doi:10.11887/j.cn.202003002　　http://journal.nudt.edu.cn

基于新型非易失内存的远程零拷贝文件系统*

韩文炳[1,2],陈小刚[1],李顺芬[1],李大刚[3],陈诗雁[3],段有康[1,2],宋志棠[1]
(1. 中国科学院上海微系统与信息技术研究所 信息功能材料国家重点实验室,上海 200050;
2. 中国科学院大学,北京 100049;3. 北京大学深圳研究生院 信息工程学院,广东 深圳 518055)

摘　要:为提升物联网与边缘计算应用中前端节点间的数据访问效率,提出了一种新型远程零拷贝文件系统。该文件系统无须借助特殊硬件,可直接基于通用网卡设备实现零拷贝的数据传输框架;充分利用新型非易失内存的随机访问特性,尽可能减少数据缓存和拷贝,提高数据访问的吞吐量。建立缓冲区池,精简并融合传统网络栈和存储栈,从而缩短文件访问路径,减少软件冗余,降低数据访问延迟。最终提供高带宽、低延迟的远程数据访问性能。测试结果表明,远程零拷贝文件系统比传统网络文件系统减少了 42.26% ~ 99.19% 的读写延迟,细粒度访问下的吞吐量最高可提升 1297 倍,显著降低了处理器开销和缓存失效次数。

关键词:新型非易失内存;物联网;边缘计算;零拷贝;网络文件系统
中图分类号:TP302　　**文献标志码**:A　　**文章编号**:1001-2486(2020)03-009-08

A novel remote zero-copy file system based on non-volatile memory

HAN Wenbing[1,2], *CHEN Xiaogang*[1], *LI Shunfen*[1], *LI Dagang*[3], *CHEN Shiyan*[3], *DUAN Youkang*[1,2], *SONG Zhitang*[1]
(1. State Key Laboratory of Functional Materials for Informatics, Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China; 2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China)

Abstract: In order to enhance data access efficiency between front-end nodes of Internet of things and edge computing applications, a novel RZCFS(remote zero-copy file system) was proposed. The zero-copy data transmission framework based on generic network interface card was realized without special hardware. The random-access character of non-volatile memory was fully utilized by RZCFS to reduce data caches and copies, which yields a significant throughput improvement for data access. A buffer pool was implemented, the traditional network stack and storage stack were simplified and converged to shorten the file access path, which can eliminate software overhead and reduce data access latency. As a result, RZCFS provides the low-latency and high-bandwidth remote data access. Simulation results show that it achieves 42.26% ~ 99.19% latency reductions for traditional NFS (network file system). The throughput of fine-grained access can reach 1297 times faster than those of NFS. The RZCFS significantly reduces the processor cycles and cache misses.

Keywords: non-volatile memory; Internet of things; edge computing; zero copy; network file system

随着物联网数据与设备的急剧增长,边缘计算成为新的发展趋势[1]。根据相关研究[2],2025 年将有超过 754 亿设备终端接入物联网。而 2019 年,人类、计算机以及其他设备总计将产生 500 ZB 数据。其中,45% 的物联网数据将在网络边缘进行分析、处理和储存[1]。在前端分布式架构物联网中,为减轻服务器压力,加速系统响应,大部分的数据任务和用户请求都将在传感节点或网关设备上进行处理[3]。然而,大量的数据与互联设备使得相邻节点间的数据交换与传输比以往更为频繁和复杂。因此,构建高效、低延迟的远程数据访问是物联网与边缘计算应用中的关键问题。

新型非易失内存(Non-Volatile Memory, NVM)的出现为物联网与边缘计算提供了新的解决方案。以相变存储器[4]、3D XPoint[5]、阻变存储器[6]和磁阻存储器[7]为代表的 NVM 技术,可以提供低延迟、字节级寻址以及持久化的数据访问性能。此外,NVM 兼具功耗低、抗辐射、存储密度大等特点。在存储级内存(Storage Class Memory, SCM)架构[8]中,NVM 直接挂载在内存总线上,中央处理器

* **收稿日期**:2018-11-28
基金项目:国家重点研发计划资助项目(2017YFA0206101,2017YFB0701703,2017YFA0206104,2018YFB0407500,SQ2017YFGX020134);国家自然科学基金资助项目(61874129,61874178,61504157,61622408);中国科学院战略性先导科技专项资助项目(XDPB12);上海市科委资助项目(17DZ2291300,18DZ2272800)
作者简介:韩文炳(1994—),男,山东济宁人,博士研究生,E-mail:hwbx@mail.ustc.edu.cn;
陈小刚(通信作者),男,副研究员,博士,E-mail:chenxg@mail.sim.ac.cn

