网络接口卡应是操作系统的一部分。

Pengcheng Xu
ETH Zurich
Switzerland
pengcheng.xu@inf.ethz.ch

Timothy Roscoe
ETH Zurich
Switzerland
troscoe@inf.ethz.ch

摘要

network interface adapter (NIC) 是云服务器中的一个关键组件，占据着独特的位置。不仅网络性能对机器的有效运行至关重要，而且与计算加速器如 GPU 不同，网络子系统必须能够响应不可预测的事件，比如网络包的到来，并以最小的延迟与相应的应用程序端点进行通信。

当前的服务器堆栈方法在灵活性、效率和性能之间进行权衡：最快的内核绕过方法将核心专用于应用程序，在接收队列上忙等待等，而更适合动态工作负载组合的更灵活的方法会在数据路径上产生更大的软件开销。

然而，我们拒绝这种权衡，我们认为这是操作系统与 NIC 之间系统状态任意（且次优）划分的结果。相反，通过利用缓存一致性互连的特性，并将 NIC 紧密集成到内核中，我们可以实现一个令人惊讶的结果：在不牺牲基于内核的网络子系统的稳健性和动态适应性的情况下，提升 RPC 工作负载优于最快的内核绕过方法的性能。

CCS Concepts

• **Networks** → **Cloud computing**; • **Software and its engineering** → **Operating systems**; • **Computer systems organization** → *Processors and memory architectures.*

Keywords

远程过程调用，无服务器计算，网络，智能网卡，缓存一致性

HOTOS '25, Banff, AB, Canada

ACM ISBN 979-8-4007-1475-7/2025/05
https://doi.org/10.1145/3713082.3730388

ACM Reference Format:
Pengcheng Xu and Timothy Roscoe. 2025. 网络接口卡应是操作系统的一部分。. In *Workshop on Hot Topics in Operating Systems (HOTOS '25), May 14–16, 2025, Banff, AB, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3713082.3730388

1 介绍

NIC 是数据中心服务器操作的核心，负责与机器进行所有关键性能的通信。它在关键方面与其他设备有所不同：在正常运行期间会发生不可预测的事件（如数据包到达）。这与其他组件形成对比，例如 GPU（操作系统向其提交应用程序任务，行为相对可预测）或本地存储设备（操作系统发出请求，假设会收到响应）。

现代服务器中的网络特点在于操作系统、应用程序和 NIC 之间的网络状态有明显的划分：大致来说，NIC 持有流的（去）复用状态，而操作系统则持有与核心间任务调度相关的状态。这一点经过了多年的演变，现代 NICs（包括研究中提出的新型架构）编码了一系列关于信任、操作系统设计和应用程序的隐式和显式假设，在 section 2中我们对这些进行了调查。

令人惊讶的是，这些假设在高性能内核旁路架构中得以保留，这些架构试图将 NIC 更加贴近应用，或者将 NIC 与处理器核心集成的 CPU 设计。这些技术带来了性能上的提升，但以灵活性为代价：旁路通常依赖于相对固定的进程到核心和队列的分配以及忙等待来实现更高的速度。这在工作负载较为静态的情况下效果很好，但对于更动态的应用程序组合则应用有限。

本文的重点在于网络接收路径，尽管它也与传输路径密切相关。我们还关注 Remote Procedure Calls (RPCs)，无论是数据中心微服务还是无服务器功能调用。虽然有些较大，但大多数 RPC 请求和响应都是较小 [23]。我们的目标是利用这一见解来减少许多工作负载中小的 RPC

调用基本上为零的 CPU 周期开销，在一个架构中，该架构仍然支持比现代内核堆栈性能更好的动态工作负载。

尽管端到端延迟主要由 RPCs 的传播时间决定，*末端系统延迟*反映了调用消耗的 CPU 周期，因此是衡量软件栈效率（反序列化、解复用、函数调度等）的良好可测量代理。

我们建议内核旁路优化已达到极限，并主张采取一种截然不同、更加以操作系统为中心的方法，其中 NIC 是一个完整的可信组件*操作系统本身*。

我们的方法结合了几个新思路。首先，我们利用*缓存一致外设互连*和成熟的协议卸载技术，在用户模式 CPU 寄存器和 NIC 之间传输数据时没有内存访问开销且不会浪费能量在空转上。其次，我们使用相同的细粒度机制让内核保持 *NIC 与当前操作系统调度状态一致*，允许 NIC 总是将数据包导向正确的用户空间进程。最后，NIC 通过相同的轻量级机制收集负载信息并响应网络中新到达的数据包，即*请求操作系统重新调度进程*。

