DOI:10.7544/issn1000-1239.2021.20200369
58(2):384-396,2021

一种基于 RDMA 多播机制的分布式持久性内存文件系统

陈茂棠[1] 郑圣安[2] 游理通[1] 王晶钰[1] 闫 田[1] 屠要峰[3] 韩银俊[3] 黄林鹏[1]
[1](上海交通大学计算机科学与工程系 上海 200240)
[2](清华大学计算机科学与技术系 北京 100084)
[3](中兴通讯股份有限公司 南京 210012)
(chenmaotang@sjtu.edu.cn)

A Distributed Persistent Memory File System Based on RDMA Multicast

Chen Maotang[1], Zheng Sheng'an[2], You Litong[1], Wang Jingyu[1], Yan Tian[1], Tu Yaofeng[3], Han Yinjun[3], and Huang Linpeng[1]
[1](*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai* 200240)
[2](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[3](*ZTE Corporation, Nanjing* 210012)

Abstract The development of persistent memory and remote direct memory access(RDMA) provides new opportunities for designing efficient distributed systems. However, the existing RDMA-based distributed systems are far from fully exploiting RDMA multicast capabilities, which makes them difficult to solve the problem of multi-copy file data transmission in one-to-many transmission, degrading system performance. In this paper, a distributed persistent memory and RDMA multicast transmission based file system(MTFS) is proposed. It efficiently transmits data to different data nodes by the low-latency multicast transmission mechanism, which makes full use of the RDMA multicast capability, hence avoiding high latency due to multi-copy file data transmission operations. To improve the flexibility of transmission operations, a multi-mode multicast remote procedure call (RPC) mechanism is proposed, which enables the adaptive recognition of RPC requests, and moves transmission operations out of the critical path to further improve transmission efficiency. MTFS also provides a lightweight consistency guarantee mechanism. By designing a crash recovery mechanism, a data verification module and a retransmission scheme, MTFS is able to quickly recover from a crash, and achieves file system reliability and data consistency by error detection and data correction. Experimental results show that MTFS has greatly increased the throughput by 10.2—219 times compared with GlusterFS. MTFS outperforms NOVA by 10.7% on the Redis workload, and achieves good scalability in multi-thread workloads.

Key words persistent memory; remote direct memory access; multicast; distributed file system; remote procedure call

摘 要 持久性内存技术与远程直接内存访问(remote direct memory access, RDMA)技术的发展,为高效分布式系统的设计提供了新的思路.然而,现有的基于 RDMA 的分布式系统没有充分利用 RDMA

收稿日期:2020-06-05;**修回日期**:2020-07-30
基金项目:国家重点研发计划项目(2018YFB1003302);上海交通大学-华为联合实验室项目(FA2018091021-202004)
This work was supported by the National Key Research and Development Program of China (2018YFB1003302) and the SJTU-Huawei Innovation Research Lab Project (FA2018091021-202004).
通信作者:郑圣安(venero@tsinghua.edu.cn)

的多播能力,难以解决 1 对多传输场景下的多拷贝文件数据传输问题,严重影响了系统性能.针对此问题,提出一种基于 RDMA 多播机制的分布式持久性内存文件系统(RDMA multicast transmission based distributed persistent memory file system, MTFS),通过低延迟多播通信机制充分利用 RDMA 多播能力,将数据高效传输到多个数据节点,从而避免了多拷贝传输操作带来的高延迟.为提升传输操作灵活性,MTFS 设计了多模式多播远程过程调用(remote procedure call, RPC)机制,实现了 RPC 请求自适应识别,并通过优化返回机制将部分传输操作移出关键路径,进一步提升传输效率.同时 MTFS 提供了轻量级一致性保障机制,通过设计故障恢复功能、数据校验系统、重传策略与窗口机制,当节点出现崩溃时进行快速恢复,并在传输出现错误时实现数据精准检测与纠正,保证了数据的可靠性和一致性.实验证明,MTFS 在各测试集上相比现有系统 GlusterFS 吞吐量提升了 10.2～219 倍.在 Redis 数据库的工作负载下,MTFS 相比于 NOVA 取得了最高 10.7%的性能提升,并在多线程测试中取得了良好的可扩展性.

关键词 持久性内存;远程直接内存访问;多播;分布式文件系统;远程过程调用

中图法分类号 TP391

新型的非易失性内存(non-volatile memory, NVM)[1-5]技术的出现为传统的计算机存储结构带来变革,其中通过内存总线连接 CPU 的 NVM 形态也被称为持久性内存(persistent memory, PM).与 DRAM 相比,持久性内存拥有相接近的传输带宽与访问延迟,但兼具了 DRAM 不具备的持久性.随着持久性内存的逐渐应用,许多研究针对持久性内存设计特异性的文件系统,其中大多数文件系统基于单台机器设计[6-10].但是目前大容量的持久性内存价格昂贵,单机存储容量难以提高.同时,持久性内存运行带来较高的 CPU 负载,单机大规模部署持久性内存会使 CPU 成为存储性能的瓶颈.这些原因使得单机持久性内存文件系统难以满足日益增长的大规模数据存储需求,必须开发基于持久性内存的分布式文件系统.

分布式系统通常使用延迟较高的传统 TCP/IP 网络构建,无法充分利用持久性内存低延迟存储的特性[11].而远程直接内存访问(remote direct memory access, RDMA)的出现与应用为分布式系统的设计与实现提供了新的思路.近年来,RDMA 相关硬件的价格下降,为 RDMA 在数据中心的大规模应用提供了可能.RDMA 通信避免了传统网络内核网络协议栈、CPU 处理和内存数据冗余拷贝等开销,其高带宽、低延迟、零拷贝和内核旁路的特性受到了业界研究者们的青睐.目前,由迈络思(Mellanox)公司开发的最新一代支持 RDMA 的 infiniband 网卡 CX-6 (ConnectX-6)能够提供高达 200 Gbps 的网络带宽和低于 600 ns 的访问延迟[12].若使用传统的固态硬盘或者机械硬盘搭配 RDMA 网络实现分布式系统,RDMA 网卡读写存储介质需要额外通过 DRAM 进行中转,存储介质自身的高延迟同样使 RDMA 难以在分布式系统中充分发挥其高性能的特性.在 RDMA 与持久性内存结合的分布式系统中,持久性内存的低延迟存储与内存访问特性,使 RDMA 可以直接对存储介质进行高效的访问,从而很好地解决了上述问题[13-14].

多播技术同样在分布式系统中发挥着重要的作用.多播技术的应用范围很广,从多媒体数据的直播分发,到数据中心的分布式文件系统,均需要多播技术的支持.尤其是在数据中心中,数据传输通常是从源节点到 2 个甚至多个目标节点,文件数据需要被复制到多个存储服务器[15].这个过程造成的延迟往往占数据中心负载的主要部分,并最终决定了系统的总体 I/O 性能[16].

尽管多播技术在系统中起到重要的作用,但是在现有的基于 RDMA 的分布式文件系统中,往往没有提供多播传输的支持[13-14].在需要 1 对多传输的场景下,这些系统通常使用每次传输到 1 个节点的方法,将数据依次推送到所有的目标节点[17].为了解决该问题,康奈尔大学的 RDMC 基于 RDMA 的 1 对 1 传输开发了多播通信框架[18].但是,由于这种框架的底层依然是基于 1 对 1 的连接进行通信,当多播操作的目标节点较多时,需要占用大量的网卡资源,导致系统难以基于框架进行有效的扩展.另外,基于框架的编程较为复杂,不利于分布式系统的应用与维护.其他解决方案在获得可扩展性的同时,往往为传输过程引入了额外的复制与延迟,同样没有办法很好地解决分布式系统中的多拷贝文件数据传输问题[19].

