zIO

Accelerating IO-Intensive Applications with Transparent **Zero-Copy IO**

Niels Kirstein

Seminar Supercomputer
Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik

19.12.2023

informatik
die zukunft

Gliederung

- Zero Copy - Einführung
- Kopien in IO-Intensiven Anwendungen
- Zero Copy Techniken
- Virtual Memory
- **zIO** (Anwendung)
  - Funktionsweise
  - Usage
  - Evaluation
- Zusammenfassung

zIO

- user library (libzIO)
- transparente Eliminierung von Kopien

Zero Copy

Vermeiden von unnötigen Datenkopien.

Warum?

- höherer Durchsatz 🚀
- geringerer Speicherverbrauch
- 1.8x höherer Durchsatz (zIO Redis)

Warum überhaupt Kopien?

- unabsichtlich
- **vereinfachen Development**
- Subsystem muss keine Modifikationen des Callers erwarten

Motivationsbeispiel

Kopieren einer Datei

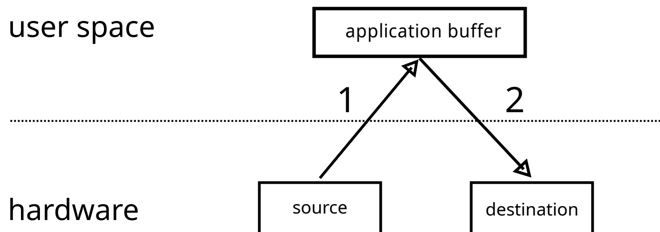


Figure 1: Ablauf: Dateikopie

Motivationsbeispiel (Kopien)

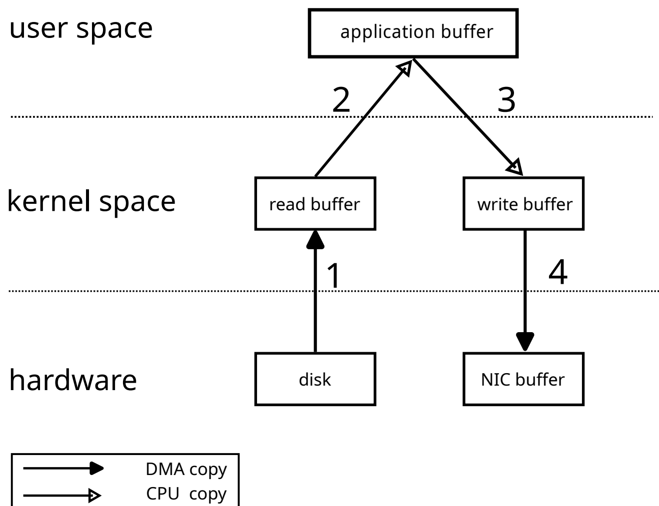


Figure 2: copies between user and kernel space

Motivationsbeispiel (Kopien)

Insgesamt **4** Kopien.

unnötige Kopien:

- kernelspace → userspace
- userspace → kernelspace

⟹ Anwendung benötigt Daten **nicht**

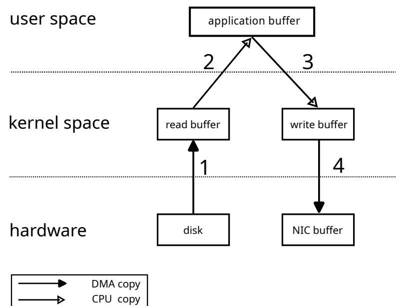


Figure 3: copies between user and kernel space

Wo treten Kopien auf?

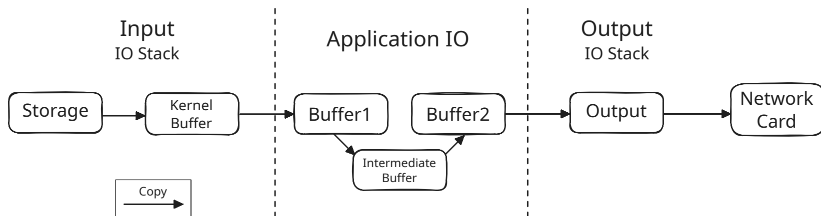


Figure 4: Kopien

- Anwendungskopien
  - zwischen Subsystemen (**Kommunikation**)
  - innerhalb eines Subsystems
- IO Stack Kopien

Kopien in IO-Intensiven Anwendungen

Kopien in IO-Intensiven Anwendungen

Anwendung	Operation	App Kopien	IO Stack Kopien
Redis	SET	4	2
	GET	2	1
MongoDB	Insert	3	2
	Read	2	2