我们探讨这对硬件和软件意味着什么，以及我们在 Enzian 研究平台上构建的原型 NIC，劳伯霍恩 来展示这个想法。

2 传统的网卡范式

大多数服务器网络堆栈以及大部分 NIC 硬件设计都是基于 Figure 1 中的模型：传入的数据包通过 Direct Memory Access (DMA) 解复用并传输到一组基于描述符的队列之一，当操作系统停止轮询该队列时使用中断进行同步。DMA 使用经过 I/O Memory Management Unit (IOMMU) 或 System Memory Management Unit (SMMU) 翻译（和保护）的地址发生，并且可以利用解复用信息或其他启发式方法将中断引导到核心。

此模型历经数十年的发展，从最初在 Xerox Alto 中使用的非常简化的以太网接口模型演变而来，以处理连接到具有数百个核心的机器的 400Gb/s 网络链路。我们将在下面讨论最近的一些替代方案，但请注意，大多数内核绕过方法仍然类似于 Figure 1，只是将某些部分从操作系统内核移动到了应用程序用户空间。更详细地说，要将一个网络数据包转换为主机上的函数调用，必须发生一系列最基本的操作是：

(1) 读取数据包内容。
(2) 执行协议处理（校验和等）。
(3) 将数据包解复用到一个内存队列中。
(4) 中断一些 CPU 核心以通知操作系统、管理程序或客户机操作系统。
(5) 执行一些通用协议处理。
(6) 识别应处理消息的操作系统进程（或线程，或任务）。
(7) 找到一个（可能不同的）核心来执行此过程。
(8) 在核心上调度该过程。
(9) 如果需要，则切换上下文到该进程。
(10) 反序列化/解组参数和函数名称。
(11) 找到函数起始的地址。
(12) 跳至本指令。

一个典型的 NIC 执行步骤 1 到 4，然后将任务移交给操作系统或应用程序。

内核旁路 如 Arrakis [18], IX [3] 和 Demikernel [24] 等方法通过替换步骤 4 为自旋或轮询，并提前将应用程序进程绑定到内存队列，从而简化步骤 5 到 9，在延迟和吞吐量与灵活性及能效之间进行权衡。数据平面被移动到应用空间，而控制平面可以留在操作系统内核中或移至专用的核心（Shinjuku [12]，Shenango [17]，Caladan [7]）或用户空间进程（ghOSt [8]）。Snap [14] 则将 CPU 核心的一部分分配给应用程序，以提供一个统一但高度可配置的抽象层 NIC，从而允许快速部署新的网络堆栈功能。

然而，所有这些方法都保留了软件与 NIC 之间的相同分工；事实上，它们都类似于 Figure 1。主要的区别在于内核/用户空间边界（以及不同的接收路径阶段执行的位置）和控制平面的设计（作为独立组件在 CPU 上的软件中实现）。很大程度上，内核绕过将原本是操作系统功能的内容转化为应用级功能，将 *NIC 更加与用户应用程序集成*。

建筑领域的其他工作已经探索了紧密的**将 NIC 与 CPU 集成**。nanoPU [10] 将由 P4 处理的数据包直接传输到 RISC-V 核心的寄存器文件中，而 CC-NIC [22] 则使用 NUMA 服务器通过仿真来探索缓存一致外围互连对 NICs 的影响。这再次保持了相同的硬件/软件边界，同时极大地优化了硬件步骤 3 和 4。