本文提出了一种基于 RDMA 多播传输机制的分布式持久性内存文件系统(RDMA multicast transmission based distributed persistent memory file system, MTFS).通过使用 RDMA 多播通信语句,实现元数据节点与数据节点之间的 1 对多通信,提升数据传输效率.具体地,MTFS 实现了低延迟多播通信机制,基于 RDMA 多播通信语句搭建内核态 RDMA 通信模块,将文件系统中 1 对多的传输请求使用多播通信机制发送,并添加无通知机制与拥塞控制来优化 RDMA 传输,从而避免了传统 1 对 1 传输机制带来的冗余传输开销.为提升传输的灵活性,对分布式文件系统各项功能提供支持,MTFS 基于多播通信机制设计多模式远程过程调用(remote procedure call, RPC)框架,发送端将请求相关信息写入 RPC 头部,接收端通过接收处理程序解析相关信息,从而定位要执行的操作地址与操作类型,保证相应文件系统功能得到执行.为保证多播传输机制的一致性,MTFS 引入轻量级一致性保障机制,利用持久性内存字节寻址的特性实现错位的快速纠正,并支持在数据无法恢复时请求重传.节点发生故障时,MTFS 为元数据节点与数据节点均提供了故障恢复机制,保证文件数据的一致性与可靠性.

本文的主要贡献有 3 个方面:

1) 提出基于 RDMA 多播机制的分布式持久性内存文件系统 MTFS,实现了内核态 RDMA 通信模块,通过将文件系统中 1 对多请求使用 RDMA 多播语句发送,避免了额外的开销.

2) 提出基于多播传输机制的多模式 RPC 设计,提升数据传输的灵活性,为分布式文件系统各项功能提供支持.

3) 引入轻量级一致性保障机制,使用冗余校验机制保证数据传输过程的可靠性,利用持久性内存字节寻址的特性实现错位的快速纠正,并为系统中的各个节点提供故障恢复功能,从而保证数据的可靠性与一致性.

1 背景介绍

本节主要介绍了持久性内存技术与远程内存直接访问技术的基本特征,同时简要介绍与 MTFS 相关的 NOVA 文件系统实现细节.

1.1 持久性内存

持久性内存是一种新兴的硬件技术,主要包括相变存储器(phase-change memory, PCM)、忆阻器、自旋矩存储器(spin-torque transfer ram, STT-RAM)和 3D XPoint 等技术[1-4],其中基于 3D XPoint 的英特尔傲腾持久内存目前已经投入市场使用[5].

持久性内存的出现打破了内外存之间的界限,颠覆了传统的存储体系结构.一方面,持久性内存作为一种内存,拥有内存的种种特性.持久性内存可直接连接于高带宽的内存总线上,传输带宽和访问延迟均与 DRAM 相接近,并支持字节寻址访问.同时,与 DRAM 相比,持久性内存具有更高的存储密度和更低的能耗,这为搭建大规模内存文件系统提供了基础.另一方面,持久性内存作为一种持久性存储介质,与传统的硬盘和 SSD 相比,具有更高的带宽和更低的访问延迟.同时,由于 CPU 可以直接对持久性内存上的数据进行访问,数据可以绕过 DRAM,无需在内存和存储之间进行迁移,数据访问的整体性能得到了大幅的提升.

持久性内存为文件系统的设计提出了新的要求.基于持久性内存的文件系统可以直接通过 load/store 指令读写持久性内存.一些文件系统专为持久性内存进行特异性设计[6-9,20],另一些文件系统则基于现有文件系统进行改动[10,21],通过添加直接访问能力,允许应用程序绕过页缓存直接访问持久性内存,从而实现了文件系统对持久性内存的适配.

1.2 远程直接内存访问

近年来,RDMA 技术在业界受到越来越广泛的关注[11,22-24].RDMA 技术允许应用程序在不告知远端 CPU 情况下,绕过内核直接访问远端内存,实现零拷贝的数据传输,从而实现高带宽且低延迟的远端内存访问[25].目前,RDMA 传输性能已远优于现有的固态硬盘和机械硬盘的读写性能,如果仍使用固态硬盘或者机械硬盘搭配 RDMA 网络实现分布式文件系统,存储介质的高延迟将使系统无法充分发挥 RDMA 网络的性能优势,同时文件系统也无法绕过 DRAM 直接写入存储介质.而持久性内存低延迟内存访问的特性,使其能有效适配 RDMA 技术,实现高效的远程存储访问.

RDMA 主要提供单边语句和双边语句 2 种传输语句支持.表 1 展示了每种传输语句对应的类型.单边语句主要包括 READ 和 WRITE 语句,这些语句可以绕过远端节点的 CPU,直接对远端内存进行读写操作.此外,RDMA 单边语句还包括 compare_and_swap 和 fetch_and_add 等原子性语句,使 RDMA 能够对远端内存进行原子性访问.双边语句主要包括 SEND 和 RECV 语句,采用类似于 socket

编程的方式,发送端和接收端均需要 CPU 参与.在发送端进行 SEND 操作之前,接收端需要提前准备 1 个 RECV 请求并放入网卡,该请求中包含待接收数据的地址.

Table 1 Verbs Type of Each Transport Verb

表 1 每种传输语句对应的语句类型

语句	单边语句	双边语句
SEND		√
RECV		√
WRITE	√	
READ	√	
ATOMIC	√	
MULTICAST		√

RDMA 通过队列对(queue pair, QP)进行传输操作.每个队列对包括 1 个发送队列(send queue, SQ)和 1 个接收队列(receive queue, RQ).当进行传输时,使用 RDMA 的程序首先根据其传输的内容填充 1 个工作请求(work request, WR),并将其发布到发送队列上.RDMA 网卡会依次处理队列上的 WR,执行对应的传输操作.当传输完成时,网卡会在完成队列(completed queue, CQ)上发布 1 个工作完成(work completion, WC)信息,通知 CPU 进行相应处理.如果是双边操作,接收端在传输进行之前还需要提交 1 个 RECV WR 并放入其接收队列.

RDMA 包括有连接和无连接 2 种形式.有连接的传输提供 2 个 QP 之间的 1 对 1 通信,若需要与多个节点进行通信,则需要创建多个 QP 分别与多个节点进行 1 对 1 通信.而无连接的传输基于数据报实现,通信节点之间不需要创建连接,每个 QP 可以和多个 QP 进行通信.用户可以选择可靠或不可靠的 RDMA 传输类型.可靠的传输可以按照顺序交付信息,并在传输失败时返回错误信息.不可靠的传输则无法提供可靠性保证,但是其通过避免发送确认信息获取更高的性能.使用不可靠传输,RDMA 通过数据链路层提供的一致性保障机制仍可以在很大程度上保证数据传输的可靠性[22].

基于传输是否有连接与是否可靠,RDMA 提供了 3 种主要的传输方式:可靠连接(reliable connection, RC),不可靠连接(unreliable connection, UC)和不可靠数据报(unreliable datagram, UD).表 2 展示了每种传输方式可以支持的传输语句.可以看到,不同的传输方式所支持的传输语句不同,RDMA 单边操作只在有连接的传输方式下支持,而 RDMA 多播传输只在 UD 模式下支持,因此用户需要根据传输需求选择对应的传输方式.

