计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220865
60(11): 2681−2695, 2023

在网存储系统研究综述

汪　庆　李俊儒　舒继武
（清华大学计算机科学与技术系　北京　100084）
（q-wang18@mails.tsinghua.edu.cn）

Survey on In-Network Storage Systems

Wang Qing, Li Junru, and Shu Jiwu
（*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084）

Abstract　Programmable network devices, represented by programmable switches and SmartNICs, are increasingly used in modern data centers to support the execution of customized data processing logic on network data transmission paths, which brings new opportunities for building high-performance in-network storage systems. However, programmable network devices have hardware resource limitations (e.g., limited expressive powers and small memory space), and there are still many challenges to fully utilize their advantages and maximize the acceleration of storage systems. We systematically review the recent research progress of in-network storage systems. First, we describe the hardware architecture and performance characteristics of programmable network devices, and based on this, we summarize two major challenges in building high-performance in-network storage systems: 1) division of labor between hardware and software, 2) fault tolerance of the storage systems. Then, according to the tasks performed by programmable network devices (data caching, distributed coordination, request scheduling, data aggregation), we classify and describe existing in-network storage systems. Moreover, using several examples of in-network storage systems, we analyze corresponding design difficulties and software technologies. Finally, we indicate open problems that need to be explored in further research on in-network storage systems, including switch-NIC collaboration, data security, multi-tenancy, and automatic function offloading.

Key words　distributed storage; programmable switch; SmartNICs; in-network storage systems

摘　要　以可编程交换机和智能网卡为代表的可编程网络设备在数据中心被越来越广泛地应用，它们支持在网络数据传输路径上执行自定义的数据处理逻辑，这为构建高性能的在网存储系统带来了新的机遇. 然而，可编程网络设备的硬件资源限制较多，如何充分发挥它们的优势、最大限度地加速存储系统仍面临着诸多挑战. 系统地综述了在网存储系统的研究进展，首先介绍了可编程网络设备的硬件结构与性能特征，并基于此总结了构建高性能在网存储系统面临的两大挑战：软硬件分工以及系统容错. 然后根据可编程网络设备执行的任务（缓存、协调、调度、聚合）对现有的在网存储系统进行分类和阐述，并以多个在网存储系统为实例分析对应的设计难点以及软件技术. 最后指明了在网存储系统进一步研究中需要着重探索的问题，包括交换机与网卡的协同、安全、多租户以及自动卸载.

关键词　分布式存储；可编程交换机；智能网卡；在网存储系统

中图法分类号　TP302

收稿日期：2022-10-09；**修回日期**：2023-03-06

基金项目：国家自然科学基金项目 (61832011)
This work was supported by the National Natural Science Foundation of China (61832011).

通信作者：舒继武（shujw@tsinghua.edu.cn）

传统的存储系统以 CPU 为中心，CPU 处理所有的存储逻辑. 然而，在如今的后摩尔时代，通用 CPU 的处理能力难以提升，这极大限制了传统存储系统的性能上限. 与此同时，传统存储系统面临来自上层应用和下层 I/O 硬件的前所未有的压力. 一方面，全球的数据量持续增长，尤其是需要实时处理的数据：据国际数据公司预测，2025 年实时性数据将达到 50 ZB[1]. 另一方面，I/O 硬件的性能也大幅提升：高速网卡的带宽已经超过 100 Gbps，如双端口 ConnectX-6 网卡的聚合带宽有 400 Gbps[2]；基于 NVMe(non-volatile memory express)协议的闪存盘能提供百万级的 IOPS(input/output operations per second). 当传统存储系统尝试利用这些高速 I/O 硬件来实时处理海量数据时，CPU 不可避免地成为了系统瓶颈，导致 I/O 硬件资源难以被充分发挥.

近些年来，以可编程交换机和智能网卡为代表的可编程网络设备在数据中心逐渐普及，这为缓解 CPU 瓶颈、构建以数据为中心的在网存储系统(in-network storage systems)带来了新的机遇. 可编程交换机与智能网卡支持用户自定义数据处理与转发过程，且具有天生的位置优势：可编程交换机是存储服务器之间的数据交换中枢，而智能网卡是存储服务器的出入口. 在网存储系统将存储功能分工至可编程交换机或智能网卡上，在网络通路上处理与存储数据，因此既能够充分发挥现代网络硬件低延迟、高带宽的优势，又能够减少数据的移动开销. 学术界和工业界近些年设计了大量的在网存储系统，从不同角度加速存储任务，获得显著的性能提升.

本文对在网存储系统进行综述，主要贡献有 3 个方面：

1)从可编程网络设备的硬件特性出发，总结了构建高性能在网存储系统面临的两大挑战：网络硬件和存储软件如何高效分工，以及在网存储系统如何容错. 此外，归纳了在硬件和软件层次针对这些挑战的现有解决方法.

2)根据可编程网络设备执行的任务，对现有的在网系统进行分类，包括在网数据缓存系统、在网数据协调系统、在网数据调度系统以及在网数据聚合系统. 针对每类在网存储系统，以具体例子分析其对应的设计难点以及软件技术.

3)对在网存储系统进行了展望，指出了研究人员未来需要着重对交换机与网卡协同、多租户、安全以及自动卸载 4 个方向进行深入探索，才能使得在网存储系统被广泛部署.

1 背景介绍

在网存储系统目前主要使用可编程交换机与智能网卡进行存储功能的加速，下面分别介绍它们的硬件结构以及性能特征.

1.1 可编程交换机

与功能固定的传统交换机不同，可编程交换机支持用户自定义网络协议以及网络包的转发逻辑. 现有的商用可编程交换机大多基于可重配置匹配表(reconfigurable match table, RMT)的硬件架构[3]，如图 1 所示. 在 RMT 架构下，可编程交换机具有多条入口流水线(ingress pipeline)和出口流水线(egress pipeline)；每条流水线包含多个阶段(stage). 网络包首先进入某个入口流水线，按照阶段顺序被交换机硬件部件处理，然后被转发至某个出口流水线，最后从对应的交换机端口流出. RMT 架构的核心可编程部件是动作-匹配表(match action table)，它定义了当网络包满足什么格式时(match 字段)，对网络包做何种操作(action 字段)：比如，用户可创建基于 IP 地址的转发表，根据网络包的 IP 地址字段进行网络包内容的修改以及路由. 此外，RMT 架构还支持有状态的数据存储；具体地，交换机具有数十 MB 的 SRAM 空间，用户能够通过定义长度固定的寄存器数组(register array)来使用这些 SRAM 空间，并在处理网络包的过程中对寄存器数组进行读写.

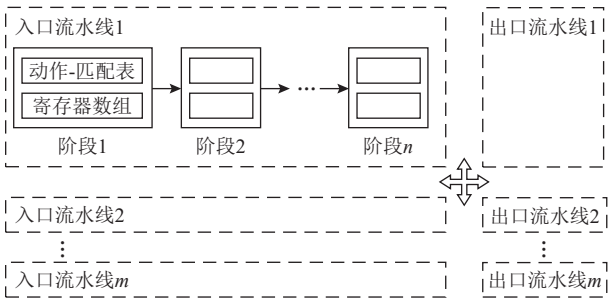

Fig. 1 Hardware architecture of RMT programmable switch

图 1 RMT 可编程交换机的硬件架构

可编程交换机拥有高带宽、低延迟的性能特征. 相比于服务器网卡，可编程交换机的聚合带宽能高出 1 个数量级：以英特尔公司的 Barefoot Tofino 系列可编程交换机[4]为例，第 1 代 Tofino 芯片聚合带宽可达到 6.4 Tbps，支持 100 Gbps 的网络端口；最新一代 Tofino 3 芯片聚合带宽可达 25.6 Tbps，支持 400 Gbps 的网络端口. 可编程交换机能够线速(line rate)转发

网络包，因此其流水线内处理逻辑的复杂度具有上限，比如不支持循环操作.

1.2 智能网卡

智能网卡是具有可编程芯片的网卡，用户可自定义网络包的处理过程，并可灵活地卸载通用 CPU 的任务. 按照数据通路模式，智能网卡可以被分为 on-path 和 off-path 这 2 类：在 on-path 智能网卡中，可编程芯片位于网络包收发路径上，即每个网络包都需要经过可编程芯片处理；而在 off-path 智能网卡中，可编程芯片在网络包收发路径之外，存在额外的交换部件来控制网络包是否被送往可编程芯片. on-path 智能网卡由于数据通路更短，网络包处理延迟比 off-path 智能网低；然而，on-path 网卡通常更难以编程.

现有智能网卡配备的可编程芯片主要分为 4 种：ARM CPU、网络处理器（network processor, NP）、专用集成电路（application specific integrated circuit, ASIC）以及现场可编程门阵列（field-programmable gate array, FPGA），它们在性能、易用性和表达能力方面各有优劣，如表 1 所示.

