中国科学: 信息科学 2023 年 第 53 卷 第 8 期: 1503–1528
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
国防科技大学建校 70 周年专题 • 评述

分离式数据中心的存储系统研究进展

舒继武*, 陈游旻, 汪庆, 王晶, 李俊儒, 廖晓坚

清华大学计算机科学与技术系, 北京 100084
* 通信作者. E-mail: shujw@tsinghua.edu.cn

收稿日期: 2023–02–14; 修回日期: 2023–05–31; 接受日期: 2023–06–15; 网络出版日期: 2023–08–17

国家重点研发计划 (批准号: 2022YFB4500302)、国家自然科学基金 (批准号: 61832011, 62202255) 和中国博士后创新人才支持计划 (批准号: BX2021154) 资助项目

摘要 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存储系统带来了新的挑战. 首先, 分析了分离式数据中心的驱动因素, 阐述了其架构特点及优势, 并综述了对应存储系统的关键技术和代表性研究工作; 然后, 围绕数据容错、异构计算及异构网络, 展望了未来的发展趋势并给出了总结.

关键词 分离式数据中心, 分离式内存, 分离式存储, 存算分离

1 引言

近年来, 全球数据呈几何级数高速增长, 大数据作为战略资源的地位日益凸显; 数据中心作为数据的载体, 是支撑未来大数据发展的重要基石. 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灵活度受限; 例如: 当内存资源短缺时, 维护人员需要购买完整的服务器进行内存扩容, 这将导致其他硬件资源 (例如磁盘和处理器) 的浪费. 此外, 大规模分布式应用通常以服务器为最小粒度进行任务调度, 不同节点间因业务的繁忙程度不同而呈现出显著的资源利用不均衡, 并导致硬件资源利用率低下; 例如, 谷歌 (Google) 公司集群内存空间利用率平均仅 45% [1]. 最后, 云服务商通常依赖虚拟化技术、分布式存储软件等将硬件资源出售给租户, 这些基础设施软件会耗费大量

引用格式: 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. 中国科学: 信息科学, 2023, 53: 1503–1528, doi: 10.1360/SSI-2023-0034
Shu J W, Chen Y M, Wang Q, et al. Progress on storage systems for disaggregated data centers (in Chinese). Sci Sin Inform, 2023, 53: 1503–1528, doi: 10.1360/SSI-2023-0034

© 2023《中国科学》杂志社

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

的 CPU (central processing unit) 资源; 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% [2]. 为此, 英特尔 (Intel)、惠普 (Hewlett-Packard) 等公司相继推出了分离式数据中心架构 (disaggregated datacenter architecture): 该架构摈弃了传统服务器的 “打包式” 架构, 将硬件资源按类别拆分为不同的硬件资源池 (例如, 处理器池、内存池、存储池等), 并通过高速网络将上述资源池互连, 从而呈现出 CPU 与内存分离、计算与存储分离、应用负载与基础设施负载分离的多维度解耦架构. 基于此架构, 维护人员可按需扩展特定的硬件资源, 不同业务可在集群内共享硬件资源, 进而提升硬件资源利用率, 同时, 基础软件功能还可以通过异构计算池灵活卸载, 降低 “数据中心税”.

受限于网络技术的发展, 分离式数据中心架构长期处于概念阶段, 惠普公司是分离式数据中心架构最早的发起者之一, 其领导的 The Machine 项目 [3] 最终以失败告终. 近年来, 新型网络技术、异构计算硬件等得到了长足发展, 例如, 英伟达 (NVIDIA) 公司推出的新一代远程直接内存访问 (remote direct memory access, RDMA) 网卡 ConnectX-7 的带宽达到 400 Gbps, 延迟低于 1 μs, 已经十分接近 DRAM (dynamic random access memory) 内存的性能; 英特尔领导的内存语义互连协议 CXL (compute express link) 能够让 CPU 与内存节点、GPU (graphics processing unit)、FPGA (field-programmable gate array) 或其他加速器之间实现高速互连, 从而满足高性能异构计算的要求; 英伟达推出的新一代数据处理器 (data processing unit, DPU) 能够对各种网络、存储和安全业务进行卸载、加速和隔离, 有效降低通用 CPU 的负担; 英特尔、英伟达等公司推出的可编程交换机、智能网卡等可以在网络数据包传输过程中灵活修改其转发逻辑, 并结合其设备内的缓存空间实现软硬件协同优化. 随着硬件技术的发展, 分离式数据中心架构的落地成为可能. 然而, 基于资源池化的分离式数据中心架构相较于传统计算机体系结构具有显著差异, 并引发一系列新的挑战, 这主要体现在以下几个层面:

访存模式. 传统服务器架构下, 处理器通过内存指令访问 DRAM 主存, 访问延迟仅数十纳秒, 带宽高达 100 GB/s; 分离式架构下, 处理器和内存池通过高速网络互连. 这一变化主要带来两方面的挑战: 首先, 基于高速网络的远程内存访存延迟和带宽相较于内存总线仍存在一定差异, 因此, 内存数据结构常用的指针追逐等将造成延迟高、带宽利用率低等问题; 其次, 当通过内存语义网络访问各层级内存时, 操作系统无法区分数据的具体存储位置, 这将导致数据放置、服务质量控制等面临挑战.

存储层级. 在传统计算机存储层级中, 硬盘、DRAM、高速缓存等存储介质的容量逐渐变小, 速度逐渐变快, 呈现出经典的 “金字塔” 结构; 然而, 异构存储资源池引入了持久性内存 (persistent memory, PM)、高速固态硬盘等新型存储介质, 这使得不同存储层级的界限变得模糊. 例如, PM 与 DRAM 的访问延迟接近 (∼100 ns)、PM 与高性能固态硬盘的带宽接近 (单个设备约 2∼6 GB/s) 等, 这使得现有的缓存、分级等数据管理策略逐渐失效.

容错模型. 传统服务器的 CPU 或内存出现故障将导致整个服务器停止服务, 而在分离式数据中心架构下, 某类硬件发生故障并不会影响其他硬件资源的正常运行. 例如, 当某一处理器发生故障时, 处理器池中的其他处理器及远端内存池均能正常运转, 由于处理器本身并不存储数据, 因而其运行逻辑可以被无缝迁移至其他处理器. 由此可见, 分离式数据中心架构独有的故障独立的特性将改变现有的容错模型.

软件开销. 目前, 高性能存储介质 (例如英特尔推出的 Optane SSD 等) 的访问延迟已低至 10 μs 内. 从软件角度, 现有的操作系统均针对传统外存 (例如磁盘等) 进行设计, 其 I/O 栈臃肿复杂, 基于中断机制的调度策略延迟高达数毫秒, 高昂的软件开销将完全掩盖高性能存储器件的低延迟特性. 从硬件角度, 现代处理器均采用多级流水线架构以屏蔽内存访问造成的纳秒级延迟 (∼100 ns), 但却无法掩盖外存存储器件造成的微秒级延迟. 美国科学院院士 Patterson 等 [4] 将上述矛盾总结为 “killer microsecond” 问题. 在分离式数据中心架构下, 异构存储池包含各类延迟不同的存储器件, 如何有效利

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

用异构存储器件, 并克服现有存储软件在延迟和效率方面难以平衡等缺陷, 是目前亟待解决的难题.

综上所述, 传统的存储系统构建方法无法应对分离式数据中心架构在访存模式、存储层级、容错模型、软件开销等方面的变化. 因此, 目前学术界和工业界正在积极地开展分离式数据中心架构与存储系统的相关研究. 本文首先介绍分离式数据中心的两大驱动因素: 新的业务挑战以及硬件技术趋势; 然后对分离式数据中心的架构进行总体描述; 在此基础上, 阐述了分离式存储系统的相关工作研究进展; 最后进行总结并展望未来研究趋势.

2 分离式数据中心的驱动因素

分离式数据中心的诞生离不开两大因素的驱动: 一方面, 新的业务挑战使得传统数据中心面临硬件资源利用率低等问题; 另一方面, 硬件技术趋势保证了将硬件资源分离后, 系统性能所受的影响可控, 并提供了硬件卸载机会, 以降低数据中心税.

2.1 新的业务挑战

传统的数据中心将计算、内存和存储资源打包至服务器中, 并将服务器作为基本组成单元. 这种方案具有灵活易部署的特点, 但目前面临着如下的业务挑战, 在资源利用率、扩展性、性能等诸多方面出现如下问题:

(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, 比如, 服务器中的存储资源可能会随 CPU 升级而淘汰, 为此需进行相应的数据迁移等 [5].

(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 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 IO 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

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

的其他资源 (即网络、内存和存储资源).

2.2 硬件技术趋势

新兴高速网络和高速存储器件是分离式数据中心架构的主要技术驱动力, 而可编程网络设备的出现为分离式数据中心架构带来了加速卸载的机遇. 本小节将依次介绍这些硬件的相关背景.

2.2.1 新兴网络技术

新兴的网络技术可提供高带宽、低延迟的远程访问能力. 这里介绍用于分离式数据中心的 3 种主要的网络技术: RDMA, CXL, 以及 NVMe-oF (NVMe over fabrics).

RDMA 网络技术. RDMA 网络在现有数据中心被广泛使用, 它提供两种异步的网络原语: 双边原语和单边原语. 双边原语和传统的 Linux 套接字接口相似: 发送端通过 `SEND` 原语发送消息, 接收端通过 `RECV` 原语预留消息缓冲区. 单边原语主要包括 `READ` 和 `WRITE`, 它们能够完全绕过接收端的 CPU, 直接读/写接收端的内存, 因此非常适合于读写远程内存池的场景. 此外, 还有一类单边原语支持原子的读后写操作, 包括比较并交换 (compare-and-swap, `CAS`) 以及获取并自增 (fetch-and-add, `FAA`), 可用于内存池上并发访问的同步与互斥. 数据中心的 RDMA 网络大多部署在以太网之上, 使用 RoCE (RDMA over converged Ethernet) 协议, 其端到端的往返延迟一般小于 3 μs. 相比于传统的内核 TCP/IP (transmission control protocol/internet protocol) 协议栈, RDMA 的极低延迟主要来源于以下两方面: 在硬件上, RDMA 网络协议栈被完全卸载至网卡; 在软件上, 应用在用户态直接与 RDMA 网卡进行交互, 避免了操作系统的介入, 达到了零拷贝的效果. 近些年 RDMA 网络的带宽增长十分迅猛, 逐渐接近于内存带宽, 以英伟达公司的 ConnectX 系列 RDMA 网卡为例: 其 2009 年发布的 ConnectX-2 网卡带宽最高为 10 Gbps, 而 2021 年发布的 ConnectX-7 网卡 [6] 带宽最高可达 400 Gbps, 提升了整整 40 倍.

CXL 网络技术. CXL 网络由英特尔公司等公司提出, 用于实现 CPU、内存、加速器之间的高效互连. 在分离式数据中心架构下, 利用 CXL 网络可以构建大内存设备, 形成内存池抽象, 并支持处理器池中的 CPU, GPU 等处理器通过同步的原生 `load/store` 指令对其进行访问. 此外, CXL 网络支持缓存一致性, 由此简化数据共享访问时的软件设计. CXL 网络的物理层与 PCIe 物理层相同, 因此可以融入成熟的 PCIe 生态. 在带宽上, CXL 网络可达到 63 GB/s (假设使用 PCIe 5.0×16). 在延迟上, 根据微软最近的数据 [7], 当 8 个 CPU 共享基于 CXL 的内存设备时, 访问延迟仅有 155 ns; 而当 64 个 CPU 共享时, 由于需要在 CXL 网络拓扑中添加中继器和 CXL 交换机, 访问延迟会随之增长, 达到 270 ns.

