Bachelor's Thesis

Efficient Synchronization of Linux Memory Regions over a Network

A Comparative Study and Implementation

Author: Felicitas Pojtinger
University: Hochschule der Medien Stuttgart
Course of Study: Media Informatics
Date: 2023-08-03
Academic Degree: Bachelor of Science
Primary Supervisor: Prof. Dr. Martin Goik
Secondary Supervisor: M.Sc. Philip Betzler

r3map

Loophole Labs

Introduction

Today’s Technological Landscape

What if there were a better way?

Thesis Structure

1. Introduction to Base Technologies
2. Overview of Access Methods
3. Implementing Select Access Methods
4. Analyzing Performance Benchmarks
5. Discussion of Benchmarks and Technologies
6. Conclusion and Future Outlook

Presentation Structure

1. Introduction
2. Methods
3. Optimizations
4. Results and Discussion
5. Implemented Use Cases
6. Future Use Cases
7. Conclusion

Access Methods

Userfaults

Principle of Locality

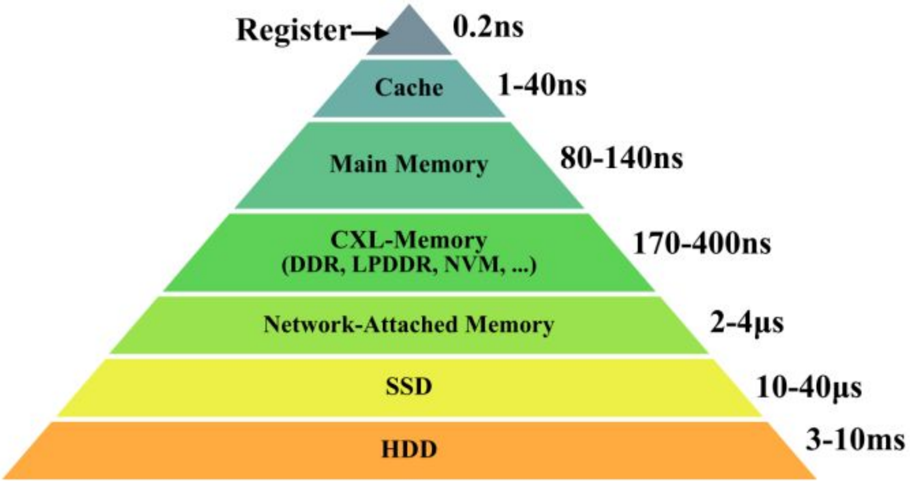

Figure 1: Latencies for different memory technologies showing, from lowest to highest latency, registers, cache, main memory, CXL memory, network-attached memory, SSDs and HDDs [12]

Memory Hierarchy

Page Faults

An Overview of Userfaults

```
// Creating the `userfaultfd` API
uffd, _, errno := syscall.Syscall(constants.NR_userfaultfd, 0, 0, 0)

uffdioAPI := constants.NewUffdioAPI(
    constants.UFFD_API,
    0,
)
// ...

// Registering a region
uffdioRegister := constants.NewUffdioRegister(
    constants.CULong(start),
    constants.CULong(1),
    constants.UFFDIO_REGISTER_MODE_MISSING,
)
// ...
syscall.Syscall(
  syscall.SYS_IOCTL,
  uffd,
  constants.UFFDIO_REGISTER,
  uintptr(unsafe.Pointer(&uffdioRegister))
)
```

Implementing Userfaults

```go
func (a abcReader) ReadAt(p []byte, off int64) (n int, err error) {
    n = copy(p, bytes.Repeat([]byte{'A' + byte(off%20)}, len(p)))

    return n, nil
}
```

```go
f, err := os.OpenFile(*file, os.O_RDONLY, os.ModePerm)
b, uffd, start, err := mapper.Register(int(s.Size()))
mapper.Handle(uffd, start, f)
```

```go
// ...
f, err := mc.GetObject(ctx, *s3BucketName, *s3ObjectName, minio.GetObjectOptions{})
b, uffd, start, err := mapper.Register(int(s.Size()))
mapper.Handle(uffd, start, f)
```

Implementing Userfaults

File-Based Synchronization

mmap

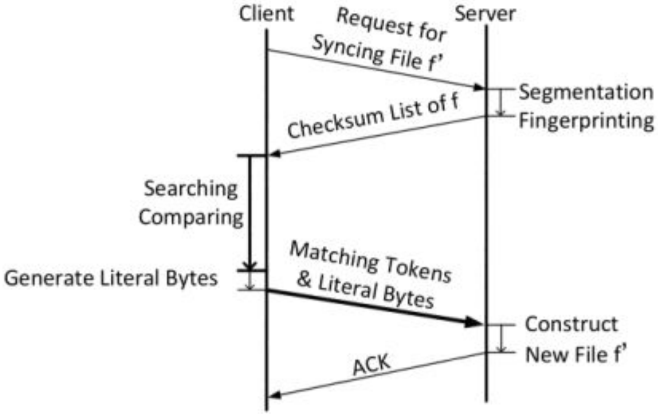

Figure 2: Design flow chart of WebRsync, showing the messages sent between and operations done for server and client in a single synchronization cycle[25]

Delta Synchronization

How can we use mmap and Delta Synchronization?

Caching Restrictions

Detecting File Changes

Speeding Up Hashing

Protocol & Multiplexer Hub

```go
// Receiving remote hashes
remoteHashes := []string{}
utils.DecodeJSONFixedLength(conn, &remoteHashes)
// ...

// Calculating the hashes
localHashes, cutoff, err := GetHashesForBlocks(parallel, path, blocksize)

// Comparing the hashes
blocksToSend := []int64{}
for i, localHash := range localHashes {
    // ...
    if localHash != remoteHashes[i] {
        blocksToSend = append(blocksToSend, j)

        continue
    }
}

// Sending the non-matching hashes
utils.EncodeJSONFixedLength(conn, blocksToSend)
```

```go
// The lock and semaphore
var wg sync.WaitGroup
wg.Add(int(blocks))

lock := semaphore.NewWeighted(parallel)

// ...

// Concurrent hash calculation
for i := int64(0); i < blocks; i++ {
    j := i

    go calculateHash(j)
}
wg.Wait()
```

```go
// Local hash calculation
localHashes, _, err := GetHashesForBlocks(parallel, path, blocksize)
// Sending the hashes to the remote
// Receiving the remote hashes and the truncation request
blocksToFetch := []int64{}
utils.DecodeJSONFixedLength(conn, &blocksToFetch)
// ...
cutoff := int64(0)
utils.DecodeJSONFixedLength(conn, &cutoff)
```

```go
case "src-control":
    // Decoding the file name
    file := ""
    utils.DecodeJSONFixedLength(conn, &file)
    // ...

    syncerSrcControlConns[file] = conn

    syncerSrcControlConnsBroadcaster.Broadcast(file)
    // ...
case "dst-control":
    var wg sync.WaitGroup
    wg.Add(1)

    go func() {
        // Subscription to send all future file names
        l := syncerSrcControlConnsBroadcaster.Listener(0)

        for file := range l.Ch() {
            utils.EncodeJSONFixedLength(conn, file)
            // ...
        }
    }()

    // Sending the previously known file names
    for file := range syncerSrcControlConns {
        utils.EncodeJSONFixedLength(conn, file)
        // ...
    }

    wg.Wait()
```

Delta Synchronization

Limitations

FUSE

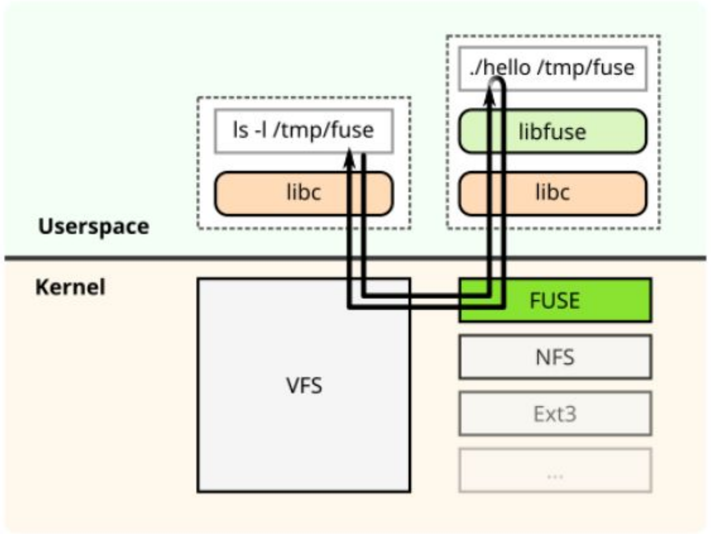

Figure 3: Structural diagram of FUSE, showing the user space components handled by the C library and the FUSE library as well as the kernel components such as the Linux VFS and the FUSE kernel module[27]

What is FUSE?

```
static int example_getattr(const char *path, struct stat *stbuf,
                           struct fuse_file_info *fi);
```

```
static int example_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                           off_t offset, struct fuse_file_info *fi,
                           enum fuse_readdir_flags flags);
```

```
static int example_open(const char *path, struct fuse_file_info *fi);
```

```
static int example_read(const char *path, char *buf, size_t size, off_t offset, struct
    fuse_file_info *fi);
```

FUSE Syscalls

FUSE for Memory Synchronization

Limitations

NBD

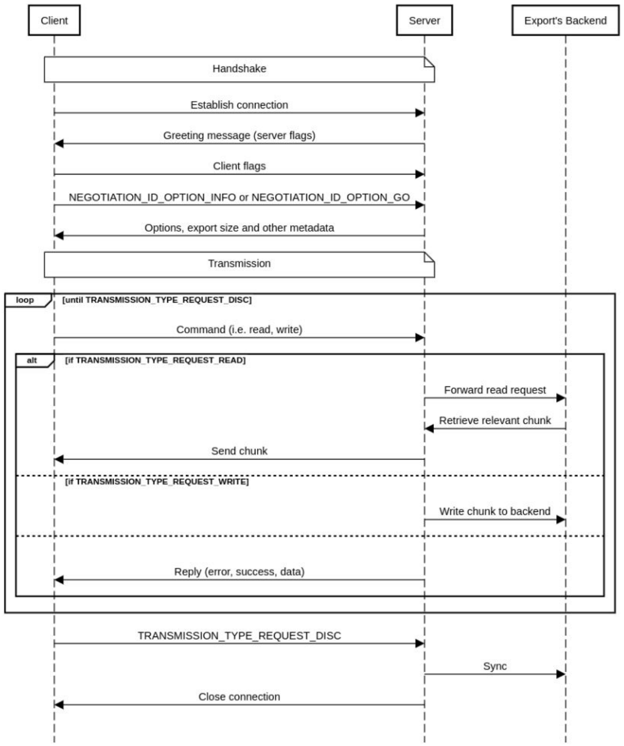

Figure 4: Sequence diagram of the baseline NBD protocol (simplified), showing the handshake, transmission and disconnect phases

