zIO: Accelerating IO-Intensive Applications with Transparent Zero-Copy IO

Tim Stamler¹, Deukyeon Hwang¹, Amanda Raybuck¹, Wei Zhang², Simon Peter³

¹The University of Texas at Austin ²Microsoft ³University of Washington
IO Copies are Common

Robust data exchange mechanism among application subsystems

IO copy call sites:

Applications (& libraries)
Eg: gRPC, Protobuf

I/O stack APIs
Eg: POSIX API (recv/send)

<table>
<thead>
<tr>
<th>Application</th>
<th>Operation</th>
<th>IO Copy call site</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>App</td>
</tr>
<tr>
<td>Redis</td>
<td>SET</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>2</td>
</tr>
<tr>
<td>Icecast</td>
<td>Cast to N clients</td>
<td>0</td>
</tr>
<tr>
<td>Ceph</td>
<td>Write</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>Anna</td>
<td>PUT</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>4</td>
</tr>
<tr>
<td>MongoDB</td>
<td>Insert</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Disk sync</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>2</td>
</tr>
<tr>
<td>Tensorflow-serving</td>
<td>Inference</td>
<td>2</td>
</tr>
<tr>
<td>Nebula Graph</td>
<td>Insert vertex</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Store a vertex</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>
IO-Intensive Apps are Increasingly Copy-Limited

More IO => more copies

High CPU overhead from copies at high throughput

Kernel-bypass IO stacks intensify the overhead

Other overheads reduced
Zero-copy IO?

Lots of work on single-stack zero-copy IO APIs:
  Network: Solaris [ATC ‘96], FreeBSD [IEEE ‘01], RDMA, netmap [ATC ‘12]
  Storage: Memory-mapped files

Cross-Stack APIs minimize copies across different IO stacks:
  Demikernel [SOSP ‘21], PASTE [NSDI ‘18], Linux sendfile

**Success has been limited:**
  Many require application modification or have non-transparent requirements
  None seek to eliminate copies within the application (even if more prevalent)
zIO: Transparent Zero-Copy IO

An open-source, transparent IO copy elimination library

Transparently interposes on IO buffer copies
Eliminates application *and* IO stack API copies
Compatible with applications using POSIX IO and libc memcpy/memmove

*zIO eliminates IO copies without application modification*
Key Insights

Assumption: much IO data remains untouched by applications
In this case, the copy doesn’t need to happen

zIO speculatively elides and tracks IO buffer copies
- Record original input buffer location when read from IO stack
- Track and elide subsequent copies of this buffer
- When writing to an output stack, present the original input buffer

Upon mis-speculation (IO buffer touched), lazily execute copy
zIO Transparent IO Copy Elision

Application

Kernel

Intercepted APIs
- IO read: read()/recv()
- memcpy()/memmove()
- IO write: write()/send()
- free()

libzio
- Input buffer recording
- Transitive buffer tracking
- Input to output resolution
- Cleaning
- Lazy copy of touched IO

I/O
Example: Application IO Copy Elision

Application

Intermediate, Unmapped Buffer

memcpy()

Original Buffer

memcpy()

Key is Accessed

read()

write()

Input IO Stack

zIO tracking:
Original Buffer
Buffer 1 -> Original
Buffer 2 -> Original

Output IO Stack

Each one page (4KB) I/O

memcpy()
IO Stack API Copy Elision

To elide IO stack API copies, zIO needs to track across the API boundary

Difficult with kernel stacks; their APIs involve system calls

Discussed in paper

With kernel-bypass IO:

Kernel-bypass IO stacks hold IO in private buffers in user space

IO stack API simply copies between app-provided and private buffers

zIO tracks IO from private buffers as the original and elides the copy
Evaluation
Evaluation Questions

Does zIO improve IO throughput by eliminating copies?

Does zIO improve the performance of real world applications?

Does zIO affect scalability?

How does zIO compare to zero-copy IO APIs?
Experimental Setup

Intel Xeon Gold 6252 CPU 24 cores @ 2.10GHz
196GB RAM
Mellanox ConnectX-5 100Gb/s Ethernet

Benchmarks:
• Network echo server
• Key-value store (Redis)
• HTTP streaming & serving (Icecast)

Four configurations:
• Linux
• Elided in-app copies (zIO)
• Kernel-bypass IO (TAS [EuroSys'19], Strata [SOSP'17])
• Elided in-app + IO stack API copies (zIO+IO)
Does zI/O Improve IO Throughput?

Network echo server with varying intermediate copies and 512KB messages
Receive data (recv), configurable number of app copies, send data (send)
Key-Value Store

YCSB Workload A (50% GET, 50% SET)

Redis with append-only file, persisting every request

<table>
<thead>
<tr>
<th>Application</th>
<th>Operation</th>
<th>Copy call site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>SET</td>
<td>App</td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>IO Stack</td>
</tr>
</tbody>
</table>

**Linux network+storage stack**

Throughput [Gb/s] vs Value size [KB]

- zlO
- Linux

**Kernel-bypass network+storage stack**

Throughput [Gb/s] vs Value size [KB]

- zlO+IO
- Kernel-bypass
HTTP Streaming

<table>
<thead>
<tr>
<th>Application</th>
<th>Operation</th>
<th>Copy call site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icecast</td>
<td>Cast to N clients</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 + N</td>
</tr>
</tbody>
</table>

Icecast streaming 1MB audio files in 64KB IO buffer chunks

- Enough listener clients to saturate Icecast server
- Using kernel-bypass IO

Network to network (1.16x higher throughput)

- Single casting client connected to Icecast

Storage to network (1.27x higher throughput)

- Icecast streams from local disk
HTTP Serving

512KB file in 64KB IO chunks, enough clients to saturate server, kernel-bypass IO

Two versions: 1. read from file, 2. mmap file (zero-copy API); both send on network

<table>
<thead>
<tr>
<th>Application</th>
<th>Operation</th>
<th>Copy call site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icecast</td>
<td>Serve to N clients</td>
<td>0</td>
</tr>
</tbody>
</table>

Throughput (Gbps)

- read TLB
- zIO+IO TLB
- mmap TLB
- read
- zIO+IO
- mmap

Why? zIO+IO can avoid TLB shootdowns with buffer reuse
Summary

zIO transparently accelerates IO intensive applications

Achieved by
1. Interposing on and eliding IO buffer copies
2. Tracking copied IO buffers, presenting the original on IO output
3. Lazily copying touched IO

1.8x speedup with Linux IO and 2.5x speedup with kernel bypass with Redis

Try it out here!
https://github.com/tstamler/zIO