计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20170849
56(2):227-239,2019

基于 RDMA 的分布式存储系统研究综述

陈游旻[1] 陆游游[1] 罗圣美[2] 舒继武[1]
[1](清华大学计算机科学与技术系 北京 100084)
[2](中兴通讯股份有限公司 南京 210012)
(chenym16@mails.tsinghua.edu.cn)

Survey on RDMA-Based Distributed Storage Systems

Chen Youmin[1], Lu Youyou[1], Luo Shengmei[2], and Shu Jiwu[1]
[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2](*ZTE Corporation, Nanjing* 210012)

Abstract RDMA (remote direct memory access) is being widely used in big data area, which allows local host to access the remote memory without the involvements of remote CPUs, and provides extremely high bandwidth, high throughput and low latency, thus helping to boost the performance of distributed storage systems dramatically. As a whole, the RDMA-enabled distributed storage systems bring new opportunity to the big data processing. In this paper, we firstly point out that simply replacing the network module in distributed systems cannot fully exploit the advantages of RDMA in both semantics and efficiency, and revolutions of storage system design are urgently needed. Then, two key aspects of efficiently using RDMA are illustrated: One is the efficient management of hardware resources, including the careful utilization of NIC an CPU cache, parallel acceleration of multicore CPUs and memory management, and the other is the reformation of the software by closely coupling the software design and RDMA semantics, which uses the new features of RDMA to redesign the data placement schemes, data indexing and distributed protocols. Relative research works of distributed file systems, distributed key-value stores, and distributed transactional systems are introduced to illustrate the above two aspects. Summarizes of the paper, and suggestions for future research are also given at the end of this paper.

Key words remote direct memory access; distributed storage; key-value store; file system; distributed transaction

摘 要 远程直接内存访问(remote direct memory access, RDMA)技术正在大数据领域被越来越广泛地应用,它支持在对方主机 CPU 不参与的情况下远程读写异地内存,并提供高带宽、高吞吐和低延迟的数据传输特性,从而大幅提升分布式存储系统的性能,因此基于 RDMA 的分布式存储系统将为满足大数据高时效处理和存储带来新的机遇.首先分析了基于 RDMA 的分布式存储系统简单替换网络

收稿日期:2017-11-08;修回日期:2018-06-05
基金项目:国家自然科学基金项目(61433008);北京市科委重大项目(D151100000815003);中兴通讯股份有限公司研究项目(PJ160628142014);深圳市科技创新委员会科技应用示范基金项目(SF20170036)
This work was supported by the National Natural Science Foundation of China (61433008), the Beijing Municipal Science and Technology Commission of China (D151100000815003), the Project of ZTE (PJ160628142014), and the Shenzhen Commission on Innovation and Technology (SF20170036).
通信作者:舒继武(shujw@tsinghua.edu.cn)

传输模块并不能充分发挥 RDMA 在语义和性能上的优势的原因,并指出存储系统架构需要变革的因素.然后阐述了高效运用 RDMA 技术主要取决于 2 个方面:第 1 方面是硬件资源的高效管理,包括网卡缓存和 CPU 缓存的合理利用、多核 CPU 的并行加速以及内存资源管理等;第 2 方面是软硬件的紧耦合设计,借助 RDMA 在语义和性能上的特性,重构新型数据组织和索引方式、优化分布式协议等.同时,以分布式文件系统、分布式键值存储和分布式事务系统为典型应用场景,分别阐述了它们在硬件资源管理和软件重构这 2 个方面的相关研究.最后,给出了总结和展望.

关键词 远程直接内存访问;分布式存储;键值存储;文件系统;分布式事务

中图法分类号 TP302.1

在后摩尔时代,多核 CPU 已经成为发展趋势,同时,内存日益廉价,这使得构建内存存储系统变为可能;在大数据计算领域,数据规模大,数据维度高,数据种类多是其典型特征,内存计算技术逐渐开始发挥重要作用.加州大学伯克利分校开发的 Apache Spark[1] 以及 SAP 公司在 2012 年推出的 HANA 内存计算[2] 已经得到工业界的广泛关注.

DRAM 相比于磁盘在带宽和延迟上均有几个数量级的优势,能极大地提升本地数据存储性能,然而,DRAM 也面临集成度低等问题,单节点内存最大只能达到几百 GB,很难满足大型应用的存储需求.构建分布式大内存系统是一种有效途径,但传统以太网带宽和延迟与 DRAM 性能具有明显差异,网络性能将成为制约系统性能的最重要因素.近年来,远程内存直接访问(remote direct memory access, RDMA)[3] 作为一种新兴的跨网数据传输技术逐渐从高性能计算走进大数据领域,并开始受到广泛关注.RDMA 技术能够在远端 CPU 不参与的情况下,绕过内核直接访问远端内存,实现零拷贝的数据传输.近年来,200 Gbps 的高速网卡已经走向市场,这与 DRAM 的带宽进一步靠近.

然而,简单地将现有的分布式存储系统中的网络模块替换为 RDMA 通信模式,而不优化上层软件逻辑的策略,并不能充分发挥 RDMA 网络的硬件优势,这主要由 3 个方面导致:

1) 硬件管理缺失.一方面,RDMA 网卡和处理器均具有独立的缓存系统,并且 RDMA 网卡的缓存空间尤为有限,因此缓存空间的管理高效与否将直接影响系统整体性能;另一方面,RDMA 网卡具有良好的并行性,多核处理器环境下数据传输并行化具有较大空间.然而,简单兼容 RDMA 的软件系统很少从硬件角度考虑 RDMA 通信的效率问题,从而忽视了缓存管理、多核并行、资源共享等方面的重要性,导致系统性能低下.

2) 软件逻辑冗余.传统软件大多采用模块化设计,软件层次分工明确.在兼容新型硬件时,如果依旧沿用了传统的软件层次,将会导致软件逻辑冗余,效率低下.例如,传统的分布式文件系统需要部署在本地文件系统之上,通过本地文件系统管理本地数据,并依靠分布式软件层构建跨节点统一视图.基于上述软件架构,客户端通过 RDMA 网络读取数据时,数据块分别经过本地文件系统镜像、页缓存、用户态缓冲区等位置,将出现多层冗余拷贝.在慢速磁盘和传统以太网环境下,内存级冗余拷贝对系统整体性能影响甚微,但是,在构建基于 RDMA 的内存级分布式文件系统时,多层数据拷贝将明显降低系统整体性能,因此,软件逻辑重构将尤为重要.

3) 分布式协议低效.在分布式系统中,多客户端并发访问数据时,往往依靠 2 阶段锁或乐观并发处理进行并发控制,避免客户端间的访问干扰;而跨节点数据一致性、系统崩溃一致性则由分布式一致性提交协议(2 阶段提交)保证.上述协议往往牵涉多个节点协同工作,流程复杂,性能低下.为此,基于传统以太网的分布式系统中,往往通过特定的数据放置策略和处理方式避免使用分布式事务[4].然而,RDMA 工作模式具有强一致性,其单向原语和原子操作可以被利用起来,从而设计新型分布式协议,满足系统高效可扩展的需求.

RDMA 技术在通信模式、网络性能等方面呈现出与传统以太网完全不同的特点.对现有模块化的软件架构进行简单的网络替换将无法充分发挥 RDMA 网络性能.为此,需要结合 RDMA 网络的硬件特性,调整软硬件结构,从原语使用、数据流优化、协议设计等方面重新设计软件逻辑,从而充分发挥 RDMA 网络的硬件优势.