NBD Protocol

NBD for Memory Synchronization

go-nbd

```go
type Backend interface {
    ReadAt(p []byte, off int64) (n int, err error)
    WriteAt(p []byte, off int64) (n int, err error)
    Size() (int64, error)
    Sync() error
}
```

```go
func Handle(conn net.Conn, exports []Export, options *Options) error
```

Server

```go
func Connect(conn net.Conn, device *os.File, options *Options) error
```

```go
// Connecting to `udev`
udevConn.Connect(netlink.UdevEvent)

// Subscribing to events for the device name
udevConn.Monitor(udevReadyCh, udevErrCh, &netlink.RuleDefinitions{
    Rules: []netlink.RuleDefinition{
        {
            Env: map[string]string{
                "DEVNAME": device.Name(),
            },
        },
    },
})

// Waiting for the device to become available
go func() {
    // ...
    <-udevReadyCh

    options.OnConnected()
}()
```

Client

Combining Server and Client into Mounts

Managed Mounts

RTT, LAN and WAN

r3map

Why not just NBD?

Chunking

Background Pull and Push

```go
type ReadWriterAt interface {
    ReadAt(p []byte, off int64) (n int, err error)
    WriteAt(p []byte, off int64) (n int, err error)
}
```

Pipeline Stages

Pipeline Components

- ArbitraryReadWriterAt
- ChunkedReadWriterAt
- SyncedReadWriterAt: Background Pull and Push

Concurrent Initialization

```go
type ManagedMountHooks struct {
    OnBeforeSync func() error
    OnBeforeClose func() error
    OnChunkIsLocal func(off int64) error
}
```

Device Lifecycle

Live Migration

Pre-Copy Migration

Post-Copy Migration

Workload Analysis

Why a new API?

Migration Protocol

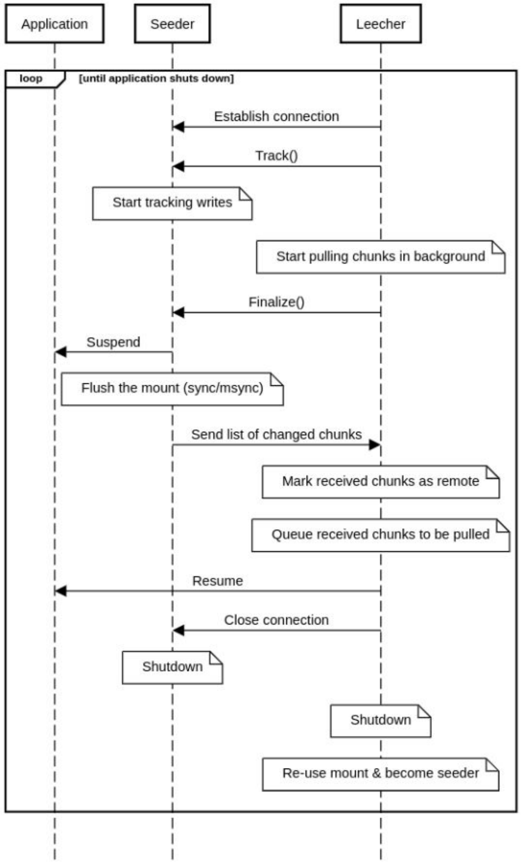

Figure 6: Sequence diagram of the migration protocol (simplified), showing the two protocol phases between the application that is being migrated, the seeder and the leecher components

```go
type SeederRemote struct {
	ReadAt func(context context.Context, length int, off int64) (r ReadAtResponse, err
		error)
	Size   func(context context.Context) (int64, error)
	Track  func(context context.Context) error
	Sync   func(context context.Context) ([]int64, error)
	Close  func(context context.Context) error
}
```

Seeder

```go
// Suspends the remote application, flushes the mount and returns offsets that have been
     written too since `Track()`
dirtyOffsets, err := l.remote.Sync(l.ctx)

// Marks the chunks as remote, causing subsequent reads to pull them again
l.syncedReadWriter.MarkAsRemote(dirtyOffsets)

// Schedules the chunks to be pulled in the background immediately
l.puller.Finalize(dirtyOffsets)

// Unlocks the local resource for reading
l.lockableReadWriterAt.Unlock()
```

Leecher

Optimizations

Pluggable Encryption, Authentication and Transport

Concurrent Backends

Remote Stores as Backends

```go
func (b *RedisBackend) ReadAt(p []byte, off int64) (n int, err error) {
    // Retrieve a key corresponding to the chunk from Redis
    val, err := b.client.Get(b.ctx, strconv.FormatInt(off, 10)).Bytes()
    // If  a key does not exist, treat it as an empty chunk
    if err == redis.Nil {
        return len(p), nil
    }
    // ...
}

func (b *RedisBackend) WriteAt(p []byte, off int64) (n int, err error) {
    // Store an offset as a key-value pair in Redis
    b.client.Set(b.ctx, strconv.FormatInt(off, 10), p, 0)
    // ...
}
```

Redis

```go
func (b *S3Backend) ReadAt(p []byte, off int64) (n int, err error) {
    // Receiving a chunk using Minio's S3 client
    obj, err := b.client.GetObject(b.bucket, b.prefix+"-"+strconv.FormatInt(off, 10),
        minio.GetObjectOptions{})
    if err != nil {
        // If an object is not found, it is treated as an empty chunk
        if err.Error() == errNoSuchKey.Error() {
            return len(p), nil
        }

        // ...
    }
    // ...
}
```

S3

```go
func (b *CassandraBackend) ReadAt(p []byte, off int64) (n int, err error) {
    // Executing a select query for a specific chunk, then scanning it into a byte slice
    var val []byte
    if err := b.session.Query(`select data from `+b.table+` where key = ? limit 1`, b.
        prefix+"-"+strconv.FormatInt(off, 10)).Scan(&val); err != nil {
        if err == gocql.ErrNotFound {
            return len(p), nil
        }

        return 0, err
    }
    // ...
}

func (b *CassandraBackend) WriteAt(p []byte, off int64) (n int, err error) {
    // Upserting a row with a chunk's new content
    b.session.Query(`insert into `+b.table+` (key, data) values (?, ?)`, b.prefix+"-"+
        strconv.FormatInt(off, 10), p).Exec()
    // ...
}
```

Dudirekta

gRPC

fRPC

Results and Discussion

Property	Value
Device Model	Dell XPS 9320
OS	Fedora release 38 (Thirty Eight) x86_64
Kernel	6.3.11-200.fc38.x86_64
CPU	12th Gen Intel i7-1280P (20) @ 4.700GHz
Memory	31687MiB LPDDR5, 6400 MT/s

Testing Environment

Access Methods

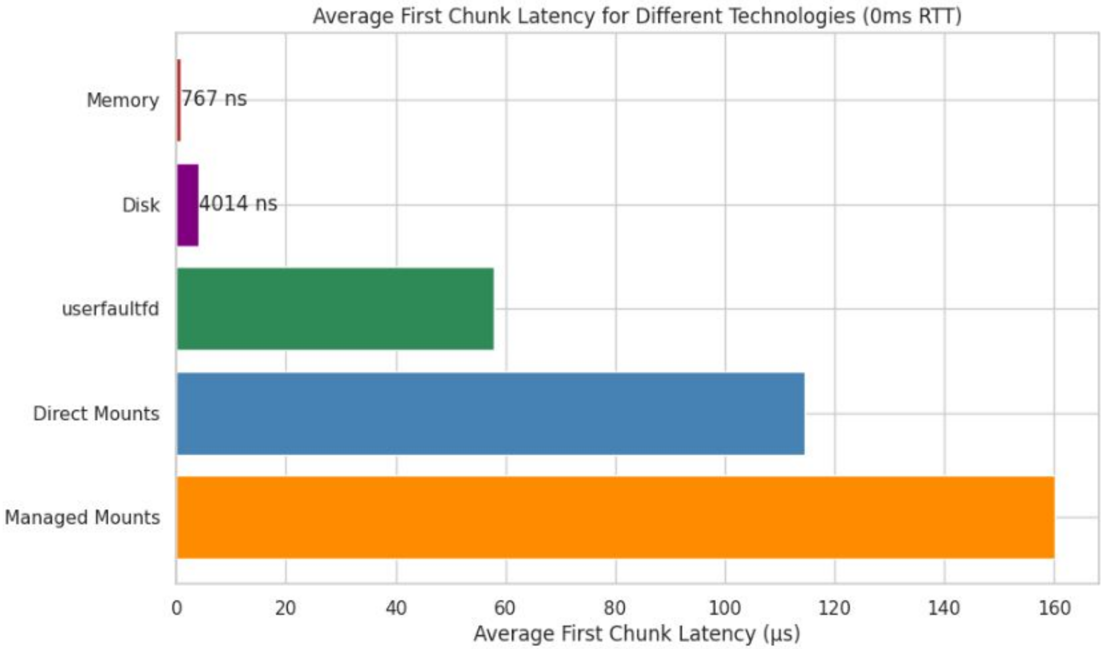

Figure 8: Average first chunk latency for different direct memory access, disk, userfaultfd, direct mounts and managed mounts (0ms RTT)