Table 2 Verbs Supported by Each Transport Type

表 2 每种传输方式支持的传输语句

语句	RC	UC	UD
SEND	√	√	√
RECV	√	√	√
WRITE	√	√	
READ	√		
ATOMIC	√		
MULTICAST			√

RDMA 提供了多播语句支持[26].多播语句是 UD 模式下双边语句的一种特殊形式.用户使用多播语句进行 RDMA 通信时,首先将所有需要通信的节点加入同一个多播组.发送信息时,目标地址设定为多播组的地址,发送端仅需要 1 次发送操作,发送成功后,所发送的信息通过交换机被分发到多播组中的各个节点.多播语句为 1 对多的传输场景提供了合适的解决方案,降低了多节点数据传输的开销,为解决基于 RDMA 的分布式系统中的多拷贝文件数据传输问题提供了有效的支持.

1.3 NOVA 文件系统

MTFS 是基于 NOVA 实现的.NOVA 是加州大学圣地亚哥分校开发的一种持久性内存文件系统[6].为更好地利用持久性内存的诸多优秀特性,NOVA 做了许多特异性的设计,使其在保证一致性的基础上提升文件系统的性能.本节讨论与 MTFS 相关的一些 NOVA 设计.

NOVA 为每个索引节点维护 1 个单独的日志链表,每块日志中存储 1 次写入的基本信息与指向写入数据页的指针,同时在 DRAM 中维护基数树索引以加速对文件数据的查找.写入操作使用写时复制机制实现,每次写入时会申请新的日志块与数据页,在日志中记录操作相关信息与指向新写入数据的指针.当数据成功写入持久性存储介质后,NOVA 更新文件日志的尾指针以及其对应的基数树索引.当读取数据时,NOVA 通过基数树索引找到对应的日志块,从日志中读取数据地址,并通过地址找到数据页并读取数据.

NOVA 使用可利用空间表管理持久性内存空间.NOVA 将可用的数据空间均分给每个 CPU 进行管理以提升并发文件访问的性能,每个 CPU 使用红黑树结构管理数据空间中的空闲块,以提升连续

数据块查找的性能.通过这种方式,NOVA 提升了持久性内存空间分配操作的并行性,减少了空间分配的争用.

NOVA 提供了故障恢复机制.当系统从故障中恢复时,首先,NOVA 需要检查崩溃前写入的日志,通过日志将未提交的事务回滚.然后,NOVA 并行扫描每个索引节点,通过日志链表恢复数据组织结构.通过日志设计,NOVA 保证系统可以从故障中恢复数据.

2 MTFS 设计

本节将详细介绍 MTFS 的系统设计.首先整体描述 MTFS 的系统架构,然后分别对 MTFS 中的低延迟多播通信机制、多模式多播 RPC 机制和轻量级一致性保障机制进行介绍.

2.1 系统架构

图 1 展示了 MTFS 的系统整体架构.系统由 1 个元数据节点和多个数据节点组成.其中元数据节点存储系统的元数据信息,包括文件的元数据信息和系统的基本配置与空间管理信息,数据节点中仅存储文件的数据.元数据节点与数据节点之间通过 RDMA 网络互连.MTFS 采用主从式架构,将数据存储在多个数据节点上,提升了数据访问的并行性.

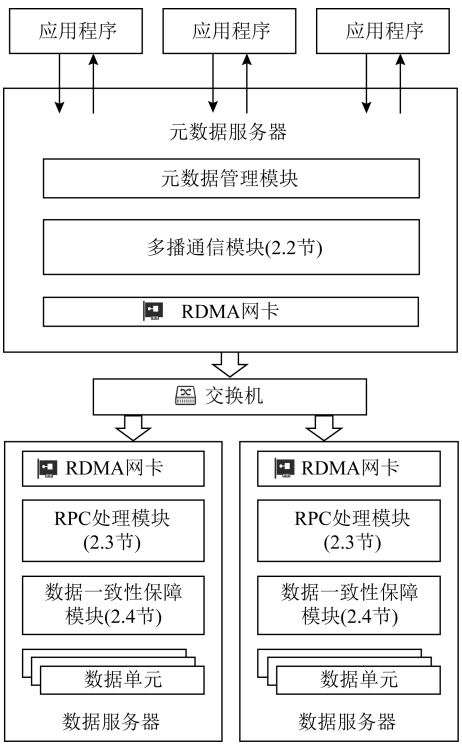

Fig. 1 Overall architecture of MTFS

图 1 MTFS 系统整体架构

应用程序通过可移植操作系统接口(portable operating system interface of UNIX, POSIX)对文件系统进行访问.以数据写入为例,当应用程序发起数据写入请求时,MTFS 通过访问元数据节点在各目标数据节点分配持久性内存空间,然后将数据写入到各数据节点中.待写入的数据通过多播通信模块(2.2 节)以 RDMA 数据报的形式由网卡发出.网络交换机收到多播数据报时,会进行分发操作,将数据报发送到多播组中的每个数据节点.数据节点通过 RPC 处理模块(2.3 节)识别数据报请求体,并通过数据一致性保障模块(2.4 节)将数据持久化到持久性内存.与此同时,元数据节点会提交该次数据写入操作并返回用户.文件系统的元数据访问则仅通过元数据节点进行,无需对数据节点进行远程访问.

2.2 低延迟多播通信机制

在分布式文件系统中,节点间的网络传输的开销非常高昂.数据节点间等待传输完成与确认需要耗费大量的时间,从而产生较高的延迟.而在 1 对多的分发场景下,传统的文件系统会发起多次网络传输请求,将相同的数据逐一发送到各个节点,增加了网卡的负载(图 2(a)).多播通信机制旨在将多个传输相同数据的请求合并为多播请求,避免网卡数据重复发送的冗余开销,从而大幅提升发送数据的效率.

当 MTFS 的元数据节点发起写请求时,数据将被同时写入所有的目标数据节点:首先多播传输模块会申请 1 块发送结构体,并根据写请求对应的数据地址、数据大小等元数据信息填写结构体的头部字段,同时将数据放入发送结构体的数据字段.然后多播传输模块会将 RDMA 传输信息添加到填写完毕的发送结构体中,并将其打包成工作请求放入发送队列进行传输;网卡依次对工作请求进行处理,将数据以 RDMA 数据报的形式发送到各个目标数据节点,由数据节点中的接收处理程序进行处理并写入持久内存.当发送操作完成之后,元数据节点会触发中断通知发送完成处理程序,将工作完成从完成队列中移出,从中获取已完成的发送结构体地址并将对应空间释放回收.执行过程如图 3 所示.

为提升系统可扩展性,MTFS 将数据节点划分为存储单元进行管理.MTFS 将数据节点每 2～3 个一组划分为存储单元,单元内的数据节点存储相同的数据,并和 MTFS 的元数据节点加入同一个多播组.多播信息仅在组内传递,由于不同存储单元间存储的数据互不相同,存储单元之间不需要进行多播