与旁路一样，当工作负载相对静态、可以绑定到专用核心并且很少闲置时，这种方法效果很好。然而，当工作负载动态变化且终端数量远多于空闲核心时，将网

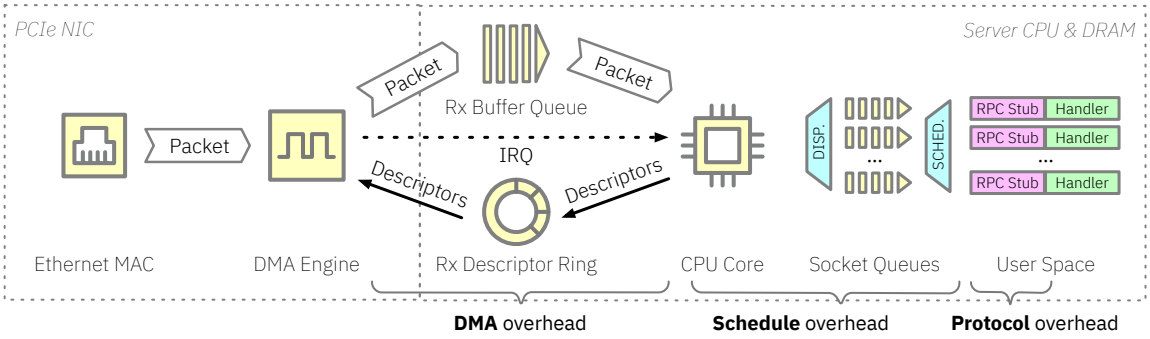

图 1: 传统 PCIe DMA 的 NIC 接收路径架构。

卡的解复用映射到核心上应用调度的前期成本很快就会变得繁琐。即使是更新的 Data Processing Unit (DPU) [1] 和 Infrastructure Processing Unit (IPU) [9] 系统也具有这些特性。

此外，紧密耦合 NIC 和 CPU 可能并不理想：网络部分不会与 CPU 同步发展，不同的工作负载具有非常不同的计算到网络 I/O 比例，因此将网卡保持为独立组件可以获得宝贵的灵活性。

3 为什么要这样划分？

硬件/软件边界的历 史稳定性在 NICs 中可以说是由于执行每一步所需的状 态。例如，步骤 10-12 需要应用特定的状态：参数格式、接口签名和代码布局。相比之下，步骤 5-9 如果不参考中央操作系统状态则无法执行。

一个关键因素是**操作系统不信任该 NIC**。内核开发者一直希望限制操作系统与 NIC [16] 之间的耦合，因为人们认为 NIC 并不能完全做到操作系统设计者所期望的那样。讽刺的是，结果仍然是设备驱动程序的复杂性不断增加，这是因为硬件供应商采用临时解决方案来向用户提供功能。这反过来意味着供应商添加到网卡的功能限于那可以很容易地暴露给用户。

此外，引入 IOMMUs 和 SMMUs 导致了一种理念：即尽可能不把 NIC 视为可信赖的设备。这是一处异常现象，因为像磁盘、CPU 内核、GPU 和 DRAM 这样的设备在很大程度上至少被操作系统的一部分所信任。造成这种情况的一个原因是人们对 SMMU 不同角色的混淆：一方面，在数据路径中提供一个方便的内存转换功能以促进设备直接传递给虚拟机，另一方面，要防火墙隔离运行在一个应用内核集上的内核与机器其余部分。

该哲学由诸如 RDMA 之类的协议构成，这些协议将 NIC 视为与主机操作系统关联较少的相对简单的设备，但仍能代表远程对等方执行内存访问操作，使用在多租户场景中最多只能算是简单的授权框架。从更侧重于操作系统的角度看待类似 RDMA 的功能，则认为 NIC 提供了靠近网络接口的到操作系统个额外的专用核心，可以执行有限数量的 RPC 操作。

一个相关因素是**架构研究人员喜欢忽略操作系统** [15]。用户应用程序则不同，因此有许多提案旨在加速步骤的子集 10-12 以减少内存延迟。Cereal [11] 提出了一个针对自定义消息格式的加速器；该加速器直接位于系统互连内部 CPU 封装上。Optimus Prime [19] 提出了一种与格式无关的转换架构，并专注于实现位于 CPU 封装内的系统互连上的加速器。Cerebros [20] 建立在 Optimus Prime 之上，旨在构建一个完全卸载的 RPC 框架。ProtoAcc [13] 则针对 Protocol Buffers，并通过直接连接到 RISC-V 核心流水线上的自定义 RoCC [2] 接口来实现加速。类似于内核旁路，这些主要目标是将应用程序静态分配给核心，并通过强加一些假设以减少步骤 5-9 来部分提高单个应用的性能。