Latency

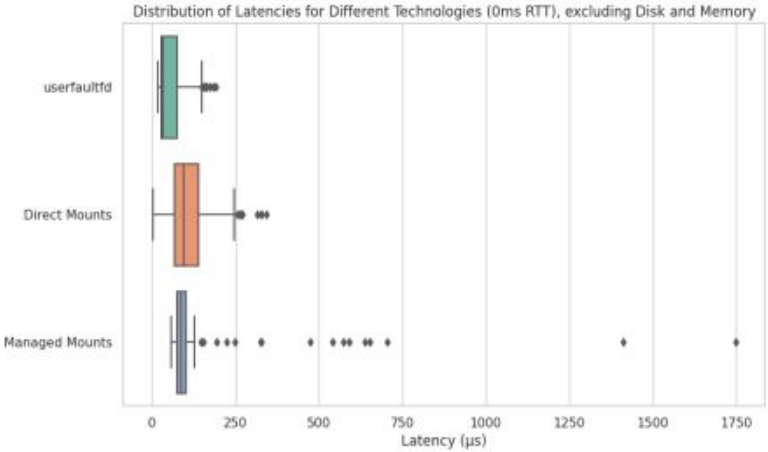

Figure 9: Box plot for the distribution of first chunk latency for userfaultfd, direct mounts and managed mounts (0ms RTT)

Latency

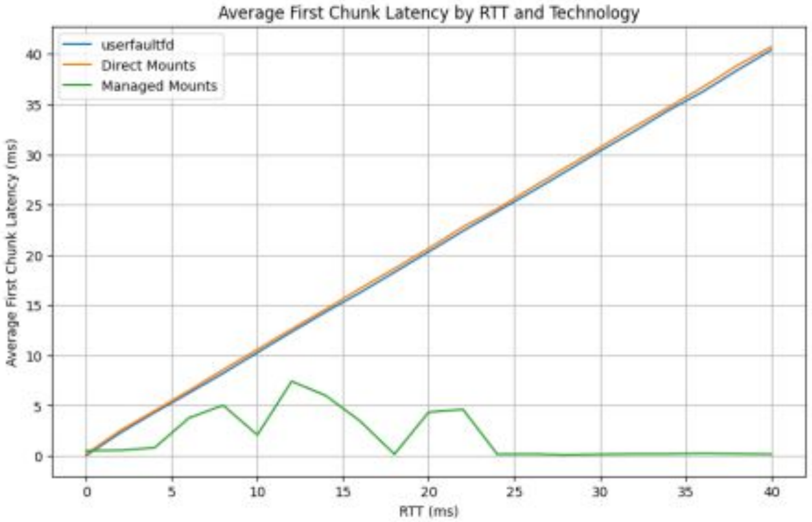

Figure 10: Average first chunk latency for userfaultfd, direct mounts and managed mounts by RTT

Latency

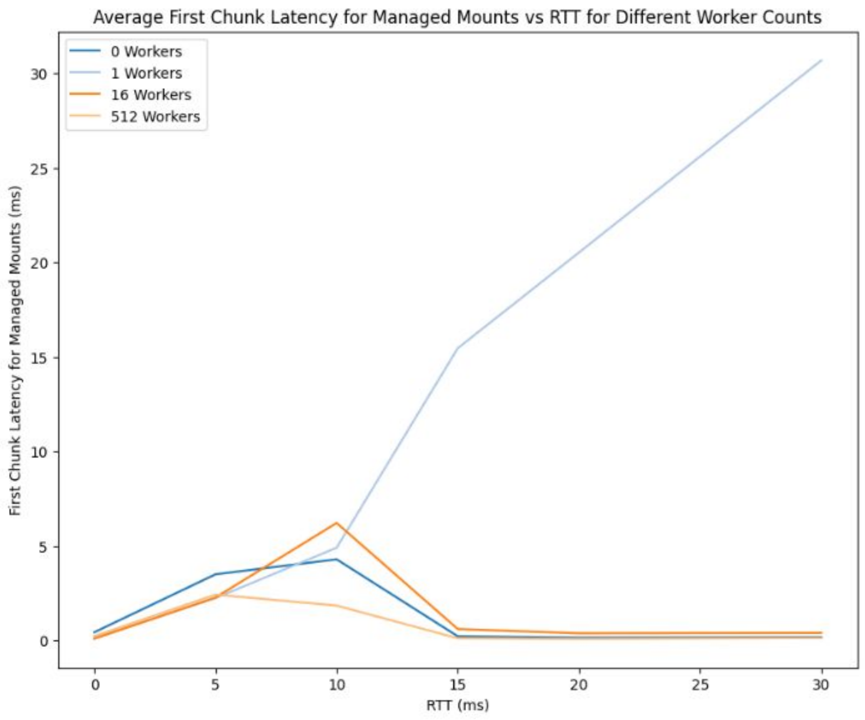

Figure 11: Average first chunk latency for managed workers with 0-512 workers by RTT

Latency

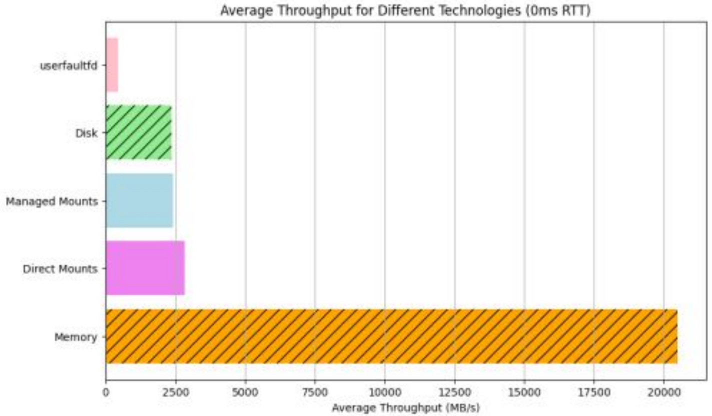

Figure 12: Average throughput for memory, disk, userfaultfd, direct mounts and managed mounts (0ms RTT)

Read Throughput

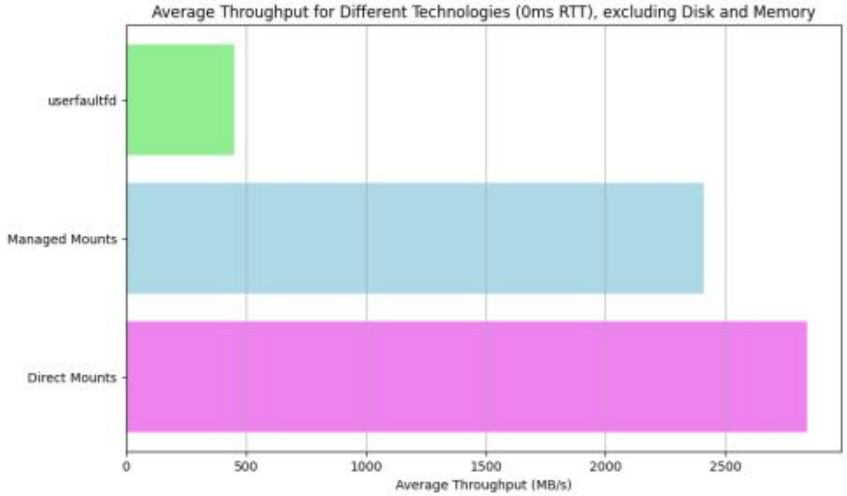

Figure 13: Average throughput for userfaultfd, direct mounts and managed mounts (0ms RTT)

Read Throughput

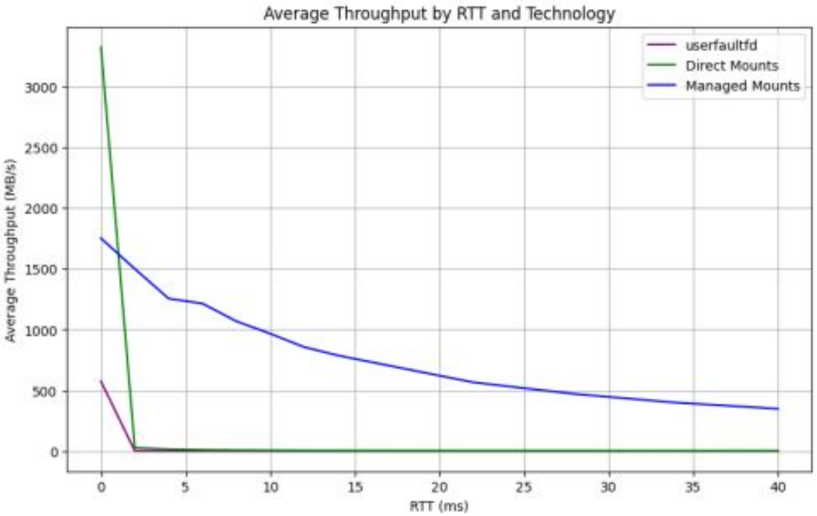

Figure 15: Average throughput for userfaultfd, direct mounts and managed mounts by RTT

