: ACM, 2020: 308-320.

[58] DAI Y, XU Y, GANESAN A, et al. From wisckey to bourbon: A learned index for log-structured merge trees[C]//Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. USENIX, 2020: 155-171.

[59] WEI X, CHEN R, CHEN H. Fast RDMA-based ordered key-value store using remote learned cache[C]//Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. USENIX, 2020: 117-135.

[60] LYU W, LU Y, SHU J, et al. Sapphire: Automatic configuration recommendation for distributed storage systems[EB/OL]. (2020-07-07)[2023-0609]. arXiv:2007.03220.

[61] MAAS M, ANDERSEN D G, ISARD M, et al. Learning-based memory allocation for C++ server workloads[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 541-556.

[62] HAO M, TOKSOZ L, LI N, et al. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network[C]// Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. USENIX, 2020: 173-190.

[63] NIKOLI D, BÖHRINGER R, BEN-NUN T, et al. Clairvoyant prefetching for distributed machine learning I/O[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2021:1-15.

[64] KHAN R, YAZDANI A, FU Y, et al. SHADE: Enable Fundamental Cacheability for Distributed Deep Learning Training[C]//21st USENIX Conference on File and Storage Technologies (FAST 23). New York: ACM, 2023:135-151.

[65] MURRAY D, ŠIMŠA J, KLIMOVIC A, et al. tf.data: a machine learning data processing framework[J]. Proceedings of the VLDB Endowment, 2021,14(12): 2945-2958.

[66] XIE M, LU Y, LIN J, et al. Fleche: an efficient GPU embedding cache for personalized recommendations[C]//Proceedings of the Seventeenth European Conference on Computer Systems. New York: ACM, 2022:402-416.

[67] XIE M, LU Y, WANG Q, et al. PetPS: Supporting Huge Embedding Models with Persistent Memory[J]. Proceedings of the VLDB Endowment, 2023,16(5): 1013-1022.

[68] RAJBHANDARI S, RASLEY J, RUWASE O, et al. Zero: Memory optimizations toward training trillion parameter models[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020:1-16.

[69] FENG Y, XIE M, TIAN Z, et al. Mobius: Fine tuning large-scale models on commodity gpu servers[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023:489-501.

[70] RUAN Z, HE T, CONG J. INSIDER: Designing In-Storage Computing System for Emerging High-Performance Drive[C]// Proceedings of the USENIX Annual Technical Conference (ATC). USENIX, 2019: 379-394.

[71] QIAO Y, CHEN X, ZHENG N, et al. Closing the B+-tree vs.LSM-tree Write Amplification Gap on Modern Storage Hardware with Built-in Transparent

Compression[C]// Proceedings of the USENIX Conference on File and Storage Technologies (FAST). USENIX, 2022: 69-82.

[72] NAWAB F, AGRAWAL D, EL ABBADI A. Dpaxos: Managing data closer to users for low-latency and mobile applications[C]//Proceedings of the International Conference on Management of Data (SIGMOD). New York: ACM, 2018: 1221-1236.

[73] CHEN X, SONG H, JIANG J, et al. Achieving low tail-latency and high scalability for serializable transactions in edge computing[C]//Proceedings of the European Conference on Computer Systems (EuroSys). New York: ACM, 2021: 210-227.

[74] GUPTA H, RAMACHANDRAN U. Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access[C]//Proceedings of the International Conference on Distributed and Event-based Systems (DEBS). New York: ACM, 2018: 148-159.

[75] SHU J, FANG K, CHEN Y, et al. TH-iSSD: Design and Implementation of a Generic and Reconfigurable Near-Data Processing Framework[EB/OL]. (2023-02-20)[2023-06-09].

[76] YANG Z, LU Y, LIAO X, et al. λ-IO: A Unified IO Stack for Computational Storage[C]//Proceedings of the USENIX Conference on File and Storage Technologies (FAST). USENIX, 2023: 347-362.

[77] NOOR J, SRIVASTAVA M, NETRAVALI R. Portkey: Adaptive Key-Value Placement over Dynamic Edge Networks[C]//Proceedings of the ACM Symposium on Cloud Computing (SOCC). ACM, 2021:197-213.

[78] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system[EB/OL]. (2008-10-31)[2023-06-09].

[79] BUTERIN V. A next-generation smart contract and decentralized application platform[J]. white paper, 2014, 3(37): 2-1.

[80] 金链盟. Fisco-Bcos [EB/OL]. (2020-01-09)[2023-06-09].

[81] Hyperledger Foundation. Hyperledger Fabric Project [EB/OL]. (2017-08-05)[2023-06-09].

[82] DINH A, WANG J, WANG S, et al. UStore: a distributed storage with rich semantics [EB/OL]. (2017-02-09)[2023-06-09]. arXiv:1702.02799.

[83] WANG S, DINH A, LIN Q, et al. Forkbase: An efficient storage engine for blockchain and forkable applications[EB/OL]. (2018-02-14)[2023-06-09]. arXiv:1802.04949, 2018.

[84] 舒继武, 陈游旻, 汪庆, 等. 分离式数据中心的存储系统研究进展[J]. 中国科学: 信息科学, 2023, 53(8): 1503-1528.

[85] 舒继武. 新型存算分离架构技术展望[J]. 中国计算机学会通讯, 2022, 18(11): 53-60.

[86] 舒继武, 陆游游, 张佳程, 等. 基于非易失性存储器的存储系统技术研究进展[J]. 科技导报, 2016, 34(14): 86-94.

[87] 陈游旻. 持久性内存存储系统关键技术研究[D]. 北京：清华大学, 2021.

[88] 毛海宇, 舒继武, 李飞, 等. 存内计算研究进展[J]. 中国科学:信息科学, 2021, 51(2):173-205.

[89] 陆游游, 舒继武. 持久性内存: 从系统软件的角度[J]. 中国计算机学会通讯, 2019, 15(1):15-20.

[90] 舒继武, 陈游旻, 胡庆达, 等. 非易失主存的系统软件研究进展[J]. 中国科学:信息科学, 2021, 51(6): 869-899.