1 背景介绍

目前有 3 类网络架构支持 RDMA 技术,分别是

IB(InfiniBand),RoCE(RDMA over converged ethernet),iWARP (internet wide area RDMA protocol). 其中 RoCE 和 IB 具有相同的上层协议栈,而前者在数据链路层则完全兼容以太网,iWARP 保留了 TCP/IP 的完整协议栈.

RDMA 允许本地 CPU 绕过操作系统,直接读写远端节点内存,该过程无需远端 CPU 的参与. 以远程写操作为例(如图 1),本地 CPU 直接以 MMIO (memory mapped I/O)的方式向网卡发起远程写命令,并传递相应参数,包括待写入数据块基地址、远端内存地址、写入数据块大小、远端注册内存密钥等信息;本地网卡收到命令之后,立即根据本地数据块基地址将数据块从主存以 DMA Read 的方式读取到网卡缓存,并发送到远端;远端网卡接收到数据块之后,以 DMA Write 的方式直接将数据写入内存对应地址. 此过程中,RDMA 无需像传统以太网一样穿越内核中的多层网络协议栈,因此实现了跨节点数据传输过程中的数据零拷贝.

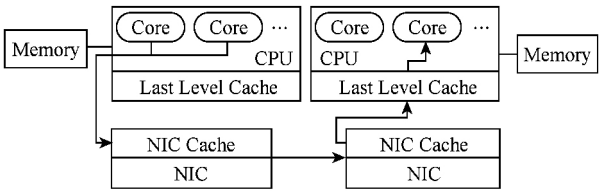

Fig. 1 Dataflow in RDMA network

图 1 RDMA 网络数据收发过程

RDMA 通信链路可以被配置为 3 种模式,分别是可靠连接(reliable connection, RC)、不可靠连接(unreliable connection, UC)和不可靠数据报(unreliable datagram, UD). 其中,UD 采用面向无连接的数据报发送方式,支持点对点和点对多的数据传输,而 RC 和 UC 模式仅支持点对点的面向连接的数据传输. 另外,UD 模式支持的最大数据传输单元为 4 KB,而 RC 和 UD 模式支持多达 2 GB 的单次数据传输.

RDMA 访问远端内存依靠 2 类原语:1)消息语义,其中 Send 和 Recv 是其典型的 1 组原语,它们类似于套接字编程中的 Send 和 Recv,在发送消息之前,接收方需提前调用 Recv 原语,用于指定接收消息的存放地址. 这类原语也被称作双向原语. 2)内存语义,该类原语包含 Read,Write 以及相应的变种(Write-with-imm 和原子操作). 这类原语能在远端 CPU 不介入的情况下直接读取或更新远端内存,它们亦被称作单向原语. 不同语义在不同的通信链路模式下具有不用的支持程度,如表 1 所示:

Table 1 RDMA Verbs and MTU Sizes in Different Modes

表 1 不同模式下的 RDMA 原语和最大传输单元(MTU)

Type	Send/Recv	Write[imm]	Read/Atomic	MTU
RC	√	√	√	2 GB
UC	√	√	×	2 GB
UD	√	×	×	4 KB

在 RDMA 编程中,网卡驱动分别提供了内核态和用户态编程接口,它们被称作 Verbs. 以 RC 模式下的点对点链路建立过程为例,应用程序需分别经历打开设备、创建保护域(protection domain, PD)、申请并注册内存、创建完成队列(completion queue, CQ)、创建收发队列(queue pair, QP)、初始化 QP 等过程. 其中,申请的内存用于收发和存储远端消息或被远端网卡直接访问. 内存在被远程访问之前需注册到网卡,注册过程包含:1)建立内存虚拟地址到物理地址的映射表,并存储到网卡缓存;2)生成密钥对,用于本地或远端内存访问控制. 收发队列包含发送队列(send queue)和接收队列(receive queue),用于存放 CPU 发起的网络请求,网卡从队列中依次处理原语,当原语处理完毕,将相应的完成信息存放到绑定的完成队列中. 创建的 QP、CQ 和注册内存均需绑定到相同 PD 中才能正常通信.

RDMA 绕过内核实现数据零拷贝,并借助硬件执行数据包的派送和解析,从而提供了高带宽、低延迟的通信特性. 但是, 将 RDMA 技术应用到分布式系统中时,也存在原语漏用、冗余拷贝、协议低效等问题急需解决. 为此,本文将结合实际系统,从以下 2 个方面分别进行阐述:1)硬件资源的高效管理,在大规模集群中,高并发的数据传输将导致 CPU 缓存、网卡缓存发生剧烈竞争,影响系统性能,另外,CPU 的多核架构为并行处理提供了机遇,如何高效调度 CPU 核、提升网络数据并行处理性能同样重要. 2)RDMA 提供了远程直接读写的新型通信原语,这打破了传统分布式系统中的跨节点数据传输模型,促使一些系统,例如分布式文件系统[5-7]、分布式键值存储系统[8-14]、分布式事务系统[15-18]等选择通过软件重构来充分发挥 RDMA 的硬件特性.

2 RDMA 与硬件管理

RDMA 技术通过硬件解析并处理网络数据包降低延迟的同时,能有效提升数据传输的并行能力.

另外,网卡将链路状态信息等核心数据缓存到网卡的缓存空间,用户程序绕过内核直接与硬件交互.因此,构建高效的基于 RDMA 的存储系统的关键因素在于结合 RDMA 的硬件特性,与系统软件层进行紧耦合设计.本节将主要从网卡缓存、CPU 缓存以及多核调度等硬件层次阐述如何设计高效的系统软件.

2.1 缓存管理

缓存系统是计算机体系结构中的重要组成部分,它将访问频繁的数据暂存到高速缓存器以加速应用,因此,缓存系统对局部性较好的应用加速效果显著.在 RDMA 网络通信中,CPU 和网卡均存在独立的缓存系统,缓存使用的好坏将直接影响着节点间数据通信性能.

网卡缓存用于暂存 CPU 发起的网络请求和相应的网络状态信息,其主要缓存 3 类数据:1)注册内存从逻辑地址到物理地址的映射表,当网卡发送数据或接收到数据时,将依据映射表查询相应物理地址;2)QP 状态,存放 QP 对应的元数据信息;3)由 CPU 发起的网络请求条目,网卡根据请求条目执行相应处理.在特定场景下,这 3 类数据都会增加网卡缓存缺失率,导致系统性能下降.

第 1 方面,内存注册后以页为单位生成映射表,默认情况页的大小为 4 KB,假设每个映射条目为 12 B,则注册 10 GB 内存需要的映射表大小为 30 MB,这将很难全部存放到网卡缓存中.FaRM[13,15] 引入内核驱动 PhyCo,它在系统启动时分配 2 GB 对齐(网卡最大支持 2 GB 的页大小)的物理连续的内存区域,并将其映射到 FaRM 进程的逻辑地址空间.通过这种方式,映射表大小将缩小为 60 B.