- mehr interne Kopien

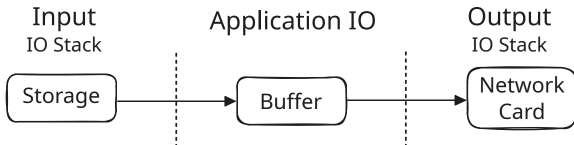


Figure 5: IO-Stack vs Application IO

Redis

- in-memory Datenbank
- key-value store (NoSQL)

Figure 6: redis logo

Kopien verwendet für:

- Read from IO Stack
- Deserealisierung
- Speichern in hash table
- Speichern in append-only file

Zero Copy Techniken [1]

Zero Copy Techniken

Einteilung

1. IO Stack APIs
2. kernel-bypass IO
3. Generelle Optimierungen

1. Zero Copy APIs - Übersicht

- Memory Map (mmap)
- sendfile
- splice
- ...

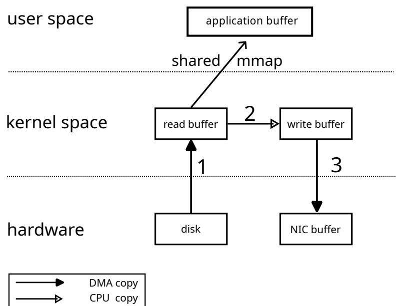


Figure 7: mmap data flow

1. Zero Copy APIs - Übersicht

- Memory Map (mmap)
- sendfile
- splice
- ...

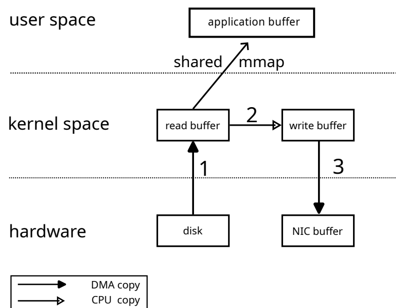


Figure 7: mmap data flow

Nachteile:

- nicht standardisiert
- eingeschränkter Einsatzbereich (Webserver)
- Hardware Unterstützung notwendig

Kernel Bypass

direkter Zugriff auf Geräte

Umgehung des Linux Kernels

$\Rightarrow$ Kein/weniger:

- Systemcalls
- Interrupts
- Context switches
- **kernelspace $\leftrightarrow$ userspace kopien**

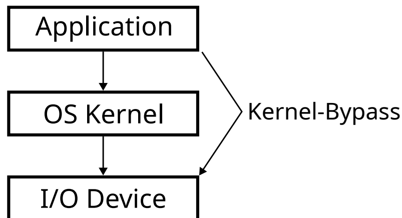


Figure 8: Kernel-Bypass IO

2. Kernel Bypass - Beispiele

- RDMA: Remote direct memory access
- DPDK: Dataplane Development Kit
- ...

- TAS: TCP Acceleration Service
- Strata (Storage)

3. Kopie Optimierungen

- Copy on Write (CoW)
- Buffer sharen
- **zIO** (paper)

Virtual Memory [2]

Virtual Memory

- Prozesse bekommen virtuellen Adressraum
- Abbildung auf physischen Speicher

Vorteile

- Speicherschutz
- Fragmentierung
- mehr Speicher
- ...

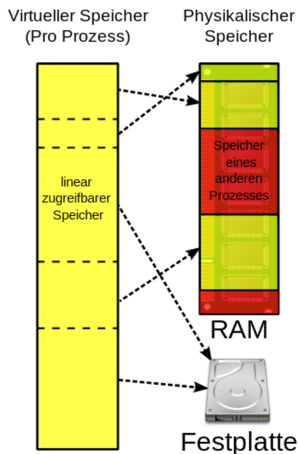


Figure 9: virtueller Speicher

Seitentabelle

- Aufteilung in Pages
- Abbildung Pages auf:
  - → **Haupt**speicher
  - → **Sekundär**speicher

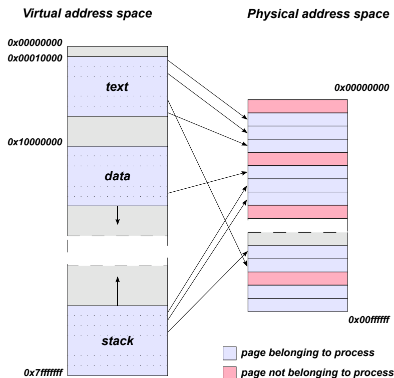
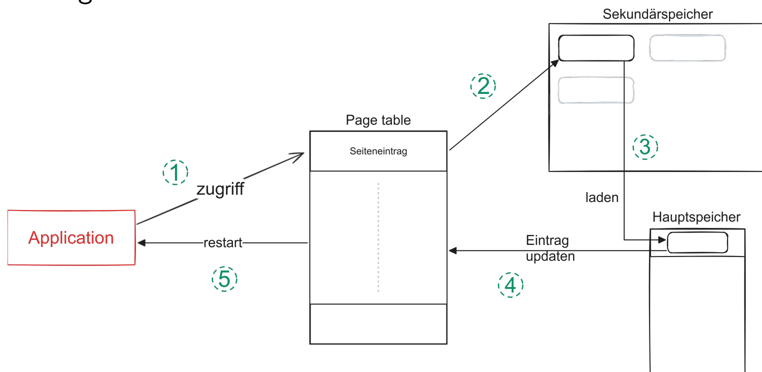