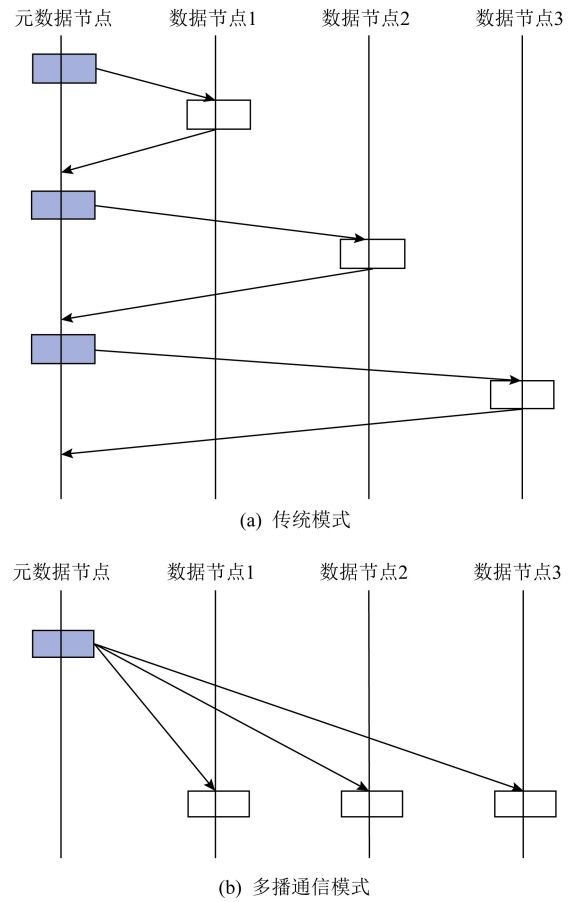

Fig. 2 Comparison of traditional and multicast schemes

图 2 传统模式与多播通信机制对比

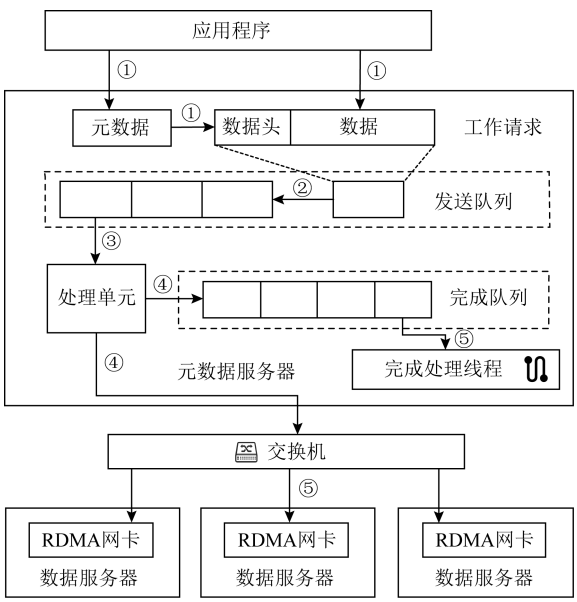

Fig. 3 Multicast transmission execution

图 3 多播通信运行过程

通信.当元数据节点发起写入请求时,MTFS 会选择剩余空间最多的存储单元,并将单元内的数据节点作为存储的目标数据节点;当元数据节点发起读取请求时,MTFS 会根据对应存储单元内各数据节点正在请求的线程数量,从中选择负载最低的数据节点读取数据.通过存储单元管理,MTFS 避免了数据冗余存储,提升了系统的可扩展性.

为减少元数据节点 CPU 的开销,MTFS 通过 RDMA 无通知机制优化多播发送流程.由于 RDMA 的多播能力由 UD 模式提供,而 UD 模式基于无连接数据报实现,不需要处理接收端的确认信息.因此,当多播请求发送完毕后,多播传输模块不会立即通知 CPU 进行处理,而是将工作完成暂存在完成队列上.当完成一定数量的工作请求之后,为了防止工作完成的堆积,下一次发送的工作请求会被设置为完成后通知.当该工作请求发送完成后,网卡会触发 CPU 中断并转入发送完成处理程序,对先前放入的工作完成进行批量处理.通过无通知发送的优化,MTFS 减少了 CPU 中断处理的次数,极大地降低了 CPU 的负担.

为了避免网络堵塞引起的发送队列拥挤,MTFS 实现了拥塞控制系统.RDMA 的发送队列长度固定,当网卡处理速率小于工作请求的增加速率时,发送队列会被填满,导致之后的工作请求无法放入发送队列.MTFS 多播传输模块会对发送队列中待发送的多播请求数目进行实时统计,并据此控制 RDMA 多播请求的发送速率.当网络拥塞时,发送队列中待发送的请求数目超过了预先设定的阈值.此时发送队列将暂缓接受工作请求直到待发送的请求数量低于阈值,从而避免了发送队列溢出造成的传输问题.

通过上述多播通信机制,MTFS 减少了文件操作过程中 RDMA 的通信次数,并充分利用 UD 模式数据报通信的优势降低了数据的传输延迟(图 2(b)).尽管 RDMA 的 UD 模式具有一定的可靠性[22],MTFS 依然通过发送端的拥塞控制机制与接收端的一致性保障机制(2.4 节),在不显著增加延迟的情况下尽可能地保证数据的可靠性与一致性.

2.3 多模式多播 RPC 机制

MTFS 使用 RPC 实现元数据节点与数据节点间通信.如图 4 所示,RPC 采用服务端主动的方式,元数据节点通过多播通信机制将 RPC 请求分发到数据节点的网卡上,并由网卡放入接收队列等待处理.接收端处理程序按到达顺序依次处理接收队列中的 RPC 请求,对参数进行解析并执行相应的操作.

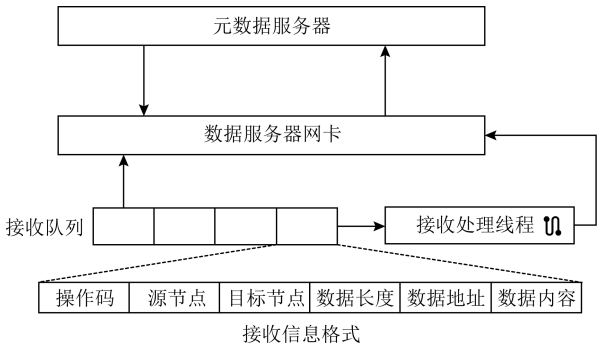

Fig. 4 RPC module design

图 4 RPC 模块设计

RPC 通过源节点字段和目标节点字段进行请求识别,从而判断该节点是否需要进行处理.由于 RPC 采用多播通信机制实现,多播组中的所有数据节点均可以接收到元数据节点的请求.为增强 RPC 的灵活性,请求头部标识了源节点与目标节点,数据节点收到请求后首先判断该节点是否需要执行操作,从而避免冗余的请求对数据节点资源的占用.

RPC 通过优化返回机制降低执行延迟.接收端根据其接收到的操作码对请求进行分类:对于不需要返回信息的请求如数据写,数据节点会直接执行相应操作,不发送返回信息;对于时效性要求较高的请求如数据读,数据节点会立即进行处理并返回完成信息;对于时效性要求较低的请求如数据迁移,元数据节点采用异步处理机制,发送 RPC 请求之后继续执行其他操作,不阻塞等待结果,当收到返回信息后再进行结果处理操作.而数据节点收到请求之后,会在优先处理其他请求之后进行处理并返回完成信息.通过 RPC 多模式分类,MTFS 减少了部分操作的传输次数,并将部分传输操作从关键路径移除,有效提升了文件操作的整体效率.

考虑到 RPC 基于多播通信机制,MTFS 的写请求使用 RPC 实现,保证写请求通过多播语句 1 对多地发送到所有的数据节点.但鉴于读请求针对单一客户端的特性,MTFS 在实现了基于 RPC 的读方法的同时,采用基于 RC 模式的 RDMA 读操作实现了对文件数据的读取操作,避免了集群规模过大时使用多播通信机制读数据造成的额外开销.用户挂载文件系统时,可以根据集群配置选择合适的读方法.

2.4 轻量级一致性保障机制

MTFS 开发了故障恢复机制以有效应对各节点上可能发生的系统崩溃.MTFS 检测到数据节点崩溃后,会将该数据节点标记成为故障节点,读请求将被分流到其他数据节点执行,不会影响系统正常运行.该数据节点中的数据可以通过元数据节点与其他数据节点进行恢复.当元数据节点崩溃时,系统停止提供服务,等待元数据节点重启,并通过文件的元数据与数据日志,将系统恢复到崩溃前的状态.通过故障恢复机制,MTFS 保障了数据的高可靠性.