NVMe-oF 网络技术. NVMe-oF 将 NVMe 存储协议扩展至网络, 支持处理器访问远程的高速存储设备, 因此能用于构建分离式数据中心架构中的存储池. 在 NVMe-oF 网络中存在 initiator 和 target 两种角色, 其中 initiator 为客户端, 用来发送 NVMe 命令; 而 target 为服务端, 用来管理存储设备并解析执行接收到的 NVMe 命令. 因此, 与 RDMA 和 CXL 的字节粒度接口不同, NVMe-oF 的数据访问粒度为块. NVMe-oF 可以运行在 RDMA, TCP 或者光纤通道之上. 由于其协议开销低、并行度高, NVMe-oF 通常能发挥 target 端存储设备的最大带宽, 而延迟也接近于本地访问. 现有的一些 RDMA 网卡, 如 ConnectX-5, 支持将 NVMe-oF 的 target 任务卸载至网卡硬件中, 即网卡硬件执行 NVMe 协议, 直接读写存储设备, 由此消除了 target 端的 CPU 介入.

表 1 对以上 3 种网络技术进行了总结对比.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

表 1 3 种网络技术的对比

Table 1 Comparison among three types of network techniques

Type	Synchronous/asynchronous	Scalability	Access granularity
RDMA	Asynchronous	High	Byte
CXL	Synchronous	Medium	Byte
NVMe-oF	Asynchronous	High	Block

2.2.2 高速存储器件

各种高速存储器件层出不穷, 使得分离式数据中心架构中的存储池的异构性大大提升. 这里主要介绍高速固态硬盘和持久性内存这两类存储器件.

高速固态硬盘. 传统的固态硬盘主要基于 NAND 介质, 访问延迟在 100 μs 量级; 此外, 由于其异地更新的特点, 需要复杂的闪存转换层支持, 并辅助以垃圾回收机制, 这造成了存储性能的不稳定. 近些年, 英特尔发布了基于 3D XPoint 介质的傲腾固态硬盘 (Optane SSD) [8]. 相比于传统基于 NAND 的固态硬盘, 傲腾固态硬盘主要具有如下 3 方面的性能优势: (1) 访问延迟极低, 在 10 μs 左右; (2) 随机访问性能与顺序访问性能接近; (3) 性能稳定, 当存储空间接近用完时, 读写性能仍然能维持在峰值.

持久性内存. 持久性内存是一类新型存储器件, 它同时拥有 DRAM 和磁盘的特点: 与 DRAM 相似, 持久性内存被连接在内存总线之上, 因此 CPU 可以通过 `load/store` 指令对其进行读写, 访问延迟极低; 与磁盘相似, 持久性内存密度高, 且能够持久性保存数据, 即其上的数据掉电后也不会丢失. 持久性内存的主要商业产品之一是英特尔发布的傲腾持久性内存 (Optane DC persistent memory) [9], 其单条容量可高达 512 GB, 且性能表现出明显的读写不对称: 写延迟为 100 ns, 而读延迟为 300 ns; 写带宽为单条 2 GB/s 左右, 而读带宽为单条 6 GB/s 左右.

2.2.3 可编程网络设备

在分离式数据中心架构中, 可编程网络设备提供了在数据通路上计算的能力, 能有效提升处理器池访问内存池和存储池的性能. 这里主要介绍可编程交换机和智能网卡这两类可编程网络设备.

可编程交换机. 传统的交换机功能固定, 能够支持的网络协议在设备出厂时就已决定, 而近些年出现的可编程交换机则颠覆了这一假设. 通过可编程交换机, 用户不仅可以自定义网络协议的格式, 并且可以编程网络包处理的逻辑 (例如修改、转发等); 此外, 可编程交换机提供一段内存区域, 用于数据的存储.

现有可编程交换机大多基于硬件流水线架构, 性能强悍: 例如英特尔公司的 Tofino 可编程交换机 [10~13], 该系列中的 Tofino 3 交换机单端口带宽最高为 400 Gbps, 聚合带宽可达 25.6 Tbps. 此外, 可编程交换机的网络包处理延迟仅为百纳秒. 为了达到如此高的性能, 可编程交换机的编程模型和硬件资源较为受限: 首先, 它只支持有限类型的运算操作和有限数目的运算步骤; 其次, 它提供的内存为高速的 SRAM, 仅有数十 MB 大小.

智能网卡. 智能网卡在普通网卡的基础上增添了可编程芯片, 用于对网络包进行自定义处理, 卸载 CPU 任务. 根据可编程芯片的种类, 不同的智能网卡具有不同的特点: 基于 ARM CPU 的智能网卡处理性能较低, 但具有成熟的工具链, 且容易编程, 典型代表为英伟达的 BlueField 智能网卡系列 [14,15]; 基于现场可编程门阵列 (FPGA) 的智能网卡性能高, 但编程难度较大, 典型代表为英伟达的 Innova 智能网卡系列 [16,17]. 目前, 这些智能网卡能提供百 Gbps 的带宽, 并且一般配备有 1~16 GB 的 DRAM 内存, 用于数据缓存等目的.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

表 2　2 种可编程网络设备的对比

Table 2　Comparison among two types of programmable network devices

	Programmable switch	SmartNIC
Chip type	ASIC	ARM/FPGA
Network bandwidth	5∼25 Tbps	10∼200 Gbps
Expressive power	Low	High
Memory capacity	∼10 MB	1∼16 GB

表 2 对以上 2 种可编程网络设备进行了对比.

3　分离式数据中心架构

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 1 [5] 所示. 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, 以及高速的 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. 计算池主要包括 CPU 池、GPU 池、FPGA 池, 以及其他异构计算资源. 在网络互连方面, RDMA 支持计算池直访内存池和持久性内存池; NVMe-oF 支持计算池直访 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, 更适合于小规模的数据中心.

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, 计算池中的处理器会配备少量的本地内存用于缓存 (如 1 GB 的 DRAM), 以减少网络访问. 类似的, 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 4∼8 颗 ARM CPU 核心), 用于执行一些管理任务 (如空间分配、垃圾回收). 这些本地资源的存在看似是与资源分离的初衷是矛盾的, 但是从整个数据中心级别来看, 资源池中的特定资源所占的比例远远大于本地对应的资源比例, 因此仍能达到高效的资源分离效果. 随着网络性能的持续提升和内存/存储硬件功能的持续丰富, 分离式数据中心架构在未来有望完全消除本地资源的存在.

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. 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, 分离式数据中心架构最为显著的优势在于:

(1) 更为彻底的存算解耦. 分离式数据中心架构不再局限于将 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 (例如计算池、内存池、机械硬盘 (HDD)/固态硬盘 (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, 并极大提升资源利用率. 例如根据微软的初步估算 [7], 当 16 颗 CPU 共享访问 DRAM 池时, 能减少整个数据中心 7% 的 DRAM 使用量, 这相当于为一个大型的云服务提供商节省了数亿美元的成本.

(2) 更为细粒度的处理分工. 分离式数据中心架构打破了传统以通用 CPU 为中心的处理逻辑, 使

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

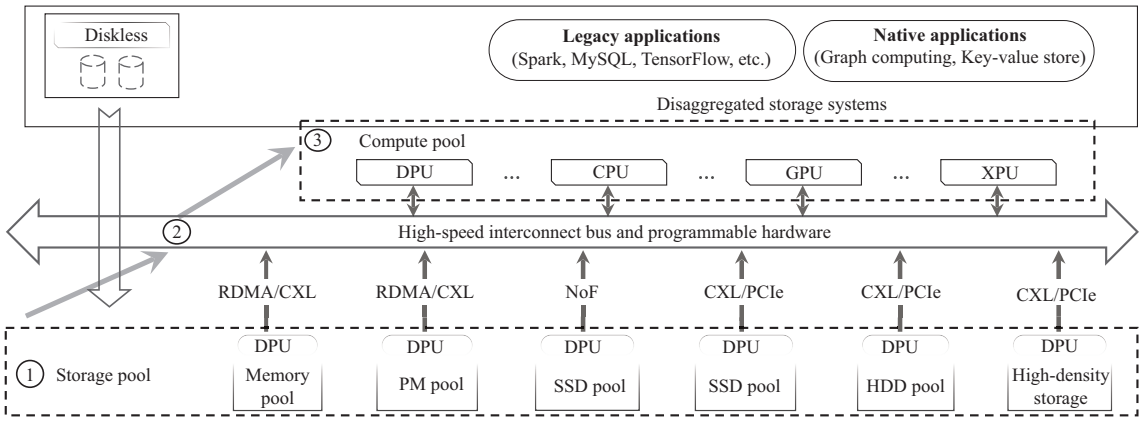


图 1 分离式数据中心的架构 [5]

Figure 1 Architecture of disaggregated data center [5]

表 3 3 种分离式数据中心实例的对比

Table 3 Comparison among three types of disaggregated data center examples

	LegoOS [1]	Clover [18]	MIND [12]
Type of local cache	Hardware-managed	Software-managed	Software-managed
Cache coherence	Not support	Not support	Support
Programmable network devices	Not equipped	Not equipped	Equipped

数据处理、聚合等原本 CPU 不擅长的任务被专用加速器、DPU 等替代, 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、降低数据中心税.

尽管具有诸多优势, 分离式数据中心架构也存在一些局限性. 首先, 由于所有的资源被网络互连, 网络的影响极大: 一方面, 计算池需跨网访问其他资源, 引入了性能开销, 正如本文第 4 节将描述的, 减少远程访问是众多分离式存储系统的优化目标; 另一方面, 网络由于升级或者设备故障会导致重新配置、重构等事件, 这将对运行在分离式数据中心之上的应用的可用性造成挑战, 如何为分离式数据中心设计高可用的动态网络配置机制是一个值得研究的课题. 其次, 在系统运维方面, 尽管分离式数据中心支持硬件资源的细粒度扩展, 可达到更佳的运维效果, 但运维的复杂度会有所增加: 分离式数据中心的异构性极高, 包括异构算力 (如 FPGA, GPU)、异构网络 (RDMA, CXL) 等, 它们的故障特征、配置方式等各不相同, 提高了相关人员的运维负担.

现有研究文献中的分离式数据中心实例大多遵循以上的描述, 但在一些设计抉择上略有差异. 表 3 [1,12,18] 展示了 3 类具有代表性的分离式数据中心实例. 对于计算池的本地缓存, 大部分实例是通过软件管理的, 而 LegoOS [1] 等少数实例将本地缓存的管理移交给硬件, 以提升性能. 同时, 计算节点之间的缓存是否需要保证一致性也是一个重要设计抉择: 大部分实例不支持缓存一致性, 以避免一致性开销, 只有少数 (如 MIND [12]) 为了简化软件设计的难度而支持缓存一致性. 此外, 有些实例配备了可编程交换机、智能网卡等可编程网络设备, 支持卸载加速 (具体例子见第 4.4 小节); 而其他实例为了降低部署成本, 并未配备这些设备.

基于分离的资源池, 研究者们构建了分离式存储系统: 一方面, 将内存资源池进行统一管理, 进而提供运行时内存给计算池, 以支持 CPU, GPU 等计算处理器正常运行进程; 另一方面, 提供块、键值、文件等接口, 用于持久可靠地存储数据. 分离式数据中心支持两类应用: 第一类是基于服务器抽象的

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

传统应用, 此时, 计算池提供处理器、内存池提供物理内存 (基于 swap 机制或者 CXL 直访)、存储池提供块设备, 三者共同组成了逻辑上的服务器 (虚拟机), 因此可以运行任何现有的应用, 用户的编程方式也无任何改变. 第二类是基于分离式资源的原生应用, 此时, 运行在处理器上的应用能感知到被拉远的内存和存储资源, 并显式调用各自的访问接口, 完成应用的功能. 这里通过键值存储系统为例, 阐述传统应用和原生应用的区别. 基于服务器抽象的键值存储系统可以是 RocksDB 等现有的成熟系统, 它们不需任何修改即可运行在计算池的处理器之上, 此时操作系统与远程资源池打交道, 提供内存分配、文件系统等具有兼容性的系统接口. 基于分离式资源的原生键值存储系统需从头设计并发索引结构 (见第 4.2.1 小节) 和数据分区策略 (见第 4.2.3 小节) 等, 根据数据访问特性显式地将数据放置在不同存储层级 (计算池的本地内存、内存池、SSD 池等), 并支持动态地数据迁移. 此外, 不同于基于服务器抽象的传统应用, 此时处理器访问远程资源池需要显式地调用 RDMA 或 NVMe-oF 等访问接口. 相比于传统应用, 由于可根据应用语义管理资源池, 并消除了数据路径上的操作系统开销, 原生应用可以达到更高的性能. 但另一方面, 原生应用面临着构建难度大的问题, 编程者需要根据应用特性进行针对性设计, 存在着诸多挑战, 例如并发控制、缓存管理等.