Table 1 Comparison Among Four Types of SmartNICs

表 1 4 种智能网卡的对比

类型	性能	易用性	表达能力
基于 ARM CPU	低	高	高
基于 NP	高	中	中
基于 ASIC	高	低	低
基于 FPGA	高	低	高

具体而言，基于 ARM CPU 的智能网卡表达能力强且易于编程，这是因为 ARM CPU 是通用处理器，且可以直接利用现有的工具链生态（如编程语言和编译器）. 但 ARM CPU 的性能比较差，针对网络包的处理能力远不如相同频率下的 x86 架构 CPU；例如，英伟达 BlueField 2 智能网卡[5] 具有 8 颗 ARMv8 A72 CPU 核心，运行频率为 2.0 GHz. 根据我们的测试，当运行基于 RDMA 的用户态测试程序，发送/接收尺寸为 64 B 的网络包时，使用这 8 颗 ARM 核心处理网络包的吞吐量只有 8 颗 2.2 GHz 英特尔 Xeon CPU 核心的 1/2. 基于 NP 的智能网卡包含大量专为网络处理设计的可编程网络处理核心，相比于 ARM CPU，它的网络处理延迟更低且并发度更高：比如 Netronome 的 Agilio CX 网卡[6] 具有 60 个 NP 核心，并且每个 NP 核心上能同时运行 8 个独立的硬件线程. 尽管性能很高，基于 NP 的智能网卡编程难度要高于 ARM CPU，且某些处理逻辑无法表达. 基于 ASIC 的智能网卡通过高速芯片进行网络处理，同时针对网络、存储、虚拟化等任务提供一定程度的可编程接口；例如英伟达 ConnectX 系列智能网卡[7] 支持上层应用自定义网络流表规则. 基于 FPGA 的智能网卡直接使用 FPGA 处理网络包，FPGA 性能高且表达能力强，但 FPGA 编程门槛比较高，典型的代表有英伟达的 Innova 智能网卡[8] 系列. 更多关于智能网卡的介绍可参见文献 [9]. 值得注意的是，相比于文献 [9]，本文侧重智能网卡和可编程交换机在存储系统中的应用与加速.

2 在网存储系统的设计挑战

可编程交换机与智能网卡为存储系统的设计带来了巨大机遇，但同时构建高性能的在网络存储系统也面临着诸多挑战，其中最主要的有：1）网络硬件和存储软件如何高效分工；2）在网存储系统如何容错. 下面依次阐述.

2.1 网络硬件和存储软件的分工

可编程网络设备的表达能力受限，不如通用 CPU. 首先，它们的计算能力受限，以可编程交换机为例，为了保证线速，用户能够定义的操作步骤有限，不支持循环和浮点数计算，而且所有的操作必须要被表达成动作-匹配表的模式；此外，在可编程交换机中，具有依赖的操作必须被摆放在不同的流水线阶段中，而流水线的总阶段数有限，这进一步限制了用户能够表达的逻辑. 其次，可编程网络设备的可用内存小：可编程交换机只具有数十 MB 左右的 SRAM，而智能网卡的内存空间也一般不超过 16 GB；此外，在可编程交换机中，SRAM 在不同流水线和不同阶段之间无法被共享，导致内存空间无法被充分利用.

但另一方面，存储系统具有大量复杂逻辑. 比如存储系统中的分布式协议（如共识协议、分布式提交协议）需要服务器之间进行多次网络消息的交互，流程复杂；存储系统中的崩溃一致性机制需要对存储介质进行多个步骤的更新操作，保证系统在崩溃之后能够恢复到一致性的状态. 此外，存储系统使用大量内存空间维护状态，比如文件系统需要几十 GB 甚至更大空间的页缓存用于加速文件访问性能. 如何缓解受限的可编程网络设备表达能力与复杂的存储系统逻辑之间的矛盾，进行网络硬件和存储软件之间高效分工是在网存储系统面临的一大挑战.

目前的研究工作主要通过存储软件抽象和网络硬件扩展 2 个方面应对这一挑战. 在存储软件抽象方

面，考虑到可编程网络设备的资源受限，现有研究工作通常不会将存储系统的所有功能全部卸载至可编程网络设备上，而是对存储系统功能进行细粒度地抽象划分，将加速效果最佳的部分实现在可编程网络设备中，而其他部分仍由存储软件实现. 比如，分布式共享内存系统 Concordia[10] 将并发控制(读写锁的管理)卸载至可编程交换机，与此相关的元数据所占空间较小但对系统性能影响大.

在网络硬件扩展方面，存在一系列研究工作提升可编程网络设备的表达能力，可具体分为 3 类，如表 2 所示.

Table 2 Summary of Switch Expressiveness Optimizations

表 2 交换机表达能力优化方案总结

优化方面	主要措施	典型方案
内存空间	提高内存共享程度	dRMT[11]，MP5[12]，TEA[13]
计算部件	优化数据表达格式	FPISA[14]
执行语义	提供新型编程模型	Packet transactions[15]

为了提高可编程交换机的内存利用率，思科公司的研究者对 RMT 架构进行扩展，提出了解耦的可重配置匹配表(disaggregated reconfigurable match table)架构[11]，将内存从交换机流水线移动至中心化的资源池中，使得不同的流水线阶段可以共享内存资源；进一步地，普渡大学的研究者提出 MP5 可编程交换机架构[12]，允许用户逻辑能够同时利用多个流水线中的内存资源；此外，卡内基梅隆大学的研究者提出 TEA 架构[13]，支持可编程交换机通过 RDMA 网络协议访问服务器集群中的 DRAM 资源，以扩充数据存储空间. 为了提高可编程交换机的计算能力，伊利诺伊大学厄巴纳-香槟分校的研究者提出一种适用于交换机高效处理的浮点数表示方法 FPISA[14]，并为此扩展了 RMT 架构. 为了扩充可编程交换机的执行语义，麻省理工学院的研究者提出网络包事务(packet transactions)的抽象[15]，为可编程交换机中的部分执行逻辑片段提供原子性保证.

2.2 在网存储系统的容错

在网存储系统引入了可编程网络设备，这不可避免地扩大了系统的故障域(failure domain). 具体而言，在传统存储系统中，系统状态仅存储在服务器内存和外存中；而对于在网存储系统，除了服务器内存和外存，系统状态还会存留在智能网卡的 DRAM 和可编程交换机的 SRAM 中. 当网卡和交换机发生崩溃时，它们存储的数据将会丢失，如何让系统容忍这种新型故障是在网存储系统的另一大设计挑战.

目前的研究工作主要从软件设计和硬件支持 2 方面应对这一挑战. 在软件设计方面，现有在网存储系统尽量只在可编程网络设备中存储软状态(soft state)，即可以从别处(如后端服务器)恢复的状态. 例如，NetCache[16] 在可编程交换机中缓存热点的键值数据，而这些数据均在后端分布式键值服务器中有最新的版本，因此可编程交换机的崩溃不会导致任何键值数据的丢失；R2P2[17] 在可编程交换机中维护每台服务器的队列长度信息，该信息丢失后可以通过询问服务器来重构.

在硬件支持方面，主要的相关研究工作有弗吉尼亚大学提出的 PMNet[18] 和卡内基梅隆大学提出的 RedPlane[19]. PMNet 将持久性内存用于可编程网络设备，以替代智能网卡的 DRAM 和可编程交换机的 SRAM；因此，可编程网络设备可以通过持久性地存储数据来容忍掉电等异常事件. 此外，PMNet 设计了一种新型的数据更新协议，如图 2 所示：当可编程交换机收到客户端发送的数据更新请求时，将内容记录在交换机的持久性日志区，然后便可提前返回完成消息给客户端，而服务器异步地处理数据更新请求；通过该协议，客户端的请求延迟可以减半. RedPlane 则采用数据复制的方式进行可编程网络设备的容错：在 RedPlane 系统中，当可编程交换机需要修改内存状态时，它会生成包含修改数据的复制请求，并发送至多台服务器；这些服务器将修改数据存储到本地的 DRAM 中，以此容忍可编程交换机的崩溃.

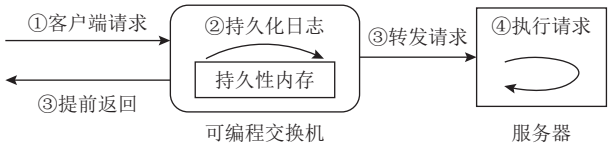

Fig. 2 Data update protocol of PMNet

图 2 PMNet 的数据更新协议

3 在网存储系统的分类与研究进展

基于可编程网络设备的在网存储系统支持在数据传输路径上执行存储任务，颠覆了传统以 CPU 为核心的存储系统架构思想. 根据可编程网络设备执行的任务，我们将在网存储系统分为 4 类：在网数据缓存系统、在网数据协调系统、在网数据调度系统以及在网数据聚合系统. 本节将依次介绍这 4 类在网存储系统，并详细分析典型的系统实例. 表 3 对这 4 类系统在多方面进行了对比.

Table 3 Comparison of Four Types of In-Network Storage Systems

表 3 4 种在网存储系统的对比

对比方面	在网数据缓存系统	在网数据协调系统	在网数据调度系统	在网数据聚合系统
网络硬件	交换机/网卡	交换机/网卡	交换机/网卡	交换机
网络硬件处理内容	数据	元数据	元数据	数据
网络硬件内存需求	高	低	低	高
复杂度	低	高	中	中
通用性	低	低	高	中
主要场景	键值缓存	分布式协议	远程过程调用	纠删码、AI 训练
典型系统	NetCache[16]，KV-Direct[20]	Concordia[10]，SwitchTx[21]	R2P2[17]，AlNiCo[22]	NetEC[23]

3.1 在网数据缓存系统

在网数据缓存系统利用可编程交换机和智能网卡的内存空间，在网络上进行数据的缓存或存储，以提供高吞吐量、低延迟的数据访问. 本节将着重介绍基于可编程交换机的 NetCache[16] 系统，以及基于智能网卡的 KV-Direct[20] 系统.