Figure 10: Seitenabbildung

Page-faults

- Page nicht im Hauptspeicher
- muss geladen werden



zIO

zIO Übersicht [3]

- Zero Copy Einführung

- Funktionsweise
- Kernel-bypass
- Verwendung
- Evaluation

zIO

Nachteile traditioneller Lösungen:

- API Restriktionen
- höhere Code Komplexität
- nicht standardisiert oder allgemein unterstützt
- **Quelltext-Anpassungen**

zIO

Nachteile traditioneller Lösungen:

- API Restriktionen
- höhere Code Komplexität
- nicht standardisiert oder allgemein unterstützt
- **Quelltext-Anpassungen**

⇒ **transparentes** Zero copy

ohne Quelltext-Anpassungen

Frage / Ziel des Papers

Einfache Entwicklung + weniger Kopien

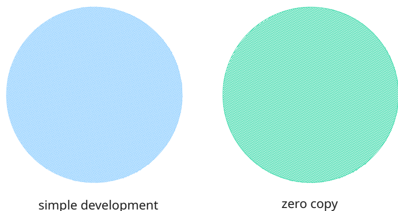


Figure 11: zero copy apis

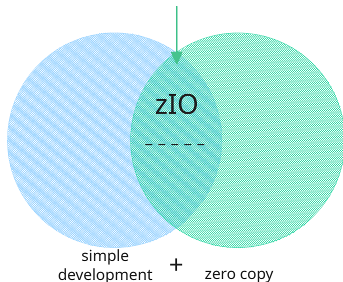


Figure 12: zIO Goals

zIO Design

- user-level library (libzIO)
- fängt **system calls** und **page faults** ab

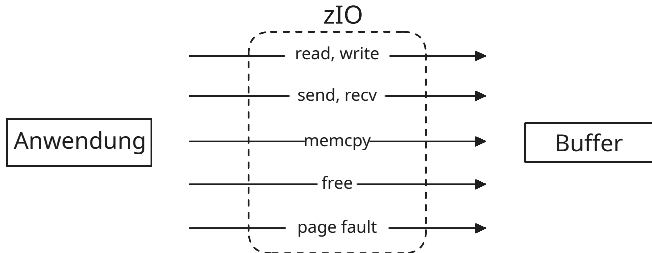


Figure 13: intercepted APIs

Funktionsweise

Funktionsweise

- read
- write
- page faults
- copy
- free

- kurzes Beispiel

Funktionsweise (Grob)

Optimistische Annahme:

- Großteil der Daten **nicht** verwendet

Funktionsweise (Grob)

Optimistische Annahme:

- Großteil der Daten **nicht** verwendet

$\Rightarrow$ **Kopien (erstmal) nicht ausführen**

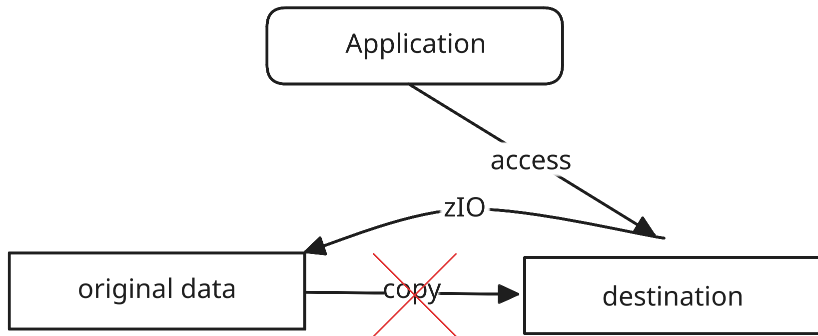


Figure 14: zIO overview

Funktionsweise (Grob)

Optimistische Annahme:

- Großteil der Daten **nicht** verwendet
- $\Rightarrow$ selten lazy Kopien

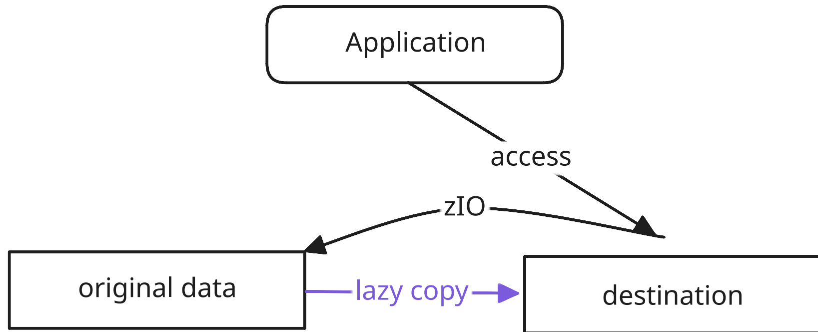


Figure 15: zIO overview

Funktionsweise (Grob)

copy → unmap page

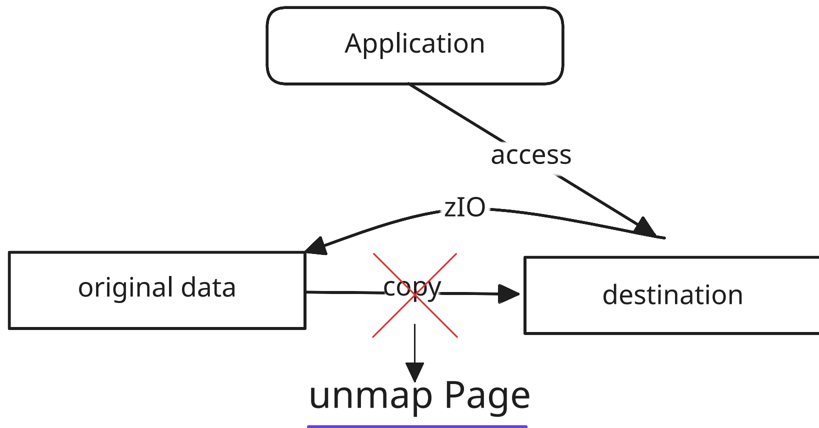


Figure 16: zIO overview unmap

Zugriff auf intermediate?

⇒ Page fault

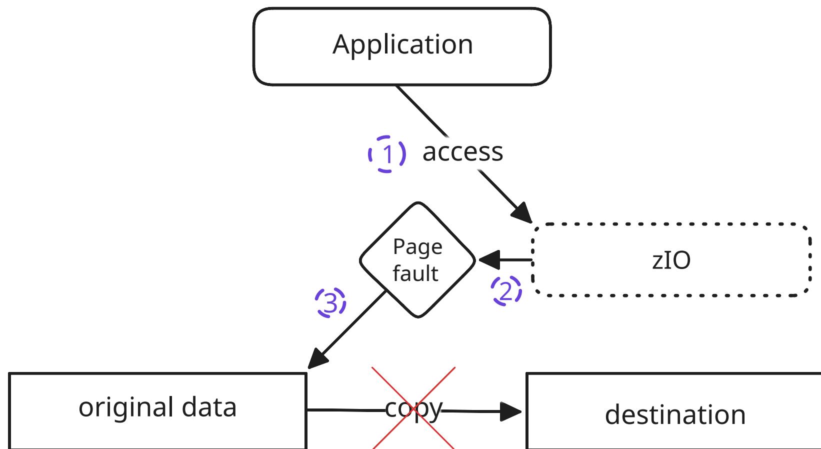
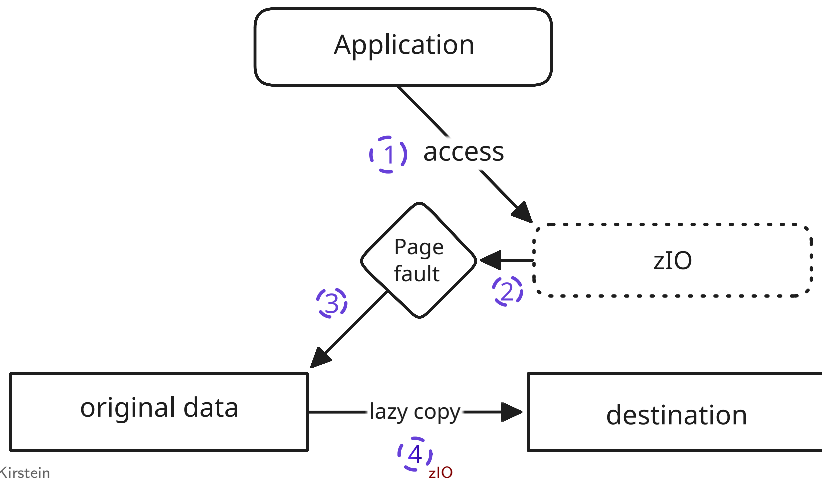


Figure 17: zIO page fault

Zugriff auf intermediate?