4 分离式存储系统及关键技术

分离式存储系统对存储资源池进行统一管理, 并将其提供给计算池中的处理器使用. 构建分离式存储系统涉及诸多关键技术, 本文将其分为 4 类.

(1) 操作系统及运行时技术. 操作系统及运行时层次的关键技术包括接口抽象技术和数据交换技术. 前者将内存资源以易用高效的形式暴露给计算池中的处理器硬件. 后者的主要任务包括识别出工作集中的热点数据, 并在本地内存和远端内存之间的快速迁移数据.

(2) 分离式内存存储技术. 分离式内存存储系统可提供低延迟的数据访问. 其核心的技术挑战在于当存在多个计算节点时, 如何保证并发读写的正确性和高性能, 这涉及到并发索引、分布式事务协议以及数据分区策略等技术.

(3) 分离式外存存储技术. 与分离式内存存储系统不同, 分离式外存存储系统采用 SSD 池等具有块接口的资源池, 需要提供给上层应用的语义更丰富, 包括对象接口以及具有目录树结构的文件接口等, 还需要考虑持久性等重要指标. 其中的关键技术包括 I/O 栈优化、对象及文件管理等.

(4) 分离式存储系统的卸载加速技术. 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可编程硬件进行数据管理、分布式协议等的卸载, 由此减少软件开销并提升性能.

4.1 操作系统及运行时技术

在分离式数据中心架构下, 操作系统及运行时的目标主要有两个: 首先, 通过接口抽象技术为上层应用 (例如存储系统) 屏蔽底层的硬件分离细节; 其次, 通过数据交换技术使得数据读写的性能尽量接近于本地内存访问.

4.1.1 接口抽象

现有的接口抽象技术主要有以下 4 种, 它们在性能和兼容性方面各有优劣, 详细对比 (包括对应的代表性系统、互连技术、兼容性、管理粒度、性能等) 如表 4 [19~40] 所示.

基于操作系统内存交换机制. 当物理内存不足时, Linux 内核的内存管理子系统有成熟的交换机

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

表 4 分离式内存接口抽象对比

Table 4 Comparison of software management of disaggregated memory

Interface abstraction	Representative systems	Interconnection	Compatibility	Management unit	Performance
Swapping	Hotpot [19], Remote Regions [20], InfiniSwap [21], Fastswap [22], Leap [23], TMO [24], Canvas [25]	RDMA	High	Page (4 KB)	Low
AutoNUMA	AutoTiering [26], Nimble [27], HotBox [28], HeMem [29]	CXL	High	Page (4 KB)	Medium
User library	AIFM [30], Carbink [31], Octopus [32,33], TH-DPMS [34], FaRM [35], soNUMA [36], Kona [37]	RDMA	Low	Arbitrary size	High
JAVA runtime	Semeru [38], Mako [39], MemLiner [40]	RDMA	High	Arbitrary size	Medium

制将内存数据替换至外存, 并释放对应的物理内存页面. 部分分离式内存管理正是沿用了这一内存交换机制, 其主要区别在于: 传统交换机制是将内存页面交换至由机械磁盘或固态硬盘构成的 swap 分区, 而分离式内存则是将内存页面交换至由远端内存节点的内存构成的 swap 分区. swap 分区由一系列交换项 (swap entry, SE) 构成, SE 通过一个全局唯一的 ID 索引, 并对应到远端内存节点上的某一具体物理内存页面. 当应用程序的访存操作触发缺页中断, 且缺页中断处理函数判定当前页面位于 swap 分区时, 首先查询由 swap 分区管理的 DRAM 缓存, 这些缓存页面通过字典树索引, 用于存放 swap 分区中刚被拉取至本地的或即将被换出的页面. 如果缓存查询命中, 则直接更新对应缓存页面的页表项, 完成页面装载. 否则, 内核需要完成一次换入操作. 该过程中, 首先通过 cgroup 判定当前进程组是否具有足够的本地物理内存空间, 如果是, 则直接通过 RDMA 读原语将存放在远端内存节点的内存页面读取至本地, 并存放至 swap 缓存, 最后完成页面装载; 与此同时, swap 分区的守护线程还会根据当前应用程序的数据访问特性进行数据预取. 相反地, 如果 cgroup 发现当前物理内存空间不足, 则需要先将最近未被访问的页面逐出至远端内存, 再执行换入操作.

基于内存访问自动均衡 (autoNUMA). 非均衡存储器访问 (non uniform memory access, NUMA) 架构是目前多处理器服务器使用最广泛的架构, 其显著特点在于 CPU 访问本地直连内存的速度快于通过核间通信机制访问与其他 CPU 直连的内存. 在 NUMA 架构下, 为避免访问远端内存而导致性能下降, 默认的内存分配策略一般优先分配与本地 CPU 直连的内存空间. 然而, 如果本地内存空间不够, 分配器仍需要从远端内存分配空间. 针对以上问题, Linux 内核引入了内存访问自动均衡 (automatic NUMA balancing, autoNUMA) 技术, 将热点访问页面从远离 CPU 的内存区域迁移至近端内存, 从而提升程序执行效率. 随着分离式内存架构受到广泛关注, autoNUMA 也随之成为 Linux 内核维护最活跃的模块. 由于 autoNUMA 最初被引入是为了解决 NUMA 架构的访问不均衡问题, 因此, 在分离式内存中, 需要不同内存层级通过具有内存语义的互连技术与 CPU 相连.

基于专用用户库. 内存交换、autoNUMA 等内核态技术具有良好的应用透明性, 但是通常以 4 KB 页面为最小管理粒度, 当应用程序的访问数据项尺寸较小时, 容易造成读写放大问题, 为此, 部分研究人员提出基于用户态专用库的接口抽象 (例如 AIFM [30], Carbink [31] 等). 以 AIFM 为例, 它将分离式内存中的数据通过对象管理; 同时, 不同对象之间可以通过智能指针进行索引, 进而为应用程序提供了高度封装的数据结构, 包括向量、列表、栈、队列、哈希表等. 这些数据结构的元素可以任意分布在不同的内存层级, AIFM 则会根据元素的访问频率实现数据的自动迁移, 而应用程序完全不需要关注具体的数据摆放策略.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

基于 JAVA 运行时. 现有工作大多从操作系统层次研究分离式内存接口抽象, 主要面向 C/C++ 等原生语言, 缺乏对托管语言 (例如 JAVA 等) 的支持. 以 JAVA 为例, 应用程序运行在 JAVA 虚拟机之上, JAVA 虚拟机负责管理线程、内存等系统资源, 并回收被释放的内存空间, 而应用程序则不用关心内存的具体摆放位置和生命周期. 基于 JAVA 运行时提供分离式内存接口抽象主要面临两方面的挑战: 首先, GC 工作线程回收被释放的内存时通常涉及图遍历操作, 当数据同时分布在近远端内存时, 局部性差的遍历操作会严重影响性能; 其次, JAVA 程序大量使用面向对象的数据结构, 涉及频繁的指针追逐操作, 进而引发大量的跨网访问. 针对以上问题, Semeru [38] 提出了一种分布式 JAVA 虚拟机方案, 其中计算节点 CPU 执行应用程序, 而内存节点的 CPU 天然具备内存亲和性, 负责运行内存管理相关的任务.

4.1.2 数据交换

分离式数据中心架构中内存层级丰富, 包含计算节点的本地内存、内存节点的远端内存等, 为了保证应用程序运行的高效性, 跨层级内存交换策略就显得尤为重要. 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.

热点追踪机制. 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, 主要包括以下几类热点内存数据的追踪机制.

• 软件插桩. 这类方法通过软件手段在访存指令前后增加额外的统计代码, 从而获得每次内存访问对应的地址、大小等信息, 进而统计出程序运行的内存访问在时间和空间上的分布. 该方法的统计信息准确、但需要运行额外的插装代码, 对性能影响较大.

• 页表标记位. Linux 内核的页表项中, 除了具体的内存页面物理地址信息, 还包含各类标志位信息, 其中读 (accessed, A) 和脏 (dirty, D) 两种权限位可用于统计应用程序的具体访存信息. 具体地, 操作系统通过守护线程将当前进程页面对应页表项的上述权限位置零, 当应用程序实际访问到某一页面时, 硬件地址翻译单元将会根据访问类型修改对应页表项的 A 或 D 权限位. 守护线程定期扫描这些权限位, 统计在过去时间窗口那些页面被应用程序访问或修改. 由于这些权限位被置零后仅能统计一次访存操作, 因此, 时间窗口越短, 其统计信息越准确, 但守护线程消耗 CPU 资源也越高; 此外, 当某一进程的内存占用量很高时, 页表项扫描开销也随之线性增高. 目前, Linux 内核的 autoNUMA 模块采用了此方法统计访存信息.

• CPU 硬件计数. 处理器一般支持基于事件的内存访问采样技术 (precise event-based sampling, PEBS). 通过 PEBS, 处理器可以在特定事件 (例如缓存行缺失等) 发生时将对应访存操作的目的地址信息存储到一个预先分配的内存空间中. 相较于普通的硬件计数手段, PEBS 可以记录更为精确的地址信息, 且支持可配置采样, 因此, 可以根据系统的精准度需求调节采样频率, 在不影响应用程序性能的同时实现最高的精准度. 目前, HeMem [29] 等采用了此方案.

数据迁移机制. 当准确获得了当前应用程序访问页面的冷热分布情况后, 需要根据页面的实际存储位置执行数据迁移, 将访问频繁的页面从远端内存读取至本地, 同时将本地访问不频繁的页面逐出至远端. 现有工作主要集中关注迁移时机、通路选择、前后台协调等问题.

在迁移时机选择方面, TMO [24] 将某一程序的内存资源紧张程度量化为由于内存资源紧张而造成的额外执行时间, 并根据这一量化指标确定当前进程所需的本地内存空间, 并将本地存放不下的页面逐出至远端内存.

在数据通路优化方面, autoNUMA 不允许在本地内存已满时将存放在远端的访问频繁的页面迁移

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

至本地. 为此, autoTiering [26] 丰富了现有的数据迁移通路, 允许在本地内存已满时将其中存放的访问不频繁的页面逐出, 以腾出空间存放远端内存中的热点页面. Fastswap [22] 在网络层面提供了不同数据流的隔离性, 使得急需的页面请求可以比预取请求拥有更高的优先级享受网络服务. Canvas [25] 重新设计了 Linux 内核数据交换模块中的 SE 分配、网络传输等, 降低了锁冲突, 提升了网络隔离性等, 在多个应用同时使用远端内存时, 性能抖动降低达 7 倍.

在前后台协调方面, 后台数据迁移任务消耗了一定的系统资源, 同时还需要和前台进行同步以避免造成数据不一致的情况发生, 这在一定程度上会影响前台应用程序的性能. 上述问题在 JAVA 虚拟机场景下更为显著: 应用程序的内存工作集往往和垃圾回收线程的内存工作集不一致, 从而造成彼此对本地内存的竞争, 进而转化为频繁的近远端内存数据交换. 为此, MemLiner [40] 提出了一种内存工作集对齐机制, 将垃圾回收线程所访问的内存对象进行重排序, 使得这些内存访问序列与工作线程尽可能相似, 进而减少对本地内存空间的竞争.