第 2 方面,RDMA 的扩展性问题还源于连接数量的增加.图 2 展示了 RDMA 网络在不同链路模式下的吞吐,我们选取 1 个节点作为 Server,启动多个线程并行地向多个节点发送(接收)32 B 的消息.其中向外发送消息称作 Outbound-Message,反之则为 Inbound-Message.如图 2 所示,在 RC 模式下,总吞吐随着连接数量的增大而减小,而 Outbound 相比于 Inbound 下降更为明显.相反,UD 模式下总吞吐不受到连接数量的影响.这主要是因为在 RC 模式下,Server 端需要与每个客户端创建 QP 并建立连接,当连接数量过多后,QP 状态信息无法全部存放到网卡缓存,导致数据频繁在网卡缓存和主存间换入换出,影响性能.而 UD 模式下,Server 端只需创建固定数量的 QP,便能与所有的客户端进行通信,因此服务端总吞吐不受到客户端数量的影响.FaSST RPC[16] 是基于 UD 的 RPC 系统,得益于 UD 的无连接通信模式,FaSST 能够线性扩展到数百个节点.然而,UD 同样也存在其他缺陷,例如不支持远程直接读写,单次最大传输数据量不能超过 4 KB,底层无拥塞控制逻辑,且有丢包或乱序风险.因此需要额外的软件控制来保障 UD 在真实应用中的可靠数据传输,但软件控制又将引入新的开销[19].

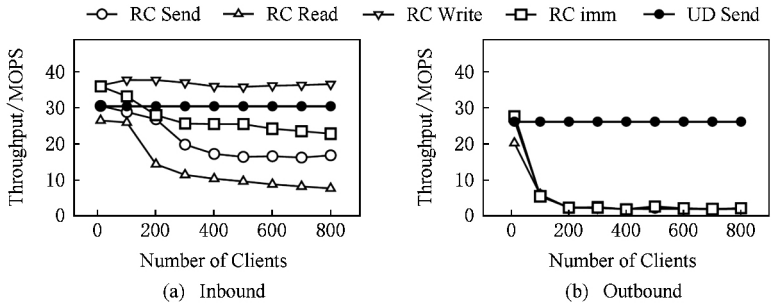

Fig. 2 Scalability issue in different transport modes of RDMA

图 2 RDMA 在不同链路模式下的扩展性问题

第 3 方面,CPU 发起网络请求时,使用 MMIO 向网卡发送命令,这些命令将被临时缓存到网卡缓存,并由网卡依次处理.然而,当 CPU 发送命令过快时,网卡缓存不足以存放新的请求,因此之前发送的未处理的请求将被换出到主存,等到被处理时再换回到网卡缓存.频繁的换入换出操作将引入大量的 PCIe 流量,这是制约网络吞吐的关键因素.图 3 源于文献[20]中的图 13(b),展示了单节点向外发起写请求时的缓存缺失情况.CPU 以批量请求的方式向网卡发起 RC Write 请求,通过控制每次批量请求包含的请求数量来调节 CPU 发起请求的速度(横坐标).其中 CX3 和 CIB 为 2 类网卡型号,且 CIB 性能强于 CX3.图 3 中 CX3 WR 和 CIB WR 分别表示在 CIB 和 CX3 环境下 RC Write 请求的速率.而 PCIeRd 则表示网卡发起的 PCIe Read 的速率,PCIe Read 速率越高,代表额外的 PCIe 流量越高,

对性能影响越大. 从图 3 中我们可以发现,CX3 环境下,当每次批量请求的数量达到 16 后,RC Write 的吞吐有一定下降,而此时 PCIeRd 的速率明显上升,表明此时 CPU 发送请求速度过快,已经存在明显的缓存缺失现象. CIB 环境下 PCIeRd 速率一直保持较低水平,且 RC Write 吞吐几乎线性提升,这说明 CIB 网卡能够支撑 CPU 发起的最快请求速度,而不发生缓存缺失现象. 因此,当网卡处理能力不及 CPU 时,需控制 CPU 请求速率,避免网络吞吐下降.

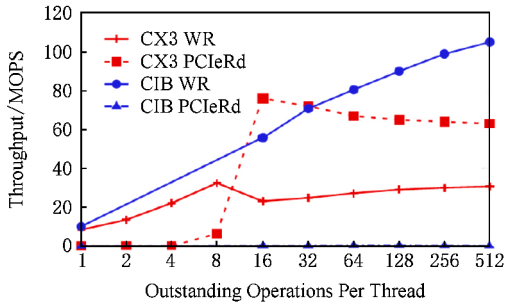

Fig. 3 NIC cache misses for RC Write operations[20]

图 3 RC Write 操作对应网卡缓存缺失率[20]

CPU 缓存在网络数据收发过程中扮演重要角色. 因特尔至强 E5 系列处理器提供了独有的数据直访 I/O(data direct I/O,DDIO)技术,它允许 PCIe 外设直接读写 CPU 的末级缓存(LLC,通常为第 3 级缓存)而绕过对主存的访问,从而降低 I/O 适配器到内存以及内存到 CPU 之间的延迟(如图 4). PCIe 外设以 Write Update 或 Write Allocate 的方更新 CPU 的 LLC. 当访问数据已经缓存到 LLC 时使用前者进行更新,而当访问数据不在缓存中时,则使用后者. Write Allocate 操作需先分配缓存空间,然后写入数据,其开销大于 Write Update. 通常情况下,用于 DDIO 的缓存区域占整个 LLC 的 10%左右,以防止外设 I/O 干扰 CPU 运行应用程序的性能. 因此,基于 RDMA 建立的通信模型下,构建过大的通信消息池将不能被完全映射到 LLC 中,从而引入大量的 Write Allocate 操作,影响系统的整体性能.

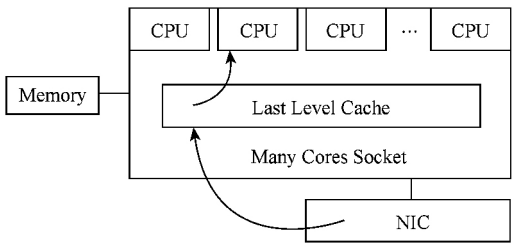

Fig. 4 NIC writes with Intel data direct I/O

图 4 因特尔数据直访 I/O 技术下网卡写操作流程

文献[9]构建的 HERD RPC 系统中,服务端静态分配固定大小的消息池,并划分为连续的内存块,用于存放远端客户端写入的新消息. 客户端通过 UC Write 将请求远程写入到消息池中,服务端处理之后则使用 UD Send 返回响应信息. 为避免客户端在写入消息时互相干扰,不同客户端被映射到消息池的不同区域. 在 HERD 中,每个消息块大小为 1 KB,最大支持的客户端数量为 200,消息池的大小为 6 MB,能完全存放到 LLC. 但这种静态映射方式限制了最大支持的客户端数量,同时每个请求的数据量也不能超过 1 KB. FaSST[16] 使用 UD Send 传递请求,数据存放地址由接收方决定,因此客户端之间可共享同一个消息池,消息池大小不会因为客户端总数量变化而发生改变.

2.2 CPU 调度

基于 RDMA 设计高效的系统软件,在 CPU 调度层面需要考虑以下 3 个方面的问题.

1) 多核并发. RDMA 网络具有良好的并行性,单条链路上的网络负载很难达到网卡的峰值处理能力. 另一方面,使用单个 CPU 核不足以处理多条链路上的网络请求,CPU 多核并行同样重要. 如图 5(a)所示,服务端与客户端创建多个 QP 连接,通过变化 QP 数量,展现了 RDMA 不同原语的最大吞吐. 我们发现,在 QP 数量少于 4 时,所有原语均不能达到网卡处理峰值. 同样,图 5(b)展现了不同 RPC 系统的吞吐率,横坐标表示改变服务端 CPU 核的数量,并启动 40 个客户端发送 RPC 请求,纵坐标表示不同 RPC 系统的吞吐率,发现 CPU 核的数量超过 4 时才能达到网卡处理峰值. 由此可见,合理增加并发度,能较大提升 RDMA 的网络性能. DaRPC[21] 是一个基于 RDMA 的 RPC 系统,它在服务端采用了精细化的并行架构:初始化阶段,服务端启动多个线程,并行处理远端请求,每个客户端均与服务端建立独立的 QP 链路,这些链路被映射到不同的服务端线程,用于并行消息处理. 另外,在 NUMA 架构下,网卡、主存等相对于 CPU 具有非对称访问特性,近端 CPU 访问网卡和主存能有效降低延迟、提升吞吐. 文献[21]表明,NUMA 友好的近端绑定策略能有效提升系统性能达 20%左右.