NetCache 由约翰霍普金斯大学提出，用于解决传统分布式内存键值系统的负载不均衡问题. 分布式内存键值系统将键值数据以某种规则分散在多台服务器的内存中，具有较好的水平扩展能力. 然而，现实世界中的负载往往是倾斜的，即少量热点的键值对（key-value pairs）会被频繁地访问，这会导致整个分布式内存键值系统负载不均衡：某些服务器承受了大量的请求，处于过载状态；而其他服务器处理的请求较少，处于空闲状态. 负载不均衡严重影响系统性能：过载的服务器限制了整个系统的总吞吐量，并导致对热点键值对的访问会经历较高的延迟. 为此，NetCache 利用可编程交换机缓存热点键值对，用于过滤对应的读请求，使得后台服务器处于负载均衡的状态.

图 3 展示了 NetCache 的架构. NetCache 系统由 4 个组件构成：1）存储服务器，将键值数据存储在 DRAM 中；2）客户端服务器，用于发送键值请求，包括查询（Get）、更新（Put）和删除（Del）；3）可编程交换机，缓存热点键值对以服务 Get 操作，并进行热点键值数据的统计；4）交换机控制器，用于向交换机中添加或删除键值对. 后端存储服务器、可编程交换机以及交换机控制器位于同一个机架中，因此所有的客户端请求均会流经该交换机.

在 NetCache 系统中，交换机与后端存储服务器协同处理客户端请求. 当交换机接收到客户端发送的 Get 请求时，会先查询本地的键值缓存；若缓存命中，则将对应的值返回给客户端；若不命中，Get 请求会被路由至请求中指定的存储服务器，然后存储服务器查询本地 DRAM，返回对应的值至客户端.

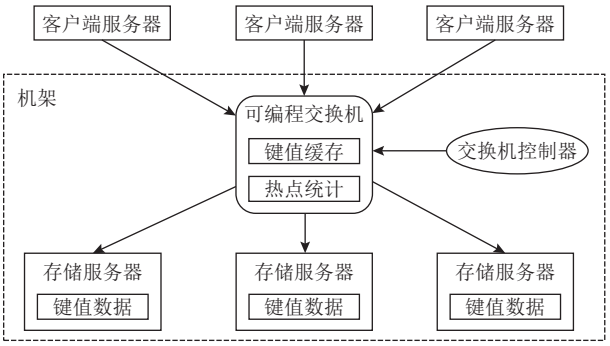

Fig. 3 Architecture of NetCache

图 3 NetCache 的架构

当交换机接收到客户端发送的 Put 或 Del 请求时，也会查询本地缓存；若缓存命中，则将该键值对标记为无效，确保未来的 Get 操作不会读到过期的数据. 然后，请求会被路由至对应的存储服务器，存储服务器更新或删除对应键值对，并回复客户端. 同时，若该键值对属于交换机的缓存，存储服务器会异步地发送更新请求至交换机，交换机更新缓存中的该键值对内容，并标记为有效.

NetCache 通过动作-匹配表和寄存器数组在交换机中构建了键值对的缓存结构. 具体地，NetCache 维护了一张查询表，每个条目对应一个键值对：其中的 match 字段是键，action 部分会产生出一份位图结构和下标. 位图结构用于定位值被存储在哪些流水线阶段的寄存器数组中，下标用于定位值在寄存器数组中的哪个元素中. 通过这种组织方式，NetCache 可以支持变长的值；但对于键，由于其存储在 match 字段中，只能是定长的. 由于交换机 SRAM 空间有限，NetCache 缓存在交换机中的键值对数量很少（64 000）；但这已经能达到良好的负载均衡效果，这得益于之前的研究结论[24]：在倾斜的负载下，只需要将 $N\times\mathrm{lb}\,N$

份最频繁被访问的键值对缓存住,就能够保证后端的负载均衡,其中 N 是后端存储服务器的数目.

NetCache 也将热点统计功能实现在交换机中. 具体地,NetCache 利用寄存器数组构建了一个计数器数组和一个计数-最小略图(count-min sketch)[25]. 其中计数器数组用于统计每份被缓存的键值对的访问次数;计数-最小略图用于近似统计未被缓存的键值对的访问次数. 当某一键值对的访问次数到达某一阈值时,交换机将其报告给交换机控制器. 交换机控制器向交换机的查询表中插入对应的条目. 为防止交换机将某一热点键值对重复地向控制器报告,NetCache 在交换机中构建了一个布隆过滤器[26],用于近似记录哪些键值对已向控制器报告.

得益于交换机的极高聚合带宽,NetCache 在 Get 操作密集的倾斜负载下能提高系统吞吐量 1 个数量级. 但 NetCache 只支持单个机架,具有扩展性问题;后续工作 DistCache[27] 通过独立缓存分配和 2 次随机选择策略将 NetCache 扩展到大规模集群. 此外,除了负载均衡场景,研究者们设计了高可靠的键值存储系统 NetChain[28]. NetChain 通过链式复制协议(Chain Replication Protocol)[29] 将每份键值对存储在多台交换机的 SRAM 上,存储方式与 NetCache 类似. 表 4 对 NetCache, DistCache, NetChain 这 3 个基于可编程交换机的缓存系统进行了总结对比.

Table 4 Summary of Caching Systems Based on Programmable Switches

表 4 基于可编程交换机的缓存系统总结

系统	针对场景	可靠性	扩展性	支持写
NetCache	负载均衡	低	低	否
DistCache	负载均衡	中	高	否
NetChain	协调服务	高	高	是

KV-Direct 系统由微软研究院提出,它将内存键值系统的功能卸载至基于 FPGA 的智能网卡上. 随着网络带宽的持续提升,服务器端 CPU 变成键值系统的主要性能瓶颈:从吞吐量上看,单个 CPU 核心处理键值操作的上限低于 8 MOPS;从延迟上看,CPU 的软件调度和排队经常导致较大的延迟波动. 因此,KV-Direct 的目标是通过高性能 FPGA 智能网卡完全消除键值访问路径上的服务器 CPU 的介入. 在 KV-Direct 中,键值数据被存储在 CPU 的 DRAM 中,智能网卡通过 DMA 操作访问 CPU DRAM. KV-Direct 重新设计了高效的索引结构、执行引擎以及缓存机制,以充分释放 FPGA 的硬件性能.

在索引结构方面,KV-Direct 构建了高效的链式哈希索引. 索引由固定数目的哈希桶构成;每个桶中存有若干哈希槽;哈希槽主要包含键值对的地址. 当插入新的键值对时,若对应的哈希桶满了,网卡则会分配新的哈希桶链接至原有哈希桶尾部,形成链式结构. 由于智能网卡与 CPU DRAM 之间为 PCIe 连接,带宽有限,且存在几百纳秒的延迟,该链式哈希索引做出了 3 个设计以减少 DMA 次数:1)每个哈希槽中内嵌了键的部分哈希值,在查询键值对时,网卡会首先进行比对,当不匹配时,则不需要读取 CPU DRAM 中的键值对. 2)尺寸较小的键值对(如 10 B)直接被存储在哈希桶中,避免了一次 DMA 访问. 3)KV-Direct 为尺寸较大的键值对和哈希桶设计了专门的分配器;分配器的主要逻辑运行在 CPU 上,但分配器中的空闲空间元数据被缓存在网卡里,因此,大部分的空间分配和释放无需网卡执行 DMA 操作.

在执行引擎方面,KV-Direct 通过乱序执行保证系统在并发语义正确的同时达到极致的吞吐量. 具体地,网卡在 FPGA 中维护了保留站(reservation station),用于追踪所有正在执行的键值操作. 保留站将哈希冲突的键值操作集合维护成队列结构,网卡按队列顺序执行这些操作,以保证并发执行的正确性. 这种基于哈希冲突的方式避免了保留站进行键的比较,极大节省了 FPGA 上的物理资源;但会引入伪冲突,即某些键不同的键值操作会被序列化执行. 此外,保留站会缓存键值操作产生的最新值,用于支持数据转发(data forwarding),提高执行效率和减少 DMA 操作次数.

在缓存机制方面,KV-Direct 将部分数据缓存在智能网卡中的 DRAM 空间(网卡 DRAM). 由于网卡 DRAM 的带宽较低(十几 GBps),传统的缓存方法会导致整个系统性能受限于网卡 DRAM 带宽. 为此,KV-Direct 设计了负载调度器,保证系统能够同时利用网卡 DRAM 和 PCIe 的带宽. 具体地,整个 CPU DRAM 空间被划分成可缓存部分和不可缓存部分. 对可缓存部分的访问会执行缓存逻辑,即相关数据会被缓存在网卡 DRAM 中,当缓存命中时会消耗网卡 DRAM 带宽但保存了 PCIe 带宽. 对于不可缓存部分,网卡对其所有访问需要 DMA 操作,只消耗 PCIe 带宽. 通过调整 2 部分 CPU DRAM 空间的比例,KV-Direct 使得整个系统可利用的带宽最大. KV-Direct 系统使用一张网卡就能达到 180 MOPS 的吞吐率,并且尾延迟低于 10 μs;同时,相比于基于 CPU 的键值系统,KV-Direct 仅使用 1/3 的功耗. KV-Direct 的不足也很明显:未考

虑分布式场景以及系统容错.