在性能（核心的静态分配）和灵活性（更多的操作系统参与）之间的这种看似权衡的背后是**操作系统与 NIC 之间细粒度交互速度慢的误解：**，NIC 并不可信，而且难以沟通。在如 Receive-Side Scaling (RSS) 的情况下，目标是在不涉及操作系统的完全情况下提供卸载（例如负载均衡）。这种假设可能适用于 DMA 描述符环，但对于通过现代 PCI Express (PCIe) 对设备寄存器的加载/存储来说则少得多，而当通过新的、缓存一致的互连如 CXL.mem 3.0 访问设备时更是如此。

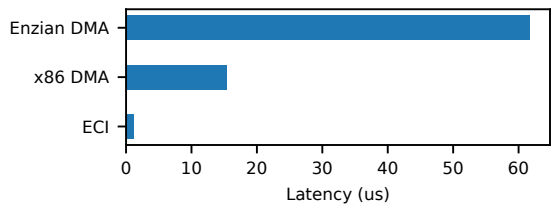

图 2: 64 字节消息的往返延迟。

4 打破僵局

我们正在追求一种不同的方法，以实现相对稳定的 RPC 和无服务器工作负载的性能优于当前的内核绕过方法，同时提供传统方法的所有灵活性，并且对于高度动态的工作负载有更好的效率。我们利用了三个新的见解。

首先，设备和核心之间的新**缓存一致性外设互连**可以彻底改变 CPU 和 NIC 之间的通信。示例包括 CXL.mem 3.0 [6]、CCIX [4]，以及 Enzian Coherence Interface (ECI) [5]。关键的是，这允许向设备发出轻量级信号：一个 NIC 可以将某些地址上的缓存操作解释为特定的信号或请求，并响应 CPU 返回信息或触发其他动作如中断。Figure 2 显示了即使使用（相对较慢的）ECI 与 DMA 在相同机器和现代 PC 服务器上的 PCIe 相比，可能实现的显著更好的交互延迟；我们预计 CXL 3.0 将带来可比的收益。

对于数据平面，此类协议允许数据包直接作为缓存行传输到目的地核心的 L1 缓存和寄存器 [21]，相比使用描述符的 DMA 提供了显著更低的延迟。

对于控制平面，CPU 核心之间的通信（运行应用程序代码或操作系统内核）是轻量级且高效的，并且可以使用常规的 MMU 机制轻松保护。它还传达丰富的信息，例如，一个 NIC 可以根据请求的地址是从其主地址空间中的哪个地址来推断核心是在用户模式下轮询还是在内核模式下轮询。

其次，**是时候信任该 NIC**。NIC 是机器操作系统功能的关键部分。与例如 GPU 不同，它是整个系统的基础设施。将其视为操作系统的一部分而非不可信的外围设备，是充分利用其硬件资源及其在数据路径中独特位置的唯一途径。

特别是，由于 NIC 负责将传入的数据包解复用到应用端点，它应该能够访问所有相关的操作系统状态：哪些进程当前在哪个核心的运行队列中，哪些正在执行，以及哪些在等待。其中一部分可以从 NIC 观察到的缓存流量推断出来，就像上面的例子一样，而任何其他状态都可以通过与 NIC 的互连以可忽略的开销显式推送。

最后，采用之前的位置使我们能够在**完全实现 RPC 分派等**上执行 NIC。将现有的加速反序列化的技术与丰富的操作系统状态知识相结合，使得 RPC 调度几乎没有任何软件开销：在常见情况下，可以在 NIC 的 Section 2 中的每个步执行操作，并使处理器核心上的挂起加载返回一个精心准备的缓存行，其中仅包含用于调度 RPC 所需的信息：只需目标函数的第一个指令的参数和虚拟地址即可跳转。

共享操作系统状态意味着在执行动态工作负载时，这种效率得以保持，在这种情况下静态关联 DMA 队列、核心、线程和插槽是不切实际的：NIC 已经有关于目标进程是否运行以及在哪里运行的信息，并可以根据情况通知该进程或操作系统。此外，操作系统可以从劳贝霍恩 获取最新的信息，了解哪些核心正在轮询以及它们的位置，从而指导调度决策。

5 实现草图

我们正在构建一个原型，劳贝霍恩 ，以展示这些想法在微服务环境中的可行性。劳贝霍恩 利用了 Enzian 研究计算机 [5] 上的大 FPGA、100Gb/s 接口和缓存一致性互连。我们概述了设计的两个关键组件：接收快速路径，以及操作系统与 NIC 之间调度状态的共享；我们将回到此设计中需要解决的其他非功能性问题，在 section 6中讨论。