MTFS 使用冗余校验机制保证了数据传输的容错性(图 5).发送端打包传输数据时,会将数据分为 2 部分并分别计算循环冗余校验(cyclic redundancy check 32, CRC32),校验结果存入发送请求的固定区域.同时,MTFS 对 2 部分数据计算奇偶校验结果并放入发送结构体.接收处理程序收到数据后会首先计算 2 部分数据的 CRC32 校验和,与收到数据中存储的校验和进行比对.若 2 份数据校验和与存储的校验和均相同,则说明本次数据传输没有发生错误;若仅有 1 份数据校验和与存储的校验和相同,说明另 1 份数据发生了传输错误.接收处理程序会通过奇偶校验结果查找错误发生的位置,计算出正确的结果并写入;若 2 份数据校验和与存储的校验和均不相同,则该次传输可能出现了大面积无法恢复的错误,接收处理程序会激活重传机制,发送 1 个重传请求到发送端,请求重新发送该数据.通过冗余校验机制,MTFS 在不显著影响性能的情况下保证了数据传输的正确性.

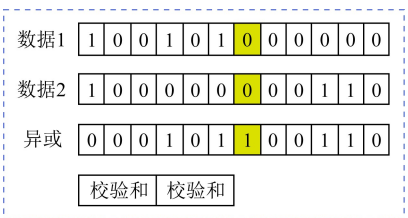

Fig. 5 Data verification mechanism

图 5 数据冗余校验机制

MTFS 使用重传机制保证文件数据的持久性.发送端申请工作请求时会指定全局唯一的 WR_ID,并附在 RDMA 发送的立即数字段一同发送.接收处理程序会记录来自同一个节点的最近 1 个请求的 WR_ID,并在新的请求到达时读取立即数字段进行比较.若新接收的请求出现乱序,接收处理程序会发送重传请求到发送端,要求重传缺失的工作请求.对于传输数据出现大面积错误的情况,接收处理程序会通过最新记录的 WR_ID 计算出错误的 WR_ID 并发送重传请求到发送端.发送端收到重传请求时,会通过 WR_ID 查找相应的发送数据进行重新打包发送.

MTFS 通过窗口确认机制保证了数据一致性.元数据节点记录了每个索引节点最后 1 次写入操作的 WR_ID.各数据节点每隔一段时间会向元数据节点报告 TAIL_WR_ID(即节点已成功接收在该 WR_ID 之前的所有请求).当元数据节点发起读请求时,会检查索引节点最后 1 次写入的 WR_ID 和目标节点报告的 TAIL_WR_ID,确认目标节点是否已经收到该索引节点的所有写请求.若未收到,则说明还有写请求正处于传输的过程中,系统会对读请求进行阻塞,直到数据节点确认已收到所有的写请求;若已收到,则元数据节点可以执行读操作.通过窗口确认机制,MTFS 避免了文件读写出现不一致,保证了数据一致性.

MTFS 避免了 RDMA 网卡的数据一致性问题.当使用 RDMA 网卡对远端地址进行数据写入时,数据可能会暂时驻留在 RDMA 网卡的易失缓存中,由网卡决定写入内存地址的时机,这可能导致数据不一致的问题[27].MTFS 采用多播发送语句,远端网卡接收到数据之后,会通知接收处理程序将数据写入持久性介质,避免数据在 RDMA 网卡缓存中驻留,保证数据的一致性.

3 系统实现

本节主要讲述了 MTFS 各项设计的实现方式与细节.虽然本文在 NOVA 的基础上实现 MTFS 的各项功能设计,但 MTFS 的设计可以基于现有的单机持久性内存文件系统实现,不局限于某个系统的设计.

3.1 文件系统相关实现

MTFS 基于 NOVA 的元数据组织实现元数据节点组织,并沿用了 NOVA 的元数据相关设计.由于 MTFS 元数据节点与数据节点分离的架构,MTFS 优化了数据日志的格式,将各目标数据节点中对应的数据地址存入日志之中.同时,MTFS 对所有的数据节点进行统一编址,使用 NOVA 的可利用空间表数据结构进行管理.每个 CPU 在每个数据节点上对应 1 个可利用空间表,并通过可利用空间表管理 1 块持久性内存空间.当执行数据写入操作时,元数据节点会从每个目标数据节点对应的区域中各分配 1 段空间,并通过 RPC 将数据远程写入.当执行数据读取操作时,系统会按照日志块中记录的远端数据的地址从远端数据节点读出数据.

3.2 RDMA 通信模块

MTFS 实现了高效的内核态 RDMA 通信模块.为了提升通用性,更好地与主流的基于 POSIX 接口的内核态文件系统适配,MTFS 在内核态实现了 RDMA 通信.同时 MTFS 将 RDMA 服务以模块的形式呈现,封装 RDMA 的相关操作,将复杂的 RDMA 操作抽象为通信建立、通信关闭和操作执行等若干类函数,对文件系统屏蔽了 RDMA 通信的相关细节,方便相关代码的移植与维护.

为了方便管理 RDMA,MTFS 使用 RDMA_CM 库管理通信.RDMA_CM 可以在 RDMA 传输建立之前,使用基于 RDMA 网络的 TCP 传输在各节点间传递 RDMA 网络参数,并据此建立 RDMA 传输.RDMA_CM 将 RDMA 通信建立流程封装成为类似于 socket 的接口,并在整个通信阶段对通信进行管理.建立多播通信时,需首先对约定好的多播地址进行解析,当确认解析无误后将解析到的设备绑定到 CM_ID,然后各节点加入该多播组并注册到子网管理器,从而完成多播通信的建立.对于多播传输请求,所有加入该多播组的节点均可以接收.通过使用 RDMA_CM 库,MTFS 简化了 RDMA 编程操作.

MTFS 实现了 RDMA 发送与接收请求的统一管理,从而支持了 RDMA 多播传输的各项优化.为了方便对网络请求进行统一管理,MTFS 在各节点上分别为 RDMA 发送与接收请求维护全局的工作请求链表.正常情况下,当有新的请求需要发送时,MTFS 申请 1 个新的工作请求,相关信息填写完毕后,会将工作请求放入链表,并生成全局唯一的 WR_ID,WR_ID 在发送时被放入数据报一起发送.当发送完成时,发送完成处理程序通过工作完成找到 WR_ID,进而找到该次工作请求并回收相关数据结构.接收请求的情况与之类似.当无通知优化功能被启用时,MTFS 会记录发送请求的数量,工作请求通知标志的默认值为 0,当一定数量的数据发送完成后,下一个工作请求的通知标志会被置为 signaled,发送完成之后通知处理程序对所有早先产生的工作完成统一进行处理.启用拥塞控制时,MTFS 使用原子变量记录发送队列上的待发送工作请求数量,通过该原子变量判断是否暂缓接受发送请求.通过对 RDMA 请求的统一管理,MTFS 实现了请求的添加、存储、删除与查找,为 RDMA 多播传输的各项优化提供了有效的支持.

3.3 RPC 与一致性保障实现

MTFS 为 RPC 传输设计了发送结构体.结构体中包含了该次操作的详细信息,包括操作码、源节点与目标节点、数据长度、数据地址和数据内容等,

整个结构体经过冗余和校验处理之后,会统一作为RDMA传输的数据进行发送.而工作请求的控制信息会根据请求类型和待发送数据进行填写,其中立即数字段填写全局唯一的WR_ID,填写完毕之后放入发送队列进行发送.接收端收到RPC请求之后,会读取数据并对各字段进行解析,执行相应操作.