除了 KV-Direct, 复旦大学还提出了基于智能网卡卸载的分布式内存键值系统 SKV[30], 其利用智能网卡中的 ARM CPU 执行系统的错误检测和副本操作.

3.2 在网数据协调系统

分布式存储系统由多台存储服务器构成, 并运行分布式协议进行服务器之间的协调(比如分布式缓存一致性与分布式事务). 传统的分布式协调方式消耗大量网络流量和服务器 CPU 资源, 效率低下. 而在网数据协调系统将分布式协议卸载至可编程网络设备上; 得益于可编程交换机和智能网卡在网络路径上的优势, 在网数据协调系统能够极大地提高分布式存储系统的性能. 本节将着重介绍分布式缓存一致性和分布式事务处理相关的研究工作.

1)分布式缓存一致性

分布式共享内存系统 Concordia[10] 由清华大学提出, 利用可编程交换机加速缓存一致性协议. 基于高速网络(如 RDMA)的分布式共享内存系统能支持图计算等大规模内存计算应用. 尽管目前网络带宽很高(如 100 Gbps), 但仍低于本地内存的访问, 且网络延迟远高于内存延迟. 因此, 在分布式共享内存系统中, 为了减少数据的远程访问, 每台服务器一般具有本地缓存. 如何保证不同服务器缓存之间的一致性是个极具挑战的问题, 而现有的缓存一致性协议需要服务器之间进行昂贵的分布式协调, 极大地降低了系统在数据共享时的性能: 基于目录的缓存一致性协议引入多次网络往返, 且当热点数据存在时, 服务器会成为系统瓶颈; 基于广播的缓存一致性协议会导致网络和 CPU 资源的消耗, 这是由于每次缓存一致性请求需要被广播到所有的服务器处理. Concordia 利用可编程交换机在网络中枢上的位置优势, 设计了高效的在网分布式缓存一致性协议. 图 4 展示了其架构.

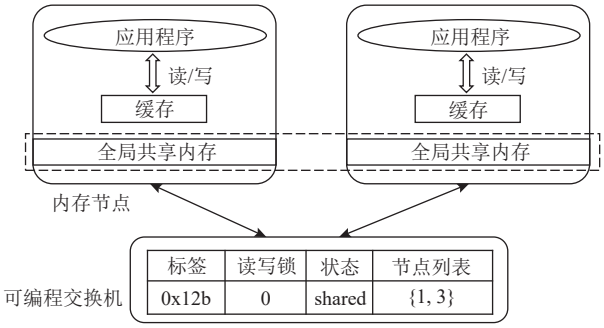

Fig. 4 Architecture of Concordia

图 4 Concordia 的架构

Concordia 是机架级别的分布式共享内存系统, 主要由内存节点和可编程交换机 2 个组件构成. 内存节点是普通的服务器, 它们将自己的 DRAM 空间分为了 2 部分: 全局共享内存以及本地缓存. 所有内存节点的全局共享内存一起构成了全局地址空间, 可以被共享读写; 内存节点的本地缓存用于缓存最近访问的全局内存, 由此减少网络访问开销. 可编程交换机记录了缓存块的信息, 并执行缓存一致性逻辑. 具体地, 如图 4 所示, 针对每份缓存块, 可编程交换机通过寄存器数组记录了相关元数据, 包括: ①缓存块的标签(tag), 即对应的全局内存地址的高位, 用于唯一标识该缓存块和索引其他的元数据; ②读写锁, 用于序列化冲突的缓存一致性请求, 比如 2 个内存节点对同一份缓存块进行写操作; ③状态, 用于描述该缓存块所处的状态, 如未共享的(unshared)、共享的(shared)或者被修改的(modified); ④节点列表, 即持有该缓存块的内存节点的集合. 利用这些元数据, Concordia 在交换机中高效地处理缓存一致性请求.

这里通过一个例子描述 Concordia 系统中的缓存一致性请求执行流程. 标签为 0x12b 的缓存块的元数据如图 4 所示, 它的状态为共享的, 被内存节点 1 和节点 3 缓存在本地; 此外, 读写锁字段为空, 代表当前不存在针对该缓存块的一致性操作. 当节点 2 需要写该缓存块对应的全局地址时, 会发现本地缓存缺失, 因此产生一个写缺失(write miss)请求发送至可编程交换机; 该请求中会捎带标签 0x12b. 当可编程交换机接收到该请求时, 首先通过标签查询到该缓存块的其他元数据. 然后, 可编程交换机尝试获得对应的写锁, 若失败, 则返回错误消息给节点 2, 节点 2 将进行请求重试. 若持锁成功, 可编程交换机根据请求类型(即 write miss)和缓存块状态(即 shared)做出相应的动作, 在这个例子中就是将一致性请求多播至节点列表对应的服务器(即内存节点 1 和节点 3). 当内存节点 1 和节点 3 收到一致性请求时, 会将本地持有的缓存块无效化, 然后发送回复给节点 2. 此外, 其中 1 个节点(节点 1 或者节点 3)会将该缓存块的内容也发送至节点 2, 这由交换机随机挑选. 当节点 2 收到所有的回复时, 将缓存块数据存放在本地缓存中, 便可继续进行对全局地址空间的读写操作. 节点 2 会发送异步的解锁请求给交换机, 请求中会捎带标签 0x12b. 当交换机接收到该解锁请求时, 则会释放该缓存块的锁, 并修改对应元数据: 状态修改为 modified, 节点列表修改为 {2}. 从该例子可以看出, Concordia 的在网缓存一致性协议在数据读写的关键

路径上只需 1 次网络往返,且不会像广播协议一样引入额外的网络和 CPU 资源的消耗.

为了解决可编程交换机内存空间小的问题,Concordia 提出动态地将缓存块的一致性管理权限在交换机与内存节点之间迁移.具体地,交换机只管理活跃缓存块的缓存一致性;对于很少触发缓存一致性协议的缓存块,它们的一致性由内存节点管理退化成传统的基于目录的协议.为了保证迁移过程中整个系统的并发正确性,Concordia 设计了细粒度的迁移协议,每次迁移只会阻塞对一份缓存块的访问.

除了 Concordia,还有一些研究工作也利用可编程交换机保证不同服务器的数据之间的一致性.华盛顿大学提出了分布式对象系统 Pegasus[31] 以解决负载不均衡问题;与 NetCache 使用缓存的方式不同,Pegasus 将热点的对象复制至多台服务器,以均摊相应的访问.在 Pegasus 系统中,可编程交换机记录热点对象所在的服务器列表,当发生写请求时更新列表,以保证后续的读操作能获得最新数据.此外,耶鲁大学提出的 Mind[32] 系统针对的是分离式内存场景,即多个计算节点访问远程内存池,并将数据缓存至本地.Mind 利用可编程交换机保证不同计算节点的缓存之间处于一致的状态.此外,约翰霍普金斯大学提出 NetLock[33],通过交换机实现高性能的锁管理器,用于上层应用保证数据一致性.NetLock 将锁资源存储在交换机的内存中,因此相比于传统基于服务器的设计,能够提高吞吐量 1 个数量级;NetLock 在交换机中为锁请求维护了队列结构,以保证能够公平地服务冲突的锁请求,降低上层应用的尾延迟.表 5 对上述在网缓存一致性系统进行了总结对比.

Table 5 Comparison of In-Network Cache Coherence Systems

表 5 在网缓存一致性系统对比

系统	针对场景	系统接口	通用性
Concordia	分布式共享内存	用户态读写	中
Pegasus	分布式对象	对象接口	低
Mind	分离式内存	OS 虚拟内存	高

2)分布式事务处理

分布式事务系统将数据划分在不同服务器中,并通过分布式并发控制和提交协议提供事务语义.这些协议存在很高的协调开销,包括网络通信、CPU 排队等,而这些开销位于事务提交的关键路径上,会导致高延迟和高冲突,严重降低系统性能.因此,研究人员利用可编程网络设备的处理能力和得天独厚的位置卸载事务系统中的协调操作,减少网络往返和 CPU 消耗.本节介绍基于可编程交换机的 Eris[34] 和 SwitchTx[21] 系统,以及基于智能网卡的 Xenic[35] 系统.