Read Throughput

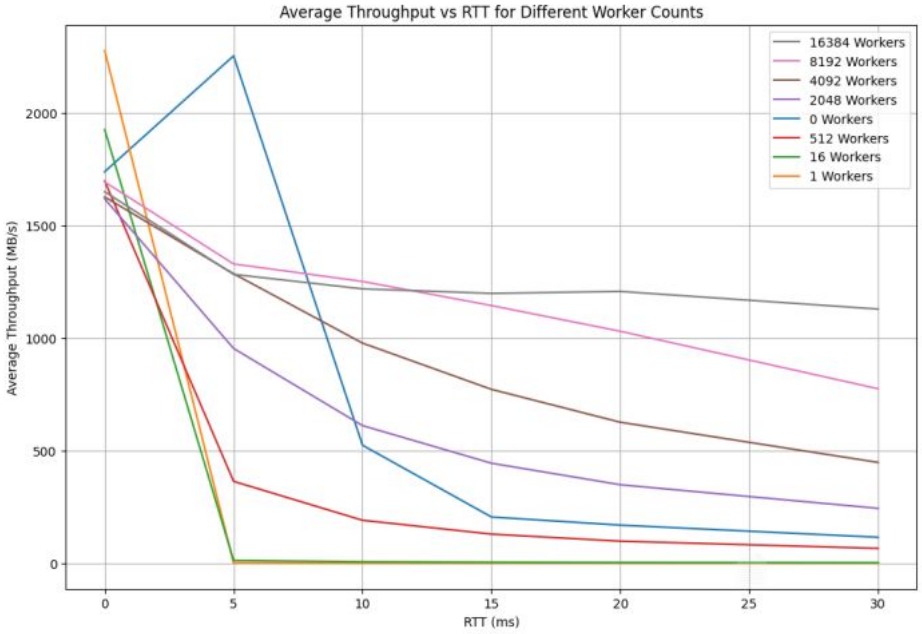

Figure 16: Average throughput for managed mounts with 0-16384 workers by RTT

Read Throughput

5.2.3 Write Throughput

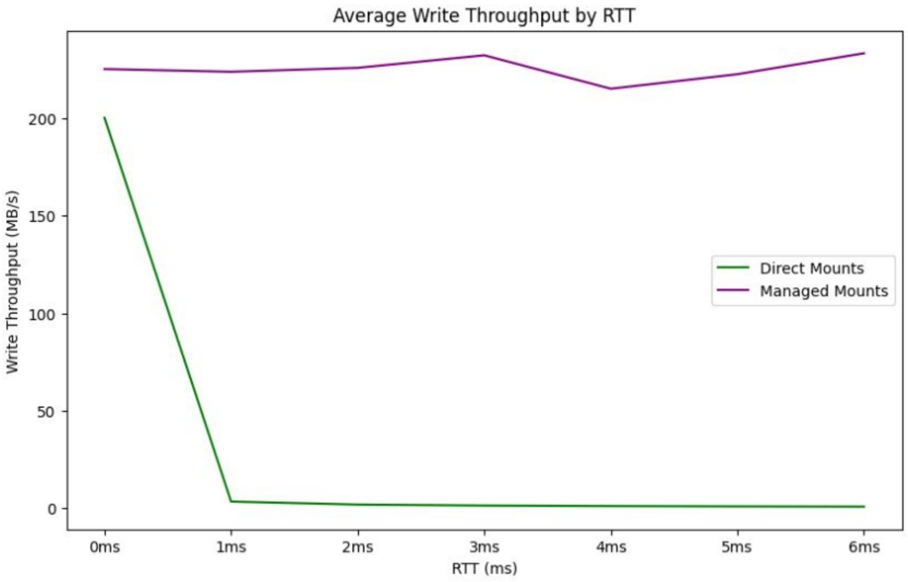

Figure 17: Average write throughput for direct and managed mounts by RTT

Write Throughput

Discussing Access Methods

Initialization

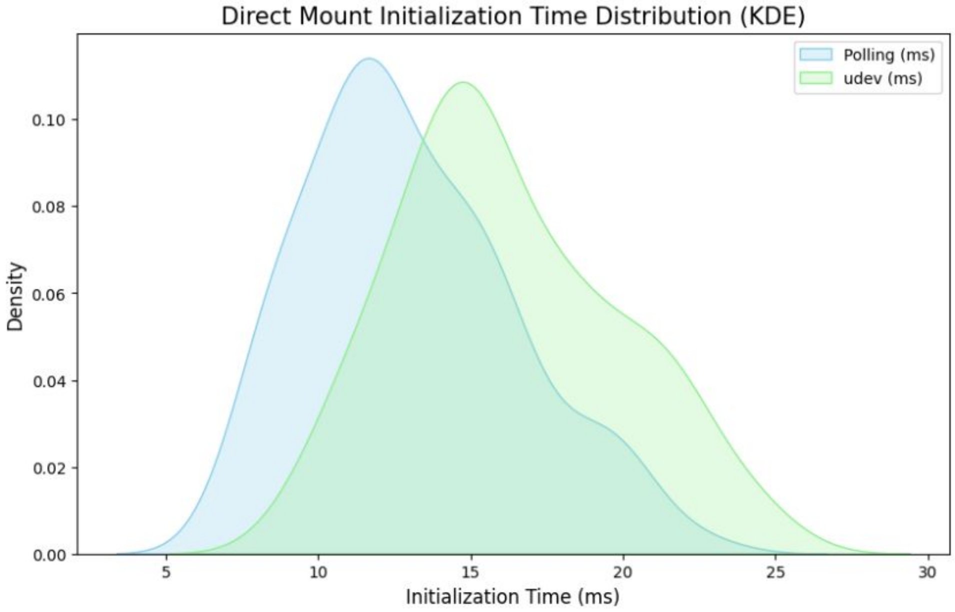

Figure 18: Kernel density estimation for the distribution of direct mount initialization time with polling vs. udev

Initialization

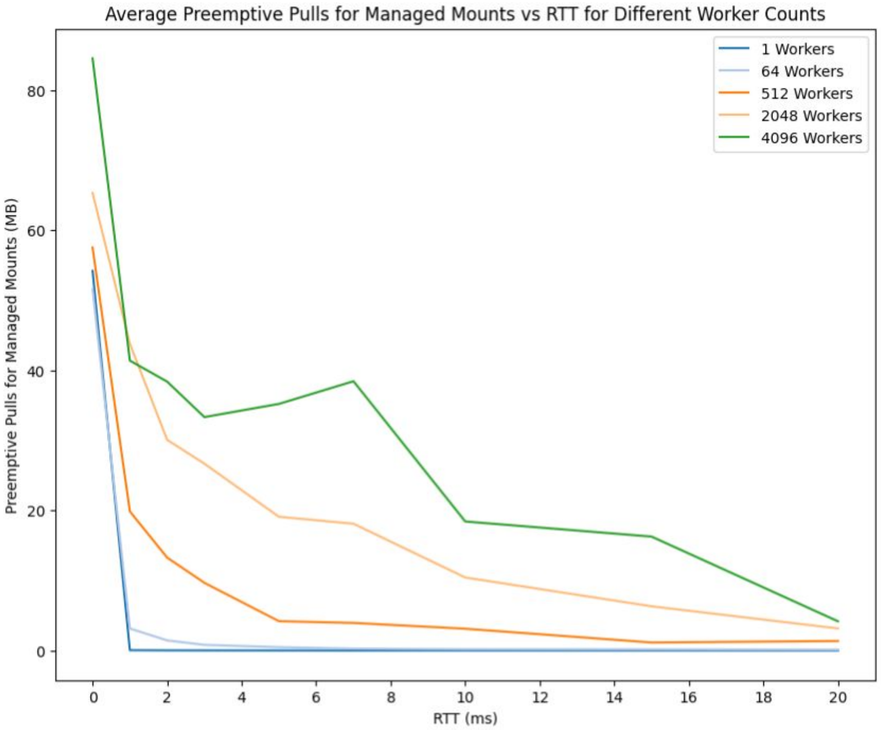

Figure 19: Amount of pre-emptively pulled data for managed mounts with 0-4096 workers by RTT

Initialization

Chunking

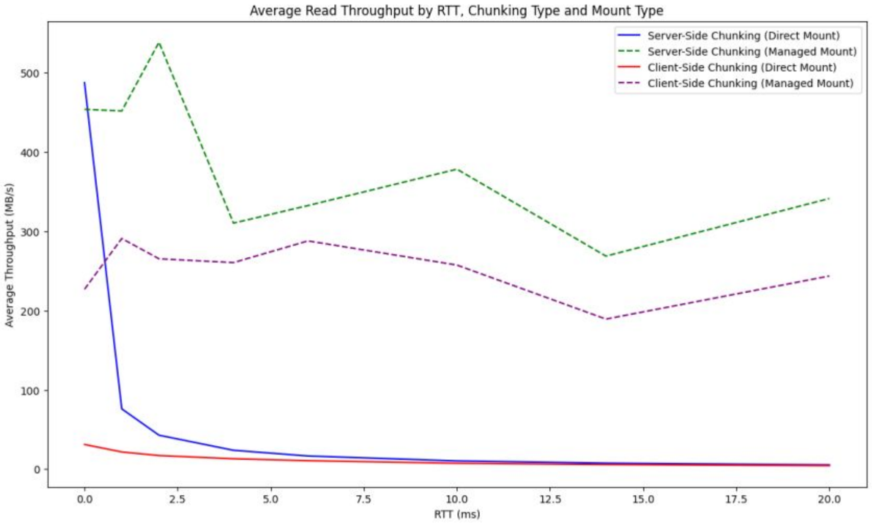

Figure 20: Average read throughput for server-side and client-side chunking, direct mounts and managed mounts by RTT

Read Throughput

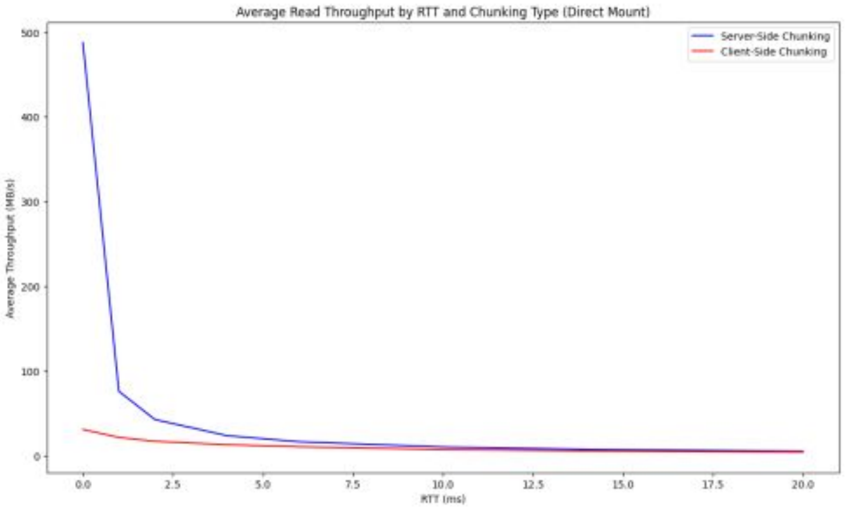

Figure 21: Average read throughput for server-side and client-side chunking with direct mounts by RTT