数据预取策略. 数据迁移机制是根据现有的热点信息调整内存页面在近远端的摆放, 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, 并提前将对应的页面存放至本地内存. Linux 内核的内存交换模块默认具备简单的预取功能, 例如, 当应用程序的过去连续两次访问的页面地址连续时, 预取模块会判定应用程序具备顺序访问特性, 并将后续的页面提前拉取到本地. 然而, 在分离式内存中, 预取页面需要跨网数据传输, 预取出错代价很高, 因此预取的准确性极为重要. 为此, Leap [23] 提出了一种基于主体趋势的预测方法, 该方法从历史访问信息中获取主体趋势信息, 而忽略其他的噪声. 具体地, 针对过去的若干次页面访问, 选取一个较小的窗口尺寸 w, 然后尝试从 w 中寻找是否存在一个主体趋势 Δ, 其重复次数超过 $\lfloor w/2 \rfloor + 1$; 若是, 则返回该趋势, 否则, 增大窗口尺寸, 再次寻找. Canvas [25] 还发现, JAVA 虚拟机中的内存访问操作大多基于指针追逐, 跳跃性较高, 导致内核级别的预取策略经常失效, 因此, Canvas 提出了两级预取机制, 即增加一层 JAVA 虚拟机的预取策略, 根据对象的指针引用信息判定接下来的内存访问地址, 从而有效提升预取精度.

4.2 分离式内存存储技术

近年来, 随着持久性内存、大容量内存等器件的商业化, 内存存储系统研究也迎来了新一波热潮 [41~71]. 然而, 现有研究大多基于服务器架构, 无法直接应用到分离式架构下. 本小节主要介绍分离式内存存储系统中的 3 个核心技术: 并发索引技术、分布式事务协议, 以及数据分区策略.

4.2.1 并发索引技术

与传统并发索引不同, 由于内存池的计算能力有限, 分离式内存存储系统中的索引一般使用 RDMA 单边原语来完成查询与修改操作. 如何设计 RDMA 友好的数据结构、如何协调并发操作是其中的主要挑战. 本小节介绍为分离式内存设计的哈希表 RACE hashing [72] 和 B^+ 树 Sherman [73].

RACE hashing 是一个全单边 RDMA 访问的分布式哈希索引. RACE hashing 采用了可扩展的哈希表结构, 并支持无锁的远程并发控制和远程扩容. 如图 2 [72] 所示, RACE 哈希表由多个子表 (subtable) 和一个目录组成. 为了减少额外的 `READ` 操作, RACE hashing 客户端在本地缓存目录, 通过本地内存访问目录, 再通过 RDMA 原语访问子表. 子表是一个一维的 bucket 数组. 为减少哈希冲突并保持高负载因子, RACE hashing 采取了多路关联度的 bucket, 采取了类似布谷鸟哈希的两路选择方案, 此外还为每两个主 bucket 提供了一个 overflow bucket 用于存放冲突的键值项. 为了减少远程 RDMA 操作次数、提高查询效率, RACE hashing 将 overflow bucket 与对应主 bucket 连续存储, 以便于一次 `READ` 操作可以将主 bucket 和 overflow bucket 一并读取.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

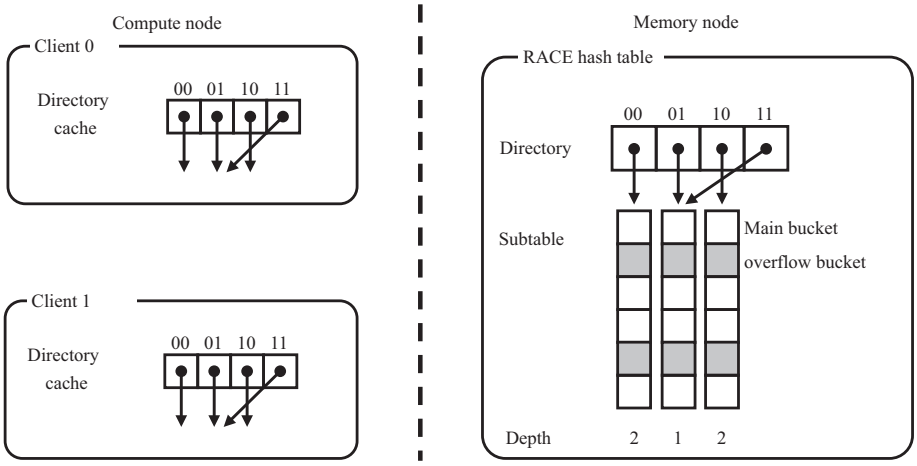


图 2 RACE hashing 的总体架构 [72]

Figure 2 The overall architecture of RACE hashing [72]

对于无锁查询、更新, 以及删除操作, 客户端首先使用 `READ` 读取对应 bucket. 由于 bucket 中的键值项不包括完整的 key 而是 key 的指纹信息, 对指纹匹配的项还需要进一步读取实际键值对象以判断 key 是否相同. 对于无锁插入操作, 客户端在 bucket 中查找是否有空槽, 有则使用 `CAS` 插入新的键值项, 否则触发远程扩容操作. 客户端执行远程扩容操作时, 先对位于内存池的目录加锁, 该锁仅防止其他客户端并发扩容, 但不会阻塞客户端执行增删改查操作. 客户端创建新的子表后更新目录中的 bucket 指针, 并将 rehash 指定的键值项移动到新的子表中.

清华大学提出了一个针对分离式内存写优化的分布式 B^+ 树 Sherman. 如图 3 [73] 所示. Sherman 的树节点被均匀分散在内存池上, 客户端节点使用 RDMA 单边原语完成对 B^+ 树的操作请求. Sherman 利用现有商用 RDMA 网卡, 结合 RDMA 网卡的硬件特性和 RDMA 友好的软件技术加速 B^+ 树的写性能.

首先, 通过 RDMA 单边原语对树型索引往往需要多次 RDMA 网络往返, 导致较高的延迟, 例如一次写操作包括加锁、读树节点、写回修改后的树节点、释放锁, 以及可能涉及到树节点分裂导致的更多 RDMA 操作. Sherman 利用 RDMA 提供的硬件级强顺序性特性, 合并多个 `WRITE` 操作以减少网络往返, 如合并写回树节点和释放锁的操作、写回分裂的新节点和原节点等. 此外, Sherman 在客户端缓存中间节点, 以减少网络往返.

其次, 在高并发、高倾斜负载下, 使用 RDMA 原子操作进行远程加锁开销较大会导致急剧的性能下降, 其中包括网络延迟本身、网卡和内存之间的 PCIe 事务, 以及大量的加锁失败重试等. Sherman 设计一种层次化的锁机制 HOCL, 显著降低了远程加锁的开销. HOCL 在分离式内存端的 RDMA 网卡内存上维护一层远程锁, 避免了网卡到内存之间的 PCIe 事务. 其次, HOCL 在客户端本地内存维护一层本地锁, 客户端需要加锁时, 首先尝试加锁本地锁, 如若加锁失败, 则不必要进一步尝试加远程锁, 减少了发送到远端网卡的请求. 客户端在释放锁时, 还会检测本地是否有其他线程请求相同的锁, 并尝试在一定公平性下将锁直接移交给其他线程, 减少了远程释放锁和其他线程加锁的操作.

最后, B^+ 树的一次写操作通常要将一个树节点重新排序并写回, 即使修改的部分可能在树节点中占据较小, 现有的一致性保证方法如校验和或版本号机制, 均需要更新整个树节点的校验和或版本号, 并将整个树节点写回. 这种较大粒度的写操作导致较严重的写放大, 降低了 RDMA 吞吐. Sherman 采

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

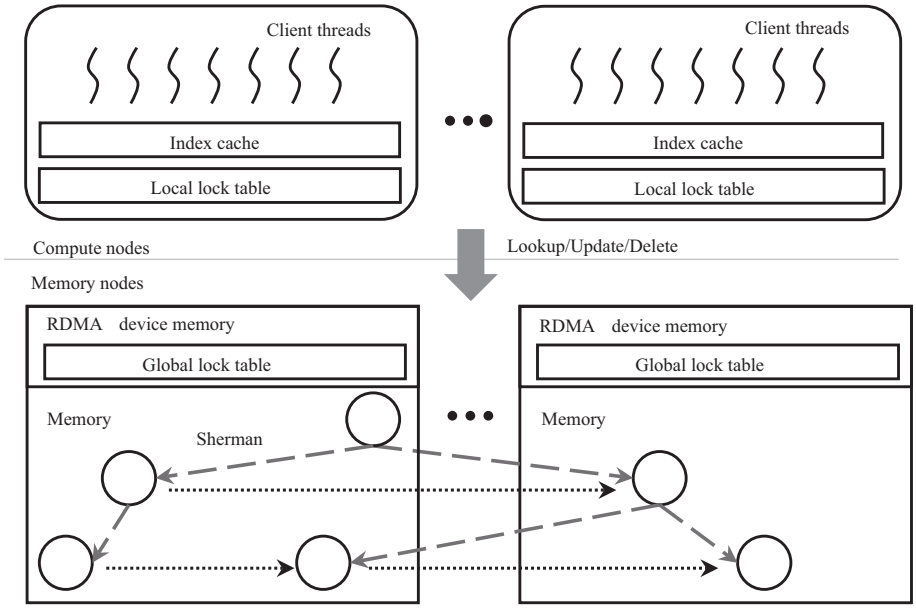


图 3 Sherman 的总体架构 [73]

Figure 3 The overall architecture of Sherman [73]

用了无序树节点的设计, 并提出了一种两级版本号的设计, 每个树节点在头部和尾部维护一组版本号, 树节点中每个键值项也在头部和尾部维护一组版本号. 树节点的版本号仅在树节点分裂、合并时更新, 键值项在插入或修改时更新. Sherman 在插入或修改操作中, 仅需要写回新增或修改的键值项, 降低了写放大.

4.2.2 分布式事务协议

在分离式内存存储系统中, 计算节点需要通过分布式事务协议保证一组操作的原子性和隔离性. 相比于传统系统, 分离式内存更侧重于如何利用 RDMA 单边原语实现高效的分布式事务协议式. 这里主要介绍 FaRM [35] 和 FORD [74].

FaRM [35] 是微软研究院提出的一个提供事务语义的分布式内存计算平台. FaRM 使用单边 RDMA 原语完成事务的执行与提交, 远端内存节点的 CPU 惰性地执行日志回收. FaRM 采用了乐观并发控制策略, 将事务分为执行和提交阶段. 在执行阶段, 协调者 (事务的发起者) 使用 `READ` 原语读取当前事务涉及的数据项, 并在本地执行事务. 在加锁阶段, FaRM 首先对写集数据加锁, 同时将数据以日志记录形式写入主副本节点, 再读取读集数据的版本号进行验证, 期间加锁失败或若读集版本号发生改变, 则事务终止. 若验证通过, FaRM 将修改数据以日志记录形式写至备份副本节点并提交备份副本. 待备份副本全部提交完成, 再向主副本节点写入提交信息, 此时标记事务完成. 随后主副本通过日志记录原地更新数据, 并释放锁. 内存节点在后台异步消化日志记录, 备份副本同时更新备份数据.