华盛顿大学提出了基于在网排序的分布式事务系统 Eris[34],通过将序号向量生成器卸载到可编程交换机中,为无依赖事务(independent transactions)[36] 指定序号向量,仅需 1 次网络通信即可完成无依赖事务,因此显著减少了分布式事务的协调开销.在 Eris 系统中,数据被分散至多个数据分区,每个数据分区由多台存储数据副本的存储服务器组成.序号向量中的 1 个序号对应 1 个数据分区.图 5 展示了 Eris 系统中无依赖事务的提交流程:

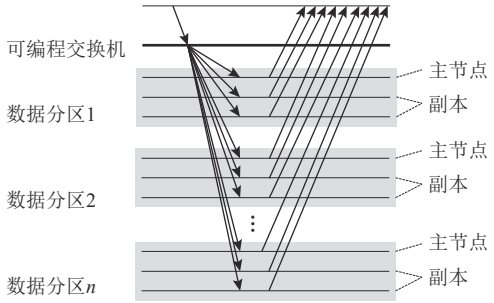

Fig. 5 Transaction commit process of Eris

图 5 Eris 的事务提交过程

在 Eris 系统中,客户端将事务请求发送给可编程交换机,可编程交换机根据事务涉及的数据分区为其生成序号向量,同时将事务请求广播至所有涉及到的数据分区;数据分区中的存储服务器按照序号顺序执行请求.交换机生成的序号向量为无依赖事务建立一个可线性化的执行顺序,而不需要额外的网络协调.为支持通用事务(比如读写集未知),Eris 将其分解为多个无依赖事务.Eris 存在着扩展性受限的问题:当系统规模扩大时中心化的序号向量生成器会成为性能瓶颈,限制系统的总吞吐量.

清华大学提出了可扩展的在网分布式事务系统 SwitchTx[21],将分布式事务协调过程抽象为多次“收集-分发”操作的组合,并将这些操作卸载到集群中的多个可编程交换机中.相较于 Eris 系统,SwitchTx 系统避免了单点瓶颈问题.图 6 展示了 SwitchTx 系统的组件构成以及事务处理流程.

SwitchTx 系统由 4 个组件构成:①客户端服务器,用于发起事务并完成事务的执行;②存储服务器,将数据存储在 DRAM 中并维护数据的版本和锁信息;③备份服务器,存储数据的备份以提供系统容错能力;④可编程交换机,完成事务提交过程多个存储服务器和备份服务器之间的协调.

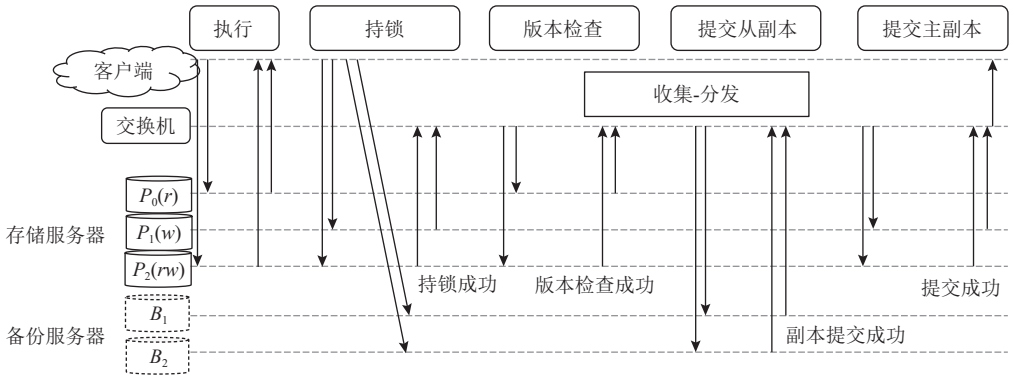

Fig. 6 Transaction processing workflow of SwitchTx

图 6 SwitchTx 的事务处理流程

SwitchTx 系统将乐观并发控制协议[37]和主从备份副本协议的协调任务卸载到交换机. SwitchTx 中的事务提交可以分为 5 个阶段，包括执行阶段、持锁阶段、版本检查阶段、提交从副本阶段以及提交主副本阶段. 这些阶段是依次进行的，其中的协调任务是同步来自当前阶段的参与者的结果，并使事务进入下一个阶段. SwitchTx 通过在交换机中设计“收集-分发”原语完成协调任务，即通过“收集”步骤统计当前阶段的执行结果，通过“分发”步骤使事务进入下一阶段. 以从持锁阶段到版本检查阶段的协调任务为例，当交换机收集到所有写参与者的包含持锁成功信息的网络包，它将向所有读参与者广播版本检查的请求.

SwitchTx 将“收集-分发”原语扩展到多个交换机，这得益于 SwitchTx 中的协调是去中心化的，即不同的事务可以使用不同的交换机来执行“收集-分发”操作. 具体地，SwitchTx 为每个事务选择一组交换机来执行“收集-分发”操作，多个交换机和服务器构成树形的拓扑结构，其中最高层交换机为树的根，其余交换机为中间节点，存储服务器、备份服务器和客户端为叶子节点. 在“收集”步骤中，非根交换机从其子节点(交换机或服务器)收集消息，并将结果发送到其父节点. 当根节点交换机收集到足够数目的消息时，则开始执行“分发”步骤，即将请求广播给对应事务阶段的参与者. 在“分发”步骤中，消息沿着树结构从根节点多播到叶子节点.

SwitchTx 只需在可编程交换机中为每个“收集-分发”操作实现一个计数器，即用寄存器数组记录“收集”步骤中已收集到的网络包数量，因此对交换机存储空间的占用小. 当可编程交换机接收到网络包时，根据其包头中标记的事务编号取出对应的计数器，增加计数器数值并与网络包携带的阈值信息对比. 若计数器数值小于阈值，交换机丢弃网络包；若计数器数值等于阈值，交换机开始“分发”步骤，并重置计数器. 其中计数器以事务编号为索引存储在哈希表中.

此外，华盛顿大学提出了分布式事务系统 Xenic[35]，利用基于 ARM CPU 的 on-path 智能网卡进行 2 方面的卸载：事务的协调任务以及数据并发索引. 具体地，Xenic 在客户端网卡存储事务的临时状态，完成事务协调任务，减少协调过程中的通信时延. Xenic 的服务端网卡利用网卡内存存储热点数据以及锁信息，消除远程数据访问的 PCIe 开销；同时利用智能网卡的 ARM CPU 处理复杂数据访问，以减少服务端 CPU 开销. Xenic 设计了智能网卡内存与主机内存协同的 Robin hood 哈希索引结构[38]，减少网卡处理远程数据访问请求时的 DMA 次数. 事务执行过程中的副本操作也完全由网卡执行，这进一步降低了主机 CPU 的消耗. 除了事务系统，一些分布式文件系统也将副本操作卸载至智能网卡：比如 LineFS[39]将包括数据副本的文件系统后台操作卸载至基于 ARM CPU 的 off-path 智能网卡，由此释放客户端的 CPU 资源，减少文件系统与计算任务之间的干扰. 表 6 对上述在网分布式事务系统进行了总结对比.

Table 6 Comparison of In-Network Transaction Systems

表 6 在网事务系统对比

系统	网络设备	扩展性	通用性	并发控制协议
Eris	交换机	低	低	确定性执行
SwitchTx	交换机	高	高	乐观并发控制
Xenic	网卡	高	高	乐观并发控制

3.3 在网数据调度系统

分布式存储系统的服务器数目不断增加，且每台服务器中的 CPU 核心数目也在持续增长. 这些趋势导致了 2 个关键问题：多 CPU 核心并发处理请求

容易产生资源冲突;服务器或CPU核心之间存在负载不均衡.在网数据调度系统利用可编程交换机和智能网卡的中心化位置优势,在网络上进行数据访问请求的调度,旨在减少多核并发的资源竞争开销,或保证服务器以及CPU核心之间的负载均衡.本节将着重介绍针对多核并发的AlNiCo系统[22],以及针对负载均衡的R2P2系统[17].

1)针对多核并发的在网数据调度系统

在单机内存事务存储系统中,系统接收来自网络的事务请求并将它们分派给工作线程.由于事务请求包括对多份数据的读/写操作,工作线程需要使用并发控制协议来执行事务,以保证事务的隔离性.但是,当存在冲突的2个事务并发执行时,事务会被中止或阻塞.中止会导致事务重试,阻塞可能会级联阻塞更多事务,中止和阻塞均会导致系统性能下降.因此事务系统需要通过合理的调度模块将冲突的事务调度到相同的工作线程,来最小化并发事务之间的冲突.现有基于软件的调度算法包括静态数据划分和基于图划分,静态数据划分无法处理动态负载,而基于图划分的方法需要对请求做批处理,即积攒大量请求后处理,引入额外的延迟.

清华大学提出了在网事务请求调度系统AlNiCo,将智能网卡与事务软件协同设计,利用智能网卡低延迟地执行调度算法,并通过软件反馈机制根据负载变化更新调度算法.图7展示了AlNiCo的整体架构:客户端通过网络连接到服务端的智能网卡并发送事务请求;智能网卡上的FPGA执行调度算法选择合适的工作线程分发请求;工作线程在执行事务请求的同时统计负载信息,用于定期更新调度算法的参数.