Read Throughput

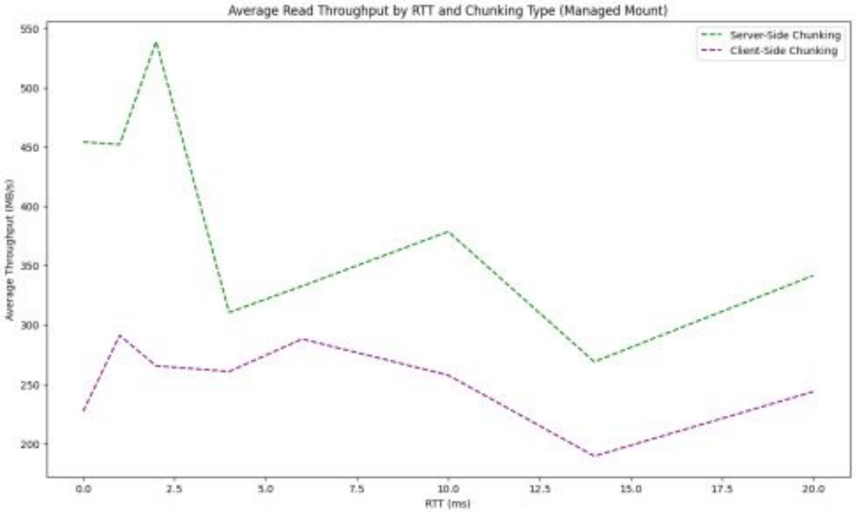

Figure 22: Average read throughput for server-side and client-side chunking with managed mounts by RTT

Read Throughput

RPC Frameworks

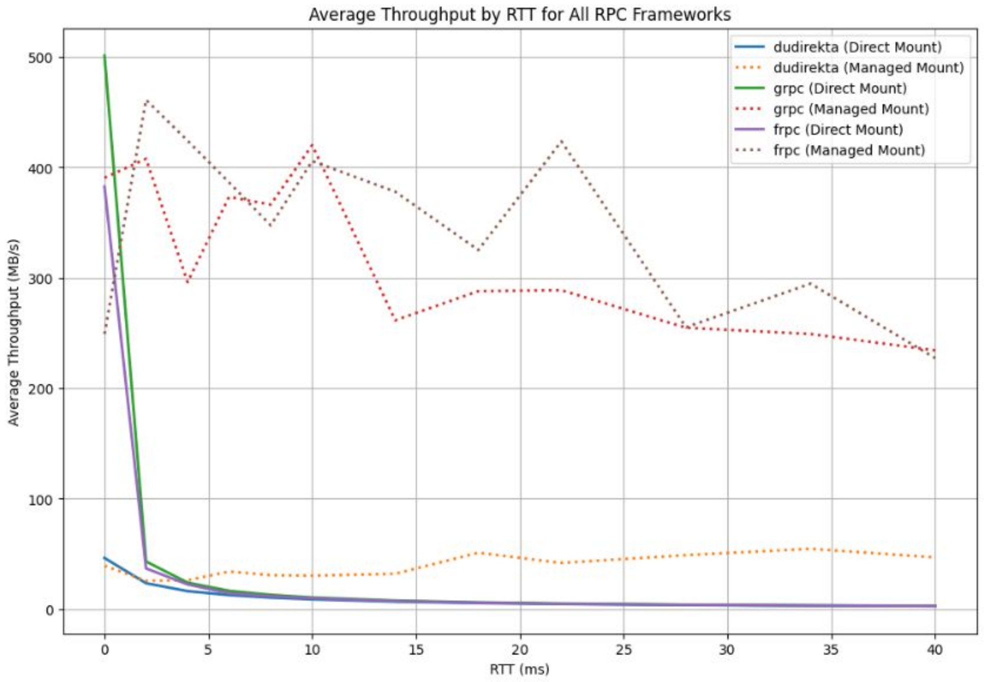

Figure 23: Average throughput by RTT for Dudirekta, gRPC and fRPC frameworks for direct and managed mounts

Throughput

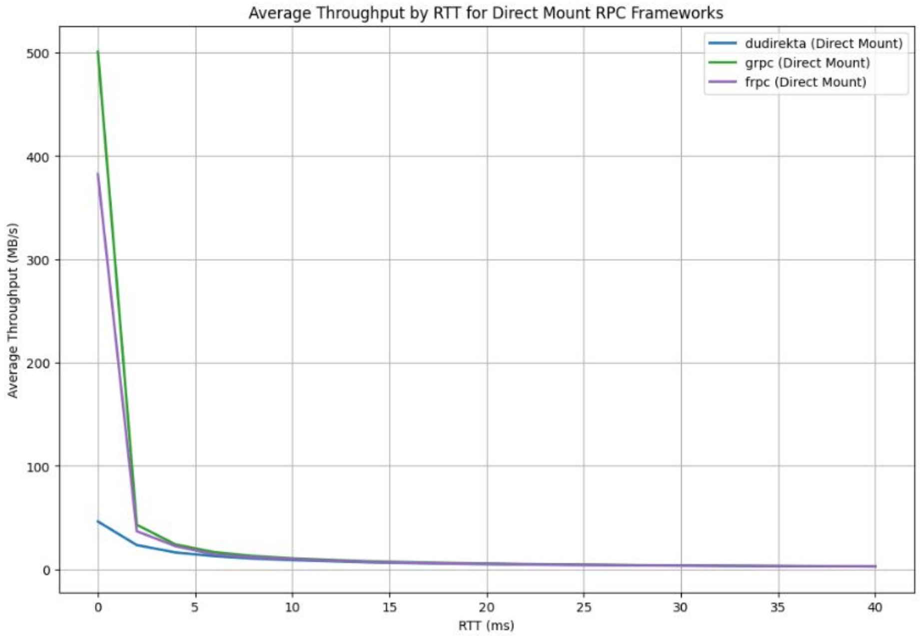

Figure 24: Average throughput by RTT for Dudirekta, gRPC and fRPC frameworks for direct mounts

Throughput

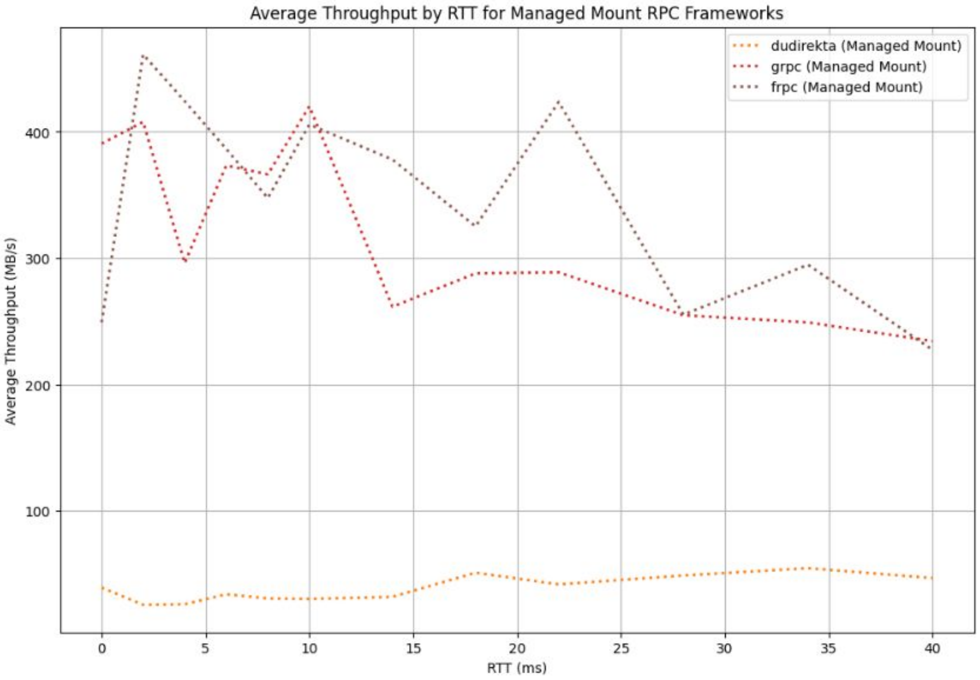

Figure 25: Average throughput by RTT for Dudirekta, gRPC and fRPC frameworks for managed mounts

Throughput

Discussing RPC Frameworks

Backends

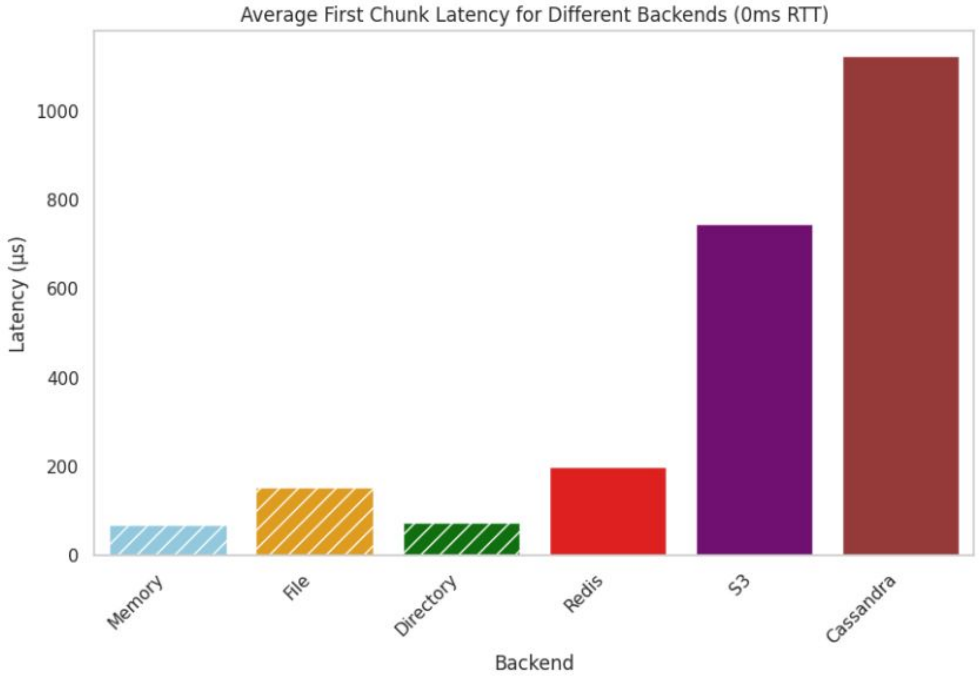

Figure 26: Average first chunk latency for memory, file, directory, Redis, S3 and ScylllaDB backends (0ms RTT)