(Central Processing Unit, CPU)可以使用 Load/Store 指令直接访问 NVM 上的持久化数据。然而,传统的文件系统和存储栈针对磁盘等慢速块设备进行设计,无法充分发挥出 NVM 的优势和性能。例如,传统磁盘需要以机械方式寻找磁道,这决定了其随机读写性能将远弱于顺序读写性能,因此存储系统会排列合并多个输入/输出(Input/Output, I/O)请求和缓存数据以提升读写性能。在磁盘环境下,相对于磁盘本身较高的读写延迟,这类软件开销可近似忽略,仅占整个数据访问延迟的 0.3%;而在 NVM 环境下,NVM 的读写延迟远低于磁盘,此时软件延迟的占比高达 94.1%[9]。因此,软件延迟已经成为 NVM 存储系统的主要性能瓶颈[10]。

1 相关工作

目前,国内外针对 NVM 的存储系统做了大量研究,以提升数据访问效率。持久性字节寻址文件系统(Byte addressable Persistent File System, BPFS)[11]利用 NVM 的可字节寻址特性,提出短路影子分页技术,实现写时复制,削弱写放大效应,但该方法需要特殊硬件支持。存储级内存文件系统(Storage Class Memory File System, SCMFS)[12]使用连续的内核虚拟地址空间管理 NVM,然而维护两套内存映射表的一致性会造成一定的性能损失。持久性内存文件(Persistent Memory File System, PMFS)[13]使用多叉平衡查找树记录文件数据块的物理地址,可持续内存文件系统(Sustainable In-Memory File System, SIMFS)[14]使用页表项管理文件数据,避开页高速缓存和面向块设备的 I/O 软件栈,从而达到较高的性能。以上工作均是优化本地 NVM 文件系统设计。

对于分布式文件系统(Distributed File System, DFS),现有研究[10, 15-16]大多将 NVM 与远程直接内存访问(Remote Direct Memory Access, RDMA)技术结合起来,实现分布式内存访问。Octopus[10]将各节点的 NVM 抽象为共享内存池,并使用哈希映射对文件数据进行布局。NVM 和 RDMA 感知分布式文件系统(NVM and RDMA aware Hadoop distributed File System, NVFS)[15]针对 NVM 和 RDMA 重新设计了 Hadoop 分布式文件系统,实现内存原语访问,加速高性能计算集群应用。Hotpot[16]将数据缓存和数据备份合并,并利用应用程序检查点维护缓存的一致性,实现高效可靠的分布式共享内存访问。然而,RDMA 通常需要特殊硬件的支持,有着较高的成本,不适合用于物联网应用场景中。通用网卡设备成本低,应用普遍,但传统网络栈中的数据拷贝和软件冗余限制了传输带宽和效率。存储感知网络栈(Storage Aware Network Stack, SANS)是一种基于通用网卡(Network Interface Card, NIC)的零拷贝传输框架[17],能够有效减少数据拷贝,充分发挥 NIC 硬件性能。

本文充分考虑了 NVM 与通用网卡的硬件特性,提出了一种新型远程零拷贝文件系统(Remote Zero Copy File System, RZCFS)。它在通用网卡设备上实现了零拷贝的网络数据传输,利用 NVM 减少数据缓存和拷贝,融合网络栈与存储栈以降低软件开销,在前端节点间建立对等的网络连接,为物联网和边缘计算等应用提供低延迟、高带宽的远程数据访问性能。

2 RZCFS 系统架构与传输机制

2.1 系统架构

图 1 分别描述了传统 DFS 和 RZCFS 的系统架构。对于一次 DFS 的 I/O 请求,应用程序先通过虚拟文件系统(Virtual File System, VFS)调用 DFS 协议,并依次经过远程过程调用、传输控制协议(Transmission Control Protocol, TCP)、网际协议(Internet Protocol, IP)等网络层协议向远程节点发送请求命令。远程节点接收到该请求后,在网络栈中解码并提交给 DFS 守护进程处理。守护进程通过 VFS 调用本地文件系统,经历通用块层、I/O 调度层以及块设备驱动层等存储软件栈访问存储设备。纵观数据的传输路径,在硬件层次上,从存储介质到内存之间存在一次数据拷贝;软件层次上,从内核空间到用户空间也有一次拷贝;此外,用户空间到网络栈还有一次拷贝。如果不考虑网络间的数据传输,那么一次 I/O 请求在两个存储节点上至少存在 6 次数据拷贝。多次的