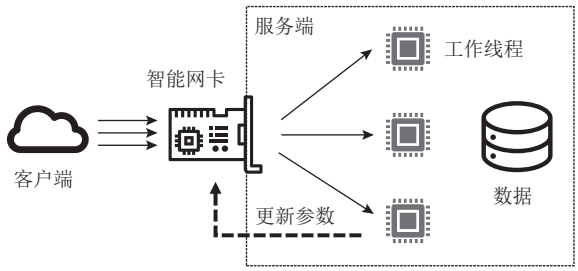

Fig. 7 Architecture of AlNiCo

图7 AlNiCo的架构

AlNiCo将调度信息分为3种类型:1)请求信息,即请求访问的数据和对应的访问方式(读或写);2)工作线程信息,即每个工作线程上正在执行或将要执行的事务;3)全局状态信息,包括数据热点等工作负载特征.AlNiCo系统将这3种调度信息编码为向量,并将调度算法设计为向量计算,以此来充分发挥网卡上FPGA的计算加速能力.具体地,AlNiCo系统用特征向量表示请求信息,向量中每个特征代表一个数据分区是否被访问和如何被访问(读/写);工作线程信息用该线程正在执行/排队的事务的特征向量的合集表示;AlNiCo将数据热点信息表示为特征向量中每个特征的权重.基于上述编码方法,AlNiCo设计了冲突感知的调度算法,利用FPGA高效地比较事务的特征向量与工作线程的特征向量之间的相似程度(越相似则代表两者冲突越大),为事务选择最可能产生冲突的工作线程.

AlNiCo在Mellanox Innova-2网卡[7]上实现了通用的可调度的远程过程调用(remote procedure call, RPC)框架.Innova-2是一款基于FPGA的off-path智能网卡.AlNiCo的RPC框架不仅可以用于实现事务请求调度算法,而且可以用于其他基于请求内容调度的应用场景.图8展示了AlNiCo系统的RPC框架:

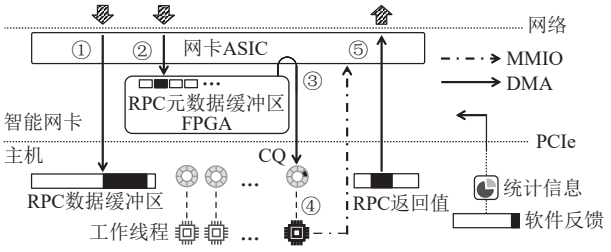

Fig. 8 RPC framework of AlNiCo

图8 AlNiCo的RPC框架

该框架通过RDMA实现服务端与客户端之间的通信,以及通过DMA和MMIO实现FPGA与主机之间的通信.AlNiCo为每个RPC请求添加固定格式的元数据,客户端可以根据调度需求将请求信息编码到元数据,在AlNiCo系统的事务调度中,元数据为事务特征向量.该RPC框架采用元数据与数据分离的设计,其在服务器主机内存上维护数据缓冲区,在FPGA上维护元数据缓冲区.RPC的处理流程为:客户端同时发起2个单边RDMA写请求分别将RPC数据(①)和RPC元数据(②)发送到它们各自的缓冲区.FPGA上的调度模块轮询元数据缓冲区,以确定新请求的到达,之后调度模块会执行调度算法选择一个工作线程.在做出调度决策之后,调度模块通过DMA操作向工作线程的接收完成队列(CQ)添加条目(③),来通知被选定的工作线程.CQ中的条目仅包含RPC数据的地址,而不包括RPC的元数据.由于RDMA写操作是保证顺序的,因此当工作线程读

取到新的 CQ 条目(④)后，可以从 RPC 数据缓冲区获得完整的 RPC 数据. 最后，工作线程执行事务请求，并通过 RDMA 写操作(⑤)回复客户端. 此外，CPU 向智能网卡暴露了 FPGA 可读的配置缓冲区，软件可以将调度器的新配置写入缓冲区，FPGA 上的调度器定期读取配置并更新.

2)针对负载均衡的在网数据调度系统

远程过程调用 RPC 框架被广泛应用在数据中心的存储系统中，是实现系统服务等级目标(service level objective, SLO)的核心组件. 随着系统规模的扩大，RPC 框架为保证可扩展性，需通过调度保证多节点之间以及节点内多核之间的负载均衡；同时，现代数据中心的存储系统如内存缓存的服务延迟仅为数百微秒，这就要求负载均衡调度本身具有极低延迟.

瑞士联邦理工学院提出 R2P2[17]，它是基于可编程交换机的负载均衡 RPC 框架. 该框架使用 JBSQ(join-bounded-shortest-queue)负载均衡策略；JBSQ 的调度质量接近单队列模型，且实际扩展性更好. R2P2 在交换机中为每个服务器维护有限深度的队列，深度通常取 2 或 3，代表该服务器未执行完的 RPC 请求数目. 图 9 展示了 R2P2 的系统架构和 RPC 处理流程.

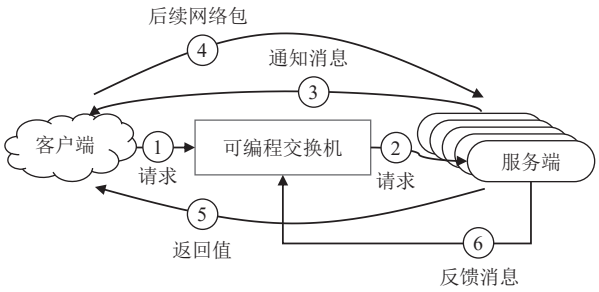

Fig. 9 Architecture of R2P2

图 9 R2P2 的架构

R2P2 通过以下阶段处理 RPC 请求：①客户端向交换机发送 RPC 请求，若 RPC 请求的尺寸较大，由多个网络包构成，则只需要发送 1 个网络包，并携带客户端的信息以及该 RPC 请求的唯一标识符；②可编程交换机读取服务器的队列信息，若存在某些服务器的队列空闲，则选择队列最短的作为目标服务器，并将该 RPC 请求转发给它和并更新队列信息；若所有服务器队列都满了，交换机则循环该 RPC 请求，直到存在空闲队列；③如果 RPC 请求由多个网络包构成，则目标服务器将发送通知消息至客户端；④客户端在收到通知消息后，将 RPC 请求的剩余部分发送到该目标服务器，该过程完全绕过交换机中的调度逻辑；⑤目标服务器执行完 RPC 请求后，将处理结果返回给客户端；⑥服务器向交换机发送反馈消息，通知自身当前的状态，包括队列空闲状况和可用性等.

由于可编程交换机的流水线硬件架构的限制，R2P2 设计了多轮的方式选择最短的队列. 具体地，在队列最大深度为 n 的系统设置下，每个 RPC 请求第 1 次经过交换机时，交换机进行第 1 轮操作，寻找队列长度小于等于 0 的服务器(队列长度信息被存储在可编程交换机的寄存器数组中)，若找到则确定了目标服务器；否则，该 RPC 请求会再次被重新送回交换机流水线，进行第 2 轮操作，寻找队列长度小于等于 1 的服务器；以此类推，若在第 n 轮操作时依旧找不到队列长度小于等于 $n-1$的服务器时，则说明所有的服务器队列均已满，该 RPC 请求将在交换机内循环，直到存在某个服务器的队列长度小于 n.

除了 R2P2 之外，约翰霍普金斯大学还提出了 RackSched[40]，这是结合服务器之间调度和 CPU 核心之间调度的统一调度框架，专为机架级别的应用设计. 服务器之间的调度由可编程交换机实现：交换机为 RPC 请求选择目的服务器以达到服务器之间的负载均衡，与 R2P2 不同，RackSched 采用树状归约算法选择最短队列从而避免网络包在交换机内的重新传输. 此外，可编程交换机追踪每台服务器上的实时负载. CPU 核心之间调度通过中心化的调度线程来实现，借鉴了单机调度系统 Shinjuku[41].

上述 R2P2 和 RackSched 系统虽然支持在服务器之间进行负载均衡调度，但忽略了数据一致性，即在存储系统中，某些数据的最新版本只存储在某些服务器中，因此交换机无法对相关 RPC 请求进行任意调度. Harmonia[42] 和 FLAIR[43] 这 2 个系统利用可编程交换支持保证数据一致性的请求调度. 具体地，它们针对的场景是副本协议，1 份数据通过共识协议被冗余地存储在不同服务器(包括 1 个主副本服务器以及多个从副本服务器)，交换机将客户端的读请求高效地调度至具有最新版本数据的服务器上. 这里的主要设计难点在于交换机如何与共识协议结合，识别哪些服务器具有读请求所需的最新数据. Harmonia 在可编程交换机中维护了细粒度哈希表，用于实时记录哪些数据存在并发的写请求，对于这些数据的读请求只能被路由至主副本，对于其余数据的读操作可被调度至任一从副本. FLAIR 将整个数据范围切分成大量的分区，在交换机中记录每个分区的稳定状态：当某个分区存在进行中的写请求时，则被标记成不稳定，对应的读请求只能被路由至主副本；对于稳定分区的读请求能以负载均衡的方式被调度至

某一从副本. 表 7 对本节涉及的在网数据调度系统进行了总结对比.

Table 7 Comparison of In-Network Date Scheduling Systems

表 7 在网数据调度系统对比

系统	网络设备	调度对象	请求类型	一致性
AlNiCo	网卡	多核	事务请求	高
R2P2	交换机	多机	通用请求	低
RackSched	交换机	多核/多机	通用请求	低
Harmonia	交换机	多机	副本读	高
FLAIR	交换机	多机	副本读	高

3.4 在网数据聚合系统

在网数据聚合系统主要利用可编程交换机带宽极高且位于网络中枢的特点, 在交换机内进行数据处理, 以提高存储系统的性能. 本节主要介绍基于可编程交换机的纠删码系统 NetEC[23].