2) 负载均衡. 服务端线程静态映射的方式能提升并行度,但每个客户端负载具有差异性,且相应的远程调用开销不尽相同,因此有可能导致个别 CPU

核处理繁忙,而其他 CPU 核空闲等待的现象. 文献[21]提出了一种"Work Stealing and Load Balancing"的管理方法,在服务端引入监控器,用于实时统计各 CPU 核的工作负载状态,当某 CPU 核的工作负载超过某阈值,则将新来的部分请求放入到 1 个全局队列中,而其他 CPU 核在不繁忙时查看全局队列,并及时处理相应请求. 这种方式有效解决了各 CPU 核负载不均衡的问题,但同时也引入了时序问题,导致早到达的 RPC 请求在晚到的请求之后被处理. 这种乱序响应需要在客户端处理逻辑中被谨慎考虑.

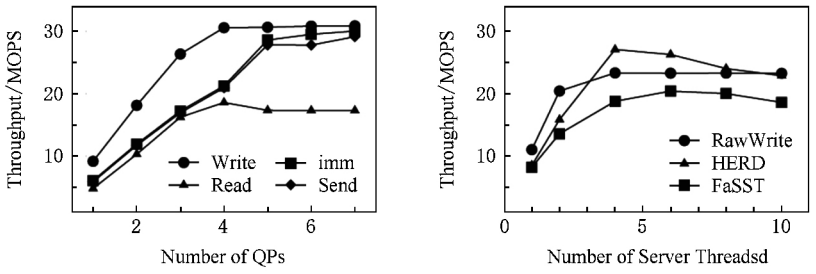

Fig. 5 Throughput with different QPs and CPU cores

图 5 不同 QP 数量或 CPU 核对吞吐的影响

3) QP 共享. 基于 RDMA 的对称系统架构[13,15-16]中,各节点同时启动服务端进程和客户端进程,并启动多个 CPU 核并行处理远端请求或发起请求. 假定集群规模为 N_s,每个节点启动 N_c 个 CPU 核,集群中的各 CPU 核之间建立全相连的拓扑结构,则每个节点需创建 QP 的数量为 $(N_s-1)\times N_c^2$,这在大规模集群中将存在严重的扩展性问题,QP 需要在 CPU 核间共享. 一种途径是任意 2 个节点之间仅创建一条链路,则每个节点间需创建的 QP 数量为 N_s-1,使得 QP 数量保持在较低水平. 但是,这种 QP 共享方式迫使多个 CPU 核并发向同 1 个节点发送网络请求时使用同 1 条 QP 链路,从而导致严重的资源竞争. 图 6 展示了 Read 请求在不同程度的 QP 共享下的单线程吞吐率,横坐标代表总共创建的 QP 数量. 从图 6 可以发现,在不共享情况下,即单个线程处理所有 QP 中的请求,性能(10.9 MOPS)远高于多个线程的共享处理模式(2 MOPS). 因此 QP 共享模式导致 CPU 利用率极低. 一种折中的方法是允许各节点间具有相同 ID 的 CPU 核之间进行通信,即每个节点建立 $N_s\times N_c$ 个 QP,这样 CPU 核之间不存在资源竞争,QP 数量也保持在较低水平.

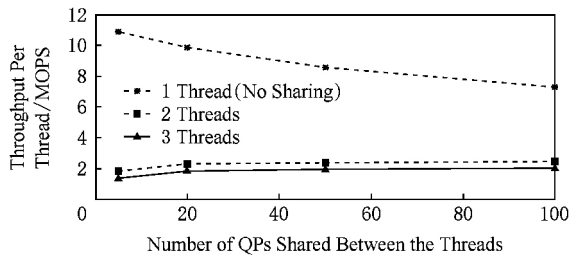

Fig. 6 Per-thread Read throughput with QP sharing[16]

图 6 QP 共享模式下 Read 请求的单线程吞吐率[16]

3 RDMA 与软件重构

RDMA 提供了不同于套接字编程的通讯接口,它允许在远端 CPU 不参与的情况下直接访问远端内存,实现数据的零拷贝传输,这在一定程度上颠覆了传统的系统架构思想,从而促使了分布式键值存储系统、文件系统以及事务系统等在软件层的变革. 软件层次的变化可基本总结为以下 3 类:分布式协议的重构、网络负载的分派均衡和数据的远程索引等方面. 本节将从多类分布式存储系统展开,并阐述软件层如何从以上 3 个方面使用和优化 RDMA 网络.

3.1 RDMA 在 Key-Value 系统中的应用

在传统存储系统中,数据的组织和索引由服务端本地执行. 一般地,客户端读取或更新服务端数据时,首先向服务端发送 RPC(remote procedure call)请求,服务端接收到请求后,迭代式地查询以树状或散列组织的数据,然后将查询或更新结果返回给客户端. Key-Value 系统采用平铺式的数据存储管理模式,仅提供类似 Get, Put 等接口,系统结构简单. 同时,RDMA 可以直接访问远端内存数据,这使得分布式键值存储系统中的数据索引模式发生改变. 近年,结合 RDMA 和 Key-Value 存储的分布式键值存储系统[8-14]被广泛研究,本节将着重介绍 Pilaf[5].

Pilaf 是纽约大学于 2013 年提出的一个内存级分布式键值存储系统,借助 RDMA 原语实现了极高的系统性能,同时有效降低了服务端 CPU 开销. Pilaf 在处理 GET 请求时,利用 RDMA 内存语义低延迟的特性,通过客户端发起多次 Read 请求完成键值查询,将数据索引任务由服务端转移到客户端(见图 7).

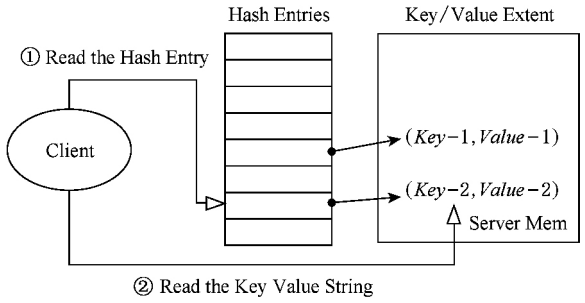

Fig. 7 Overall architecture of Pilaf[8]

图 7 Pilaf 的总体架构[8]

如图 7 所示,Pilaf 的键值对通过散列表索引,散列表和键值对统一存放在内存区域中,散列表中的各表项存放键值对的内存地址,用于索引真实的键值对. 存放上述内容的内存空间在系统初始化时分配,并注册到网卡,使得客户端可远程直接访问,客户端在接入时获取服务端注册内存的虚拟地址. 客户端在执行 Get 操作时,首先计算出 Key 相应散列值,并根据散列值确定对应表项在散列表中的偏移;然后执行 RDMA Read 读取散列表在该偏移下的内容,如果表项包含 1 个有效地址,则根据该地址执行第 2 次 Read 操作获取键值内容. 如果获取的键值对匹配,则成功返回;如果散列表项为空,或者键值对不匹配,则线性查找下 1 个表项,直到查询成功.