FORD [74] 是一个面向分离式内存的分布式事务系统. FORD 提出了若干优化, 有效减少了事务执行的网络往返. 图 4 [74] 描述了 FORD 的事务执行流程, 协调者首先使用合并的 `CAS+READ` 操作, 通过一次网络往返完成加锁和读数据, 并且并行地使用 `WRITE` 将回滚日志写到副本节点. 随后, 协调者通过 `READ` 验证读集的版本号. 当验证通过并收到所有回滚日志的 ACK 后, 协调者使用合并的 `CAS+WRITE` 操作, 将数据置为不可见状态并更新数据, 此时事务标记为提交状态. 最后, 协调者使用 `CAS` 将数据置

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

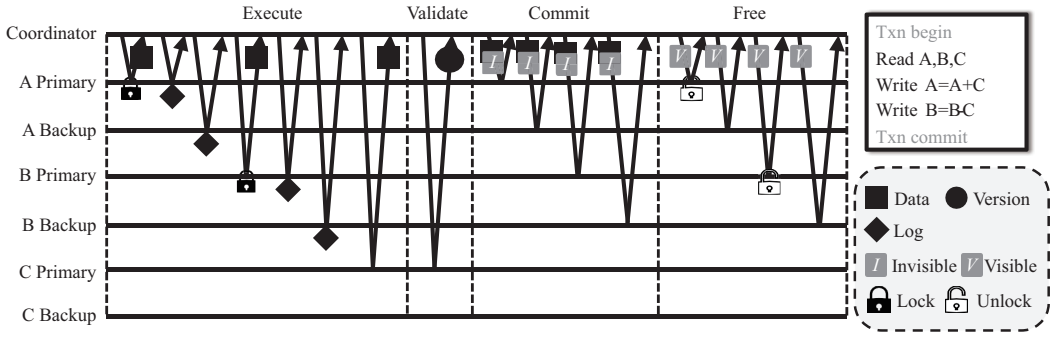


图 4 FORD 的事务执行流程 [74]

Figure 4 Transaction processing in FORD [74]

为可见状态并释放锁.

与 FaRM 相比, FORD 减少了事务执行的网络往返次数. 但 FORD 预先向所有副本写入回滚日志, 在事务失败时, 这些回滚日志写操作浪费了较多网络资源. FaRM 的内存节点需要计算资源消化日志, 在系统吞吐较高时, 内存节点的计算资源将成为系统瓶颈.

4.2.3 数据分区策略

分离式内存存储系统包含多台计算节点和内存节点, 因此存在数据分区的问题, 即 (1) 数据被如何划分到不同的内存节点, (2) 某个计算节点能访问哪些数据. 这里通过几个典型的分离式内存存储系统对数据分区策略进行阐述.

Clover [18] 是普渡大学 (Purdue University) 和加利福尼亚大学 (University of California) 提出的一个基于分离式持久性内存的键值系统. Clover 将数据平面与控制平面分离. 其中位于计算节点的客户端直接与内存节点进行数据平面的交互, 完成键值数据的读写功能. Clover 使用额外的元数据服务器进行控制平面的交互, 元数据服务器负责记录键值数据的辅助元数据、管理分离式内存的分配和垃圾回收等功能. Clover 的分区策略较为简单: 内存节点中数据未做任何分区, 同时所有的计算节点均可访问所有数据.

DINOMO [75] 是德克萨斯大学奥斯汀分校 (University of Texas at Austin) 提出的一个同时实现高性能、高扩展性和轻量在线重配置的分离式持久性内存键值存储系统. DINOMO 采取了键值空间逻辑分区的策略, 客户端负责指定分区的键值存储访问操作. DINOMO 并不对键值数据进行分区存储, 而是共享地存储在一起. 这种分区策略允许客户端高效使用本地数据缓存, 同时, 在增加或删除客户端时, 不影响数据的存储, 不需要对数据进行重新切分. DINOMO 客户端执行写操作时通过 RDMA 单边原语将数据以日志形式写到远程内存, 并在本地缓存数据. 远程内存节点负责将不同客户端写入的日志异步地合并更新到索引中. 为了计算节点缓存失效后也能正确找到数据, 内存节点需要及时吸收日志并更新索引. 因此, DINOMO 内存节点的计算资源在系统吞吐较高时将成为系统瓶颈.

AsymNVM [76] 是清华大学提出的分离式内存架构, 其支持多种远程数据结构的访问. 在 AsymNVM 中, 数据根据键空间被分区, 即每个分区有单独的数据结构. 对于写操作, 某个分区只能被单个客户端访问, 由此高效支持客户端缓存, 并避免一致性管理开销; 对于读操作, 不同客户端能访问同一个分区, 由此提升了整体系统读操作的负载均衡和扩展性. 表 5 [18,75,76] 对基于分离式存储系统的分区策略进行了总结对比.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

表 5 分离式存储系统的分区策略总结

Table 5 Summary of partitioning in disaggregated in-memory key-value store

Representative systems	Clover [18]	DINOMO [75]	AsymNVM [76] writers	AsymNVM [76] readers
Data storage strategy	Shared	Shared	Partition	Partition
Client access strategy	Shared	Partition	Partition	Shared
Client cache friendliness	Low	High	High	Low
Load balancing	High	Medium	Low	Medium
Memory scale-out overhead	Low	Low	High	High
Computation scale-out overhead	Low	Low	High	Low

表 6 3 种 I/O 栈的对比

Table 6 Comparison among three types of I/O stacks

	ReFlex [77]	i10 [78]	ccNVMe [79]
Userspace/kernel	Userspace & kernel	Kernel	Kernel
Network	TCP/IP	TCP/IP	RDMA
Transaction	Not support	Not support	Support

4.3 分离式外存存储技术

分离式外存存储系统的设计初衷是将存储资源池化以提高资源利用率, 并赋予计算和存储资源独立扩展的能力. 早期具有代表性的分离式外存存储系统包括存储区域网络 (storage area network, SAN) 和网络附接存储 (network attached storage, NAS) 等. 而在现代分离式数据中心架构下, 随着高性能网络 (例如 RDMA) 和存储技术 (例如 NVMe) 的出现和持续发展, 近年来的工作主要通过降低软件开销以充分发挥硬件性能, 具体包括优化 I/O 栈、设计高性能对象和文件系统等.

4.3.1 I/O 栈优化

与本地外存存储系统不同, 分离式外存存储系统通过高速网络访问 SSD 等存储池介质, 因此需要对网络和 IO 栈进行协同优化, 以避免复杂交互导致网络或存储的带宽无法被充分利用. 这里主要介绍 3 个典型的研究工作: ReFlex [77], i10 [78], 以及 ccNVMe [79], 表 6 进行了总结和对比.

斯坦福大学 (Stanford University) 的研究人员提出 ReFlex [77] 将网络和存储栈深度融合. 如图 5 [77] 所示, ReFlex 使网络接收逻辑 (图 5 中 ① 和 ②) 和本地 NVMe 发送逻辑 (图 5 中 ③ 和 ④) 在不被中断的情况下连续执行, 这避免了频繁的线程上下文切换, 降低了响应延迟. 本地 NVMe 完成逻辑 (图 5 中 ⑤ 和 ⑥) 和数据传输 (图 5 中 ⑦ 和 ⑧) 也以类似的方式绑定在一起连续执行. Reflex 同时采用零拷贝、批量异步处理等技术降低软件开销, 提升了系统吞吐率. 为保证多租户的延迟需求, ReFlex 在 I/O 调度器中设计了请求开销模型. 在该模型中, ReFlex 首先需要预先获得存储盘的延迟随着已使用带宽增加的变化曲线, 其次以对延迟要求最高的租户为基准限定系统可用的最大带宽. 多租户的聚合带宽必须低于该最大带宽, 否则减少该存储盘上的租户数量以满足所有租户的延迟需求.

康奈尔大学 (Cornell University) 的研究人员提出 i10 [78] 以优化现有 Linux 内核中基于 TCP 的 NVMe 存储栈的性能. i10 的核心设计是将 I/O 请求以细粒度方式进行分组并进行组内请求合并. 具体来说, 对于某个应用, i10 首先枚举应用运行的所有 CPU 核和远端存储盘, 并列举所有 “CPU 核 – 存储盘” 的二元组, i10 为各二元组创建独有的调度队列. i10 根据请求的发起 CPU 核和终点存储盘

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

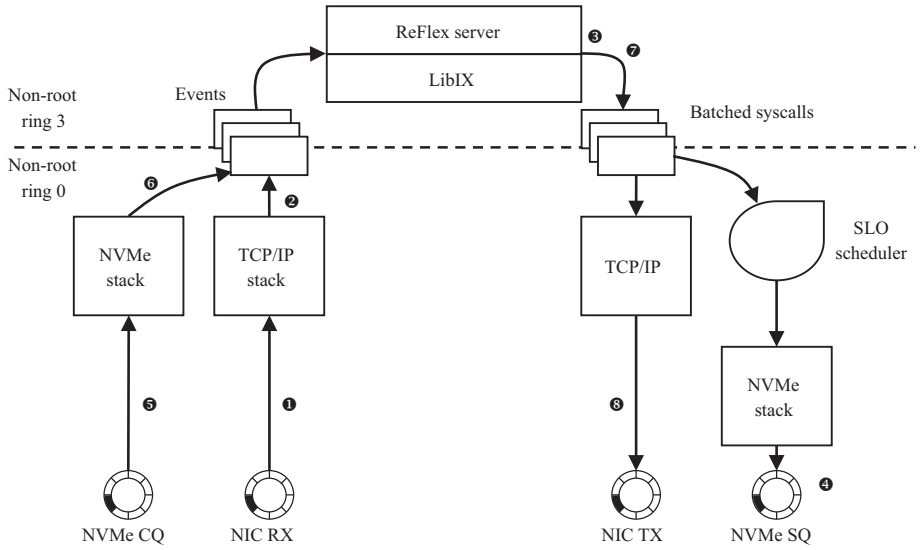


图 5　ReFlex 处理流程 [77]

Figure 5　Workflow of ReFlex [77]

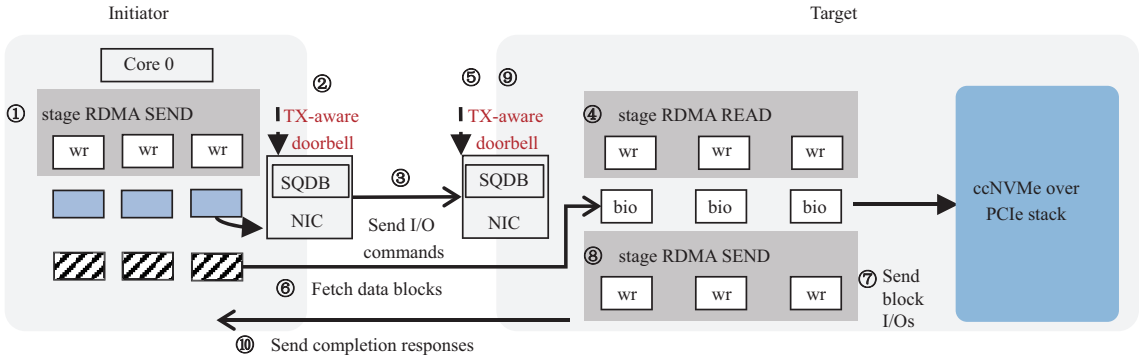


图 6　(网络版彩图) ccNVMe over RDMA 工作流程 [79]

Figure 6　(Color online) Workflow of ccNVMe over RDMA [79]

位置分发请求到特定的调度队列, 以降低请求在同一队列中互相干扰的概率, 从而达到降低延迟的目的. 此外, 对于各调度队列, i10 使用了批量处理技术, 只有当请求数量足够多或者超过设定的时间阈值时, 才将请求统一从存储栈发至网络栈, 这也降低了请求在网络栈和存储栈之间上下文切换开销.

清华大学的研究人员提出 ccNVMe [79], 将事务语义融合至网络存储协议, 降低了存储协议的开销. ccNVMe 的核心设计是尽可能以事务 (一组需要同时完成的请求) 为单位交换各类数据. 如图 6 [79] 所示, ccNVMe 沿用了 NVMe-oF 的设计, 包含发起端和接收端两个部分. 不同于以请求为基本操作单位的 NVMe-oF, ccNVMe 以事务为基本操作单位. 对于发起的 I/O 请求, ccNVMe 的发起者尽可能将属于同一事务 `SEND` 操作聚集在主机内存, 并通过一次门铃操作通知网卡处理请求, 这避免了主机和网卡间频繁的通信. 与发起端的操作类似, 接收端解析完 `SEND` 操作中附带的 NVMe I/O 命令后, 积攒多个 `READ` 操作, 并通过一次门铃操作通知网卡从发起端拉取数据. 最后, 当请求被具有崩溃一致性保证的本地 ccNVMe 存储栈处理完成后, 系统积攒同一事务的 `SEND` 操作并一次性将相关的请求完成信号返回给发起端.