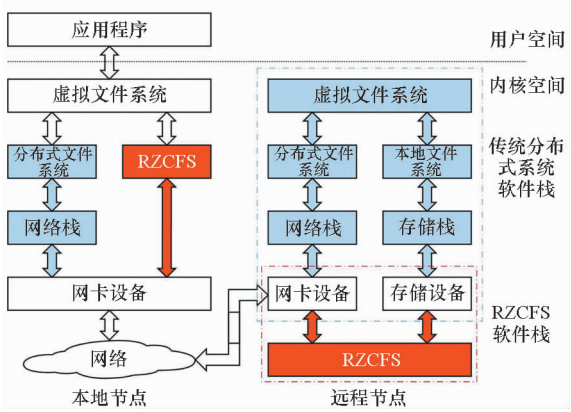


图 1 系统架构

Fig. 1 System architecture

数据拷贝既加重了 CPU 的负担,降低了数据吞吐量,同时也增加了系统功耗。

RZCFS 使用 SANS 网络传输框架,去除了 TCP/IP 等传统网络协议栈,直接与网卡驱动交互,在数据链路层发送和接收数据请求。远程节点接收到请求命令后,在数据链路层将请求提交给 RZCFS 处理。由于 NVM 存储介质可直接连接 CPU,因此 RZCFS 可以旁路页高速缓存、通用块层等存储软件栈,直接访问 NVM 上的文件数据。RZCFS 不需要依赖于本地文件系统,它融合了网络栈与存储栈,可以同时支持本地文件管理和远程文件访问,有效缩短了文件访问路径。从数据迁移的角度来看,借助于 SANS 框架,RZCFS 实现了文件数据的零拷贝传输。RZCFS 对存储栈的精简,去除了存储介质到内存的拷贝。当应用程序使用内存映射(Memory MAP, MMAP)等用户态访问方式时,内核空间到用户空间的拷贝也被去除。因此,RZCFS 可以实现远程文件数据的零拷贝访问。

2.2　零拷贝传输机制

SANS 零拷贝网络传输机制不需要特殊网络硬件的支持,可以有效减少处理器开销,降低传输延迟。本文将 SANS 应用到 RZCFS 的远程文件数据传输中,以实现远程零拷贝的文件访问。

文件系统的数据访问往往以4 KB为单位,与内存页大小一致;RZCFS 中,文件数据也是按 4 KB分块的;文件数据和用户缓存虽然有连续的虚拟地址,但物理地址是不连续的,它们散落在内存的不同位置,均按 4 KB 地址对齐。

通用网络控制器[18]的硬件特性包括:支持分散聚合直接内存访问(Direct Memory Access, DMA);DMA 地址需要按 4 B 对齐;最大传输单元为 1518 B,不支持巨型帧,这意味着需要使用多个网络包去发送 4 KB 数据。由于 NVM 直接挂载到内存总线上,所以网络控制器可以 DMA 寻址 NVM 空间。

结合上述特性,RZCFS 应用 SANS 框架来传输文件数据,以下是 4 KB 数据传输的例子。

图 2 中,SANS 将需要传输的 4 KB 数据划分为 3 段,长度分别是 1496 B,1496 B,1104 B。数据发送时,每个数据段添加 16 B 以太网包头,从而得到 1512 B、1512 B、1120 B 长度的 3 个网络包。在接收端,同样将 4 KB 的接收缓冲区划分为 3 段。约定接收缓冲区起始地址为 0x0010,则设置第一个数据包接收地址为 0x0A28,第二个为 0x0450,第三个为 0x0000。SANS 由高地址到低地址反向发送 3 个数据块,即分段 3 对应网络包 1,分段 2 对应网络包 2,分段 1 对应网络包 3。在接收节点上,网络包 1 先到达 0x0A28 位置;网络包 2 到达 0x0450 位置,并将网络包 1 的包头覆盖掉;网络包 3 到达 0x0000 位置,将网络包 2 的包头覆盖掉。SANS 通过反向发送数据,使后一网络包覆盖前一网络包的包头,从而拼接出连续的 4 KB 数据块,称为反向覆盖方法。

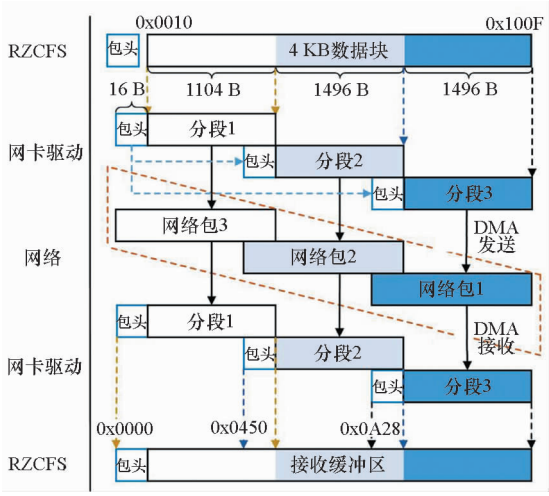


图 2　零拷贝传输机制

Fig. 2　Zero copy transmission mechanism

上述传输过程都可以通过 DMA 硬件完成,CPU 只需配置好 DMA 发送和接收的物理地址,即可在没有数据拷贝的前提下实现 4 KB 数据块的自动分片和拼接,极大地降低处理器的开销。然而,SANS 会引入包头覆盖问题,即接收网络包 3 时,其包头会覆盖掉接收缓冲区起始地址之前的 16 B 数据。在后续章节中,RZCFS 会通过缓冲区管理和预存机制来避免。

3　RZCFS 设计与实现

3.1　RZCFS 布局结构

如图 3 所示,RZCFS 的组成部分包括超级块,索引节点(inode)表,位图、数据空间。其中,超级块记录 RZCFS 的全局信息,如起始物理地址、inode 数量等;inode 表存放本地文件的 inode 结构;位图记录数据空间的数据块使用情况;数据空间划分为 4 KB 数据块,存储本地文件数据。另外,RZCFS 分配一段 NVM 空间作为缓冲区池,存放网络接收的命令包和数据包,以缓冲远程文件的 inode 和数据。