Put 操作需要更新散列表和键值对,在多客户端并行访问场景下,会出现数据冲突访问,因此将 Put 操作的逻辑完全交由客户端并不现实. 在 Pilaf 中,所有的更新操作(包括 Put,Del 等)将按照传统方式转交给服务端执行. 具体方式为:客户端向服务端发起相应更新操作的 RPC 请求,服务端收到请求之后,在本地执行查询更新,然后返回更新结果. 在读写并发场景下,以上工作流程会引入脏读. 例如,在服务端更新某一键值对的时候,客户端同时使用 Read 读取正在更新的键值对,则有可能读取到 1 个中间状态的值. Pilaf 引入自校验方法来解决脏读问题. 如图 8 所示.

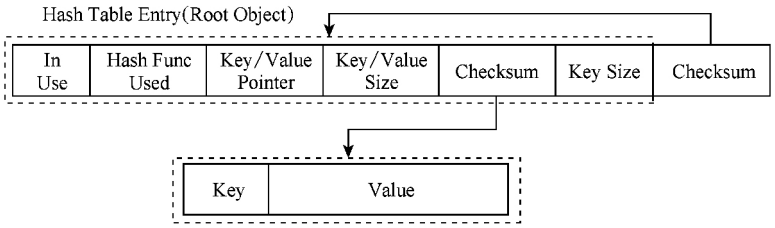

Fig. 8 Self-verifying Hash table structure[8]

图 8 自校验散列表数据结构[8]

在散列表的每一个表项中,添加 2 层校验码,服务端在处理更新请求时,首先更新键值对内容,然后根据键值对内容计算校验码,并更新散列表对应表项的第 1 个校验值,然后根据表项更新第 2 个校验值. 当客户端执行 Get 操作时,通过 2 次匹配校验码来判断是否发生脏读. 如果匹配失败,则等待随机时间,再次获取键值内容,直至匹配成功. Pilaf 借助多次 Read 请求将 Get 处理逻辑转移到客户端,这种远端索引方式一定程度降低了服务端的处理开销. 同时,依靠 RDMA 的硬件优势,系统处理请求的能力相比于传统方式提升显著.

表 2 详细描述了多种 KV 存储系统的实现差异. HERD[9] 广泛地测试了 RDMA 各类原语的性能差异,总结出 Pilaf 在读取操作中将引入多次 RDMA Read 操作,效率低下. 为此,HERD 设计基于 RDMA 的高性能 RPC 系统,所有的操作依旧通过服务端处理,从而每次远程请求将只引入 1 次往返网络交互. HERD 同时还考虑到 RDMA 网络的扩展性问题,采用 UC Write 和 UD Send 分别作为客户端发送请求和服务端发送反馈信息的原语. HERD 能同时保证 Get/Put 请求的高性能,且客户端数量能轻松扩展到数百个. HydraDB[10] 则面向通用性中间件,可作为系统缓存层或独立的存储层,并提供数据复制功能,保障数据容错,同时结合 NUMA 架构进行性能优化. HydraDB 也使用 RDMA Read 执行 Get 请求,并增加客户端缓存提升性能. 在 KV 缓存系统中,引入 RDMA 单向原语将导致服务端对客户端访问特性无法感知,为解决

上述问题,C-Hint[12]通过客户端与服务端的协同设计,以提升 KV 缓存系统的命中率. RFP[11]针对 RDMA 的 InBound 和 OutBound 原语性能不对称问题,提出客户端主动执行发送和接收的新型 I/O 机制. Nessie[14]则配合使用 Read、Write 以及原子操作(CAS)原语,将 KV 访问逻辑完全转交到客户端,从而避免服务端 CPU 参与存储服务. FaRM[13]将 KV 存储系统构建在具有事务接口的分布式共享内存处理平台上,该系统将会在后文进行进一步介绍. 目前国内针对 KV 存储系统的研究也已经相应展开,但大多基于现有系统进行改造,性能方面局限性较大[22].

Table 2 Comparison of Implementation Details Between Different KVStores

表 2 不同键值存储系统实现细节对比

KVStore	Get Forward	Get Back	Put Forward	Put Back	Cache	Replication	Transaction
Pilaf[8]	Read	Read	Send	Recv	No	No	No
HERD[9]	UC Write	UD Send	UC Write	UD Send	No	No	No
HydraDB[10]	Read	Read	Write	Write	No	Yes	No
RFP[11]	Write	Read	Write	Write	No	No	No
C-Hint[12]	Read	Read	Write	Write	Yes	No	No
FaRM[13]	Read	Read	Write	Write	No	Yes	Yes
Nessie[14]	Read	Read	Write +CAS	Write +CAS	No	No	No

3.2 RDMA 在文件系统中的应用

总体上,RDMA 更多地被应用到键值存储和分布式事务系统,而分布式文件系统由于结构复杂,很难充分发挥出 RDMA 的硬件特性. 目前,也有部分分布式文件系统开始尝试支持 RDMA 网络,以提供更高的性能[6-7]. 但是,这些系统大多采用了模块化的软件设计,将网络传输、文件存储和控制逻辑严格分离,而在引入 RDMA 网络的时候,仅仅采用了简单的网络通讯模块替换,而不是选择重构文件系统的内部逻辑,因此取得的效果甚微.

清华大学于 2017 年提出的分布式持久性内存文件系统 Octopus[23],通过紧密结合 RDMA 特性,重新设计了文件系统软件逻辑. 具体地,各个节点将数据存储区注册到内存,并共享到集群使之可被远程直接访问,进而构建持久性共享内存,而元数据区域则由服务节点进行本地管理(如图 9). Octopus 通过引入持久性共享内存以降低数据冗余拷贝,进而提供接近硬件的读写带宽;引入客户端主动式数据传输来重新均摊客户端和服务端之间的网络负载;引入自识别远程过程调用协议以提供低延迟元数据访问性能.

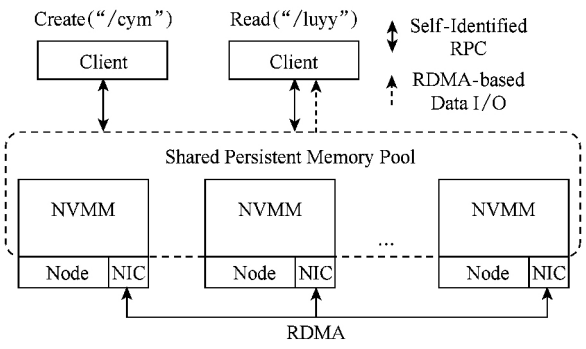

Fig. 9 The overall architecture of Octopus[23]

图 9 Octopus 整体架构[23]

1) 持久性共享内存

现有的分布式文件系统构建在本地文件系统之上,因而存在大量的数据冗余拷贝. 以文件读写为例,数据块需要逐层拷贝到 TCP/IP 协议栈中的 mbuf、用户态缓冲区、内核态页缓存、文件系统镜像等位置,最终统计到 1 次文件读写将引入 7 次数据拷贝,这种低效的系统设计模式严重限制了 RDMA 网络的硬件特性. Octopus 则提出了持久性共享内存,用于直接管理文件系统数据. 这种设计移除了本地文件系统层,同时让客户端远程直接访问,成功将数据拷贝次数降低到 4 次.