4 实 验

本节将MTFS与其他文件系统对比,对各项性能进行评估.首先介绍实验环境配置,然后从微观测试、Filebench测试、Redis测试3个方面对比MTFS与现有的分布式和单机持久性内存文件系统,详细比较并分析相关性能差异.

4.1 实验配置

实验所使用的实验平台环境配置信息如表3所示.各节点均部署4块Intel Optane DC 128 GB持久性内存,并采用APP-Direct模式.各节点通过一块Mellanox ConnectX-5 RDMA网卡与其他节点通信,网卡配置为Infiniband模式运行,并连接到Infiniband交换机.

Table 3 Platform Configuration

表3 实验平台配置

实验参数	配置
CPU	Intel® Xeon® Gold 6240×2
内存	32 GB×8
持久性内存	Intel Optane DC Persistent Memory 128 GB×4
操作系统	Ubuntu18.04, Linux 4.13
网络	Mellanox ConnectX-5

MTFS使用NOVA的代码作为基础,在Linux 4.13内核上实现.MTFS基于Mellanox OFED实现了内核态RDMA通信模块,并整合入文件系统.MTFS最大可支持同一广播域内的所有节点组成的集群,本实验中MTFS默认在3个节点组成的集群上部署,包括1个元数据节点与2个数据节点,其中2个数据节点属于同一个存储单元.各节点采用128 GB持久性内存作为存储介质.

本实验主要将MTFS与同样运行在持久性内存与RDMA上的分布式文件系统GlusterFS进行比较[28],对各项性能进行评估.鉴于MTFS基于单机文件系统NOVA实现,本实验同样将与NOVA进行性能比较.由于NOVA部署在单台机器上,公平起见,MTFS在实验中数据吞吐量均表示集群中单个节点的吞吐量.

4.2 微观测试

实验使用Fio进行微观测试,展示MTFS在1对多的场景下进行写操作的吞吐量[29].在测试过程中采用随机读写,分别通过I/O大小变化和文件大小变化,测试Fio的总体吞吐量.

图6展示了MTFS与其他对比系统在不同的I/O大小下的性能比较,其中文件大小采用128 KB.可以得知,MTFS与NOVA写操作吞吐量均较高,相差在5%以内;而GlusterFS的性能最低,与MTFS相比在不同的I/O大小下有12～15.6倍的差距.这主要是因为多播传输有效降低了1对多传输过程的开销,从而提升节点吞吐量.GlusterFS在写入时,客户端会首先计算出数据对应的多个brick的位置,然后通过多次RDMA操作将数据分别写入各个brick对应的数据节点[28],而MTFS写入时仅需1次RDMA多播操作,即可将数据写入所有数据节点,大幅降低写入操作延迟;同时GlusterFS并没有针对持久性内存与RDMA进行特异性设计,难以充分利用硬件性能优势.因此MTFS性能大幅优于GlusterFS.考虑到MTFS各数据节点互为备份并提供了一致性保障机制,保证数据持久化操作得到执行,因此系统将一致性保障与持久化操作放到远端数据节点执行,元数据节点在提交RDMA多播操作后,不需要等待完成信息确认数据写入,只需承担RDMA网络发送延迟开销.元数据节点的其他操作与NOVA相似.因此尽管MTFS将数据保存了2份备份,与单机文件系统NOVA相比仍取得相近的性能表现.

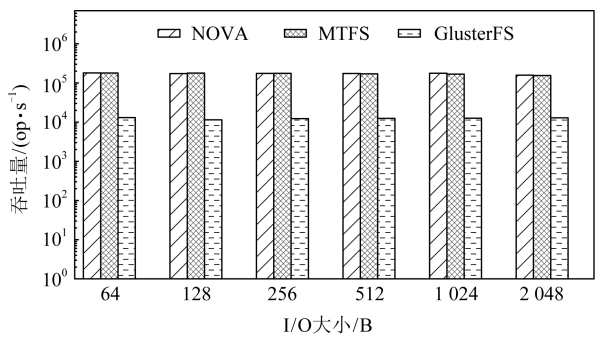

Fig. 6 Throughput of write syscall with varying I/O sizes

图6 写系统调用在不同I/O大小下的吞吐量

图7展示了MTFS与其他对比系统在不同的文件大小下的写系统调用吞吐量,其中I/O大小为2 KB.从图7中可以看到,与GlusterFS相比,在文件

大小为 2 KB 时,MTFS 性能相比 GlusterFS 提升了 145 倍,而在文件大小为 2 GB 时,MTFS 对比 GlusterFS 仍然有 10.2 倍的提升.同时,随着文件大小增长,MTFS 和 NOVA 性能均未出现明显下降.这是因为 MTFS 与 NOVA 虽然采用日志结构,但相应的在 DRAM 中维护了作为数据索引的基数树,当文件大小增长时,文件数据访问延迟并不会出现明显增大.

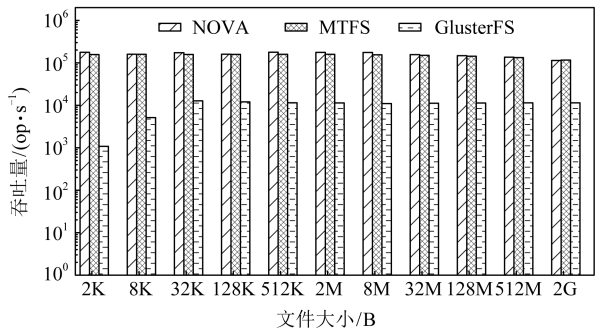

Fig. 7 Throughput of write syscall with varying file sizes

图 7 写系统调用在不同文件大小下的吞吐量

为了检验 MTFS 在集群中的可扩展性,实验测试了 MTFS 在不同规模集群中的性能表现,结果如图 8 所示.可以得知,随着集群中节点数量增加,在不同的 I/O 大小下,MTFS 吞吐量都能够基本保持不变,展现出良好的可扩展性.

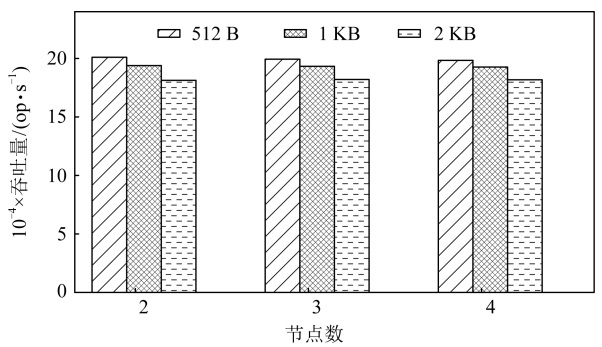

Fig. 8 Throughput of write syscall in clusters with varying sizes

图 8 写系统调用在不同规模集群中的吞吐量

4.3 Filebench

MTFS 使用 Filebench[30] 测试评估真实负载下的表现.实验选择 Filebench 的工作负载 Randomwrite 来评估 MTFS 的性能.实验采用以下参数进行测试:负载平均文件大小 2 KB,数据操作的平均 I/O 大小从 64 B~2 KB 不等.

图 9 展示了各文件系统在 Filebench 的 Randomwrite 工作负载上的性能表现.可以得知,MTFS 与 NOVA 性能相接近,在 I/O 大小为 2 KB 时,MTFS 相比 NOVA 性能有 11%的提升.主要原因是当 I/O 大小增大时,NOVA 进行持久化相关操作的延迟随之增大,而 MTFS 将持久化相关操作放在远端数据节点进行处理,从关键路径移除,减小了数据操作的延迟.与 GlusterFS 相比,MTFS 性能仍然有 13.7~219 倍的提升.