Latency

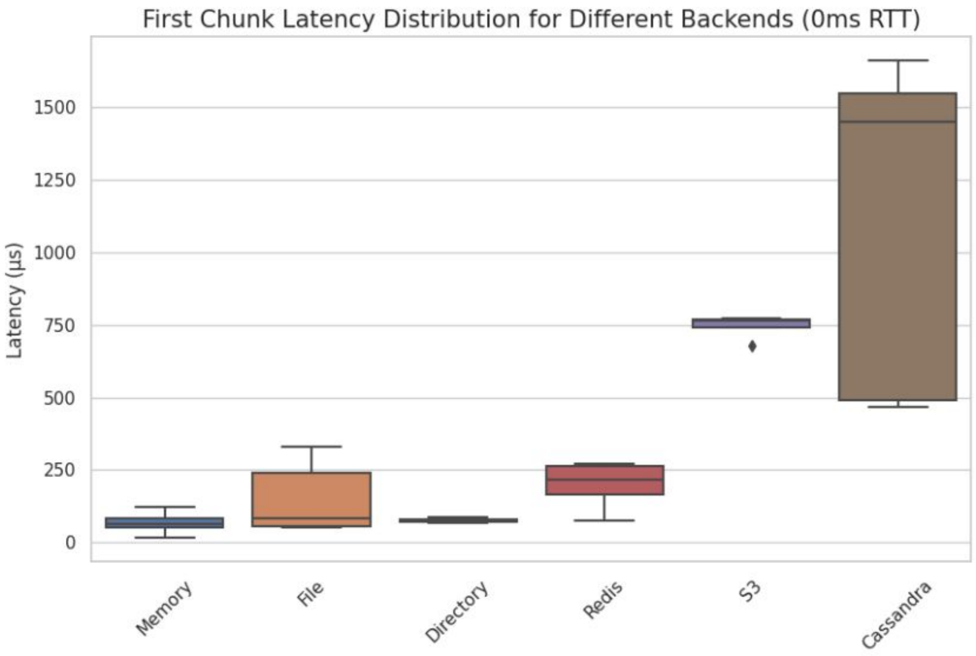

Figure 27: Box plot of first chunk latency distribution for memory, file, directory, Redis, S3 and ScylllaDB (0ms RTT)

Latency

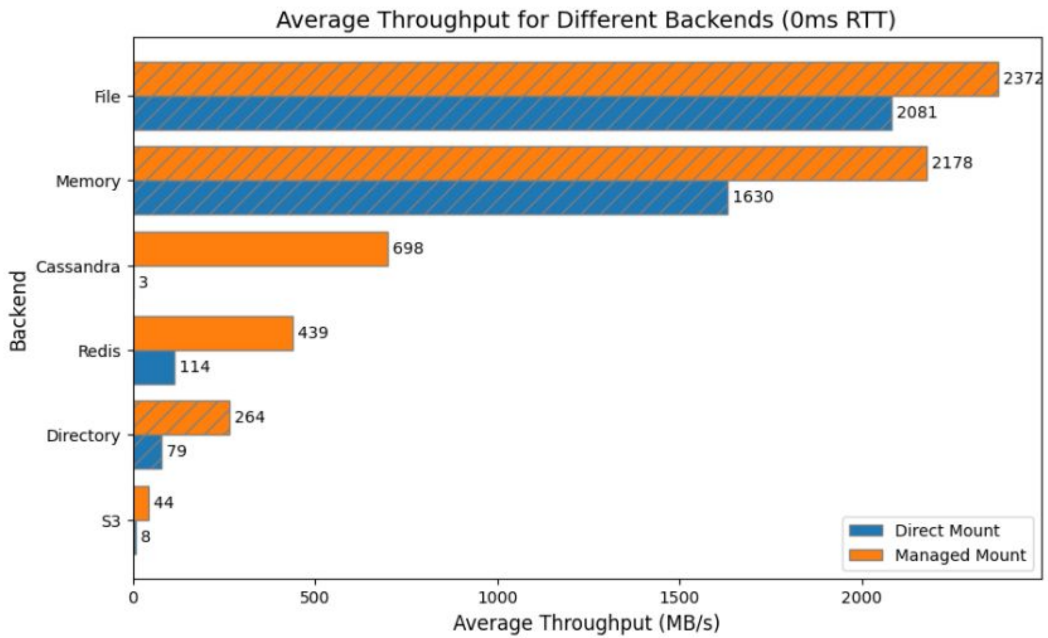

Figure 28: Average throughput for memory, file, directory, Redis, S3 and ScylllaDB backends for direct and managed mounts (0ms RTT)

Throughput

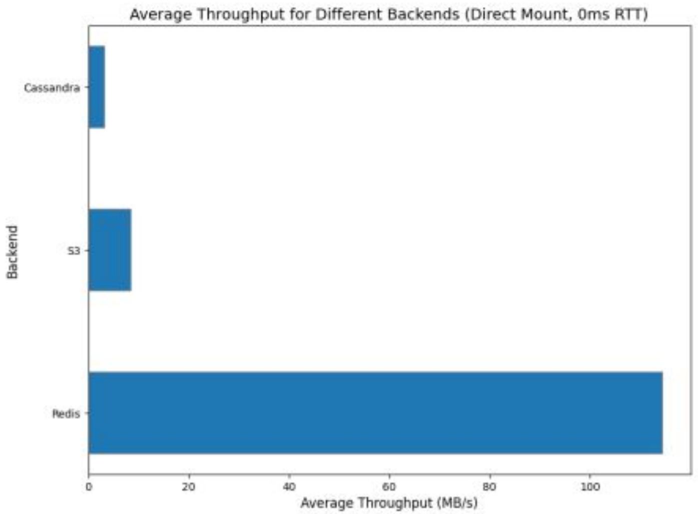

Figure 29: Average throughput for Redis, S3 and ScylllaDB backends for direct mounts (0ms RTT)

Throughput

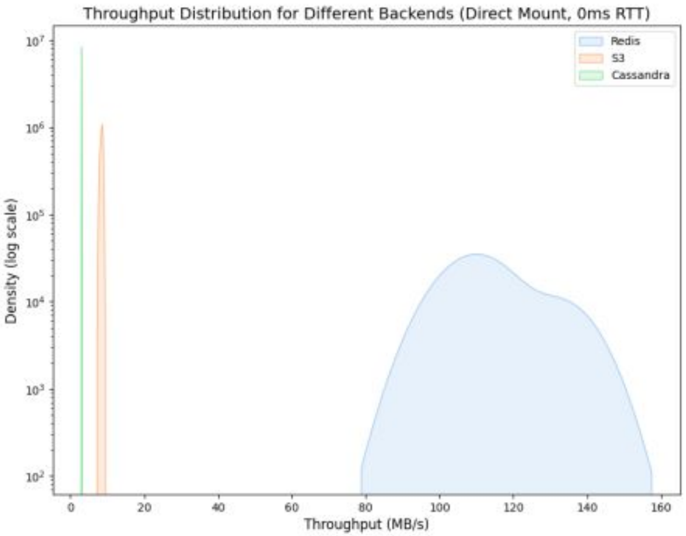

Figure 30: Kernel density estimation (with logarithmic Y axis) for the throughput distribution for Redis, S3 and ScylllaDB for direct mounts (0ms RTT)

Throughput

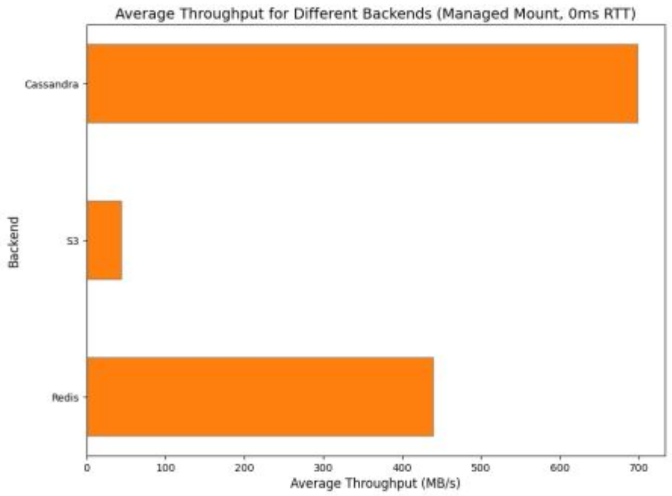

Figure 31: Average throughput for Redis, S3 and ScylllaDB backends for managed mounts (0ms RTT)

Throughput

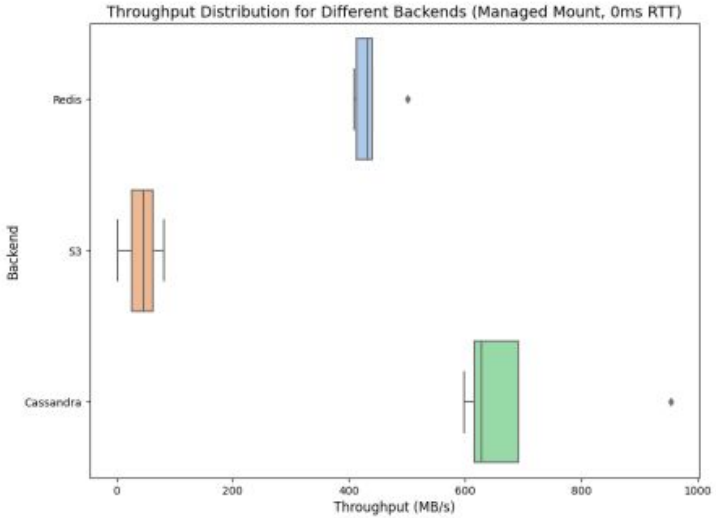

Figure 32: Box plot for the throughput distribution for Redis, S3 and ScylllaDB for managed mounts (0ms RTT)