5.1 接收快速路径

Figure 3 展示了最小接收路径的概览，在接收进程已经在系统中的至少一个核心上执行并且该进程的至少一个执行线程准备好处理请求的情况下。

劳贝霍恩 解复用并反序列化一个传入的 RPC 请求包，使用操作系统内核（以及间接地应用程序服务）提前提供的信息，以提供（1）一个进程和通信端点，（2）该进程中与请求对应的代码指针和数据指针，以及对应于步骤 1-3、5-6、10 和 11 的调用参数，在 section 2中。这可以通过使用现有的 NIC 技术变体来实现，如协议卸载和 RPC 反序列化加速（例如 [19]）。

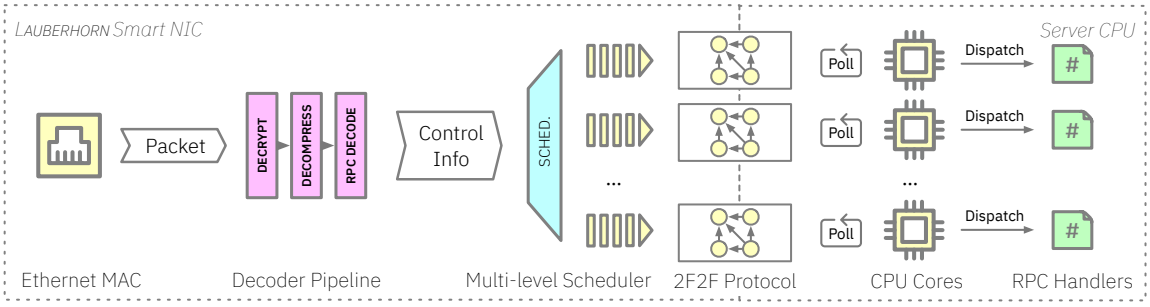

图 3: 劳伯霍恩 接收路径概述。

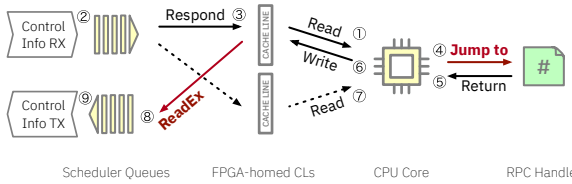

图 4: NIC 和 CPU 之间的劳伯霍恩 协议

在 FPGA 实现中，以太网帧从以太网 MAC IP 模块流进，并通过各种流模式头部解码器将数据包分离并移除以太网、IP 和 UDP 头部，将其存储在 SRAM 中供任何需要缓存的上层协议使用。

劳贝霍恩 然后将最小的数据结构传递到目标通信端点，该结构包括目标代码和数据指针以及 RPC 参数。它使用 Ruzhanskaia 等。[21](Figure 4) 描述的协议扩展来完成这一操作。每个端点包含一组托管在 NIC 上的缓存行：两条控制行加上多条辅助的行，用于处理大于单个缓存行（Enzian 上为 128 B）的有效负载。发送路径使用一组类似的、不相交的缓存行。

为了接收一个请求，一个进程会向一条控制缓存行发出加载请求以接收该请求。当适当的包到达并被解码后，NIC 将会用上述列出的数据响应此加载；在此之前核心会被阻塞（而不是自旋）。

CPU 现在已经在寄存器和 L1 缓存中拥有了启动用户程序第一条指令所需的所有信息，并且已经处于正确的地址空间。它执行了 RPC 处理程序并将 RPC 结果写入同一个 控制缓存行中，然后加载第二条 控制线以供下一个数据包使用。劳贝霍恩 观察到了对第二条线的加载操作，因此知道了 CPU 已经完成了第一个请求的服务。在响应第二个缓存行的读取之前，NIC 通过一致性协议发出一个获取独占权限请求第一条（包含 RPC 响应）从 CPU 的缓存中并将其发送到网络上。最后，当下一个数据包到达并被解码时，劳贝霍恩 对 CPU 在第二个控制缓存行上的读取做出响应。

当然，劳贝霍恩 不能在一致性协议中不超时的情况下阻止核心无限期地的缓存填充，这会导致无法恢复的“总线错误”，并使系统处于不一致状态。我们通过在 15 毫秒后返回重试 虚拟消息来避免这种情况，将轮询开销（包括总线流量和 CPU 空转）几乎降至零，并提高能效。