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

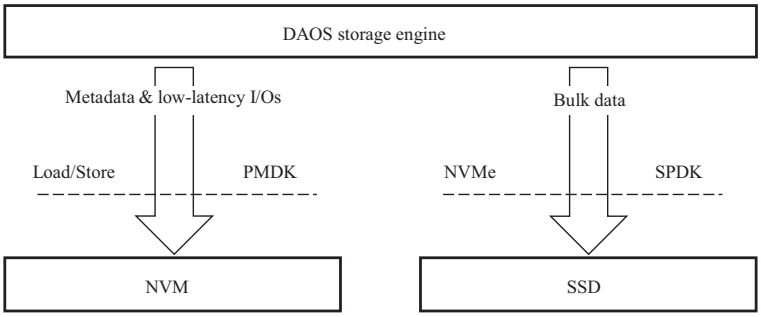


图 7 DAOS 系统架构 [81]

Figure 7 Architecture of DAOS [81]

4.3.2 分离式对象管理

分离式对象存储系统提供对象接口, 可作为上层文件系统和数据库系统等的基石. 分离式对象管理技术旨在以轻量而高效的方式管理存储池中的对象数据, 这里介绍典型的分离式对象存储系统 LighStore [80], RADOS [81], 以及 DAOS [82].

麻省理工学院 (Massachusetts Institute of Technology) 的研究人员提出 LightStore [80], 将存储功能精炼并实现在只有 2.5 英寸大小的存储卡中, 其中包含网络直连的键值接口层、对象处理模块、闪存转换层和闪存芯片. 不同于传统架构使用的 “主机 – 存储” 协议 (例如 NFS 和 SMB-CIFS), LightStore 精简了存储协议, 提供键值存储接口. 应用所在的节点只需实现额外的适配层即可直接接入 LightStore. LightStore 将计算密集的存储协议分离, 而将其余核心的存储功能固化在存储节点中, 提升了系统整体的能效比.

RADOS [81] 是开源软件 Ceph 的分布式对象存储引擎, 负责多节点的存储资源池化并实现资源池内的负载均衡、数据副本、恢复等功能. RADOS 以固定大小的对象 (例如 2 MB) 为基本操作单元, 并通过哈希算法将对象组织成多个放置组, 最后通过 CRUSH [83] 算法将放置组内对象分发到存储节点上的本地存储引擎 (例如 BlueStore [84]). RADOS 将计算和存储分离, 使得两者故障域独立. 利用这一特点, 普林斯顿大学的研究人员基于 RADOS 加速了应用层的恢复重启时间 [85].

Intel 主导的 DAOS [82] 项目是专门针对持久性内存和 NVMe 存储构建的高性能存储系统. 如图 7 [81] 所示, DAOS 存储引擎将元数据、低延迟 I/O 和索引操作直接通过内存接口写入持久性内存中, 并通过 SPDK 将大块数据批量写回闪存存储中. DAOS 设计了多层抽象: 例如, DAOS 引入池抽象将任意数量的底层设备资源 (如 NVMe SSD 和持久性内存) 打包, 在一个池内, DAOS 允许用户创建多个独立的容器 (即命名空间) 以共享使用池内的硬件资源.

4.3.3 分离式文件管理

分离式文件系统的核心包括两部分: 文件数据的管理和文件元数据的管理. 前者主要关注数据布局、数据一致性等方面, 后者主要关注元数据扩展性. 这里介绍典型的分离式对象存储系统 HailStorm [86], PolarFS [87], 以及 RIOFS [88].

洛桑联邦理工学院 (École Polytechnique Fédérale de Lausanne) 的研究人员提出的 HailStorm [86] 将分布式文件系统的计算和存储分离, 使得计算和存储资源能独立扩展且不互相干扰. HailStorm 包括客户端、代理和文件服务 3 部分, 并对上层数据库系统等应用提供 POSIX 文件接口. 客户端位于计算节点, 负责系统的计算任务 (如 compaction 操作). 代理位于计算节点, 负责系统计算任务的调度

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

和资源分配. 存储服务端位于存储节点, 负责存储池化、数据布局和热点迁移. HailStorm 的分离设计缓解了存储系统的 CPU 竞争问题, 这一设计在存储设备对 CPU 的需求变高时尤为重要.

阿里巴巴公司提出的 PolarFS [87] 将数据存储池化由单独的数据服务器实现. 数据被切分成多个块, 并通过 PolarSwitch 分发到各块服务器中. 各块服务器之间通过优化的并行 Raft 共识协议保证分布式一致性. 此外, PolarFS 将集群的元数据等控制信息单独抽离并由 PolarCtrl 进行集中管理.

清华大学研究人员提出的 RIOFS [88] 用于加速基于分离式存储的文件系统性能. RIOFS 提取文件系统中的数据存储顺序 (例如日志数据块和提交块之间的持久化顺序), 携带在各请求的 I/O 命令中, 并临时纪录在 NVMe 的持久性内存区域 (persistent memory region, PMR) 中. RIOFS 凭借记录的顺序信息让文件系统的日志请求乱序执行, 并能在系统崩溃等乱序情况下将文件系统恢复到正确的状态. 由于日志请求能被乱序执行, RIOFS 还引入了请求合并等技术进一步降低 CPU 开销.

4.4 分离式存储系统的卸载加速技术

在分离式数据中心架构中, 交换机和网卡负责资源池内部以及资源池之间通信任务. 可编程交换机和智能网卡等可编程硬件提供了卸载用户自定义逻辑的能力, 这为减少存储软件开销、提升分离式存储系统性能带来了新的机遇. 本小节将介绍相关工作.

4.4.1 基于可编程交换机的硬件卸载

分离式存储系统利用可编程交换机上的内存存储系统元数据信息, 凭借其线速处理能力和中心化的位置来卸载机柜内的资源管理和冲突协调任务, 以减少软件开销且提高系统并发. 这里将介绍交换机卸载缓存一致性、地址空间管理, 以及冲突协调的相关研究工作.

Concordia [11] 由清华大学提出, 利用可编程交换机加速缓存一致性协议. 缓存是分离式存储系统中减少远程访问数据提高系统性能的关键组件, 为保证不同计算节点间缓存的一致性, 需要复杂的缓存一致性协议. 常见的缓存一致性协议包括基于目录的缓存一致性协议和基于广播的缓存一致性协议, 当缓存数据存在读写冲突时, 缓存一致性协议会消耗额外的网络资源和 CPU 资源, 成为系统性能瓶颈. Concordia 利用可编程交换机设计了高效的在网分布式缓存一致性协议, 利用可编程交换机存储缓存元数据信息, 并利用交换机中心化位置减少缓存一致性协议中的网络通信次数.

耶鲁大学 (Yale University) 提出的 MIND [12] 系统构建了共享的全局虚拟地址空间, 将地址翻译和权限管理功能卸载到可编程交换机, 同时利用可编程交换机维护计算节点之间的缓存一致性. 在分离式内存中, 粗粒度的内存块管理会导致计算节点间产生大量错误冲突问题, 而细粒度的内存块管理会增大内存管理所需的存储空间, 导致无法将所有地址翻译表项存储在可编程交换机有限的内存中, 为此 MIND 系统设计了有界分割算法来动态调整内存块的管理粒度.

MIND 系统主要由 3 部分组成: 计算节点、内存节点和可编程交换机. 具体地, 如图 8 [12] 所示, 计算节点运行用户计算任务, 同时用少量本地内存作为远端内存的缓存; 内存节点仅用于向计算节点提供内存资源, 所有的内存访问操作通过单边 RDMA 操作完成, 内存节点不需要消耗 CPU 资源; 可编程交换机的控制平面维护系统的全局视图, 完成内存分配、权限管理等操作, 可编程交换机的控制平面负责完成地址翻译操作以及加速缓存一致性协议.

MIND 系统在可编程交换机中实现了地址翻译操作和缓存一致性协议. 具体地, 来自用户进程的所有内存读取/写入操作首先查询计算节点的本地内存缓存, 如果被访问页面没有本地内存缓存, 计算节点会触发缺页中断, 并使用 RDMA 请求从内存资源池中读取页面, 同时驱逐冷数据缓存. MIND 在可编程交换机中同时维护计算节点与交换机之间的 RDMA 连接以及内存节点与交换机之间的 RDMA

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

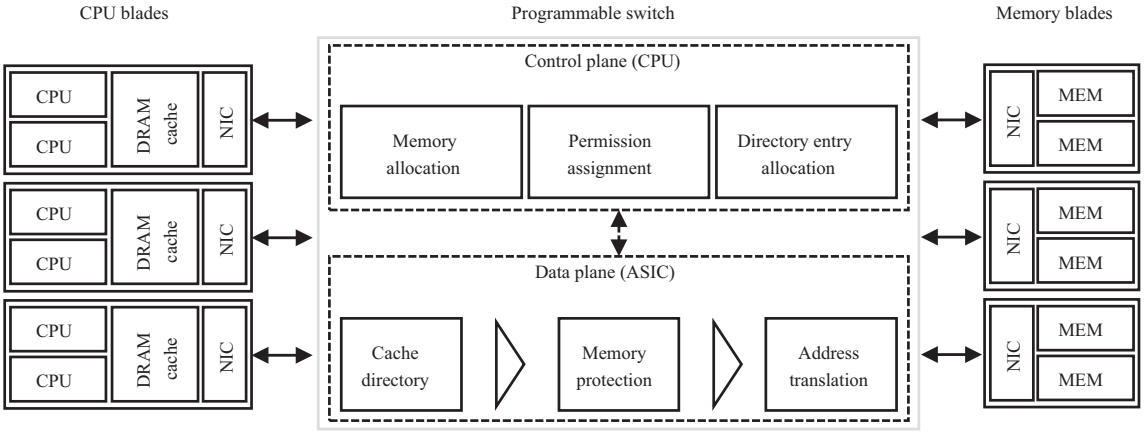


图 8 MIND 系统架构 [12]

Figure 8 Architecture of MIND [12]

连接. 可编程交换机将来自计算节点的 RDMA 读写请求中的全局统一地址映射到对应内存节点的虚拟地址, 修改网络包中地址等字段后将 RDMA 请求转发到对应内存节点, 如果该内存访问请求需要更新其他计算节点的缓存块, 则触发缓存一致性逻辑, 交换机发送网络包将其他计算节点的缓存块标记为无效.

MIND 系统通过有界分割算法动态确定每个缓存块的大小. 有界分割算法首先将整个虚拟地址空间划分为多个连续缓存块, 每个缓存块包含多个页面, 且大小固定. MIND 系统统计每个缓存块在一段时间内产生伪冲突的次数; 当伪冲突的次数超过系统设定阈值时, MIND 系统将一个缓存块分成两个相等的缓存块, 并创建一个新的缓存目录条目. MIND 系统设置缓存块的最小大小为一个页面的大小 (4 KB), 因此该算法被称为有界分割算法. MIND 系统利用有界分割算法在处理伪冲突开销和维护缓存目录项开销的权衡中取得动态平衡.

加州大学圣地亚哥分校 (University of California, San Diego) 提出的 Warm [89] 系统利用可编程交换机序列化写操作, 以加速分离式内存中的并发冲突处理. 典型的分离式内存存储系统 Clover 中, 写操作需要通过 `CAS` 操作修改数据指针避免写冲突, 针对 Clover 系统, Warm 系统通过在交换机中缓存指针将 `CAS` 操作变成 `WRITE` 操作, 从而序列化并发写请求, 同时缓存在交换机中的指针还可以服务读操作, 以减少读操作的重试次数.