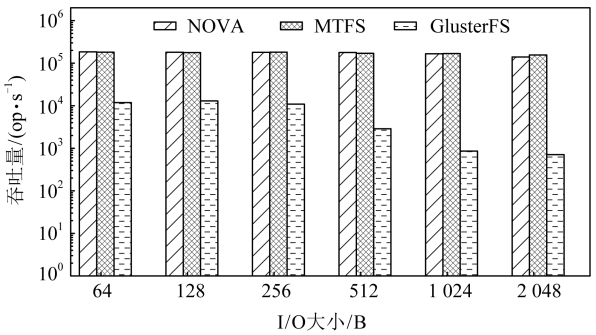

Fig. 9 Throughput of Randomwrite with varying I/O sizes

图 9 Randomwrite 在不同 I/O 大小下的吞吐量

4.4 Redis

为进一步测试 MTFS 在真实负载环境下的表现,实验采用 Redis[31] 作为负载测试各文件系统的表现.Redis 是一个高性能的 key-value 数据库,在企业级应用中被广泛使用.实验采用 Redis 的 AOF 机制实现数据持久化,持久化策略使用每修改同步策略,数据大小从默认 2 B~2 KB 不等.实验比较 Redis 在不同文件系统下的每秒请求数.

图 10 展示了各文件系统运行 Redis 时的性能表现.与上述微观测试和 Filebench 测试采用 I/O 密集型负载不同,Redis 包含大量的计算操作,这导致 GlusterFS 数据操作上的性能缺陷在一定程度上被掩盖,但 MTFS 与 GlusterFS 相比仍取得了 25.5~

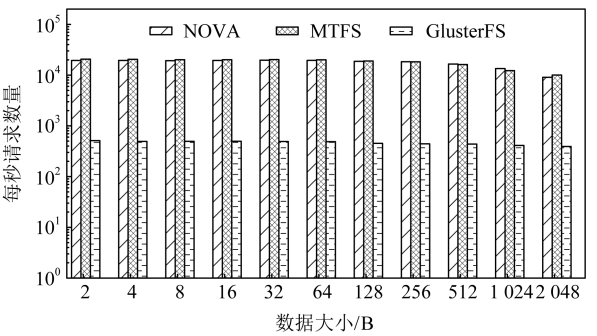

Fig. 10 Performance of Redis with varying data sizes

图 10 Redis 在不同数据大小下的性能

42.1 倍的性能提升.与 NOVA 相比,MTFS 在 Redis 负载上取得了更优异的性能表现,主要原因是 Redis 操作中 I/O 密集度下降减轻了网卡的负载,使得 RDMA 多播操作发送延迟下降,进而提升了数据传输效率.

实验在 Redis 负载上对各文件系统进行了线程扩展能力测试,结果如图 11 所示.可以看到,MTFS 随线程数量增加,表现出良好的线程可扩展性.

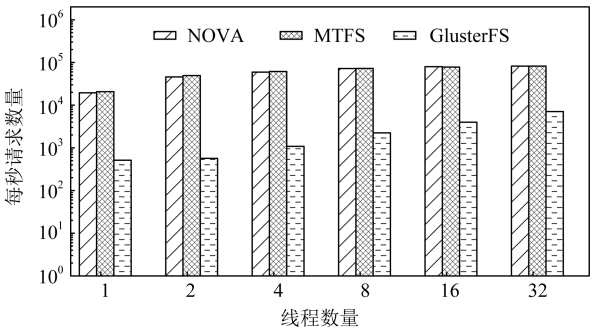

Fig. 11 Performance of Redis with varying thread numbers

图 11 Redis 在不同线程数下的性能

5 相关工作

新兴的持久性内存技术与 RDMA 技术分别为本地内存访问与远程网络通信访问提供了优化的可能,现有的一些工作基于这 2 种技术提出了针对性设计.

5.1 RDMA 相关优化

文献[11]提出了一个基于 RDMA 的分布式共享内存系统 FaRM.FaRM 使用单边 RDMA 语句实现了无锁读取和 RPC 操作,避免了双边 RDMA 操作带来的通信延迟与 CPU 开销.与基于 TCP 的系统相比,它的设计使其在 RDMA 网络上获得了巨大的性能提升.但是,FaRM 使用了 RC 模式的单边 RDMA 操作,对于每组连接都需要创建至少 1 对 QP,而过多的 QP 会带来 RDMA 网卡缓存不足和处理器争用的问题,这限制了系统在大规模集群中的表现.MTFS 基于多播语句实现,每对 QP 都可以发送信息到所有的节点,从而大幅减少了对 RDMA 网卡相关资源的需求.

文献[22]通过实验证明了在 UD 模式中,RDMA 传输依然具有非常高的可靠性,并基于该结论提出了 FaSST.FaSST 使用 UD 模式的双边 RDMA 操作开发了 RPC 系统,利用 UD 模式数据报传输的优势,通过降低发送端延迟提高系统性能,并取得了良好的可扩展性.然而 FaSST 只能通过 QP 进行 1 对 1 数据发送,当需要将相同的数据传输到不同的目的地时,需要进行多次发送操作.尽管 MTFS 同样基于 UD 模式的双边 RDMA 操作,但是在上述情况下,MTFS 使用多播语句可以将发送操作降低到 1 次,从而有效降低了发送端的延迟与资源占用.

文献[23]提出了一种基于混合 RDMA 语句的分布式事务系统 DrTM+H.该系统使用了乐观并发控制实现事务,按照乐观并发控制中的特性比较不同 RDMA 语句的表现,并在事务执行的每个阶段选择最适合的语句进行操作,取得了良好的性能表现.MTFS 也使用了混合 RDMA 语句,选择了多种 RDMA 语句分别进行读取和写入操作,但是与 DrTM+H 相比,MTFS 将元数据操作放在本地执行,仅将数据放在远端存储,通过减少 RDMA 操作次数降低了事务执行延迟.

5.2 基于持久性内存的分布式文件系统

文献[17]提出一个分布式文件系统 NVFS,在现有的 HDFS 基础上进行了改进,使系统可以更好地利用持久性内存技术和 RDMA 技术的特性.尽管 HDFS 表现良好,但 HDFS 设计的复杂性增大了改进的难度,导致 NVFS 无法充分利用硬件的性能.而其他类似的工作只是简单地用 RDMA 替换了现有系统中的网络,同样无法充分地利用新技术的特性[28].

文献[13]同样将持久性内存与 RDMA 功能进行了紧密结合,开发出分布式文件系统 Octopus.Octopus 基于 RDMA 的 write_with_imm 语句构建 RPC,该语句可以在进行单边写操作的同时携带 32 b 立即数字段,通过在该字段中编码 RPC 相关信息,为接收端的相关处理提供了一定的灵活性与便利.但是在该语句中,当远端节点的网卡接收到立即数字段时,需要触发中断来通知 CPU 处理立即字段,从而失去了 RDMA 单边写操作无需 CPU 处理的优势,而仅 32 b 的字段大小也无法充分实现灵活性.为了满足灵活性的需求,MTFS 采用了 UD 模式的双边操作语句,并设计了大小可调整的头部字段来实现 RPC 操作.

上述系统虽然均表现良好,但由于缺少对 RDMA 多播通信支持,导致难以解决分布式系统中的多拷贝数据传输问题,造成一定的性能损失.MTFS 充分利用 RDMA 多播通信能力,通过多播传输解决了多拷贝数据传输问题,有效提升了系统数据传输效率.