1. ⇒ Page fault
2. ⇒ lazy copy



Funktionsweise Kopien

Woher kennen wir den `original buffer`?

Verhalten bei Kopien (`memcpy(src, dest)`)

- `dest` **speichern** als *intermediate*
- *original buffer* finden
- intermediate buffer: **unmappen**
- **page faults** abfangen mit `userfaultfd`
- original buffer: **read only**

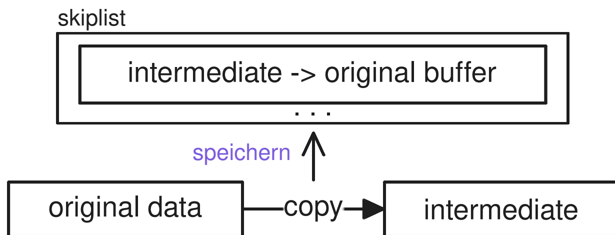


Figure 19: zIO Verhalten bei Kopien

Funktionsweise Input

IO Input: `read(src, dest)`

- `dest` als original buffer speichern

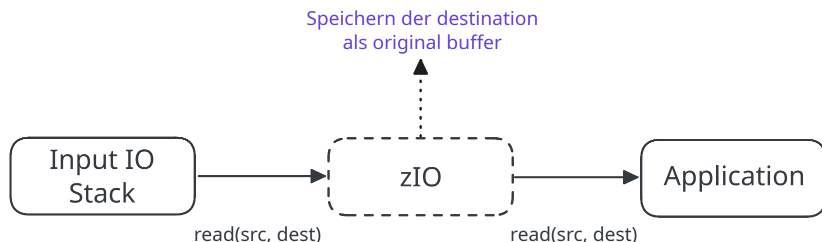


Figure 20: zIO Verhalten bei IO Stack Input

Funktionsweise Output

Bei Ausgabe an IO Stack:

- Original Buffer providen

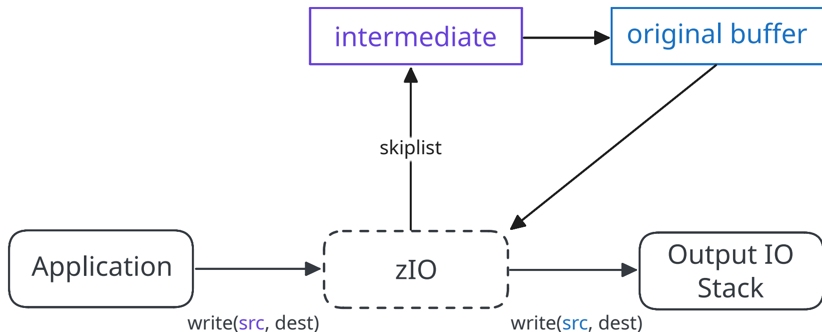


Figure 21: zIO Verhalten bei Ausgabe an IO Stack

Funktionsweise Übersicht

1. IO Input: `read(src, dest)`

   `dest` als original buffer speichern

2. Kopien (`Memcpy(src, dest)`):

   - Kopie nicht ausführen
   - unmap buffer → page faults abfangen
   - Speichern `dest` als `intermediate`