全局变量 Net Local 结构用于连接 RZCFS 和

网卡驱动。它包含超级块和缓冲区池的虚拟地址和物理地址、缓冲区管理信息、网络控制器信息、RZCFS 与网卡驱动的函数句柄等。通过 Net Local、RZCFS 和网卡驱动都可以方便地访问缓冲区池和数据空间。

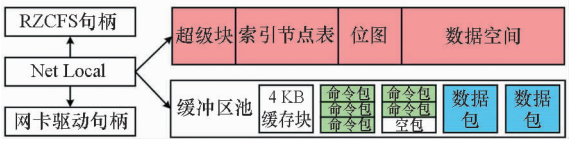


图 3 RZCFS 布局结构

Fig. 3 RZCFS layout

3.2 缓冲区管理

RZCFS 有两种网络包:数据包和命令包。两者通过以太包头中的 type 字段进行区分,数据包的 type 字段为 0x8000,命令包为 0x8001。根据 SANS 框架,每 3 个数据包传输 4 KB 数据,长度依次为 1512 B、1512 B、1120 B。命令包用于发送 RZCFS 的远程调用命令,返回执行结果或远程文件元数据,长度为 256 B。缓冲区池为数据包和命令包提供一个统一的接收管理方案。

如图 4 所示,将缓冲区池按 4 KB 大小分块并对齐。根据 SANS 框架,每个缓存块分为 3 段,按照数据包方式设置接收地址。由于命令包的长度始终小于数据包,所以该缓存块既可以接收 3 个数据包,也可以接收 3 个命令包。RZCFS 使用循环链表管理所有缓存块,链表结点包括一个物理地址,即缓存块的起始物理地址;一个状态位,用于标记这个块是否空闲。RZCFS 只缓存元数据,不会缓存文件数据。RZCFS 由高地址到低地址反向访问缓冲区池,每次选择一个空闲的缓存块设置接收地址。反向遍历缓冲区池使得包头被接收在低地址侧的相邻缓存块末端,而不会破坏之前缓存块的数据。如果低地址侧的相邻缓存块缓存了元数据,由于命令包长度远小于数据包长度,16 B 包头仍旧不会覆盖到其中的有效数据。由此避免了缓冲区池内的包头覆盖问题。

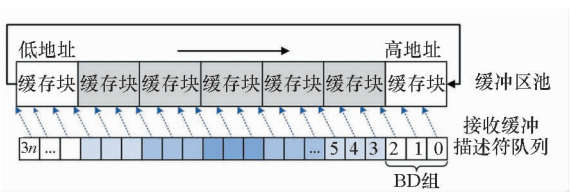


图 4 缓冲区管理

Fig. 4 Buffer pool management

网络包的接收地址通过接收缓冲描述符 RX BD来设置。每个 RX BD 对应一个网络包,并存放着该包的长度、状态、接收地址等信息。根据 SANS 框架,每 3 个 RX BD 配置一个缓存块,称为一个 BD 组。初始化时,将所有 RX BD 都配置为缓冲区池内空闲缓存块的物理地址。每当接收新的网络包后,被释放的 RX BD 以 BD 组为单位配置为空闲缓存块的物理地址,继续为 RZCFS 提供接收缓冲区。

命令包与数据包的混杂可能导致当前的 RX BD队列没有按组对齐,从而无法正确拼接数据包。RZCFS 使用空包机制(Nothing OPeration, NOP)克服这个问题。当 RZCFS 发送命令包请求远程数据时,会将本地 RX BD 中多余 BD 的数目也发送给远程节点。由于 RX BD 的总数是对 3 整除的,所以多余 BD 的数目即当前 RX BD 序号除 3 的余数。当远程节点接收到命令包后,先发送若干空包消耗掉多余 RX BD,然后再发送数据包。此时本地节点上的 RX BD 队列是按 BD 组对齐的,可以正确接收文件数据块。显然,在每次请求中,最多发送两个 NOP 包。因此,对于大容量的数据传输,NOP 包带来的冗余非常小。

由于缓冲区池和数据空间区的分块都按 4 KB地址对齐,所以当数据块按 SANS 框架拼接后,其物理地址也是按 4 KB 对齐的,从而满足文件系统或应用程序的访问要求。

3.3 远程读操作

图 5 是读远程文件数据流图。远程节点接收到读文件请求命令包后,RZCFS 解析命令包中的参数,查找文件数据的物理地址。远程节点先发送 NOP 包,然后按照 SANS 框架依次发送数据包。本地节点在缓冲区池中接收文件数据块。如果应用程序使用可移植操作系统接口(Portable Operating System Interface of UNIX, POSIX)方式读文件,还需要将文件数据从缓冲区池拷贝至用户缓冲区;如果使用 MMAP 方式,RZCFS 只需在缺页中断处理时返回缓冲区池中的文件数据物理地址,完成数据映射。从图 5 可以看出,在远程文