此外，这提供了一种干净地取消调度进程的机制：虽然操作系统仍可以在任何时候抢占正在运行的进程，但一个因这种通信负载而阻塞的核心提供了一个有用的同步点。劳伯霍恩 可以通知操作系统进程已被阻塞，操作系统（或 NIC）可以向进程的核心发送一个 Inter-Processor Interrupt (IPI)，然后劳贝霍恩 可以向进程发送一个重试消息，解除其阻塞状态并使其立即进入内核。

5.2 解复用和调度

劳贝霍恩 的关键新颖之处在于它如何利用精确的内核调度状态来分派请求。在快速情况下，请求可以直接到达正确的进程而无需内核干预，因为劳贝霍恩 意识到哪个核心正在运行该进程并等待持有请求的缓存行。有效地保持这种状态在上下文切换中是最新的，在一定程度上是由于 CPU 和由互连启用的 NIC 之间的通信具有极低的延迟。

当没有核心在运行接收到的数据包的目标进程时，劳博霍恩 快速将请求传递给内核，允许它调度目标进程并以软件方式传递解组后的请求。Figure 5 将这种方法与传统的 Linux 调度循环进行了比较。

一个运行常规内核线程的 CPU 核心使用 Section 5 中的协议来监控一对控制缓存行以接收请求；劳贝霍恩 可

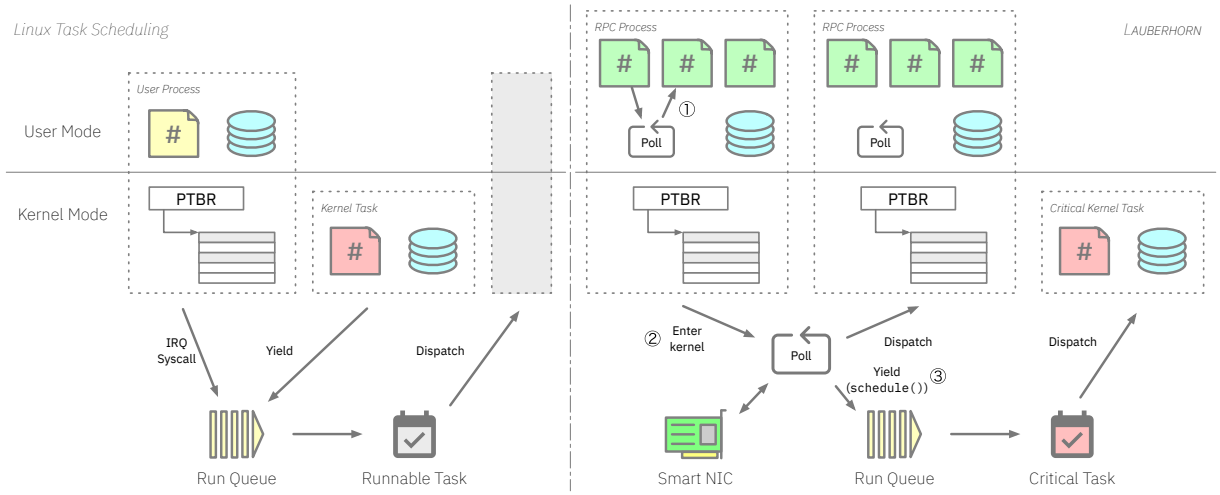

图 5: 正常任务调度与 NIC 驱动的 RPC 隔离域调度之间的比较。

以将针对任何进程的请求分发到这个 CPU 核心和端点，此时 CPU 切换到相应的进程来处理该请求。由于它是一个常规内核线程，因此会定期调用调度 () ③ 并能处理诸如读复制作更新等常规关键内核操作。

此后，①核心保持在同一进程中，并在不同的对控制缓存行上运行用户模式循环，这些缓存行由劳伯霍恩专用于该进程。此时，将请求分派到此服务几乎不涉及任何软件开销：核心执行的加载立即返回要跳转到的地址。

