ChainIO: Bridging Disk and Network Domains with eBPF

Anonymous Authors

ABSTRACT

Modern data-driven services—from analytical databases and key-value stores to stream processors—suffer high tail-latencies because each disk read and subsequent packet send/recv incurs a separate user-kernel crossing and redundant buffer copy. While Linux's `io_uring` now supports both block and socket I/O with asynchronous, batched submissions, it does not provide zero-copy transfers between storage and network domains; AF_XDP delivers high-performance packet I/O but is siloed to the network stack. No existing framework transparently unifies these mechanisms end-to-end. We present ChainIO, an eBPF-based system that intercepts and rewrites I/O syscalls, uses ring buffers to pass data descriptors directly between `io_uring` and AF_XDP, and orchestrates in-kernel execution to chain disk reads into network sends (and vice versa) with full POSIX semantics, fallback safety for unsupported cases, and zero application changes. Our prototype works with unmodified binaries and improves ClickHouse's TPC-H query throughput by up to 39% (while cutting context switches by 59% and CPU use by 14%), and reduces 99th-percentile latency by 30% in a distributed key-value store. ChainIO thus offers a general, safe, and high-performance path for cross-domain I/O optimization in diverse data-intensive workloads.

1 INTRODUCTION

Modern data-driven services—such as distributed analytical databases—often suffer from high tail latency because of inefficient user–kernel crossings and redundant buffer copies. To address this, the Linux kernel introduced io_uring[1], an asynchronous I/O interface that uses shared ring buffers to reduce system calls and context switches. While `io_uring` supports storage and networking, it lacks zero-copy across domains and incurs overhead when chaining disk reads to network sends. XDP (eXpress Data Path) is an in-kernel eBPF framework offering sub-microsecond latency and zero-copy forwarding. Similarly, AF_XDP[3] offers superior network performance but can't integrate directly with storage operations. No existing solution provides a unified framework for automatically identifying and optimizing these cross-domain dependencies without application modifications.

Previous research has approached this challenge from several angles without fully solving the cross-domain problem: FlexSC [4] and MegaPipe batch syscalls but focus on single domains, while DPDK/SPDK and Demikernel [7] bypass the kernel entirely but require extensive application modifications. eBPF-based solutions like XRP [9] and BPF-oF [6] accelerate specific I/O paths (NVMe reads, remote storage) without addressing cross-domain dependencies. Even architectural innovations like IX [2] that redesign the OS with separate control and data planes remain siloed in network or storage specialization, leaving a critical gap for workloads that chain operations across both domains.

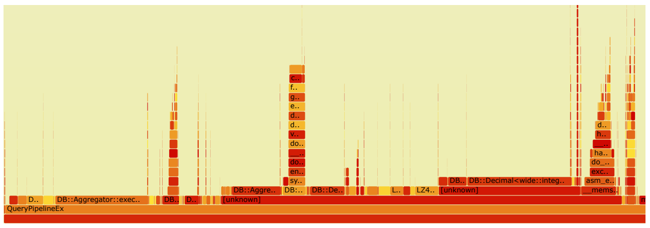

Figure 1: Flamegraph of ClickHouse Server showing syscall overhead

As a distributed SQL database, ClickHouse's[5] `MergeTree` engine exemplifies this cross-domain problem, shown in the profiling data in Figure 1. A typical query in ClickHouse follows this path: client query → column-file reads (`pread()`) → distribution to remote shards (`send()`) → network stack → remote node's `recv()` → disk lookup → response aggregation. Its columnar storage engine issues large numbers of small, random `read()` calls against compressed column files and mark-file offsets, with each compressed-block fetch and metadata lookup translating into a user-kernel transition. In distributed setups, remote-shard requests add further `send()` and `recv()` calls for data fetches and Raft heartbeats. Our profiling shows that the blocking `read()` syscall alone consumes 25% of query time, while small network receives (heartbeats, shard updates) account for another 2%. The cumulative cost of these syscalls—exacerbated by Spectre/Meltdown mitigations—introduces tens of microseconds of overhead per transition, multiplying into hundreds of milliseconds on fan-out queries. When a query spans dozens of remote partitions, each extra transition adds up quickly, creating a critical bottleneck for interactive dashboards and real-time analytics that cannot be solved by optimizing either storage or networking in isolation.

We introduce ChainIO, a unified syscall-chaining framework that bridges both domains. Our solution dynamically rewrites POSIX I/O calls into batched submissions, unifies memory management across domains through shared regions, coordinates cross-domain operations while preserving correctness, and adaptively optimizes for tail latency. ChainIO requires no application modifications, achieving significant performance gains by eliminating redundant context switches and memory copies. This design can extend to other services with mixed storage and network workloads under POSIX easily.

2 DESIGN AND IMPLEMENTATION

ChainIO (Figure 2) architecture combine dynamic binary rewriting with in-kernel eBPF programs to eliminate redundant context switches while preserving POSIX semantics. Implemented in 2000 lines of C/C++ and eBPF code, ChainIO maintains compatibility with unmodified ClickHouse binaries. ChainIO comprises three integrated components:

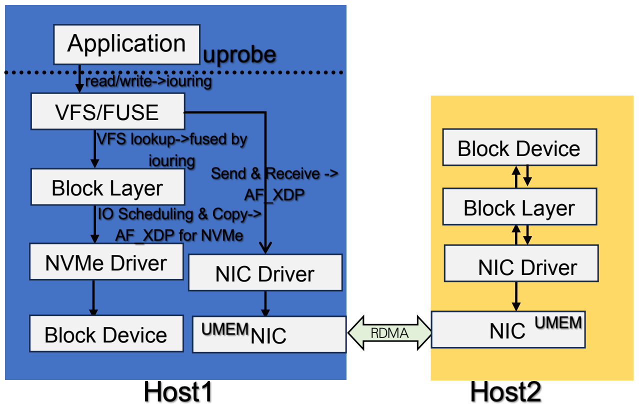

Figure 2: ChainIO Architecture

Cross-Domain Ring Bridge. Unlike simply deploying io_uring and AF_XDP side-by-side, our core innovation is a ring design unifying storage and network operations through shared memory, implemented via BPF shared memory maps that bridge user-space `io_uring` rings with in-kernel XDP processing. This unified descriptor format supports both disk SQEs (`io_uring` descriptors) and network SQEs (XDP frame metadata), enabling atomic cross-domain operations and automatic dependency tracking so that network sends fire immediately after disk reads complete—without extra context switches. More specifically, ChainIO register a contiguous user-space memory region (UMEM) with both `io_uring` and AF_XDP, using it as direct DMA buffers for NVMe operations and zero-copy packet buffers for optimized network traffic. A custom slab allocator manages UMEM efficiently, and multiple eBPF programs—including an XDP packet router for steering optimized-path traffic into UMEM and an IO completion handler for triggering chained operations—coordinate end-to-end execution entirely in-kernel.

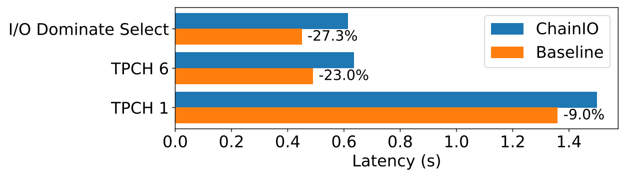

Figure 3: Latency Comparison Across TCP-H Queries

Syscall-Chaining Engine. To chain POSIX I/O calls end-to-end, ChainIO uses eBPF to intercept `read()`, `send()`, and `recv()` syscalls and reroute them into a unified submission ring. A userspace uprobes[8] handler then converts each intercepted call into a batched `io_uring` request. To adapt to Clickhouse, We instrument key MergeTree routines (e.g., `MergeTree::readMark()` and `MergeTree::readData()`) using USDT probes to capture compressed-column reads and associated metadata, preserving semantic context across syscall boundaries with minimal overhead. It can be extended to other services easily by changing the metadata probes.

Tail-Latency Optimizations. To reduce high-percentile latencies, ChainIO employs three complementary techniques: dynamic batch sizing, priority-aware scheduling, and lightweight in-kernel preemption. A user-space coordinator continuously monitors operation latencies and dynamically tunes the submission batch size—flushing smaller batches under load to bound the 99th-percentile latency below 150. We also prioritize latency-sensitive tasks, such as metadata lookups and NURaft heartbeats, over bulk column scans to ensure critical operations complete promptly. Finally, we route distributed query messages and Raft heartbeats through AF_XDP for zero-copy, low-latency network delivery.

3 EVALUATION

We evaluated ChainIO on ClickHouse (v21.8) running on CloudLab servers with Intel Xeon Silver 4314 CPUs, 128GB RAM, and dual-port 100Gb Mellanox ConnectX-6 NICs. Each server has a Samsung PM1725a NVMe SSD. We measured performance using TPC-H at scale factor 20 on a single NVMe-SSD, comparing our ChainIO SQPOLL + HugePage + Registered-File configuration against a Thread-poll + pread baseline. As shown in the Figure 3, for I/O-bound queries such as Q6, average latency improves by up to 23%. When running a narrow column scan (SELECT SUM(LENGTH(l_comment))), latency improves by 27.3%. Row throughput increases by up to 39% for I/O-dominated workloads. 99th-percentile latency for short-running metadata queries also improves by 3.2×.

REFERENCES

[1] Jens Axboe. Efficient io with io_uring. Linux Kernel report, 2019. urlhttps://kernel.dk/io$_u ring.pdf$.

[2] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. {IX}: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65, 2014.

[3] Killian Castillon du Perron, Dino Lopez Pacheco, and Fabrice Huet. Understanding delays in af_xdp-based applications. In *ICC 2024-IEEE International Conference on Communications*, pages 5497–5502. IEEE, 2024.

[4] Livio Soares and Michael Stumm. {FlexSC}: Flexible system call scheduling with {Exception-Less} system calls. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[5] ClickHouse Team. Clickhouse: The fastest analytical database for observability ml genai. https://clickhouse.com/, 2025. Accessed: 2025-05-17.

[6] Ioannis Zarkadas, Tal Zussman, Jeremy Carin, Sheng Jiang, Yuhong Zhong, Jonas Pfefferle, Hubertus Franke, Junfeng Yang, Kostis Kaffes, Ryan Stutsman, et al. Bpf-of: Storage function pushdown over the network. *arXiv preprint arXiv:2312.06808*, 2023.

[7] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. The demikernel datapath os architecture for microsecond-scale datacenter systems. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*, pages 195–211, Virtual Event, Germany, 2021. ACM.

[8] Yusheng Zheng, Tong Yu, Yiwei Yang, Yanpeng Hu, XiaoZheng Lai, and Andrew Quinn. bpftime: userspace ebpf runtime for uprobe, syscall and kernel-user interactions, 2023.

[9] Yuhong Zhong, Haoyu Li, Yu Jian Wu, Ioannis Zarkadas, Jeffrey Tao, Evan Mesterhazy, Michael Makris, Junfeng Yang, Amy Tai, Ryan Stutsman, and Asaf Cidon. XRP: In-Kernel storage functions with eBPF. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 375–393, Carlsbad, CA, July 2022. USENIX Association.