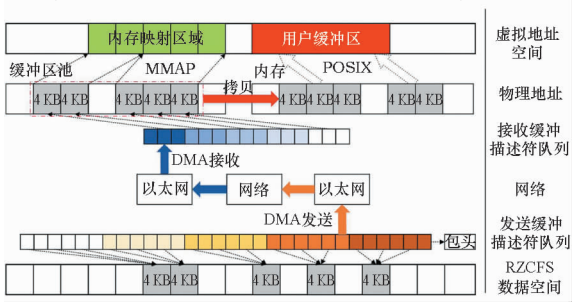


图 5 读远程文件数据流图

Fig. 5 Data flow diagram of reading remote file

件的读路径中,文件数据的发送与接收均是通过 DMA 完成,POSIX Read 存在一次数据拷贝,MMAP 方式不需要数据拷贝。

3.4　远程写操作

RZCFS 使用预存机制解决数据空间区内的包头覆盖问题。在配置接收地址时,如果 RZCFS 检测到数据块位于数据空间区,则保存该数据块起始地址前的 16 B 数据。在接收数据包时,如果检测到接收地址位于数据空间区,则在数据块拼接完成后,将对应的 16 B 数据恢复到原来位置。数据覆盖仅发生在 BD 组内的最后一个数据包被接收后,并在中断上下文中立刻恢复为原有数据,所以应用程序不会访问到错误的文件数据。

图 6 是写远程文件数据流图。远程节点接收到写文件请求命令包后,根据命令包参数查找被写文件的数据块地址。由于远程节点网络包的默认接收地址为缓冲区池,为了减少数据拷贝,RZCFS 会重新配置 RX BD 指向文件数据块的物理地址。RX BD 重设完毕后,远程节点向本地节点发送写请求应答命令包。由于第一个 RX BD 已经被网络控制器硬件占用,无法重设其接收地址,所以本地节点先发送一个 NOP 包,消耗掉该 BD 后再发送文件数据。多发送一个 NOP 包来代替一次数据拷贝无疑是值得的,尤其是大块数据写入的情况。从图 6 可以看出,在远程文件写路径中,POSIX 和 MMAP 均不需要数据拷贝,数据通过 DMA 直接接收在文件系统的数据空间区。

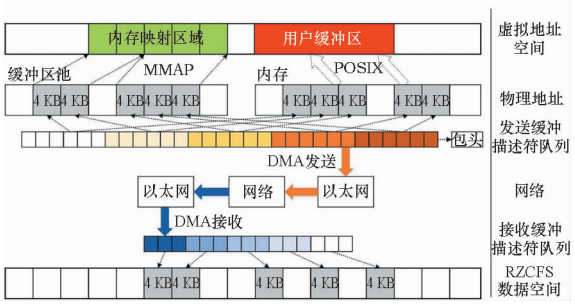


图 6　写远程文件数据流图

Fig. 6　Data flow diagram of writing remote file

4　实验结果与分析

4.1　原型系统

由于支持内存接口的大容量 NVM 产品还未商用,因此本文使用 DRAM 代替 NVM 开展实验。与 NVM 相比,DRAM 同样支持随机访问且读写速度更快,RZCFS 主要考虑了 NVM 的随机访问特性,因此使用 DRAM 模拟 NVM 测试性能是合理的。由于 RZCFS 的优化途径是减少软件冗余和拷贝次数,而 NVM 的拷贝延迟比 DRAM 更高,因此 RZCFS 在真实的 NVM 介质上将会比 DRAM 模拟环境下获得更高的性能提升。

本文将 ZedBoard 开发平台[19]作为物联网中的计算节点,使用内存模拟 NVM 介质,实现并构建 RZCFS。同时基于虚拟内存盘(Ramdisk)建立第四代扩展文件系统(fourth Extended file system, Ext4),并在此之上挂载网络文件系统(Network File System, NFS),作为对比方案。为了避免性能损失,禁用了 Ext4 的日志功能。具体硬件配置见表 1。