Throughput

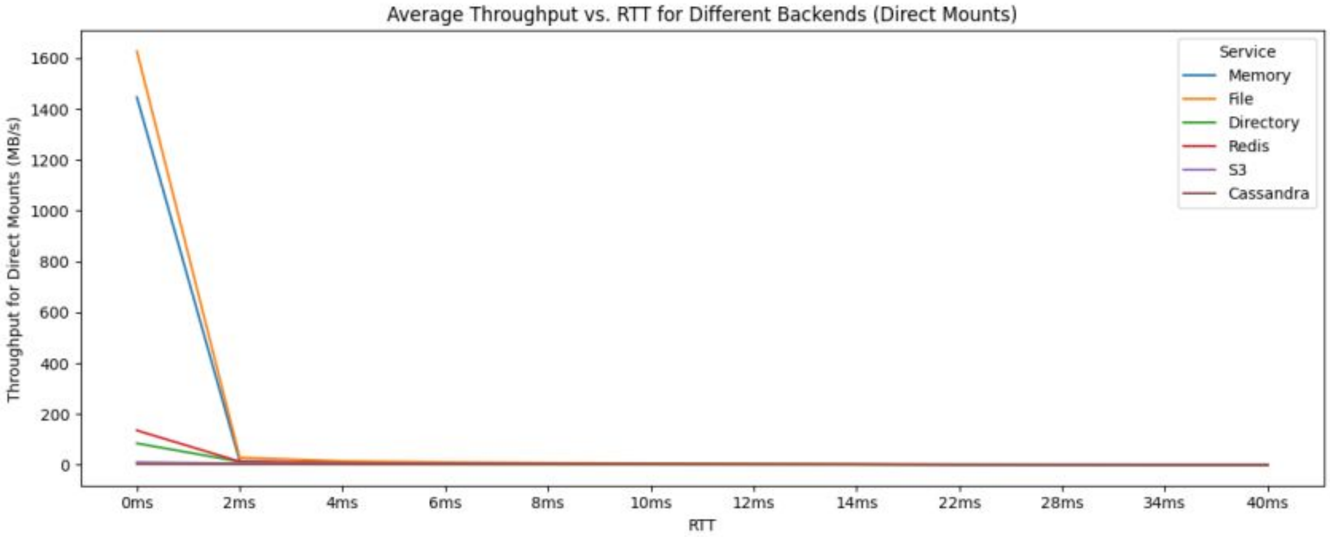

Figure 33: Average throughput for memory, file, directory, Redis, S3 and ScylllaDB backends for direct mounts by RTT

Throughput

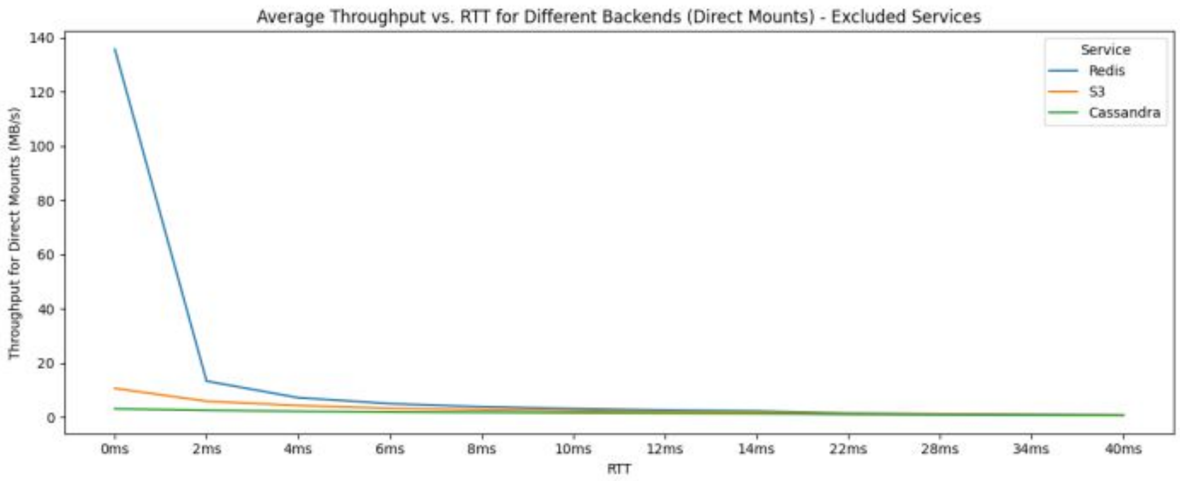

Figure 34: Average throughput for Redis, S3 and ScylllaDB backends for direct mounts by RTT

Throughput

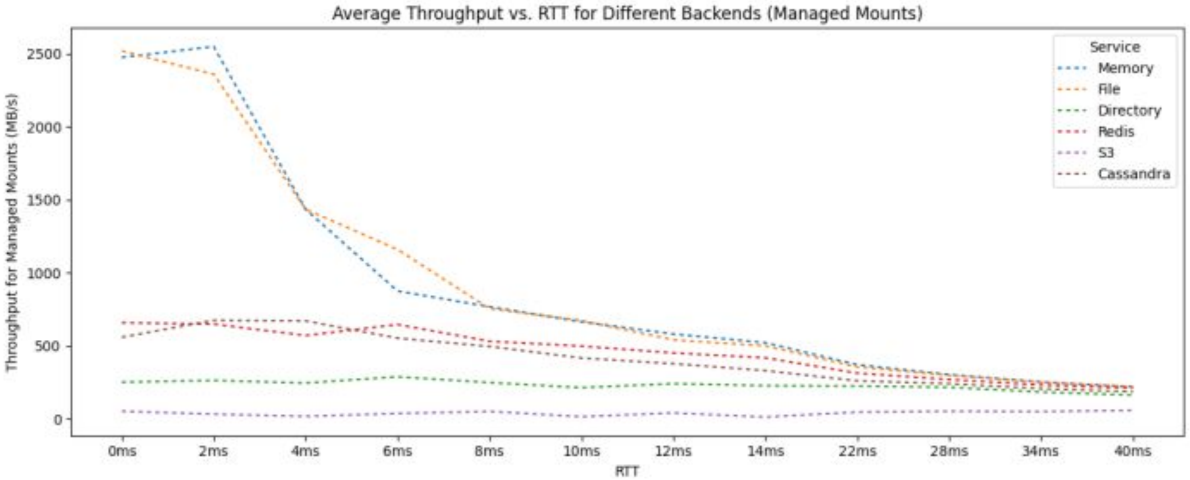

Figure 35: Average throughput for memory, file, directory, Redis, S3 and ScylllaDB backends for managed mounts by RTT

Throughput

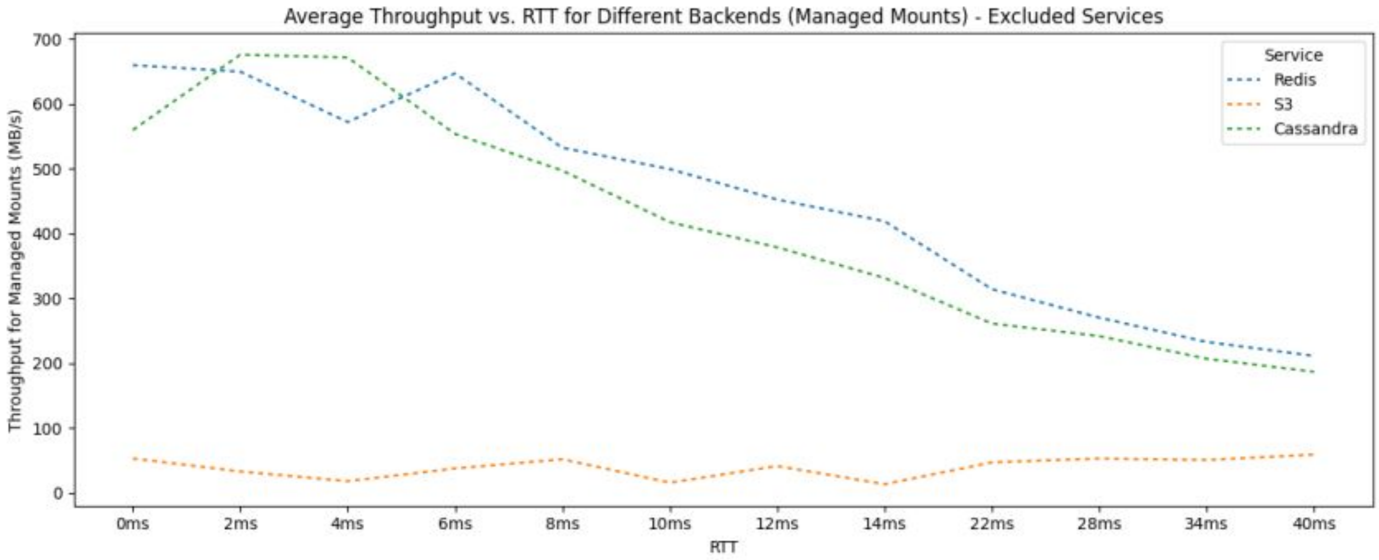

Figure 36: Average throughput for Redis, S3 and ScylllaDB backends for managed mounts by RTT

Throughput

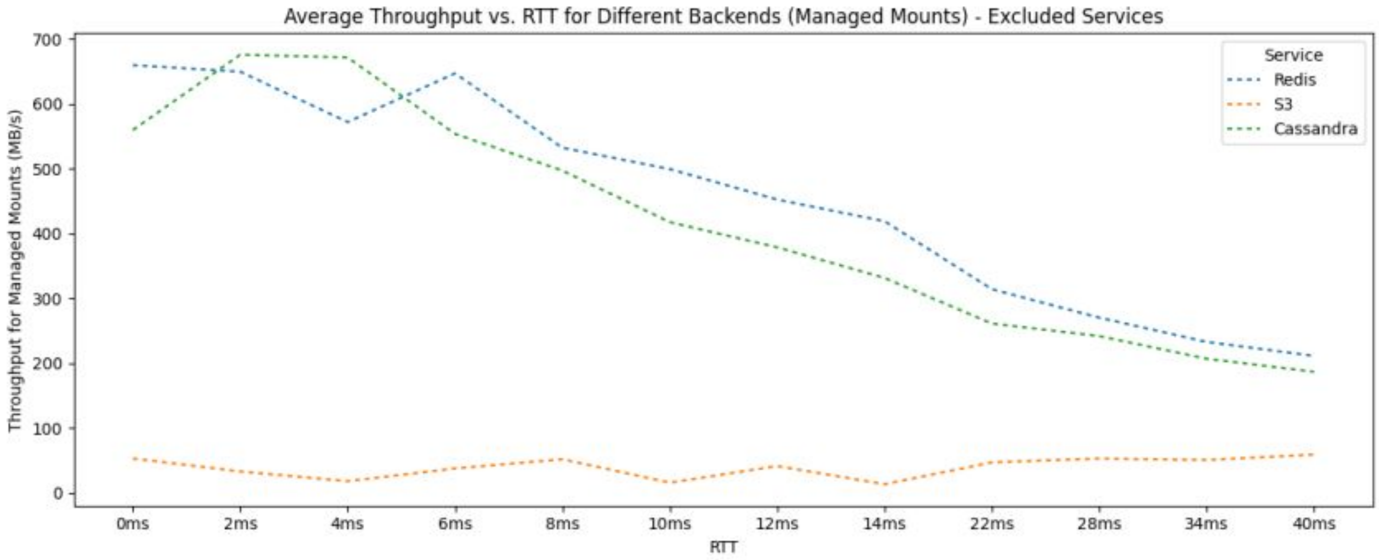

Figure 36: Average throughput for Redis, S3 and ScylllaDB backends for managed mounts by RTT

Throughput

Discussing Backends

Implemented Use Cases

ram-dl

Usage

> TL;DR: "Upload" RAM with `ram-ul`, "download" the RAM with `ram-dl`, done!