2) 客户端主动式数据传输

通常情况下,文件读写将引入 1 次网络往返请求,以文件读取为例,客户端主动发起文件读请求,服务端收到请求之后,查询并装填数据,然后将数据返回给客户端. 我们称这种传输模式为服务端主动式数据传输,这种工作模式在传统以太网下工作高效,但是当转移到 RDMA 网络上后 CPU 占用率极高,成为了系统瓶颈. 为此提出一种客户端主动式数据传输机制,它具有以下 3 个步骤: ①客户端向服务端发起文件读写请求;②服务端查询文件元数据信息,并将元数据直接返回给客户端;③客户端根据

元数据信息提供的远程地址,直接执行 RDMA Read/Write.其中步骤①②通过发起 RPC 请求完成,而步骤③由客户端主动执行,服务端 CPU 不参与文件数据的传输,这种设计引入了更多的网络往返次数,但是将服务端 CPU 负载转移到客户端,从而提升了并发文件读写性能.

3) 自识别远程过程调用

RDMA 提供了微秒级别的网络延迟,为充分利用 RDMA 低延迟的特性,Octopus 设计并实现了高效的 RPC 系统. RPC 主要用于文件元数据访问和分布式事务的协调提交. RDMA 的 2 类原语均可用于设计 RPC 系统,基于 Send/Recv 的 RPC 实现简单,但其性能不如内存语义,UD 模式具有良好的扩展性和高性能,但其诸多缺陷导致并不适合应用到基于可靠传输的文件系统场景. 内存语义具有极低延迟,在构建 RPC 系统时,由于其单向性,服务端必须开启线程轮询扫描内存来检测新消息,这会导致较高的 CPU 开销,且客户端数量上升后,CPU 需扫描更大范围的内存空间,这也会影响 RPC 延迟. Octopus 选择了 Write-with-imm 作为远程请求的原语,它可在报头携带 32 b 立即数,用于存放客户端元数据信息,指导服务端快速定位新消息;同时,该原语将消耗服务端提前发起的 Recv 请求,因此是双向原语;基于 Write-with-imm 的 RPC 系统能帮助服务端快速检测到完成信息,根据立即数快速定位到新消息,保证 RPC 性能的同时提供稳定的传输延迟.

俄亥俄州立大学提出的 NVFS[7] 将非易失内存和 RDMA 结合起来,用于加速 HDFS. 但由于 HDFS 本身软件设计厚重,NVFS 很难充分发挥 NVM 和 RDMA 的硬件特性. IBM 最近提出的 Crail[5],其元数据跨节点访问基于 DaRPC[21],其中,DaRPC 是一个基于 RDMA 的 RPC 系统,它将消息处理和网络传输紧密结合,并提供高吞吐、低延迟的跨网传输性能. Crail 相比于 Octopus 多 1 次冗余拷贝,因此其带宽不如 Octopus;Crail 同时也无法保证数据的一致性和持久性,为此,Crail 引入多级存储架构,并通过后端存储进行数据持久化.

3.3 RDMA 在分布式事务中的应用

RDMA 具有 cache 强一致性,这意味着,RDMA 网卡写入的最新数据能及时被 CPU 读取,同时,网卡总是发送 CPU 最新更新的数据; RDMA 还具有原子性,它提供 2 种原子操作原语:CAS 和 FAA①,它们可以并发更新同一个内存地址上的64 b值而保证 1 次操作的原子性. RDMA 的上述特性将促使分布式协议的重构.

DrTM[17-18] 是上海交通大学 2015 年提出的一种分布式事务系统,它巧妙地结合了 RDMA 和硬件事务内存(hardware transactional memory, HTM),利用它们之间的强一致性设计并实现了一套高效的分布式事务协议. 事务内存提供了 XBEGIN,XEND 和 XABORT 接口,能够在硬件层面控制对内存的冲突访问,一旦冲突发生,只有 1 个线程成功修改,而其他线程均无条件终止. 事实上,RDMA 的远程冲突访问也将导致本地的事务性内存访问终止,因此,DrTM 利用这种强一致性重新设计了事务协议.

DrTM 在执行事务逻辑时,首先通过 RDMA Read 将存储在远端的读集和写集搜集到本地,然后依靠 HTM 在本地执行数据更新,最后使用 RDMA Write 将更新过的数据写回到远端(如图 10 中 Case1). 其中,图 10 中 Case2~Case5 展示了本地和分布式事务执行冲突的不同场景:图 10 中 Case2 场景中,M1 搜集远端数据时,M2 正在执行本地更新,HTM 将强制终止执行,2 个冲突事务之间实现了序列化. 而图 10 中 Case2 和 Case3 场景下,M1 开始搜集远端数据时,M2 还未开始本地执行,因此 M2 事务不会终止,之后 M2 也访问并修改了相同的数据项,冲突事务之间没实现严格的序列化. 为解决上述问题,DrTM 借助 RDMA 原子操作实现了一套分布式锁,并强制在搜集远端数据之前首先进行数据项加锁. 这样在图 10 中 Case3~Case5 场景中,M2 的本地/远程读写阶段将被阻塞,从而实现事务间的序列化. DrTM 借助 RDMA 的强一致性和 HTM 的原子性,重新构造了分布式事务协议,同时基于 RDMA 提供的原子操作原语设计了高效的独占锁和共享锁,用于细粒度的事务并发控制. 因此,RDMA 提供的一致性以及原子性的特点将促使相应的分布式协议发生重大变化.

FaRM[13,15] 是 Microsoft Research 在 2014 年提出的一个基于 RDMA 的分布式内存计算平台. FaRM 提供基于分布式事务的共享内存读写接口,通过乐观锁和 2 阶段提交协议来保证事务执行的原子性和可序列化. 如图 11 是 FaRM 的事务执行逻辑,其中

① COMP_AND_SWAP 和 FETCH_AND_ADD 缩写为 CAS 和 FAA.

包含 1 个协调者(C)和 3 个参与者(P_1,P_2,P_3)以及他们的备份节点(B_1,B_2,B_3). 事务分为执行阶段和提交阶段,图 11 中虚线箭头代表 Read,实线箭头代表 Write. 在执行阶段,协调者首先查询内存区标识符,通过散列函数计算相应远端地址,发起 Read 请求,远程读取事务涉及的读集和写集,然后根据读取的数据在本地执行事务逻辑. 在提交阶段,首先要对写集对应数据项加锁,协调者以 RPC 的方式将加锁请求写到参与者的消息区,参与者接收到消息后,对数据项加锁,如果加锁失败或数据项版本发生变化,则返回失败,事务终止. 加锁阶段完成后,需再次核对读集对应版本. 若以上 2 步均成功执行,则先后对备份节点和参与者执行数据提交. 为提高事务读写的并发性能,FaRM 通过增加 cache line 粒度的版本控制实现无锁读,在不加锁的情况下执行事务读取,并感知事务的一致性,判断是否发生脏读. 不同于 DrTM 的是,FaRM 无需 CPU 支持 HTM 功能,而是通过乐观锁进行并发控制,仅在事务提交时进行冲突检查;另一方面,FaRM 在事务执行过程中进行数据备份,从而提高了数据可用性.