这在假设“热点”服务的数量少于可用核心数量的情况下运行良好——Enzian 上有 48 个核心，而现代服务器级处理器通常有数百个。

用户模式循环可以通过多种方式放弃 CPU②。进程可以在执行时通过执行系统调用来自愿释放 CPU；在核心被加载控制缓存行阻塞的情况下，当它接收到 RPC 负载或来自劳贝霍恩 的重试 消息时会发生这种情况。或者，内核和劳贝霍恩 可以合作通过向核心发送中断来完全抢占用户进程，然后如上所述使用随后的重试 恢复它（允许其接收中断）。请注意，这可以由内核调度器发起，或者基于其收集到的每个服务器进程瞬时负载统计数据由劳贝霍恩 发起。该方法因此也支持根据负载动态调整用于 RPC 的核心数量。

许多数据中心部署使用非抢占式内核以提高吞吐量。劳贝宏恩 向内核提供动态负载信息（再次使用内核模式控制通道）来在 RPC 服务和非 RPC 进程之间重新分配核心。正如我们已经抢先处理了等待缓存行的用户空间线程，任何等待在劳贝霍恩 的不可抢占内核线程都可以通过从 NIC 发送一个退休 消息来进行重新分配。

6 开放问题和关注点

应用程序线程、操作系统内核进程、缓存一致性协议以及 NIC 之间在劳贝霍恩 中的细粒度并发交互是微妙的，系统的正确运行要求我们确保所有竞争条件都是良性的。幸运的是，我们发现这个问题非常适合使用 TLA+ 进行规范，并且可以相对容易地进行模型检查以验证其正确性。

劳伯霍恩 如前所述，将支持全功能的 RPC 高效交互，但缺乏一些在实际数据中心环境中变得重要的非功能性特性。虽然加密可以使用相当标准的技术来处理，但对于追踪、调试和统计的支持则呈现出进一步与操作系统紧密集成的有趣特性。

对于大消息，直接的低延迟方法变得效率较低，最好回到基于 DMA 的传输方式，因为此时吞吐量比延迟更重要。权衡将取决于平台，以 Enzian 为例，这种情况大约在 4KiB 时发生。

嵌套的 RPCs 将受益于快速创建专用端点以接收 RPC 回复的能力。与 NIC 的细粒度交互应使创建此延续成为一个成本低廉的操作，并带来显著的性能优势。

标准操作系统接口 NIC 和应用程序接口 NIC 的设计是一个开放性问题，我们希望通过构建原型劳贝霍恩 来解答这个问题，并在此过程中改进我们提供的接口。

7 致谢

我们感谢匿名审稿人对论文提出的建设性意见，以及恩济安团队其他成员的支持和想法。本工作部分由谷歌系统研究小组的捐赠资助，对此我们表示感激。

References

[1] Amazon Web Services. *The Security Design of the AWS Nitro System*, Nov. 2022. https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.html.

[2] Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., and Koenig, J. The Rocket Chip Generator.

[3] Belay, A., Prekas, G., Klimovic, A., Grossman, S., Kozyrakis, C., and Bugnion, E. Ix: a protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (USA, 2014), OSDI'14, USENIX Association, p. 49 – 65.

[4] CCIX Consortium and others. Cache Coherent Interconnect for Accelerators (CCIX), May 2024.

[5] Cock, D., Ramdas, A., Schwyn, D., Giardino, M., Turowski, A., He, Z., Hossle, N., Korolija, D., Licciardello, M., Martsenko, K., Achermann, R., Alonso, G., and Roscoe, T. Enzian: an open, general, CPU/FPGA platform for systems software. In *ASPLOS '22: Proceedings of the Twenty-Seventh International Conference on Architectural Support for Programming Languages and Operating Systems* (February 2022).

[6] Consortium, C. Compute Express Link (CXL) version 3.0, Aug. 2022.

[7] Fried, J., Ruan, Z., Ousterhout, A., and Belay, A. Caladan: Mitigating Interference at Microsecond Timescales. pp. 281–297.

[8] Humphries, J. T., Natu, N., Chaugule, A., Weisse, O., Rhoden, B., Don, J., Rizzo, L., Rombakh, O., Turner, P., and Kozyrakis, C. ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event Germany, Oct. 2021), ACM, pp. 588–604.

[9] Humphries, J. T., Natu, N., Kaffes, K., Novaković, S., Turner, P., Levy, H., Culler, D., and Kozyrakis, C. Tide: A Split OS Architecture for Control Plane Offloading, Oct. 2024. arXiv:2408.17351.