3. Datenkonsistenz durch page faults

4. IO Output: `write`

Speichern der Adressen

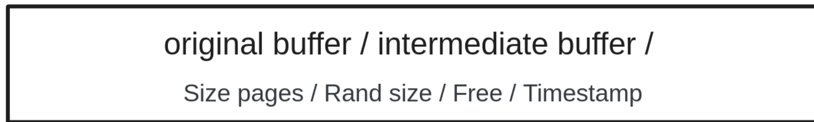


Figure 22: Aufbau Skiplist

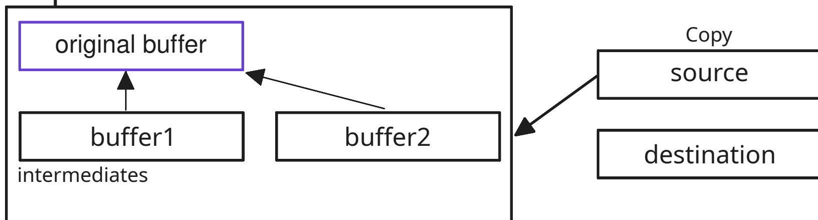


Figure 23: Skiplist suche

Eliminierung auf Seitenebene

Buffer nicht page-aligned ❓

- *page faults* agieren auf Seitenebene

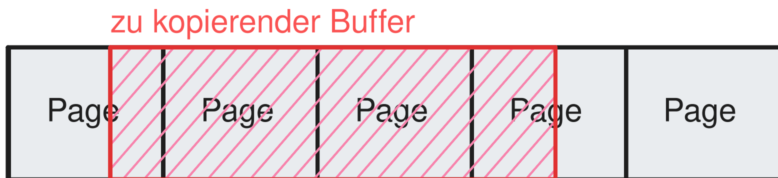


Figure 24: unaligned buffer

Eliminierung auf Seitenebene

Buffer nicht page-aligned ❗

- ⇒ **Kopieren von Rändern** 👍
  - klein ⟶ geringer Overhead
  - häufiger benutzt (Header)

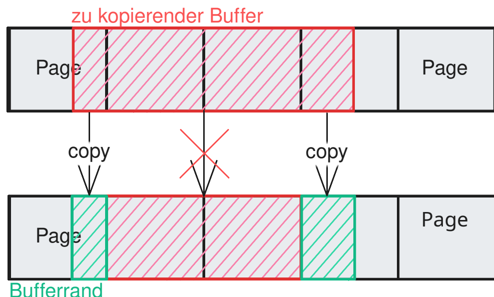


Figure 25: Rand Kopien

Speicher freigeben

Freigeben (`free`) eines

- intermediate Buffers
  - Anwendung benötigt Speicher nicht mehr
  - Löschen des Skiplist Eintrags

Speicher freigeben

Freigeben (`free`) eines

- intermediate Buffers
    - Anwendung benötigt Speicher nicht mehr
    - Löschen des Skiplist Eintrags

$\Rightarrow$ **Erfolgreich Kopie eliminiert**

Speicher freigeben

Freigeben (`free`) eines

- intermediate Buffers
  - Anwendung benötigt Speicher nicht mehr
  - Löschen des Skiplist Eintrags

$\Rightarrow$ **Erfolgreich Kopie eliminiert**

- original Buffers
  - **nur** markieren als *free*
  - da Buffer evtl. noch benötigt (Page fault, Output)
  - $\rightarrow$ Garbage collected

Funktionsweise Beispiel

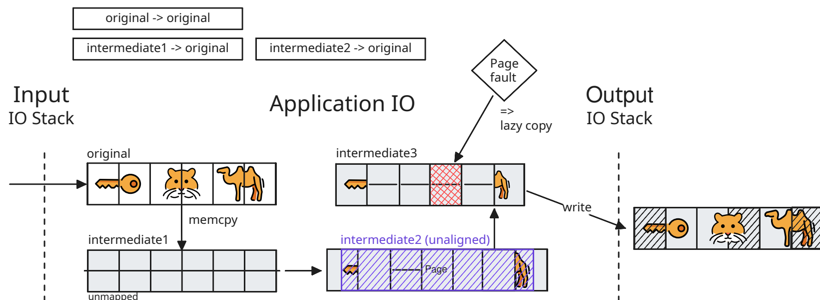


Figure 26: Beispiel zIO

Kernel Bypass

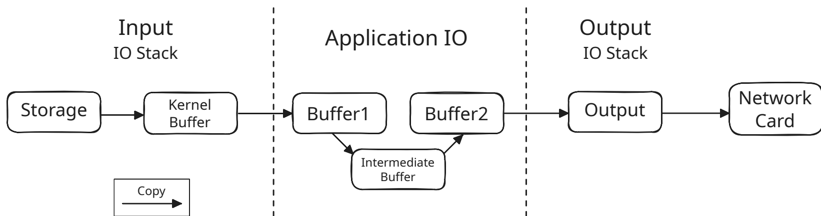


Figure 27: Application IO vs Stack IO

Mehr Kopie Eliminierungen 🚀

- Application IO
- IO Stack

Kernel Bypass IO

Kernel-Bypass IO Stacks

- shared library calls
- shared memory

⇒ keine Syscalls notwendig
⇒ **aber Quelltext Anpassung**

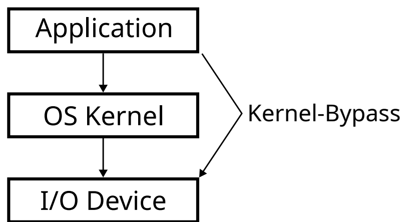


Figure 28: Kernel-Bypass IO

Kernel Bypass IO

Nur implementiert für:

- TAS network stack
- Strata file system

NVMM

- **persistenter** Hauptspeicher (NVMM)
- original + intermediate buffer sind NVM
- ORP: Optimistic receiver persistence
- ⇒ keine Kopien in Storage notwendig

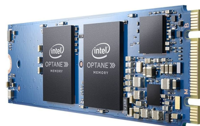


Figure 29: intel optane NVM Memory

NVMM

- **persistenter** Hauptspeicher (NVMM)
- original + intermediate buffer sind NVM
- ORP: Optimistic receiver persistence
- ⇒ keine Kopien in Storage notwendig

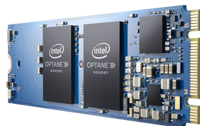


Figure 29: intel optane NVM Memory

TAS

zIO Usage

zIO

- dynamically loaded

zIO + Kernel-bypass IO Stack (zIO + IO)

- TAS Network
- Strata filesystem

Usage

https://github.com/tstamler/zIO

```
> LD_PRELOAD copy_interpose.so ./programm
```

- lädt custom Syscalls

Für NVMM Nutzung:

- spezielle Hardware
- Emulation mit DRAM

Evaluation

Evaluation

Anwendungen

- Simple Echo Server
- Redis

Vergleiche

- Vanilla Linux
- Anwendungskopie (zIO)
- Anwendungs + IO Stack API Kopie (zIO + IO)

Simple Echo server

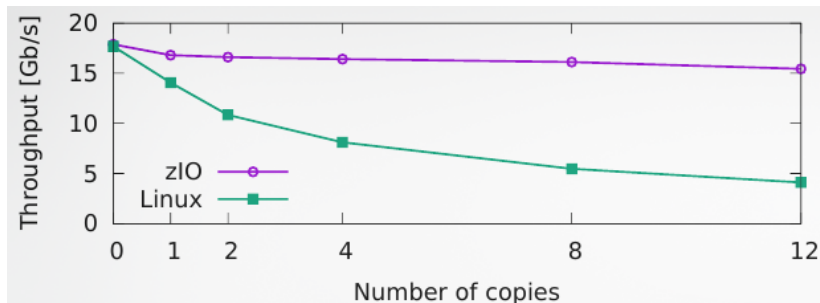


Figure 30: Linux vs. zIO Durchsatz bei versch. Kopien

- 12 Kopien: **3.8** mal höherer Durchsatz

Simple Echo server

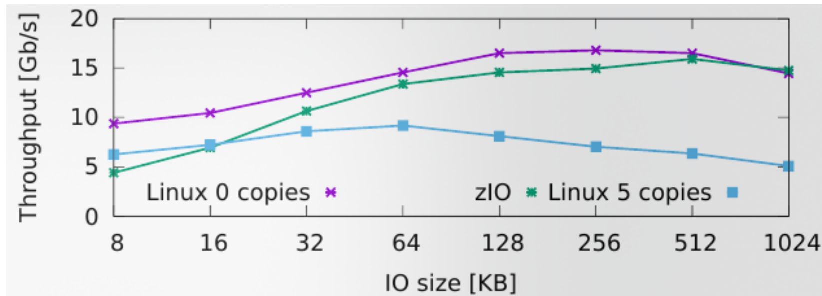


Figure 31: Linux vs. zIO Durchsatz gegen IO Size und Anzahl Kopien

- 5 Kopien: Nahe an *zero copies*
- $<$ 16KB: tracking aus

Simple Echo server (2)

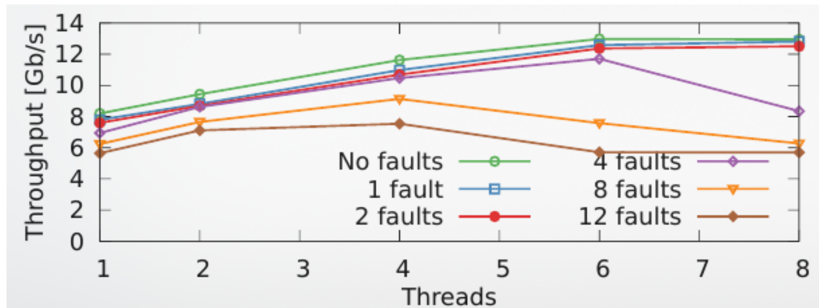


Figure 32: Durchsatz gegen Pagefaults

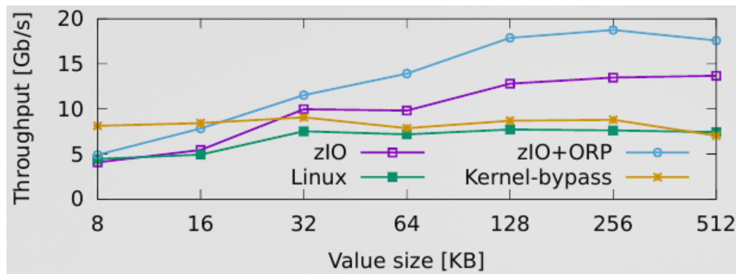
- Ab 2 page faults: performance drop
- page faults: aufwendig