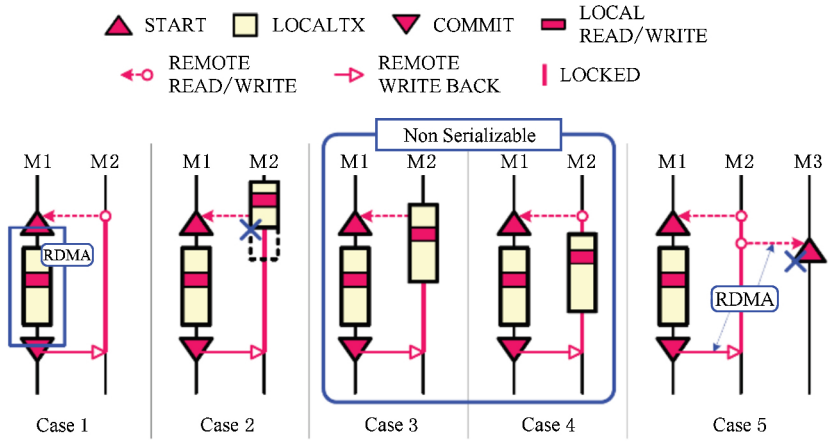

Fig. 10 The cases of conflicts between local and remote txs[17]

图 10 本地和分布式事务之间的冲突情形[14]

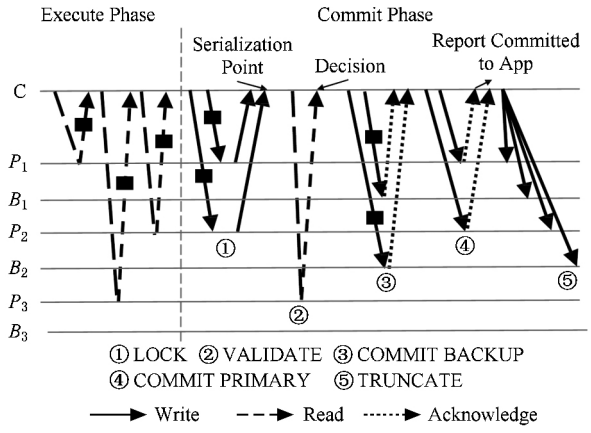

Fig. 11 FaRM commit protocol[15]

图 11 FaRM 的事务提交协议[15]

CMU 于 2016 年提出的分布式事务协议 FaSST[16],则侧重于考虑 RDMA 网络的高性能和可扩展,他们发现 UD Send 在实际通信场景中根本不出现丢包情况,且表现出极高的性能和扩展性,为此,他们借用 FaRM 的分布式事务协议模型,设计并实现了新型分布式事务,其性能远高于 DrTM 和 FaRM.

3.4 RDMA 的通用性优化

RDMA 在上述 3 类系统中得到了广泛的研究与应用. 此外,RDMA 还在 I/O 负载分派、数据远程备份等方面提供了新的方式,本节将着重介绍与 RDMA 相关的通用性优化技术.

1) 网络负载重分派

RDMA 单向原语在执行过程中仅需发送方 CPU 参与,这区别于传统套接字的收发模式. 因此从 CPU 执行 I/O 负载角度来看,RDMA 单向原语为网络负载的重分派提供了机遇. 从 3.3 节事务提交协议中可以看到,FaRM 在事务执行阶段、版本核对阶段、备份节点/参与者提交阶段以及无锁读阶段均使用了单向原语,参与者或备份节点 CPU 完全不参与事务执行逻辑. 通过网络负载的重新分派,协调者组织执行事务的大部分逻辑,而参与者较少参与事务提交,将 CPU 资源更多的应用在本地执行逻辑上. 实验表明,90 节点的 FaRM 系统运行 TATP 测试集,可提供 140 MTPS 的事务执行速度. 另外,3.1 节提到的 Pilaf,HydraDB 等键值存储系统将 Get 请求的网络负载从服务端重新分派到客户端,一定程度上

减轻了服务端 CPU 开销,同时提升了系统吞吐. 而 Octopus 则进一步改变数据 I/O 通路,将传统的服务端主动返回文件数据的设计模式改为"服务端仅返回文件数据地址,客户端主动通过 RDMA Read 或 Write 执行文件 I/O"的新型 I/O 机制,从而大幅降低文件服务器 CPU 开销,并提升了数据并发存取吞吐.

2) 数据备份与持久化

新型非易失存储器[24-26]提供接近内存的访问速度以及字节寻址的访问方式,同时还能像磁盘一样提供持久性存储. 持久性内存的出现改变了计算机体系结构中存储金字塔的持久性和易失性边界,数据持久化由传统的软件控制方式变为执行 CPU 刷写指令的硬件控制方式[27]. 因此,存储系统在组织数据时需要精细化执行数据持久化操作或异地备份,以保障数据存储可靠性和故障可恢复. 然而,CPU 持久化刷写指令代价高,异地备份则引入额外的网络开销,上述原因都将成为制约非易失内存性能的重要因素. RDMA 提供极低的网络延迟(~1 μs),这与非易失内存的访问延迟几乎在同一个数量级,因此,Mojim[28]提出一种基于 RDMA 的新型数据布局和备份方案,并提供了高可靠、高性能的存储服务.

Mojim 是一个构建在 NVMM 上的内核态系统,图 12 描述了 Mojim 的系统架构. 它可以向上层提供基础的数据读写接口和不同级别的持久化接口. Mojim 采用了包含主层和辅层的双层架构,其中主层包括 1 个主节点和 1 个镜像节点,辅层包含一到多个备份节点. 根据不同的持久化级别,写入 Mojim 的数据同步或异步地流入到主节点、镜像节点和备份节点. Mojim 包含的持久化级别主要包括 M-sync,M-async 和 M-syncsec. 其中,以 M-sync 模式写入时,Mojim 将数据首先写入到本地(不执行 CPU 缓存数据刷写),然后将数据传输到镜像节点,等待对方网卡反馈确认信息后成功返回. 而镜像节点异步将数据传递到备份节点. 以 M-async 模式写入时,主节点首先刷写缓存数据,然后将数据传输到镜像节点,在不等待确认信息时直接成功返回. M-syncsec 则同步等待数据写入到备份节点之后才成功返回. 以上 3 类持久写方式具有不同级别的可靠性、可用性和一致性.

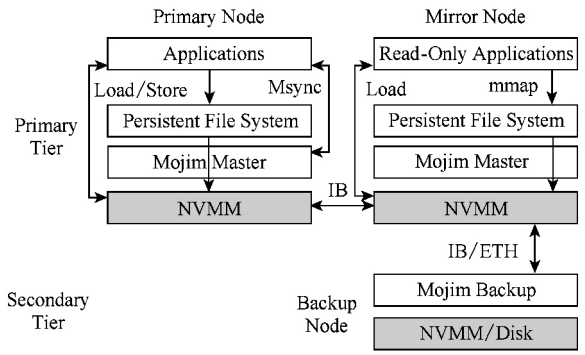

Fig. 12 Mojim architecture[28]

图 12 Mojim 的系统架构[28]

实验显示,基于 RDMA 的异地备份方案性能强于基于本地缓存刷写的持久化方案,这主要是因为 CPU 刷写缓存数据以 cache line 为粒度,且并发的数据持久写被 CPU 强制顺序性执行,并行度严重受到限制. 因此,RDMA 网络为分布式持久性内存系统中的数据持久化和远程备份提供了新的机遇. 另一方面,基于远程刷写的原语[29]也有望出现.