1. Upload RAM

On a remote (or local) system, first start `ram-ul`. This component exposes a memory region, file or directory as a fRPC server:

```
$ ram-ul --size 4294967296
2023/06/30 14:52:12 Listening on :1337
```

2. Download RAM

On your local system, start `ram-dl`. This will mount the remote system's exposed memory region, file or directory using fRPC and r3map as swap space, and umount it as soon as you interrupt the app:

```
$ sudo modprobe nbd
$ sudo ram-dl --raddr localhost:1337
2023/06/30 14:54:22 Connected to localhost:1337
2023/06/30 14:54:22 Ready on /dev/nbd0
```

This should give you an extra 4GB of local memory/swap space, without using up significant local memory (or disk space):

```
# Before
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            30Gi       7.9Gi       6.5Gi       721Mi        16Gi        21Gi
Swap:          8.0Gi          0B       8.0Gi

# After
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            30Gi       7.9Gi       6.5Gi       717Mi        16Gi        21Gi
Swap:           11Gi          0B        11Gi
```

🚀 **That's it!** We hope you have fun using `ram-dl`, and if you're interested in more like this, be sure to check out r3map!

ram-dl

tapisk

```go
func (b *TapeBackend) ReadAt(p []byte, off int64) (n int, err error) {
	// Calculating the block for the offset
	block := uint64(off) / b.blocksize

	// Getting the physical record on the tape from the index
	location, err := b.index.GetLocation(block)
	// ...

	// Creating the seek operation
	mtop := &ioctl.Mtop{}
	mtop.SetOp(ioctl.MTSEEK)
	mtop.SetCount(location)

	// Seeking to the record
	syscall.Syscall(
		syscall.SYS_IOCTL,
		drive.Fd(),
		ioctl.MTIOCTOP,
		uintptr(unsafe.Pointer(mtop)),
	)
	// ...

	// Reading the chunk from the tape into memory
	return b.drive.Read(p)
}
```

tapisk

Future Use Cases

Improving Cloud Storage Clients

Universal Database, Media and Asset Streaming

Universal App State Mounts and Migrations

Conclusion

Thanks!

Author: Felicitas Pojtinger
University: Hochschule der Medien Stuttgart
Course of Study: Media Informatics
Date: 2023-08-03
Academic Degree: Bachelor of Science
Primary Supervisor: Prof. Dr. Martin Goik
Secondary Supervisor: M.Sc. Philip Betzler

r3map

Remote mmap: High-performance remote memory region mounts and migrations in user space.

hydrun CI passing | go version >=1.20 | reference | chat 1 users

Overview

r3map is a library that simplifies working with remote memory regions and migrating them between hosts.

It can ...

- **Create a virtual `[]byte` or a virtual file that transparently downloads remote chunks only when they are accessed**: By providing multiple frontends (such as a memory region and a file/path) for accessing or migrating a resource, integrating remote memory into existing applications is possible with little to no changes, and fully language-independent.
- **`mmap` any local or remote resource instead of just files**: By exposing a simple backend interface and being fully transport-independent, r3map makes it possible to map resources such as a **S3 bucket, Cassandra or Redis database**, or even a tape drive into a memory region efficiently, as well as migrating it over an RPC framework of your choice, such as gRPC.
- **Enable live migration features for any hypervisor or application**: r3map implements the APIs which allow for zero-downtime live migration of virtual machines, but makes them generic so that they can be used for any memory region, bringing live migration abilities to almost any hypervisor or application with minimal changes and overhead.
- **Overcome the performance issues typically associated with remote memory**: Despite being in user space, r3map manages (on a typical desktop system) to achieve **very high throughput (up to 3 GB/s)** with **minimal access latencies (~100µs)** and **short initialization times (~12ms)**.
- **Adapt to challenging network environments**: By implementing various optimizations such as **background pull and push**, two-phase protocols for migrations and concurrent device initialization, r3map can be deployed not only in low-latency, high-throughput local datacenter networks but also in more constrained networks like the public internet.

pojntfx/r3map

High-performance remote memory region mounts and migrati
in user space.

Go ★ 21

Efficient Synchronization of Linux Memory Regions over a Network: A Comparative Study and Implementation

Bachelor's thesis by Felicitas Pojtinger.

University: Hochschule der Medien Stuttgart
Course of Study: Media Informatics
Date: 2023-08-03
Academic Degree: Bachelor of Science
Primary Supervisor: Prof. Dr. Martin Goik
Secondary Supervisor: M.Sc. Philip Betzler

Deliverance CI failing

Abstract

Current solutions for access, synchronization and migration of resources over a network are characterized by application-specific protocols and interfaces, which result in fragmentation and barriers to adoption. This thesis aims to address these issues by presenting a universal approach that enables direct operation on a memory region, circumventing the need for custom-built solutions. Various methods to achieve this are evaluated on parameters such as implementation overhead, initialization time, latency and throughput, and an outline of each method's architectural constraints and optimizations is provided. The proposed solution is suitable for both LAN and WAN environments, thanks to a novel approach based on block devices in user space with background push and pull mechanisms. It offers a unified API that enables mounting and migration of nearly any state over a network with minimal changes to existing applications. Illustrations of real-world use cases, configurations and backends are provided, together with a production-ready reference implementation of the full mount and migration APIs via the open-source r3map (remote mmap) library.

Felicitas Pojtinger, 2023
Efficient Synchronization of Linux Memory Regions over a Network
A Comparative Study and Implementation

Felicitas Pojtinger
pojntfx · she/her

Developer specializing in Go, DevOps and modern frontend technologies ✨

Edit profile

Sponsors dashboard

325 followers · **89** following

@loopholelabs
Stuttgart, European Union
00:35 (UTC +02:00)
felicitas@pojtinger.com
https://felicitas.pojtinger.com/
@pojntfx
@pojntfx@mastodon.social
in/pojntfx
https://bsky.app/profile/felicitas.pojtinger.com

Achievements

x3 x3

Beta Send feedback

Organizations

pojntfx / README.md

Hi there 👋

I'm **Felicitas** 🌊, an experienced software engineer with a passion for **systems programming in Go**, all things **DevOps** and the **modern web** 🌐.

pojntfx (Felicitas Pojtinger)

1.61k Contributions in 2023
198 Public Repos
Joined GitHub 6 years ago
felicitas@pojtinger.com

Top Languages by Repo

If you want to find out more about my projects and what I'm currently working on, check out my 🔗 website, ➕ follow me on Bluesky where I post about all things I find interesting 🌌 or check out some of my **featured repos** 👇:

Systems Development	
weron (★ 1307 🛠 Go ⚖ AGPL-3.0 📅 2023) Overlay networks based on WebRTC.	go-nbd (★ 296 🛠 Go ⚖ Apache-2.0 📅 2023) Pure Go NBD server and client library.
liwasc (★ 152 🛠 Go ⚖ AGPL-3.0 📅 2021) List, wake and scan nodes in a network.	bofied (★ 151 🛠 Go ⚖ AGPL-3.0 📅 2021) Modern network boot server.
ram-dl (★ 70 🛠 Go ⚖ Apache-2.0 📅 2023) A tool to download more RAM (yes, seriously!)	stfs (★ 28 🛠 Go ⚖ AGPL-3.0 📅 2022) Simple Tape File System (STFS), a file system for tapes and tar files.
r3map (★ 21 🛠 Go ⚖ Apache-2.0 📅 2023) High-performance remote memory region mounts and migrations in user space.	tapisk (★ 6 🛠 Go ⚖ AGPL-3.0 📅 2023) Expose a tape drive as a block device.

Apps	
keygaen (★ 94 🛠 Go ⚖ AGPL-3.0 📅 2023) Sign, verify, encrypt and decrypt data with PGP in your browser.	multiplex (★ 19 🛠 Go ⚖ AGPL-3.0 📅 2022) Synchronized torrent streaming for distributed watch parties.
html2goapp (★ 16 🛠 Go ⚖ AGPL-3.0 📅 2021) CLI and web app to convert HTML markup to go-app.dev's syntax.	connmapper (★ 6 🛠 TypeScript ⚖ AGPL-3.0 📅 2023) Visualize your system's internet connections on a globe.

Development Tooling	
octarchive (★ 67 🛠 Go ⚖ AGPL-3.0 📅 2023) Simple tool to back up all repos on a GitHub/Gitea account to a local folder.	pojde (★ 62 🛠 Shell ⚖ AGPL-3.0 📅 2021) Develop from any device with a browser.
hydrapp (★ 35 🛠 Go ⚖ Apache-2.0 📅 2023) Build fast apps that run everywhere with Go and a browser engine of your	alpimager (★ 27 🛠 Go ⚖ AGPL-3.0 📅 2021) Build custom Alpine Linux images with