[10] Ibanez, S., Mallery, A., Arslan, S., Jepsen, T., Shahbaz, M., Kim, C., and McKeown, N. The nanoPU: A Nanosecond Network Stack for Datacenters. pp. 239–256.

[11] Jang, J., Jung, S. J., Jeong, S., Heo, J., Shin, H., Ham, T. J., and Lee, J. W. A Specialized Architecture for Object Serialization with Applications to Big Data Analytics. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (Valencia, Spain, May 2020), IEEE, pp. 322–334.

[12] Kaffes, K., Chong, T., Humphries, J. T., Belay, A., Mazières, D., and Kozyrakis, C. Shinjuku: Preemptive Scheduling for {usecond-scale} Tail Latency. pp. 345–360.

[13] Karandikar, S., Leary, C., Kennelly, C., Zhao, J., Parimi, D., Nikolic, B., Asanovic, K., and Ranganathan, P. A Hardware Accelerator for Protocol Buffers. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event Greece, Oct. 2021), ACM, pp. 462–478.

[14] Marty, M., de Kruijf, M., Adriaens, J., Alfeld, C., Bauer, S., Contavalli, C., Dalton, M., Dukkipati, N., Evans, W. C., Gribble, S., Kidd, N., Kononov, R., Kumar, G., Mauer, C., Musick, E., Olson, L., Rubow, E., Ryan, M., Springborn, K., Turner, P., Valancius, V., Wang, X., and Vahdat, A. Snap: a microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2019), SOSP '19, Association for Computing Machinery, p. 399 – 413.

[15] Mogul, J., Baumann, A., Roscoe, T., and Soares, L. Mind the Gap: Reconnecting Architecture and OS Research. In *Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS-XIII)* (Napa, CA, USA, May 2011).

[16] Mogul, J. C. TCP offload is a dumb idea whose time has come. In *9th Workshop on Hot Topics in Operating Systems (HotOS IX)* (Lihue, HI, May 2003), USENIX Association.

[17] Ousterhout, A., Fried, J., Behrens, J., Belay, A., and Balakrishnan, H. Shenango: Achieving High {CPU} Efficiency for Latency-sensitive Datacenter Workloads. pp. 361–378.

[18] Peter, S., Li, J., Zhang, I., Ports, D. R. K., Woos, D., Krishnamurthy, A., Anderson, T., and Roscoe, T. Arrakis: The Operating System is the Control Plane. In *11th Symposium on Operating Systems Design and Implementation (OSDI'14)* (Broomfield, Colorado, USA, October 2014).

[19] Pourhabibi, A., Gupta, S., Kassir, H., Sutherland, M., Tian, Z., Drumond, M. P., Falsafi, B., and Koch, C. Optimus Prime: Accelerating Data Transformation in Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne Switzerland, Mar. 2020), ACM, pp. 1203–1216.

[20] Pourhabibi, A., Sutherland, M., Daglis, A., and Falsafi, B. Cerebros: Evading the RPC Tax in Datacenters. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event Greece, Oct. 2021), ACM, pp. 407–420.

[21] Ruzhanskaia, A., Xu, P., Cock, D., and Roscoe, T. Rethinking Programmed I/O for Fast Devices, Cheap Cores, and Coherent Interconnects, Sept. 2024. arXiv:2409.08141 [cs].

[22] Schuh, H. N., Krishnamurthy, A., Culler, D., Levy, H. M., Rizzo, L., Khan, S., and Stephens, B. E. CC-NIC: a Cache-Coherent Interface to the NIC. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating*

Systems, Volume 1 (La Jolla CA USA, Apr. 2024), ACM, pp. 52–68.

[23] Seemakhupt, K., Stephens, B. E., Khan, S., Liu, S., Wassel, H., Yeganeh, S. H., Snoeren, A. C., Krishnamurthy, A., Culler, D. E., and Levy, H. M. A Cloud-Scale Characterization of Remote Procedure Calls. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz Germany, Oct. 2023), ACM, pp. 498–514.

[24] Zhang, I., Raybuck, A., Patel, P., Olynyk, K., Nelson, J., Leija, O. S. N., Martinez, A., Liu, J., Simpson, A. K., Jayakar, S., Penna, P. H., Demoulin, M., Choudhury, P., and Badam, A. The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event Germany, Oct. 2021), ACM, pp. 195–211.