4.4.2 基于智能网卡的硬件卸载

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侧的智能网卡扩展 RDMA 语义或存储协议.

苏黎世联邦理工学院 (Swiss Federal Institute of Technology in Zurich) 提出的 StRoM [90] 系统将数据处理操作卸载到内存节点的智能网卡上, 通过近数据处理来减少数据移动开销. StRoM 扩展了 RDMA 的语义, 能够实现多步数据访问操作和网络处理操作. StRoM 系统基于 FPGA 实现, 兼容现有的 RoCE 协议, 允许用户在网卡中实现自定义的数据处理逻辑. 图 9 [90] 描述了 StRoM 系统中网络包的处理流程, 来自计算节点的网络包首先经过 RoCE 网络栈处理, 对于普通 RDMA 请求, RoCE 网络栈直接通过 DMA 引擎完成数据的读写操作; 对于其他请求, 网络包的操作码字段会包含对应逻辑的

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

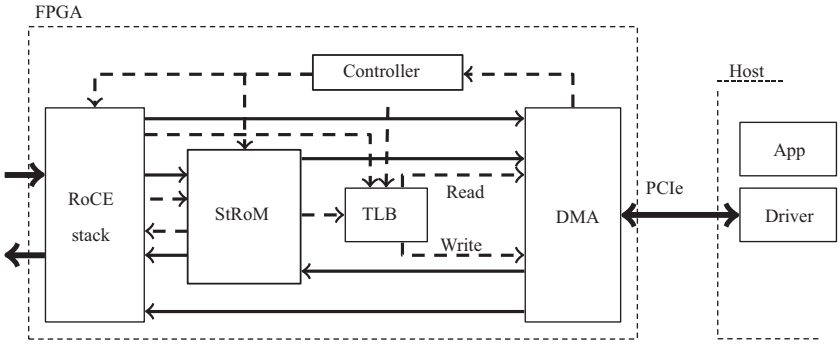


图 9 StRoM 处理流程 [90]

Figure 9 Workflow of StRoM [90]

标志符, 网卡将执行用户定义的复杂的计算逻辑以及 DMA 操作.

StRoM 系统实现了 4 种数据处理逻辑. 第 1 种操作是通过指针追逐对内存池中数据结构进行遍历. 遍历数据结构是一种内存密集型但计算量低的操作. 在单机系统中, 数据结构遍历通常受到内存延迟的限制, 在分离式内存中访问远程数据结构时, 由于内存读写需要网络往返, 遍历延迟显著增加. 通过将指针追逐卸载到内存节点网卡可以将多次网络往返变为一次网络往返, 远程链表遍历以及链式哈希表等数据结构都可从该操作中受益. 第 2 种操作是远程数据块一致性读取. 当数据读取超过缓存块大小时, 计算节点需要通过乐观读方法保证读取数据的一致性, 具体地, 计算节点单边读操作访问数据, 如果内存节点数据同时被修改, 计算节点可以通过校验每个缓存块上的版本发现数据不一致, 进而进行重试, 重试过程需要一次额外网络往返. StRoM 系统将版本验证和重试卸载到智能网卡, 远程数据块一致性读取即可通过一次网络往返完成. 第 3 种操作是网卡支持将计算节点发送的数据分发到不同的内存区域或处理器核心. 基于原生 RDMA 操作的数据重新布局操作, 需要计算节点本地完成数据布局后通过多次 RDMA 写操作将数据分散到内存池的多个位置, StRoM 通过内存节点网卡硬件卸载这一操作能够节省计算节点 CPU 开销. 第 4 种操作是数据流的基数估计操作. 基数估计操作可以用于统计数据集中不同数据项的数量, 将基数估计操作卸载至智能网卡避免了内存节点 CPU 的开销, 相较于用计算节点实现的方法, 内存节点智能网卡卸载能够避免计算节点之间的同步开销.

苏黎世联邦理工学院提出的 Farview [91] 系统在 StRoM 系统的基础上在内存节点的网卡上实现了对读数据流的数据处理操作, 并利用此操作向计算节点提供复杂的 SQL 接口. Farview 系统在智能网卡上设计了操作流水线, 能够支持选择、投影、分组、去重、正则匹配等数据查询操作以及加/解密和序列化/反序列化等数据操作, Farview 系统中的计算节点通过将上述操作组合来实现单边的复杂 SQL 查询操作.

华盛顿大学 (University of Washington) 提出的 Gimbal [92] 系统针对分离式存储场景, 为 NVMe-oF 增加了多租户支持, 将单个 SSD 硬件抽象为多个虚拟 SSD 提供给计算节点的用户, 并向用户提供公平的服务质量保证. Gimbal 系统实现在支持闪存捆绑 (just-bunch-of-flash, JBOF) 的基于 ARM 的智能网卡上. 为评估和控制不同 SSD 的硬件性能, Gimbal 在网卡上实现读写操作的动态开销估计模块和速率控制模块, 基于这两个子模块 Gimbal 系统实现了基于延迟的 SSD 拥塞控制算法; 为保证多个客户端的公平性, Gimbal 系统实现了一个基于轮询算法的公平调度器. 此外, 阿里巴巴公司提出的 Solar [93] 系统针对分离式存储场景, 将计算节点的存储操作的控制路径卸载到计算节点网卡, 在不降低存储系统性能的同时减少计算节点 CPU 的开销, 同时避免存储操作对用户任务的干扰, 比如缓存污染, 进而

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

使得云厂商可以将计算节点作为裸金属设备向用户出售.

5 总结与展望

本文首先从传统数据中心架构在新业务场景下面临的挑战和硬件技术趋势出发, 阐述了分离式数据中心架构的形态及其优势. 基于此, 本文对分离式数据中心的存储系统相关研究工作进行了详细地综述; 这些研究工作从一定程度上证明了: 通过精心的软件设计, 并辅助于高速的现代硬件, 分离式数据中心能够达到接近甚至超越传统数据中心的性能, 并且还拥有其他无可比拟的优势, 包括极致的硬件资源利用率、弹性扩缩容等. 展望未来, 作者认为分离式数据中心架构具有以下研究趋势.

(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 [94]. 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. 此外, 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. 最后, 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. 此外, 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. 最后, 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: 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: 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.

参考文献

1 Shan Y Z, Huang Y T, Chen Y L, et al. LegoOS: a disseminated, distributed os for hardware resource disaggregation. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2018. 69–87

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

2 Kanev S, Darago J P, Hazelwood K, et al. Profiling a warehouse-scale computer. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. New York: ACM, 2015. 158–169
3 HB Labs. The machine: a new kind of computer. 2022. https://www.hpl.hp.com/research/systems-research/themachine
4 Barroso L, Marty M, Patterson D, et al. Attack of the killer microseconds. Commun ACM, 2017, 60: 48–54
5 舒继武. 新型存算分离架构技术展望. 中国计算机学会通讯, 2023, 18: 1–8
6 Nvidia. ConnectX-7. 2022. https://www.nvidia.com/en-au/networking/ethernet-adapters/
7 Li H C, Berger D S, Novakovic S, et al. Pond: CXL-based memory pooling systems for cloud platforms. 2022. ArXiv:2203.00241
8 Intel. Intel Optane DC SSD series. 2022. https://www.intel.com/content/www/us/en/products/details/memory-storage/data-center-ssds/optane-dc-ssd-series.html
9 Intel. Intel Optane persistent memory. 2022. https://www.intel.com.au/content/www/au/en/products/docs/memory-storage/optane-persistent-memory/overview.html
10 Intel. Intel Tofino intelligent fabric processors. 2022. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-3-product-brochure.html
11 Wang Q, Lu Y Y, Xu E C, et al. Concordia: distributed shared memory with in-network cache coherence. In: Proceedings of the 19th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2021. 277–292
12 Lee S, Yu Y P, Tang Y P, et al. MIND: in-network memory management for disaggregated data centers. In: Proceedings of the 28th Symposium on Operating Systems Principles. New York: ACM, 2021. 488–504
13 Li J R, Lu Y Y, Zhang Y M, et al. SwitchTx: scalable in-network coordination for distributed transaction processing. In: Proceedings of the VLDB Endowment. New York: VLDB, 2022. 2881–2894
14 Nvidia. NVIDIA BlueField data processing units. 2022. https://www.nvidia.com/en-us/networking/products/data-processing-unit/
15 Kim J, Jang I, Reda W, et al. LineFS: efficient SmartNIC offload of a distributed file system with pipeline parallelism. In: Proceedings of the 28th Symposium on Operating Systems Principles. New York: ACM, 2021. 756–771
16 Nvidia. Innova-2 Flex. 2022. https://www.nvidia.com/en-au/networking/ethernet-adapters/
17 Li J R, Lu Y Y, Wang Q, et al. AlNiCo: SmartNIC-accelerated contention-aware request scheduling for transaction processing. In: Proceedings of the 27th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2022. 951–966
18 Tsai S Y, Shan Y Z, Zhang Y Y. Disaggregating persistent memory and controlling them remotely: an exploration of passive disaggregated key-value stores. In: Proceedings of the 26th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2020. 33–48
19 Shan Y Z, Tsai S Y, Zhang Y Y. Distributed shared persistent memory. In: Proceedings of the 8th Symposium on Cloud Computing. New York: ACM, 2017. 323–337
20 Aguilera M K, Amit N, Calciu I, et al. Remote regions: a simple abstraction for remote memory. In: Proceedings of the 24th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2018. 775–787
21 Gu J C, Lee Y, Zhang Y W, et al. Efficient memory disaggregation with INFINISWAP. In: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2017. 649–667
22 Amaro E, Branner-Augmon C, Luo Z, et al. Can far memory improve job throughput? In: Proceedings of the 15th European Conference on Computer Systems. New York: ACM, 2020
23 Al Maruf H, Chowdhury M. Effectively prefetching remote memory with leap. In: Proceedings of the 26th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2020. 843–857
24 Weiner J, Agarwal N, Schatzberg D, et al. TMO: transparent memory offloading in datacenters. In: Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2022. 609–621
25 Wang C X, Qiao Y F, Ma H R, et al. Canvas: isolated and adaptive swapping for multi-applications on remote memory. In: Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2023. 234–246

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

26 Kim J, Choe W, Ahn J. Exploring the design space of page management for multi-tiered memory systems. In: Proceedings of the 27th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2021. 715–728

27 Yan Z, Lustig D, Nellans D, et al. Nimble page management for tiered memory systems. In: Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019. 331–345

28 Bergman S, Faldu P, Grot B, et al. Reconsidering OS memory optimizations in the presence of disaggregated memory. In: Proceedings of the 23rd International Symposium on Memory Management. New York: ACM, 2022. 1–14

29 Raybuck A, Stamler T, Zhang W, et al. HeMem: scalable tiered memory management for big data applications and real NVM. In: Proceedings of the 28th Symposium on Operating Systems Principles. New York: ACM, 2021. 392–407

30 Ruan Z Y, Schwarzkopf M, Aguilera M K, et al. AIFM: high-performance, application-integrated far memory. In: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2020. 315–332

31 Zhou Y, Wassel H M G, Liu S H, et al. Carbink: fault-tolerant far memory. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2022. 55–71

32 Lu Y Y, Shu J W, Chen Y M, et al. Octopus: an RDMA-enabled distributed persistent memory file system. In: Proceedings of the 23rd USENIX Annual Technical Conference. Berkeley: USENIX Association, 2017. 773–785

33 Zhu B, Chen Y, Wang Q, et al. Octopus^{+}: an RDMA-enabled distributed persistent memory file system. ACM Trans Storage, 2021, 17: 1–25

34 Shu J, Chen Y, Wang Q, et al. TH-DPMS: design and implementation of an RDMA-enabled distributed persistent memory storage system. ACM Trans Storage, 2020, 16: 1–31