NetEC 由清华大学提出, 其可利用可编程交换机加速纠删码系统的数据重构性能. 相比于副本机制, 纠删码在保证相同数据可靠性的同时能够大幅度降低存储空间的使用. 例如, 在广泛使用的里德-所罗门码(RS 码)[44] 中, 针对 k 份原始数据块, 能够计算出 r 份校验块; 这 k 份原始数据块能够通过这 $k+r$ 份中的任意 k 份重构出来, 因此最多能够容忍 r 个错误. 纠删码的主要缺陷是数据重构性能低下. 假设分布式存储系统使用 $RS(k, r)$ 编码, 当某台服务器崩溃后, 恢复丢失的任何一份数据块需要从 k 台其他服务器读取相应的数据块, 然后进行向量点积计算重构. 此时, 整个系统的重构速度受限于接收端网卡带宽: 假设网卡带宽为 B, 恢复大小为 M 的数据需要的时间为 $k\times M/B$. NetEC 的基本思想是在交换机上完成纠删码的重构过程, 由此克服接收端网卡的带宽瓶颈并消除 CPU 的计算开销. 图 10 展示了其架构.

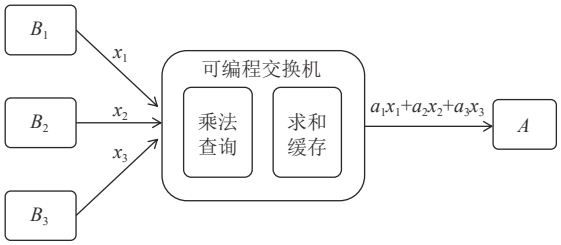

Fig. 10 Architecture of NetEC

图 10 NetEC 的架构

在图 10 中的例子里, 交换机根据服务器 B_1, B_2, B_3 中的数据进行纠删码的重构, 重构的结果被存储在服务器 A 中. B_1, B_2, B_3 将数据块发送至可编程交换机, 可编程交换机主要完成 2 项任务: 1)针对每个字(word), 即图 10 中的 x_i, 进行伽罗华域的乘法, 将其乘以常数 a_i; 2)将来自不同服务器的乘法结果进行求和, 获得重构的结果 $a_1x_1+a_2x_2+a_3x_3$. 由于交换机的硬件限制, NetEC 对这 2 项任务进行了精心设计. 由于交换机不支持乘法运算, NetEC 将乘法运算过程转换为查表和加法. 具体地, 在 NetEC 中, 每个 word 为 16 b, 对于 16 b 的每个数字, 可编程交换机预先存储了对数查询表和指数查询表. 因此, 对于 x_i 和 a_i 相乘, 交换机先通过查询对数查询表获得 $\mathrm{lb}(x_i)$ 的值, 而 $\mathrm{lb}(a_i)$ 的值是预先知道的; 然后, 将 $\mathrm{lb}(x_i)$ 和 $\mathrm{lb}(a_i)$ 相加, 获得 $\mathrm{lb}(a_ix_i)$; 最后, 通过查询指数查询表, 获得 a_ix_i 的值. 对于求和操作, NetEC 设计了求和缓存结构, 对于每个 word, 分配初始值为 0 的缓存区, 当计算完 a_ix_i 后, 交换机将 a_ix_i 与现有缓存区的值相加并更新缓存区. 同时, 交换机维护了位图, 用于记录来自哪些服务器的 word 已经完成了乘法和求和. 若所有服务器的 word 都已被处理, 最后的重构结果将被发送至服务器 A. 为了防止不同发送端的数据传输不同步导致交换机内求和缓存结构空间占用过大, NetEC 复用了 TCP 协议栈的功能, 保证重构过程中发送端服务器在交换机中暂存的求和结果不超过 TCP 接收窗口的大小. NetEC 被集成进 HDFS[45] 中, 相比于原有纠删重构方法, 提升重构速度最高达 9 倍, 且完全消除了重构过程的 CPU 开销.

除了纠删码场景, 利用可编程交换机进行数据聚合还能够加速分布式机器学习训练系统. 微软研究院提出了 SwitchML[46] 系统, 将机器学习训练过程中的模型参数聚合卸载至可编程交换机. 针对可编程交换机不支持浮点数计算的问题, SwitchML 设计了服务器与交换机协同设计的方法: 服务器将需要聚合的浮点参数进行量化, 转换成定点数, 因此交换机只需要进行定点数的聚合. 此外, 清华大学提出了 ATP 系统[47], 利用多台交换机协同加速机器学习的训练任务, 且能高效支持多个训练任务共同运行的多租户场景; 沙特阿拉伯阿卜杜拉国王科技大学提出了针对稀疏训练任务的数据聚合系统 OmniReduce[48], 并将部分聚合算法卸载至可编程交换机; 其他工作如 iSwitch[49] 和 Flare[50] 设计了加速数据聚合的定制化交换机硬件架构, 其中 iSwitch 采用了 FPGA 硬件, Flare 采用了 PsPIN[51] 硬件. 此外, Flare 进一步支持用户自定义聚合操作处理的数据类型. 表 8 对上述在网数据聚合系统进行了对比总结.

Table 8 Comparison of In-Network Data Aggregation Systems

表 8 在网数据聚合系统对比

系统	交换机硬件	支持稀疏数据	支持自定义类型
NetEC	RTM	否	否
SwitchML	RTM	否	否
ATP	RTM	否	否
OmniReduce	RTM	是	否
iSwitch	FPGA	否	否
Flare	PsPIN	是	是

4 总结与展望

本文首先从可编程网络硬件(包括可编程交换机和智能网卡)的特性出发,展开分析了构建在网存储系统面临的挑战,并对现有研究工作详细地分类与剖析. 现有研究工作利用可编程网络硬件对存储系统的不同模块进行加速,包括数据缓存、协调、调度以及聚合,能够显著提高存储系统的性能. 然而,研究人员仍然需要在4个方面进行深入探索,才能让在网存储系统广泛普及到数据中心和超算中心.

1)交换机与网卡协同. 现有的在网存储系统大多孤立地使用可编程交换机或者智能网卡,无法做到全方位的存储功能卸载. 而未来的在网存储系统应该是全编程的:可编程交换机和智能网卡协同工作、互相补充. 其中可编程交换机执行服务器之间的任务,智能网卡执行服务器内部的任务. 例如,对于在网缓存系统,可利用交换机缓存全局最热的数据,而利用智能网卡缓存服务器中较热的数据,以达到整体性能的最优. 当交换机与网卡协同设计时,存储系统的故障域将进一步扩大,这需要引入新的高效容错机制.

2)多租户. 当在网存储系统被部署至云环境时,需高效地支持多租户,即多租户之间要进行资源的共享和隔离. 具体地,多个租户需分时复用可编程交换机和智能网卡的计算、内存资源;同时,当出现资源竞争时,多个租户之间需达到较好的性能隔离,不会相互影响. 支持多租户需要编译器和网络硬件体系结构的共同支持,为不同租户卸载的存储功能高效分配可编程网络设备的硬件资源. 例如,对于在网数据聚合系统,该如何分配可编程交换机的内存空间和带宽,以满足多个租户的服务等级目标. 目前,已有少量研究工作利用智能网卡进行了存储虚拟化的探索,例如芝加哥大学提出的LeapIO系统[52]. 在多租户环境下,在网存储系统的容错将变得愈发复杂,需考虑某个在网存储系统的失效不会影响其他租户系统的可用性.

3)安全. 目前越来越多的网络数据为了安全考虑被加密,此时就需要可编程交换机和智能网卡能够高效地处理加密的数据. 在存储系统软件设计方面,我们需要首先分析清楚哪些数据需加密,比如用户请求;哪些无需加密,比如存储系统的元数据. 在网络硬件设计方面,需要让交换机和网卡支持同态加密,在加密的网络数据上进行处理和计算.

4)自动卸载. 从头构建可商用的高可靠在网存储系统极其困难,需要大量的工程代码和测试验证. 如果能够将现有成熟的存储系统如Memcached[53],Ceph[54]中的某些模块自动卸载至可编程交换机和智能网卡,就能既利用现有的系统代码,又能享受到可编程网络设备带来的性能红利. 这需要研究自动卸载技术,自动分析现有存储系统代码并将某些模块自动卸载至可编程网络设备. 这里面存在诸多技术挑战,比如如何识别对哪些模块的卸载会带来性能收益,如何保证部分模块被卸载后整体系统功能上依旧正确.

作者贡献声明: 汪庆负责文献的搜集整理、论文整体架构的设计和论文主要内容的撰写;李俊儒负责论文部分内容的撰写;舒继武负责论文结构的讨论和修改.