Redis

- in-memory Datenbank
- key-value store (NoSQL)
- Kopien u.a. an *Append-only file* (AOF) (Siehe Kopien …)

Workload 100% WRITE

- ORP: *Optimistic receiver persistence* (NVMM)



- 4 IO Anwendungskopien
- zIO: **1.8x**

- 2 Kopien (Network Input –> Anwendung –> Storage Output)
- ORP: **2.5x**

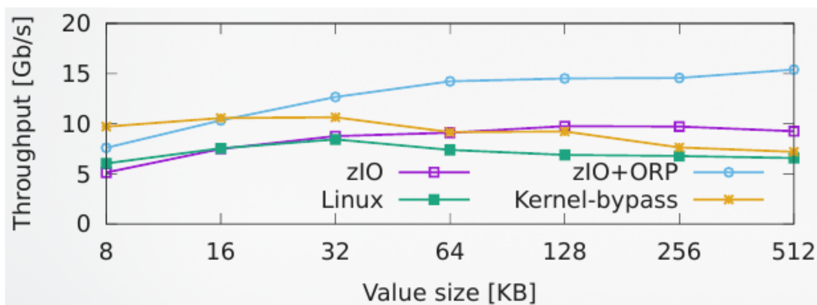
Workload YCSB (Yahoo! Cloud Serving Benchmark)

Workload A

- Viele `UPDATE` Operationen
- 50/50 READ/WRITE

50% READ:

- Weniger Anwendungskopien
- zIO: **1.3x**
- Mehr IO Stack Kopien
- ORP: ~**2.0x**



Zusammenfassung

Zusammenfassung

- Auftreten von Kopien
- Zero Copy: Techniken

Motivation

- Nachteile existierender Lösungen
- zIO: **transparentes** Zero Copy

Zusammenfassung

zIO Design

- Großteil Daten nicht verwendet
- Optimistisches Eliminieren von Kopien
  - Abfangen von System Calls
  - Datenkonsistenz durch Page faults
  - Speichern `intermediate` Buffer

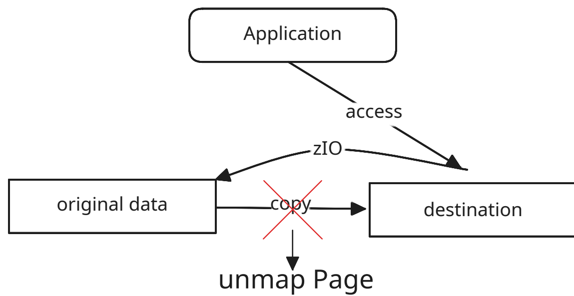


Figure 33: zIO-overview

Zusammenfassung

Evaluation

- **3.8** mal höherer Durchsatz (**Simple** Echo Server)
- **1.3x - 1.8** (Redis)
- Mehr mit NVMM möglich (~2.0x)

Quellen

Bildquellen

virtueller Speicher: Von Virtual_memory.svg: Ehambergderivative work: Shaddim (talk) - Virtual_memory.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=14926975

Seitenabbildung: Von Traced by User:Stannered, original by en:User:Dysprosia - en:Image:Virtual address space and physical address space relationship.png, BSD, https://commons.wikimedia.org/w/index.php?curid=1895627

Alle Bilder aus dem Abschnitt Evaluation: aus dem Paper zIO [3]

Beispiel Grafik Folie 27, 43: Inspiriert durch Paper zIO [3]

Quellen

[1] E. B. washuu, “ZeroCopy : Techniques , benefits and pitfalls,” Available: https://api.semanticscholar.org/CorpusID:13983599

[2] Wikipedia contributors, “Virtual memory — Wikipedia, the free encyclopedia.” 2023. Available: https://en.wikipedia.org/w/index.php?title=Virtual_memory&oldid=1185908051

[3] T. Stamler, D. Hwang, A. Raybuck, W. Zhang, and S. Peter, “zIO: Accelerating IO-Intensive applications with transparent Zero-Copy IO,” in *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, Carlsbad, CA: USENIX Association, Jul. 2022, pp. 431–445. Available: https://www.usenix.org/conference/osdi22/presentation/stamler