RDMA 区别于传统的以太网通信方式,它能提供 Read,Write 等远程直接访问的单向原语,在数据传输过程中无需远端 CPU 的参与,这促使系统设计时重新考虑数据的组织和索引方式,以及网络负载的均衡分派;RDMA 的低延迟访问特性彻底改变了传统分布式系统设计中对"本地"和"异地"的性能权衡,为数据的多机备份和持久化提供了新的思路;RDMA 还能保证缓存数据的强一致性,处理器总能够读取最新数据,同时还提供 2 种原子操作,用于数据项的原子性更新,这为设计新型分布式协议提供了新的机遇.

4 总结与展望

本文分别从硬件管理和软件重构 2 个方面阐述了对 RDMA 的相关研究,详细分析了 RDMA 的内在硬件特性、系统软件设计的变革以及当前 RDMA 的典型应用场景. 从硬件角度来看,RDMA 网络并行数据传输会受到多处硬件资源的制约,例如网卡缓存、CPU 缓存等,因此在系统架构上要同时要兼顾其硬件特性,在系统扩展性、可靠性和高性能中权衡,在 CPU 资源开销与网络资源共享之间合理取舍. 从软件重塑角度看,RDMA 区别于传统网络的数据传输接口和其极低的网络延迟改变了现有的软件设计方式,为数据组织与索引、网络传输重分派和分布式协议的重构等方面提供了新的方法,合理的软件重构更能充分发挥 RDMA 网络的优势,提升系统的整体性能.

迈洛斯公司目前已经提供超过 200 Gbps 的传输速率的 RDMA 网卡,传输延迟早已突破 1μs,其迅速提升的网络传输性能,正促使 RDMA 技术被逐渐应用到数据中心.针对其扩展性问题,迈洛斯引入动态链接传输机制(dynamically connected transport, DCT)实现动态创建/销毁 QP 链接,以保证 QP 数量维持稳定;Oracle 提出的 Sonoma 处理器则将网卡和 FPGA 集成到片上,能轻松支持数千个 QP 链路.为增强 RDMA 网卡设备的灵活性,迈洛斯还提供了可编程式 RDMA 网卡,用于协助用户进行应用数据传输加速、数据压缩/解压缩和其他功能性优化,为上层系统设计提供了更强的灵活性.RDMA 具有独特的数据传输模式和极高的数据传输性能,与此同时,其引入的扩展性等问题正在逐步解决.与 RDMA 紧耦合的分布式系统软件设计将为大数据处理和存储带来新的机遇.

参 考 文 献

[1] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets [C] //Proc of the 10th USENIX Workshop on Hot Topics in Cloud Computing. Berkeley, CA: USENIX Association, 2010: No. 10

[2] Färber F, Sang K C, Primsch J, et al. SAP HANA database: Data management for modern business applications [C] //Proc of the 31st ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2012: 45-51

[3] Wikipedia. RDMA [OL]. [2018-04-14]. https://en.wikipedia.org/wiki/Remote direct memory access

[4] Decandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store [C] //Proc of the 21st ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2007: 205-220

[5] Stuedi P, Trivedi A, Pfefferle J, et al. Crail: A high-performance I/O architecture for distributed data processing [J]. IEEE Data Engineering Bulletin, 2017, 40(1): 38-49

[6] Song W, Gkountouvas T, Birman K, et al. The freeze-frame file system [C] //Proc of the 7th ACM Symp on Cloud Computing. New York: ACM, 2016: 307-320

[7] 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 the 30th Int Conf on Supercomputing. New York: ACM, 2016: No. 8

[8] Mitchell C, Geng Yifeng, Li Jinyang. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store [C] //Proc of the 22nd USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2013: 103-114

[9] Kalia A, Kaminsky M, Andersen D G. Using RDMA efficiently for key-value services [C] //Proc of the 38th ACM Conf on SIGCOMM. New York: ACM, 2014: 295-306

[10] Wang Yandong, Zhang Li, Tan Jian, et al. Hydradb: A resilient RDMA-driven key-value middleware for in-memory cluster computing [C] //Proc of the 17th Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2015: No. 22

[11] Su Maomeng, Zhang Mingxing, Chen Kang, et al. RFP: When RPC is faster than server-bypass with RDMA [C] //Proc of the 12th European Conf on Computer Systems. New York: ACM, 2017: No. 1

[12] Wang Yandong, Meng Xiaoqiao, Zhang Li, et al. C-hint: An effective and reliable cache management for RDMA-accelerated key-value stores [C] //Proc of the ACM Symp on Cloud Computing. New York: ACM, 2014: No. 23

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

[14] Cassell B, Szepesi T, Wong B, et al. Nessie: A decoupled, client-driven key-value store using RDMA [J]. IEEE Transactions on Parallel & Distributed Systems, 2017, 28(12): 3537-3552

[15] Dragojević A, Narayanan D, Nightingale E B, et al. No compromises: Distributed transactions with consistency, availability, and performance [C] //Proc of the 25th Symp on Operating Systems Principles. New York: ACM, 2015: 54-70

[16] 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

[17] Wei Xingda, Shi Jiaxin, Chen Yanzhe, et al. Fast in-memory transaction processing using RDMA and HTM [C] //Proc of the 25th Symp on Operating Systems Principles. New York: ACM, 2015: 87-104

[18] Chen Yanzhe, Wei Xingda, Shi Jjaixin, et al. Fast and general distributed transactions using RDMA and HTM [C] //Proc of the 7th European Conf on Computer Systems. New York: ACM, 2016: No. 26

[19] Dragojevic A, Narayanan D, Castro M. RDMA reads: To use or not to use? [J]. IEEE Data Engineering, 2017, 40(1): 3-14

[20] Kaminsky A K M, Andersen D G. Design guidelines for high performance RDMA systems [C] //Proc of the 25th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2016: 437-450

[21] Stuedi P, Trivedi A, Metzler B, et al. DaRPC: Data center RPC [C] //Proc of the 5th ACM Symp on Cloud Computing. New York: ACM, 2014: No. 15

[22] An Zhongqi, Du Hao, Li Qiang, et al. Memcached optimization on high performance I/O technology [J]. Journal of Computer Research and Development, 2018, 55(4): 864-874 (in Chinese)

(安仲奇,杜昊,李强,等. 基于高性能 I/O 技术的 Memcached 优化研究[J]. 计算机研究与发展,2018,55(4):864-874)

[23] Lu Youyou, Shu Jiwu, Chen Youmin, et al. Octopus: An RDMA-enabled distributed persistent memory file system [C] //Proc of the 26th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 773-785

[24] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable dram alternative [J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 2-13

[25] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology [J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 24-33

[26] Strukov D B, Snider G S, Stewart D R, et al. The missing memristor found [J]. Nature, 2008, 453(7191): 80-83

[27] Lu Youyou, Shu Jiwu, Sun Long. Blurred persistence in transactional persistent memory [C] //Proc of the 31st Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2015: No. 13

[28] 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

[29] Talpey T. Remote persistent memory access-workload scenarios and RDMA semantics [OL]. [2017-03-31]. https://www.openfabrics.org/images/eventpresos/2017presentations/405_RemotePM_TTalpey.pdf

Chen Youmin, born in 1993. PhD candidate. His main research interests include distributed systems and storage systems.

Lu Youyou, born in 1987. PhD, assistant professor. Member of CCF. His main research interests include non-volatile memories and file systems.

Luo Shengmei, born in 1971. PhD. Member of CCF. His main research interests include cloud storage, cloud computing, and big data.

Shu Jiwu, born in 1968. Professor and PhD supervisor. Outstanding member of CCF. His main research interests include nonvolatile memory systems and technologies, network (cloud/big data) storage systems, storage security and reliability, and parallel and distributed computing.