参 考 文 献

[1] Seagate. The digitization of the world: From edge to core [EB/OL]. [2022-09-20].https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[2] Nvidia. ConnectX-6 [EB/OL]. [2022-09-20].https://www.nvidia.com/en-us/networking/ethernet/connectx-6/

[3] Bosshart P, Gibb G, Kim H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99−110

[4] Intel. Intel Tofino intelligent fabric processors [EB/OL]. [2022-09-20].https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-3-product-brochure.html

[5] NVIDIA. NVIDIA BlueField data processing units [EB/OL]. [2022-09-20].https://www.nvidia.com/en-us/networking/products/data-processing-unit/

[6] Netronome. Agilio CX SmartNICs [EB/OL]. [2022-09-20].https://www.netronome.com/products/agilio-cx/

[7] Nvidia. ConnectX SmartNICs [EB/OL]. [2022-09-20].https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/

[8] Nvidia. Innova-2 Flex [EB/OL]. [2022-09-20].https://www.nvidia.com/en-au/networking/ethernet-adapters/

[9] Ma Xiaoxiao, Yang Fan, Wang Zhan, et al. Survey on smart network interface card[J]. Journal of Computer Research and Development, 2022, 59(1): 1−21 (in Chinese)
(马潇潇, 杨帆, 王展, 等. 智能网卡综述[J]. 计算机研究与发展, 2022, 59(1): 1−21)

[10] Wang Qing, Lu Youyou, Xu Erci, et al. Concordia: Distributed shared memory with in-network cache coherence[C]//Proc of the 19th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2021: 277−292

[11] Chole S, Fingerhut A, Ma Sha, et al. dRMT: Disaggregated programmable switching[C/OL]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2017 [2023-02-09].https://doi.org/10.1145/3098822.3098823

[12] Shrivastav V. Stateful multi-pipelined programmable switches [C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2022: 663−676

[13] Kim D, Liu Zaoxing, Zhu Yibo, et al. TEA: Enabling state-intensive network functions on programmable switches[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2020: 90−106

[14] Yuan Yifan, Alama O, Fei Jiawei, et al. Unlocking the power of inline floating-point operations on programmable switches[C]//Proc of the 19th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2022: 683−700

[15] Sivaraman A, Cheung A, Budiu M, et al. Packet transactions: High-level programming for line-rate switches[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2016: 15−28

[16] Jin Xin, Li Xiaozhou, Zhang Haoyu, et al. NetCache: Balancing key-value stores with fast in-network caching[C]//Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 121−136

[17] Kogias M, Prekas G, Ghosn A, et al. R2P2: Making RPCs first-class datacenter citizens[C]//Proc of the 44th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2019: 863−880

[18] Seemakhupt K, Liu Sihang, Senevirathne Y, et al. PMNet: In-network data persistence[C]//Proc of the 48th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 804−817

[19] Kim D, Nelson J, Ports D R K, et al. RedPlane: Enabling fault-tolerant stateful in-switch applications[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2021: 223−244

[20] Li Bojie, Ruan Zhenyuan, Xiao Wencong, et al. KV-Direct: High-performance in-memory key-value store with programmable NIC[C]//Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 137−152

[21] Li Junru, Lu Youyou, Zhang Yiming, et al. SwitchTx: Scalable in-network coordination for distributed transaction processing[C]//Proc of the 48th Int Conf on Very Large Databases. New York: ACM, 2022: 2881−2894

[22] Li Junru, Lu Youyou, Wang Qing, et al. AlNiCo: SmartNIC-accelerated contention-aware request scheduling for transaction processing[C]//Proc of the 47th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2022: 951−966

[23] Qiao Yi, Kong Xiao, Zhang Menghao, et al. Towards in-network acceleration of erasure coding[C]//Proc of the Symp on SDN Research. New York: ACM, 2020: 41−47

[24] Fan Bin, Lim H, Andersen D G, et al. Small cache, big effect: Provable load balancing for randomly partitioned cluster services[C]//Proc of the 2nd ACM Symp on Cloud Computing. New York: ACM, 2011: 264−275

[25] Cormode G, Muthukrishnan S. An improved data stream summary: The count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1): 58−75

[26] Luo Lailong, Guo Deke, Ma R T B, et al. Optimizing Bloom filter: Challenges, solutions, and comparisons[J]. IEEE Communications Surveys & Tutorials, 2018, 21(2): 1912−1949

[27] Liu Zaoxing, Bai Zhihao, Liu Zhenming, et al. DistCache: Provable load balancing for large-scale storage systems with distributed caching[C]//Proc of the 17th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 143−157

[28] Jin Xin, Li Xiaozhou, Zhang Haoyu, et al. NetChain: Scale-free sub-RTT coordination[C]//Proc of the 15th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 35−49

[29] Van Renesse R, Schneider F B. Chain replication for supporting high throughput and availability[C]// Proc of the 6th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2004: 91−104

[30] Sun Shangyi, Zhang Rui, Yan Ming, et al. SKV: A SmartNIC-offloaded distributed key-value store[C]//Proc of IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2022: 132−142

[31] Li Jialin, Nelson J, Michael E, et al. Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories[C]//Proc of the 14th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 387−406

[32] Lee S, Yu Yanpeng, Tang Yupeng, et al. Mind: In-network memory management for disaggregated data centers[C]//Proc of the 28th ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2021: 488−504

[33] Yu Zhuolong, Zhang Yiwen, Braverman V, et al. NetLock: Fast, centralized lock management using programmable switches[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2020: 126−138

[34] Li Jialin, Michael E, Ports D R K. Eris: Coordination-free consistent transactions using in-network concurrency control[C]//Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 104−120

[35] Schuh H N, Liang Weihao, Liu Ming, et al. Xenic: SmartNIC-Accelerated Distributed Transactions[C]//Proc of the 28th ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2021: 740−755

[36] Cowling J, Liskov B. Granola: Low-overhead distributed transaction

coordination[C]//Proc of the 37th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2012: 223−235

[37] Kung H T, Robinson J T. On optimistic methods for concurrency control[J]. ACM Transactions on Database Systems, 1981, 6(2): 213−226

[38] Celis P, Larson P A, Munro J I. Robin hood hashing[C]//Proc of the 26th Annual Symp on Foundations of Computer Science. Piscataway, NJ: IEEE, 1985: 281−288

[39] Kim J, Jang I, Reda W, et al. LineFS: Efficient SmartNIC offload of a distributed file system with pipeline parallelism[C]//Proc of the 28th ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2021: 756−771

[40] Zhu Hang, Kaffes K, Chen Zixu, et al. RackSched: A microsecond-scale scheduler for rack-scale computers[C]//Proc of the 14th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 1225−1240

[41] Kaffes K, Chong T, Humphries J T, et al. Shinjuku: Preemptive scheduling for μ second-scale tail latency[C]//Proc of the 16th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2019: 345−360

[42] Zhu Hang, Bai Zhihao, Li Jialin, et al. Harmonia: Near-linear scalability for replicated storage with in-network conflict detection[C]//Proc of the 45th Int Conf on Very Large Databases. New York: ACM, 2019: 375−388

[43] Takruri H, Kettaneh I, Alquraan A, et al. FLAIR: Accelerating reads with consistency-aware network routing[C]//Proc of the 17th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 723−737

[44] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems[J]. Software: Practice and Experience, 1997, 27(9): 995−1012

[45] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system[C]//Proc of the 26th Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2010: 133−142

[46] Sapio A, Canini M, Ho C Y, et al. Scaling distributed machine learning with in-network aggregation[C]//Proc of the 18th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2021: 785−808

[47] Lao C L, Le Yanfang, Mahajan K, et al. ATP: In-network aggregation for multi-tenant learning[C]//Proc of the 18th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2021: 741−761

[48] Fei Jiawei, Ho C Y, Sahu A N, et al. Efficient sparse collective communication and its application to accelerate distributed deep learning[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2021: 676−691

[49] Li Youjie, Liu Iou-Jen, Yuan Yifan, et al. Accelerating distributed reinforcement learning with in-switch computing[C]//Proc of the 46th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2019: 279−291

[50] De Sensi D, Di Girolamo S, Ashkboos S, et al. Flare: Flexible in-network allreduce[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2021: 14−29

[51] Di Girolamo S, Kurth A, Calotoiu A, et al. A RISC-V in-network accelerator for flexible high-performance low-power packet processing[C]//Proc of the 48th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 958−971

[52] Li Huancheng, Hao Mingzhe, Novakovic S, et al. LeapIO: Efficient and portable virtual NVMe storage on ARM socs[C]//Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 591−605

[53] Nishtala R, Fugal H, Grimm S, et al. Scaling Memcache at Facebook[C]//Proc of the 10th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2013: 385−398

[54] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system[C]//Proc of the 7th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2006: 307−320

Wang Qing, born in 1997. PhD. His main research interests include storage systems and memory systems.

汪　庆, 1997 年生. 博士. 主要研究方向为存储系统和内存系统.

Li Junru, born in 1997. PhD candidate. His main research interests include programmable network and distributed storage systems.

李俊儒, 1997 年生. 博士研究生. 主要研究方向为可编程网络和分布式存储系统.

Shu Jiwu, born in 1968. PhD, professor, PhD supervisor. His main research interests include non-volatile memory systems and technologies, storage security and reliability, and parallel and distributed computing.

舒继武, 1968 年生. 博士, 教授, 博士生导师. 主要研究方向为非易失内存存储系统与技术、存储安全与可靠性、并行与分布式计算.