表 1　硬件配置参数

Tab. 1　Hardware parameters

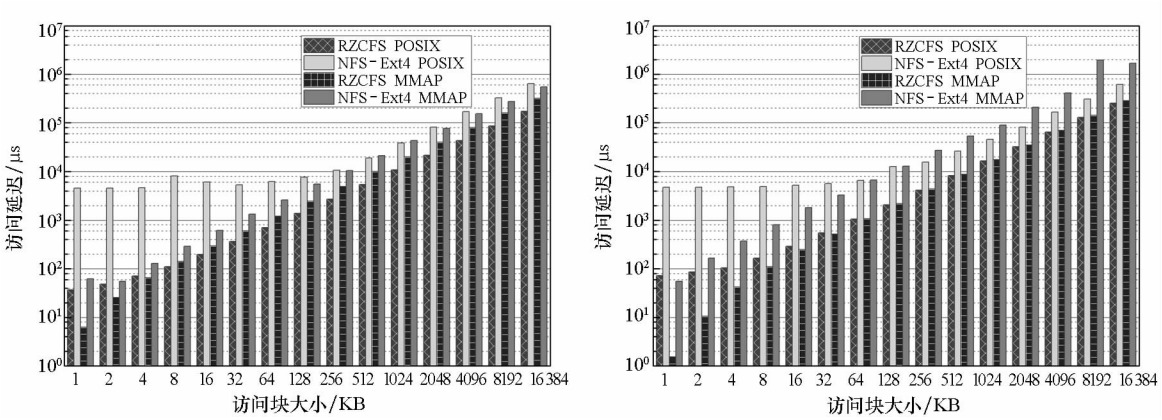
系统组件	配置信息
CPU	Dual ARM Cortex™ – A9 667 MHz
一级缓存	32 KB 指令缓存,32 KB 数据缓存
二级缓存	512 KB
内存	512 MB,第三代双倍数据率接口(Double Data Rate 3, DDR3),533 MHz
网络控制器	千兆位以太网控制器,支持 DMA 分散聚合

4.2　延迟与吞吐量测试

本文使用文件系统标准测试工具 IOzone[20]测试 RZCFS 和 NFS – Ext4 远程数据的访问性能。图 7展示了两种方案访问块大小从 1 KB 到16 MB 的情况,分别使用 POSIX 和 MMAP 方式访问时的平均读、写延迟。在所有访问粒度和方式下,RZCFS 都表现出更低的读、写延迟。在最好情况下(块大小为 1 KB 时),RZCFS 比 NFS – Ext4 减少了 99.19% 的读延迟,减少了 98.44% 的写延迟。RZCFS 通过零拷贝机制和精简软件栈,缩短了文件访问路径,降低了读、写延迟。随着块大小的增大,RZCFS 的性能优势变小。这是由于块大小越大,相关的 I/O 调用次数和数据拷贝的开销就越少,这缓解了 NFS – Ext4 的多层软件栈和多次数据拷贝带来的性能损失。但即便在 16 MB 块大小下,RZCFS 仍旧可以减少 42.26% 的读延迟和 58.82% 的写延迟。

图 8 是 RZCFS 和 NFS – Ext4 的吞吐量性能对比。在 POSIX 方式下,RZCFS 的顺序读和顺序写性能分别是 NFS – Ext4 的 3.68 ~ 63.95 倍和 2.43 ~ 46.86 倍;随机读和随机写性能提升更大,分别是 3.61 ~ 90.07 倍和 5.74 ~ 86.41 倍。这是因为,尽管 NFS – Ext4 的数据存储在内存中,但访问过程仍然需要经过多层网络协议与块设备软件层次,从而产生不必要的数据拷贝。RZCFS 通过

融合网络栈与存储栈降低软件开销,并提供零拷贝访问以减少数据迁移,从而大幅提高了文件系统的吞吐量。另外,顺序写的吞吐量高于顺序读是由于顺序写需要创建新的数据块,造成了性能开销。随机写比随机读的吞吐量高是因为写操作是零拷贝访问,而读操作需要一次拷贝。在MMAP方式下,RZCFS在各种访问粒度和读写模式下的吞吐量都比较稳定。相比于NFS-Ext4,RZCFS显著提高了随机读写性能。在块大小为1 KB时,RZCFS的随机读和随机写速度分别快996倍和1297倍。这是因为NFS-Ext4基于页高速缓存实现MMAP,大量小块数据的随机访问导致了频繁的缺页中断和上下文切换,严重影响了文件系统性能。