6 总 结

持久性内存与 RDMA 技术的出现,为分布式系统的设计提供了新的思路.现有的基于 RDMA 的分布式系统未能充分利用 RDMA 的多播能力,难以解决多拷贝文件数据的传输问题.本文提出一种基于 RDMA 多播机制的分布式持久性内存文件系统 MTFS,通过低延迟多播通信机制将数据高效传输到多个数据节点,从而避免了多拷贝传输操作.为提升传输操作的灵活性,MTFS 设计了多模式多播 RPC 机制,并通过优化返回机制将部分操作移出关键路径,进一步降低传输延迟.同时 MTFS 提供了轻量级一致性保障机制,保证了数据的可靠性和一致性.实验结果表明,MTFS 在各测试集上性能比 GlusterFS 高 10.2～219 倍,并在 Redis 负载上相比于 NOVA 取得了最高 10.7%的性能提升.MTFS 在大规模数据存储场景中有着广阔的应用前景.

参 考 文 献

[1] Zhang Hongbin, Fan Jie, Shu Jiwu, et al. Summary of storage system and technology based on phase change memory [J]. Journal of Computer Research and Development, 2014, 51(8): 1647-1662 (in Chinese)
(张鸿斌, 范捷, 舒继武, 等. 基于相变存储器的存储系统与技术综述[J]. 计算机研究与发展, 2014, 51(8): 1647-1662)

[2] Shu Jiwu, Lu Youyou, Zhang Jiacheng, et al. Research progress on non-volatile memory based storage system [J]. Science & Technology Review, 2016, 34(14): 86-94 (in Chinese)
(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)

[3] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology [C] //Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 24-33

[4] Kültürsay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative [C] //Proc of 2013 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2013: 256-267

[5] Intel. Optane DC persistent memory [OL]. 2019 [2020-05-24]. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[6] Xu Jian, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories [C] //Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 323-338

[7] Dong Mingkai, Chen Haibo. Soft updates made simple and fast on non-volatile memory [C] //Proc of 2017 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 719-731

[8] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: 1-15

[9] Zheng Shengan, Huang Linpeng, Liu Hao, et al. Hmvfs: A hybrid memory versioning file system [C] //Proc of the 32nd Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2016: 1-14

[10] Wilcox M. Add support for NV-DIMMs to ext4 [OL]. 2014 [2020-04-10]. https://lwn.net/Articles/613384

[11] Dragojević A, Narayanan D, Castro M, et al. FaRM: Fast remote memory [C] //Proc of the 11th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 401-414

[12] Mellanox. ConnectX®-6 VPI card 200Gb/s InfiniBand & Ethernet adapter card [EB/OL]. 2019 [2020-05-25]. https://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-6_VPI_Card.pdf

[13] Lu Youyou, Shu Jiwu, Chen Youmin, et al. Octopus: An RDMA-enabled distributed persistent memory file system [C] //Proc of 2017 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 773-785

[14] Yang Jian, Izraelevitz J, Swanson S. Orion: A distributed file system for non-volatile main memory and rdma-capable networks [C] //Proc of the 17th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 221-234

[15] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[16] Venkataraman S, Panda A, Ousterhout K, et al. Drizzle: Fast and adaptable stream processing at scale [C] //Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 374-389

[17] Islam N S, Wasi-ur-Rahman M, Lu Xiaoyi, et al. High performance design for HDFS with byte-addressability of NVM and RDMA [C] //Proc of 2016 Int Conf on Supercomputing. New York: ACM, 2016: 1-14

[18] Behrens J, Jha S, Birman K, et al. RDMC: A reliable RDMA multicast for large objects [C] //Proc of the 48th Annual IEEE/IFIP Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2018: 71-82

[19] Jha S, Behrens J, Gkountouvas T, et al. Building smart memories and cloud services with Derecho [R/OL]. New York: Cornell University, 2017 [2020-02-04]. https://pdfs.semanticscholar.org/b304/f0aaff45cabe3e328f560bd0591e8193d601.pdf

[20] Zheng Shengan, Hoseinzadeh M, Swanson S. Ziggurat: A tiered file system for non-volatile main memories and disks [C] //Proc of the 17th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 207-219

[21] Silicon Graphics. XFS: Ahigh-performance journaling filesystem [OL]. 2013 [2020-01-15]. https://xfs.org/

[22] Kalia A, Kaminsky M, Andersen D G. FaSST: Fast, scalable and simple distributed transactions with two-sided RDMA datagram RPCs [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 185-201

[23] Wei Xingda, Dong Zhiyuan, Chen Rong, et al. Deconstructing RDMA-enabled distributed transactions: Hybrid isbetter![C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 233-251

[24] Zhang Yiying, Yang Jian, Memaripour A, et al. Mojim: A reliable and highly-available non-volatile memory system [C] //Proc of the 20th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 3-18

[25] Dunning D, Regnier G, McAlpine G, et al. The virtual interface architecture[J]. IEEE Micro, 1998, 18(2): 66-76

[26] Mellanox. Mellanox OFED for linux user manual [OL]. 2018 [2020-01-08]. https://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v4_4.pdf

[27] Kim D, Memaripour A, Badam A, et al. Hyperloop: Group-based NIC-offloading to accelerate replicated transactions in multi-tenant storage systems [C] //Proc of 2018 Conf of the ACM Special Interest Group on Data Communication. New York: ACM, 2018: 297-312

[28] Gluster Inc. GlusterFS RDMA transport [OL]. 2014 [2019-12-15]. https://gluster.readthedocs.io/en/latest/Administrator%20Guide/RDMA%20Transport/

[29] Jens Axboe. Fio: Flexible I/O tester [OL]. 2014 [2020-01-15]. http://freecode.com/projects/fio

[30] File System and Storage Lab in Stony Brook University. Filebench [OL]. 2019 [2020-02-21]. https://github.com/filebench/filebench

[31] Salvatore Sanfilippo. Redis [OL]. 2018 [2020-03-04]. https://redis.io/

Chen Maotang, born in 1996. Master candidate. His main research interests include remote direct memory access, persistent memory and file systems.

陈茂棠,1996 年生.硕士研究生.主要研究方向为 RDMA、持久性内存和文件系统.

Zheng Shengan, born in 1991. Postdoctoral researcher. His main research interests include persistent memory and file systems.

郑圣安,1991 年生.博士后研究员.主要研究方向为持久性内存和文件系统.

You Litong, born in 1994. PhD candidate. His main research interests include non-volatile memories, file system, and key value store system. (litong.you@sjtu.edu.cn)

游理通,1994 年生.博士研究生.主要研究方向为非易失性内存、文件系统和键值存储系统.

Wang Jingyu, born in 1994. PhD candidate. Her main research interests include storage system and remote direct memory access. (wjy114@sjtu.edu.cn)

王晶钰,1994 年生.博士研究生.主要研究方向为存储系统和 RDMA.

Yan Tian, born in 1994. Master candidate. His main research interests include non-volatile memories and file systems. (officialyan@sjtu.edu.cn)

闫　田,1994 年生.硕士研究生.主要研究方向为非易失性内存和文件系统.

Tu Yaofeng, born in 1972. PhD candidate, senior member of CCF. His main research interests include cloud computing, big data and machine learning. (tu.yaofeng@zte.com.cn)

屠要峰,1972 年生.博士研究生,CCF 高级会员.主要研究方向为云计算、大数据和机器学习.

Han Yinjun, born in 1977. Senior engineer. His main research interests include non-volatile memories, file system, and database system. (han.yinjun@zte.com.cn)

韩银俊,1977 年生.高级工程师.主要研究方向为非易失性内存、文件系统和数据库系统.

Huang Linpeng, born in 1964. PhD, professor and PhD supervisor. His main research interests include distributed systems and in-memory computing. (lphuang@sjtu.edu.cn)

黄林鹏,1964 年生.博士,教授,博士生导师.主要研究方向为分布式系统和内存内计算.