35 Dragojević A, Narayanan D, Nightingale E B, et al. No compromises: distributed transactions with consistency, availability, and performance. In: Proceedings of the 25th Symposium on Operating Systems Principles. New York: ACM, 2015. 54–70

36 Novakovic S, Daglis A, Bugnion E, et al. Scale-out NUMA. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2014. 3–18

37 Calciu I, Imran M T, Puddu I, et al. Rethinking software runtimes for disaggregated memory. In: Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021. 790–92

38 Wang C X, Ma H R, Liu S, et al. Semeru: a memory-disaggregated managed runtime. In: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2020. 261–280

39 Ma H R, Liu S, Wang C X, et al. Mako: a low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In: Proceedings of the 43rd International Conference on Programming Language Design and Implementation. New York: ACM, 2022. 92–107

40 Wang C X, Ma H R, Liu S, et al. MemLiner: lining up tracing and application for a far-memory-friendly runtime. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2022. 35–53

41 Shu J W, Chen Y M, Hu Q D, et al. Development of system software on non-volatile main memory. Sci Sin Inform, 2021, 51: 869–899 [舒继武, 陈游旻, 胡庆达, 等. 非易失主存的系统软件研究进展. 中国科学: 信息科学, 2021, 51: 869–899]

42 Wang Q, Lu Y Y, Li J R, et al. Nap: a black-box approach to NUMA-aware persistent memory. In: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2021. 93–111

43 Wang Q, Lu Y Y, Wang J, et al. Replicating persistent memory key-value stores with efficient RDMA abstraction. In: Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2023. 1–15

44 Chen Y M, Lu Y Y, Yang F, et al. FlatStore: an efficient log-structured key-value storage engine for persistent memory. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020. 1077–1091

45 Wang J, Lu Y Y, Wang Q, et al. Pacman: an efficient compaction approach for log-structured key-value store on

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

persistent memory. In: Proceedings of the 28th USENIX Annual Technical Conference, 2022. 773–788
46 Ou J X, Shu J W, Lu Y Y. A high performance file system for non-volatile main memory. In: Proceedings of the 11th European Conference on Computer Systems. New York: ACM, 2016. 1–16
47 Chen Y, Shu J, Ou J, et al. HiNFS: a persistent memory file system with both buffering and direct-access. ACM Trans Storage, 2018, 14: 1–30
48 Lu Y, Shu J, Sun L. Blurred persistence in transactional persistent memory. In: Proceedings of the 31st International Conference on Massive Storage Systems and Technology. Piscataway: IEEE, 2015. 1–13
49 Volos H, Tack A J, Swift M M. Mnemosyne: lightweight persistent memory. In: Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011. 91–104
50 Hu Q, Ren J, Badam A, et al. Log-structured non-volatile main memory. In: Proceedings of the 23rd USENIX Annual Technical Conference. Berkeley: USENIX Association, 2017. 2–44
51 Bhandari K, Chakrabarti D R, Boehm H J. Makalu: fast recoverable allocation of non-volatile memory. In: Proceedings of the ACM Sigplan International Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM, 2016. 677–694
52 Coburn J, Caulfield A M, Akel A, et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In: Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011. 105–118
53 Kannan S, Gavrilovska A, Schwan K. pVM: persistent virtual memory for efficient capacity scaling and object storage. In: Proceedings of the 11th European Conference on Computer Systems. New York: ACM, 2016. 1–16
54 Oukid I, Lasperas J, Nica A, et al. FPTree: a hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory. In: Proceedings of the International Conference on Management of Data. New York: ACM, 2016. 371–386
55 Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory. In: Proceedings of the 22nd Symposium on Operating Systems Principles. New York: ACM, 2009. 133–146
56 Lu Y, Shu J, Sun L, et al. Loose-ordering consistency for persistent memory. In: Proceedings of the 32nd International Conference on Computer Design. Piscataway: IEEE, 2014. 216–223
57 Kolli A, Rosen J, Diestelhorst S, et al. Delegated persist ordering. In: Proceedings of the 49th IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE, 2016. 1–13
58 Narayanan D, Hodson O. Whole-system persistence. In: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2012. 401–410
59 Zhao J, Li S, Yoon D H, et al. Kiln: closing the performance gap between systems with and without persistence support. In: Proceedings of the 50th IEEE/ACM IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE, 2017. 421–432
60 Memaripour A, Badam A, Phanishayee A, et al. Atomic in-place updates for non-volatile main memories with Kamino-Tx. In: Proceedings of the 12th European Conference on Computer Systems. New York: ACM, 2017. 499–512
61 Chi P, Lee W C, Xie Y. Making B^+-tree efficient in PCM-based main memory. In: Proceedings of the International Symposium on Low Power Electronics and Design. New York: ACM, 2014. 69–74
62 Huang Y, Pavlovic M, Marathe V, et al. Closing the performance gap between volatile and persistent key-value stores using cross-referencing logs. In: Proceedings of the 24th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2018. 967–979
63 Tolia N, Tolia N, Ranganathan P, et al. Consistent and durable data structures for non-volatile byte-addressable memory. In: Proceedings of the USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2010
64 Chen S, Jin Q. Persistent B^+-trees in non-volatile main memory. In: Proceedings of the 41st International Conference on Very Large Data Bases. Berlin: Springer, 2015
65 Yang J, Wei Q, Chen C, et al. NV-tree: reducing consistency cost for NVM-based single level systems. In: Proceedings of the USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2015
66 Hwang D, Kim W H, Won Y, et al. Endurable transient inconsistency in byte-addressable persistent B^+-tree. In: Proceedings of the 16th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2018

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

67 Zuo P, Hua Y. A write-friendly and cache-optimized hashing scheme for non-volatile memory systems. IEEE Trans Parallel Distrib Syst, 2018, 29: 985–998

68 Zuo P, Hua Y, Wu J. Write-optimized and high-performance hashing index scheme for persistent memory. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2018. 461–476

69 Nam M, Cha H, Chio Y, et al. Write-optimized dynamic hashing for persistent memory. In: Proceedings of the USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2019

70 Lee S K, Lim K H, Song H, et al. WORT: write optimal radix tree for persistent memory storage systems. In: Proceedings of the USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2017. 257–270

71 Xie M H, Lu Y Y, Wang Q, et al. PetPS: supporting huge embedding models with persistent memory. In: Proceedings of the VLDB Endowment. New York: VLDB, 2023. 1013–1022

72 Zuo P F, Sun J H, Yang L, et al. One-sided RDMA-conscious extendible hashing for disaggregated memory. In: Proceedings of the 27th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2021. 15–29

73 Wang Q, Lu Y Y, Shu J W. Sherman: a write-optimized distributed B^+tree index on disaggregated memory. In: Proceedings of the International Conference on Management of Data. New York: ACM, 2022. 1033–1048

74 Zhang M, Hua Y, Zuo P F, et al. FORD: fast one-sided RDMA-based distributed transactions for disaggregated persistent memory. In: Proceedings of the 20th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2022. 51–68

75 Lee S, Ponnapalli S, Singhal S, et al. DINOMO: an elastic, scalable, high-performance key-value store for disaggregated persistent memory (extended version). 2022. ArXiv:2209.08743

76 Ma T, Zhang M X, Chen K, et al. AsymNVM: an efficient framework for implementing persistent data structures on asymmetric NVM architecture. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020. 757–773

77 Ana K, Heiner L, Christos K. ReFlex: remote flash ≈ local flash. In: Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017. 345–359

78 Jaehyun H, Cai Q Z, Tang A, et al. TCP ≈ RDMA: CPU-efficient remote storage access with i10. In: Proceedings of the 17th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2020. 127–140

79 Liao X, Lu Y, Yang Z, et al. Efficient crash consistency for NVMe over PCIe and RDMA. ACM Trans Storage, 2023, 19: 1–35

80 Chanwoo C, Jinhyung K, Junsu I, et al. LightStore: software-defined network-attached key-value drives. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019. 939–953

81 Sage A, Scott A, Ethan L, et al. Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2006. 307–320

82 Intel. DAOS v2.2. 2022. https://docs.daos.io/v2.2/

83 Sage A, Scott A, Ethan L, et al. CRUSH: controlled, scalable, decentralized placement of replicated data. In: Proceedings of the ACM/IEEE Conference on Supercomputing. New York: ACM, 2006. 122–134

84 Abutalib A, Sage A, Michael K, et al. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. New York: ACM, 2019. 353–369

85 Li N Q Q, Kalaba A, Freedman M, et al. Speculative recovery: cheap, highly available fault tolerance with disaggregated storage. In: Proceedings of the 28th USENIX Annual Technical Conference. Berkeley: USENIX Association, 2022. 271–286

86 Laurent B, Ashvin G, Willy Z. Hailstorm: disaggregated compute and storage for distributed LSM-based databases. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020. 301–316

87 Cao W, Liu Z J, Wang P, et al. PolarFS: an ultra-low latency and failure resilient distributed file system for shared

https://engine.scichina.com/doi/10.1360/SSI-2023-0034

storage cloud database. In: Proceedings of the VLDB Endowment. New York: VLDB, 2018. 1849–1862
88 Liao X J, Yang Z, Shu J W. RIO: order-preserving and CPU-efficient remote storage access. 2022. ArXiv:2210.08934
89 Grant S, Snoeren A C. In-network contention resolution for disaggregated memory. In: Proceedings of the Workshop on Resource Disaggregation and Serverless. New York: ACM, 2021. 1–7
90 Sidler D, Wang Z K, Chiosa M, et al. StRoM: smart remote memory. In: Proceedings of the 15th European Conference on Computer Systems. New York: ACM, 2020. 1–16
91 Korolija D, Koutsoukos D, Keeton K, et al. Farview: disaggregated memory with operator off-loading for database engines. 2021. ArXiv:2106.07102
92 Min J, Liu M, Chugh T, et al. Gimbal: enabling multi-tenant storage disaggregation on smartnic JBOFs. In: Proceedings of the ACM SIGCOMM Conference. New York: ACM, 2021. 106–122
93 Miao R, Zhu L J, Ma S, et al. From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud. In: Proceedings of the ACM SIGCOMM Conference. New York: ACM, 2022. 753–766
94 Lee Y, Al Maruf H, Chowdhury M, et al. Hydra: resilient and highly available remote memory. In Proceedings of the 20th USENIC Conference on File and Storage Technologies. Berkeley: USENIX Association, 2022. 181–198

Progress on storage systems for disaggregated data centers

Jiwu SHU*, Youmin CHEN, Qing WANG, Jing WANG, Junru LI & Xiaojian LIAO

Department of Computer Science and Technology, Tsinghua University, Beijing 100084*, China*
* Corresponding author. E-mail: shujw@tsinghua.edu.cn

Abstract Exponential growth in data has resulted in unprecedented challenges regarding data storage and management for data centers globally. It is becoming increasingly difficult to meet business needs due to the deficiencies of the traditional server-centric data center architecture in resource utilization, scalability, and performance. Recently, the disaggregated data center architecture has attracted considerable attention from academia and industry. In this architecture, hardware resources are decoupled into different hardware resource pools (such as processor, memory, and storage pools). These resource pools are interconnected through a high-speed network and can be scaled independently as needed; moreover, the pooled hardware resources can be flexibly shared among different applications, resulting in higher utilization. However, the disaggregated data center architecture presents significant differences in memory access mode, storage hierarchy, fault tolerance model, and software overhead, which elicits new challenges in building storage systems on top of such novel architectures. In this work, factors driving disaggregated data centers are analyzed, and their architectural features and advantages are described. Furthermore, the key technologies and representative research work on disaggregated storage systems are summarized. Lastly, future development trends, including memory-level data reliability and heterogeneous computing and networking, are outlined.

Keywords disaggregated datacenter, disaggregated memory, disaggregated storage, separation of compute and storage

https://engine.scichina.com/doi/10.1360/SSI-2023-0034