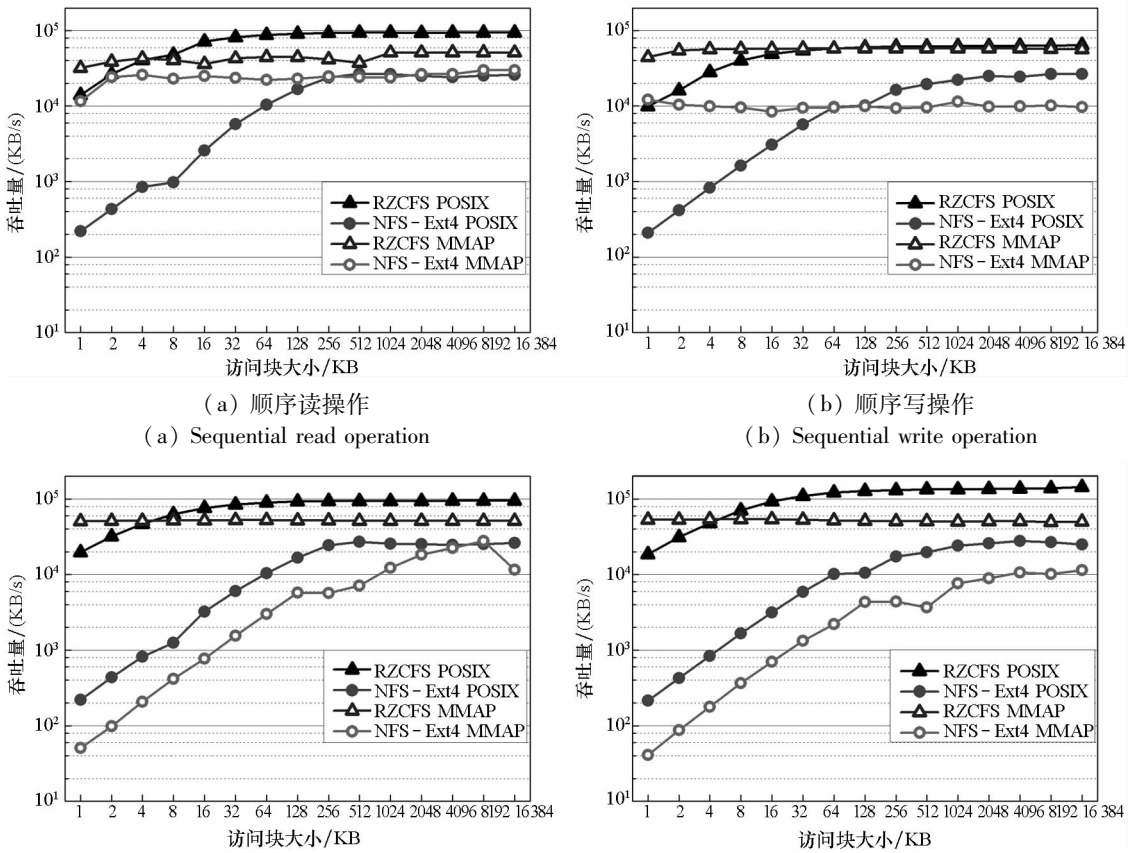


(a) 读操作
(a) Read operation

(b) 写操作
(b) Write operation

图7 数据访问延迟

Fig. 7 Latency of data access



(a) 顺序读操作
(a) Sequential read operation

(b) 顺序写操作
(b) Sequential write operation

(c) 随机读操作
(c) Random read operation

(d) 随机写操作
(d) Random write operation

图8 吞吐量性能

Fig. 8 Throughput performance

4.3　数据拷贝分析

RZCFS 主要通过减少数据拷贝来提升数据访问性能。为了验证数据拷贝的影响,本文实现了一拷贝的远程写操作,即先在缓冲区池内接收文件数据,再将其拷贝至数据空间区。如图 9 所示,零拷贝方案的吞吐量是一拷贝方案的 1.54 ~ 1.87 倍。块大小为 1 KB 时,零拷贝性能提升最高,这说明减少数据拷贝可以有效提升小块数据的访问性能。

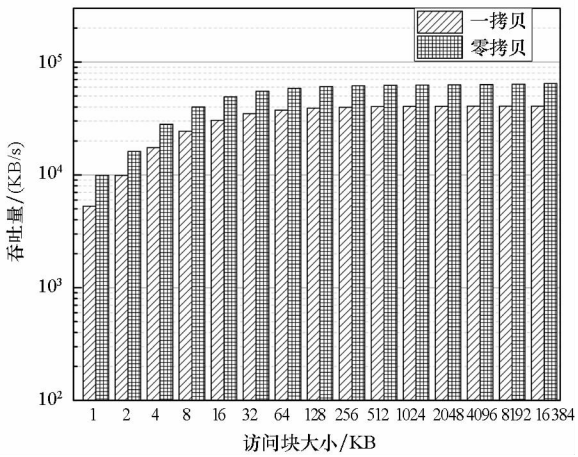


图 9　一拷贝性能分析

Fig. 9　Performance analysis of one-copy

4.4　CPU 周期与缓存失效分析

为验证 RZCFS 的性能细节,本文使用性能分析工具 perf[21] 抓取了 MMAP 下小文件随机读写的 CPU 周期和缓存失效次数。在图 10 中,随着块大小的减少,NFS - Ext4 所消耗的 CPU 周期和产生的缓存失效急剧增加。在块大小为 1 KB 时,RZCFS 最多减少了 98.59% 的 CPU 周期,降低了 99.03% 的缓存失效。这是由于 RZCFS 的数据访问避免了多层网络协议和存储软件层次,使用零拷贝机制一次性传输多个数据页,从而大大减少了 CPU 的负担与缓存替换。

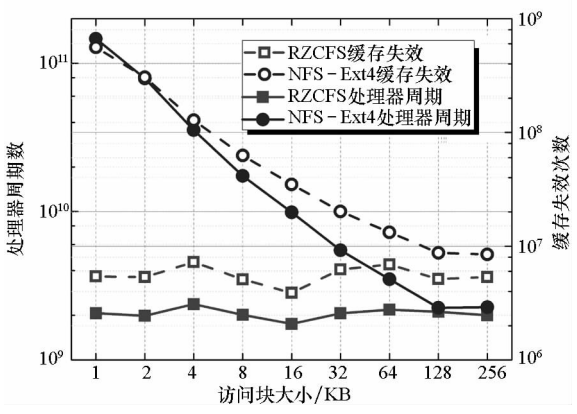


图 10　CPU 周期与缓存失效分析

Fig. 10　Analysis of CPU cycles and cache misses

5　结论

本文为提升物联网与边缘计算中前端节点间的数据访问效率,提出了一种新型远程零拷贝文件系统。它基于通用网卡设备实现零拷贝的网络数据传输;充分发挥 NVM 的优势,避免数据缓存和迁移;精简并融合网络栈与存储栈,缩短文件访问路径,降低数据访问延迟。最终在节点间建立对等的网络连接,提供高效、低延迟的远程数据访问性能。实验结果表明,本文提出的远程零拷贝文件系统有效减少了访问延迟,大幅提升了吞吐量性能,显著降低了处理器开销和缓存失效次数,为 NVM 在物联网和边缘计算等领域的应用优化提供了技术支撑。

参考文献(References)

[1] Shi W, Cao J, Zhang Q, et al. Edge computing: vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637 - 646.

[2] Lucero S. IoT platforms: enabling the Internet of Things[R/OL]. (2016 - 03 - 15)[2018 - 05 - 10]. https://cdn.ihs.com/www/pdf/enabling-IOT.pdf.

[3] 陈海明, 崔莉, 谢开斌. 物联网体系结构与实现方法的比较研究[J]. 计算机学报, 2013, 36(1): 168 - 188.
CHEN Haiming, CUI Li, XIE Kaibin. A comparative study on architectures and implementation methodologies of Internet of things[J]. Chinese Journal of Computers, 2013, 36(1): 168 - 188. (in Chinese)

[4] Burr G W, Breitwisch M J, Franceschini M, et al. Phase change memory technology[J]. Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena, 2010, 28(2): 223 - 262.

[5] Intel. Introducing Intel Optane technology-bringing 3D XPoint memory to storage and memory products[EB/OL]. (2016 - 03 - 15)[2018 - 05 - 10]. https://newsroom.intel.com/press-kits/introducing-intel-optane-technology-bringing-3d-xpoint-memory-to-storage-and-memory-products/.

[6] Akinaga H, Shima H. Resistive random access memory (ReRAM) based on metal oxides[J]. Proceedings of the IEEE, 2010, 98(12): 2237 - 2251.

[7] Åkerman J. Toward a universal memory[J]. Science, 2005, 308(5721): 508 - 510.

[8] Burr G W, Kurdi B N, Scott J C, et al. Overview of candidate device technologies for storage-class memory[J]. IBM Journal of Research and Development, 2008, 52(4.5): 449 - 464.

[9] Lee E, Bahn H, Yoo S, et al. Empirical study of NVM storage: an operating system's perspective and implications[C]// Proceedings of IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, 2014: 405 - 410.

[10] Lu Y, Shu J, Chen Y, et al. Octopus: an RDMA-enabled distributed persistent memory file system[C]//Proceedings of USENIX Annual Technical Conference (USENIX ATC 17), 2017: 773 - 785.

[11] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory[C]//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009: 133 - 146.

[12] Wu X, Reddy A L. SCMFS: a file system for storage class memory [C]//Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, 2011: 39.

[13] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory[C]//Proceedings of the Ninth European Conference on Computer Systems, 2014.

[14] Sha E H M, Chen X, Zhuge Q, et al. Designing an efficient persistent in-memory file system[C]//Proceedings of Non-Volatile Memory System and Applications Symposium (NVMSA), IEEE, 2015: 1 - 6.

[15] Islam N S, Wasi-ur-Rahman M, Lu X, et al. High performance design for HDFS with byte-addressability of NVM and RDMA[C]//Proceedings of the International Conference on Supercomputing, 2016: 8.

[16] Shan Y, Tsai S Y, Zhang Y. Distributed shared persistent memory [C]//Proceedings of the Symposium on Cloud Computing, 2017: 323 - 337.

[17] Chen S, Li D, Chen X, et al. Storage-aware network stack for NVM-assisted key-value store [C]//Proceedings of 27th International Conference on Computer Communication and Networks (ICCCN), 2018: 1 - 9.

[18] Xilinx Inc. Zynq - 7000 all programmable SoC technical reference manual (v1. 12. 1)[EB/OL]. (2017 - 12 - 06)[2017 - 12 - 20]. https://www. xilinx. com/support/documentation/user_guides/ug585-Zynq-7000-TRM. pdf.

[19] Digilent Inc. ZedBoard (REV D)[EB/OL]. [2017 - 11 - 20]. http://zedboard. org/product/zedboard.

[20] Norcott W D. Iozone filesystem benchmark [EB/OL]. (2016 - 01 - 23) [2018 - 05 - 10]. http://www. iozone. org.

[21] Wiki P. Perf: Linux profiling with performance counters[EB/OL]. (2015 - 09 - 28)[2018 - 05 - 10]. https://perf. wiki. kernel. org/index. php/Main_Page.