Virtual Machine Memory Overcommitment with UserfaultFD

KVM Forum | September 2024

NUTANIX

Table of Contents

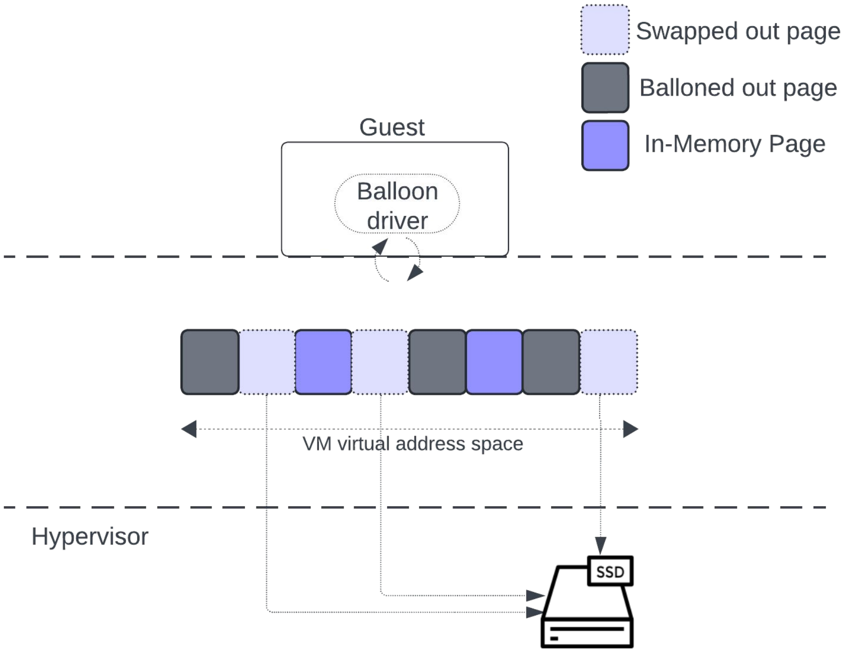
Background

NUTANIX

Memory Overcommitment

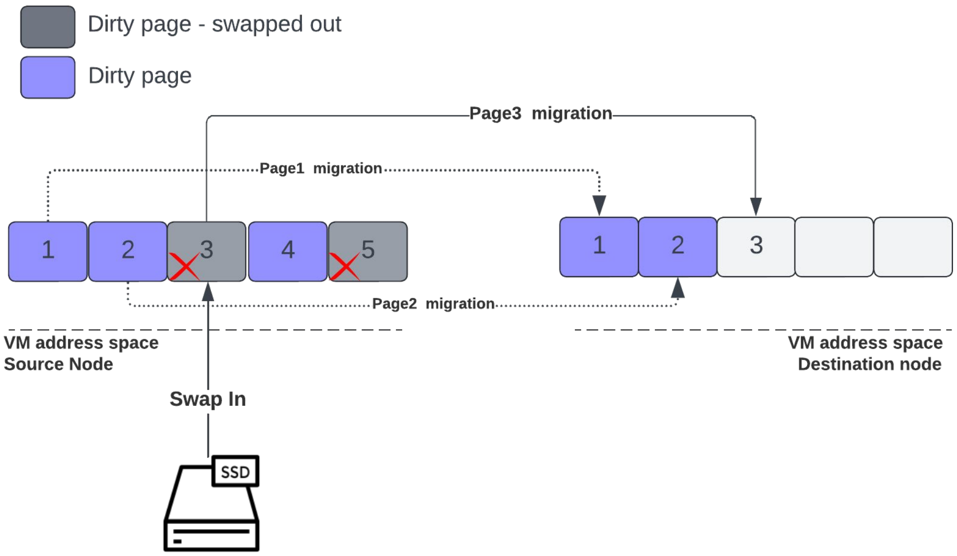
A hypervisor can allocate more guest-physical memory to virtual machines (VMs) than the available host-physical memory (RAM).

- Ballooning: A balloon driver inside the guest OS inflates to occupy more memory, which the hypervisor can then reclaim from VM.
- Hypervisor Swap: The hypervisor can swap memory pages from VMs to disk, freeing up RAM.



Live Migration with Memory Overcommitment

- Transfer all dirty pages from source to destination
- Swapped out pages are faulted in and transferred to destination



Current Challenges

NUTANIX

Live Migration Challenges

- Live migration touches all guest memory, causing page faults on swapped-out pages
  - Page faults are costly
  - Impact on live migration **data transfer throughput**
  - Causes **heavy page thrashing**

- Slower data transfer give guest more time to dirty memory, creating a vicious cycle
  - Overall **huge time to migrate** VMs

- Live migration disturbs guest **active working set**
  - Faulted pages become MRU
  - Causes bad page reclamation decisions

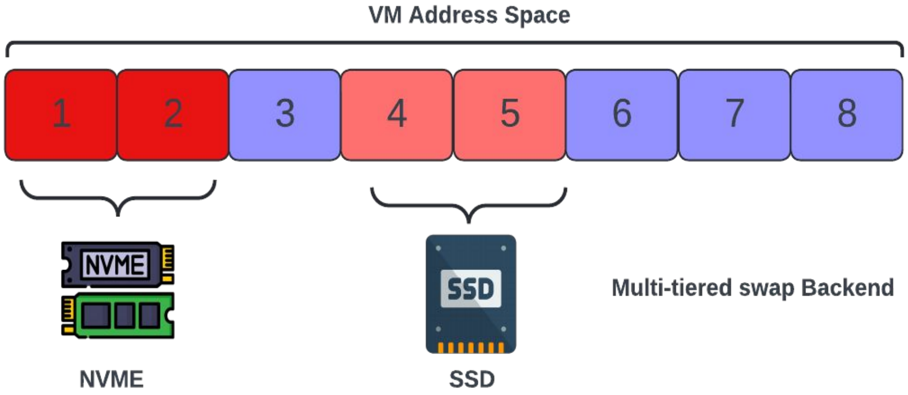
Lack of control for user customizations

- Need **full control over swapped out pages** and choose where to place it
- Make it **lightweight** by removing bloat that we don’t need
- Enable **swap policies specific to each VM**

Swap Tiering

If we have control over swapped out pages, we can optimally utilize hierarchy of multiple swap devices.

- HDD, SATA SSD, NVMe SSD devices and temporary RAM read only caches
- Faster devices are costlier and have limited capacity
- Swap out warmer pages to faster devices



Design and Workflow

NUTANIX

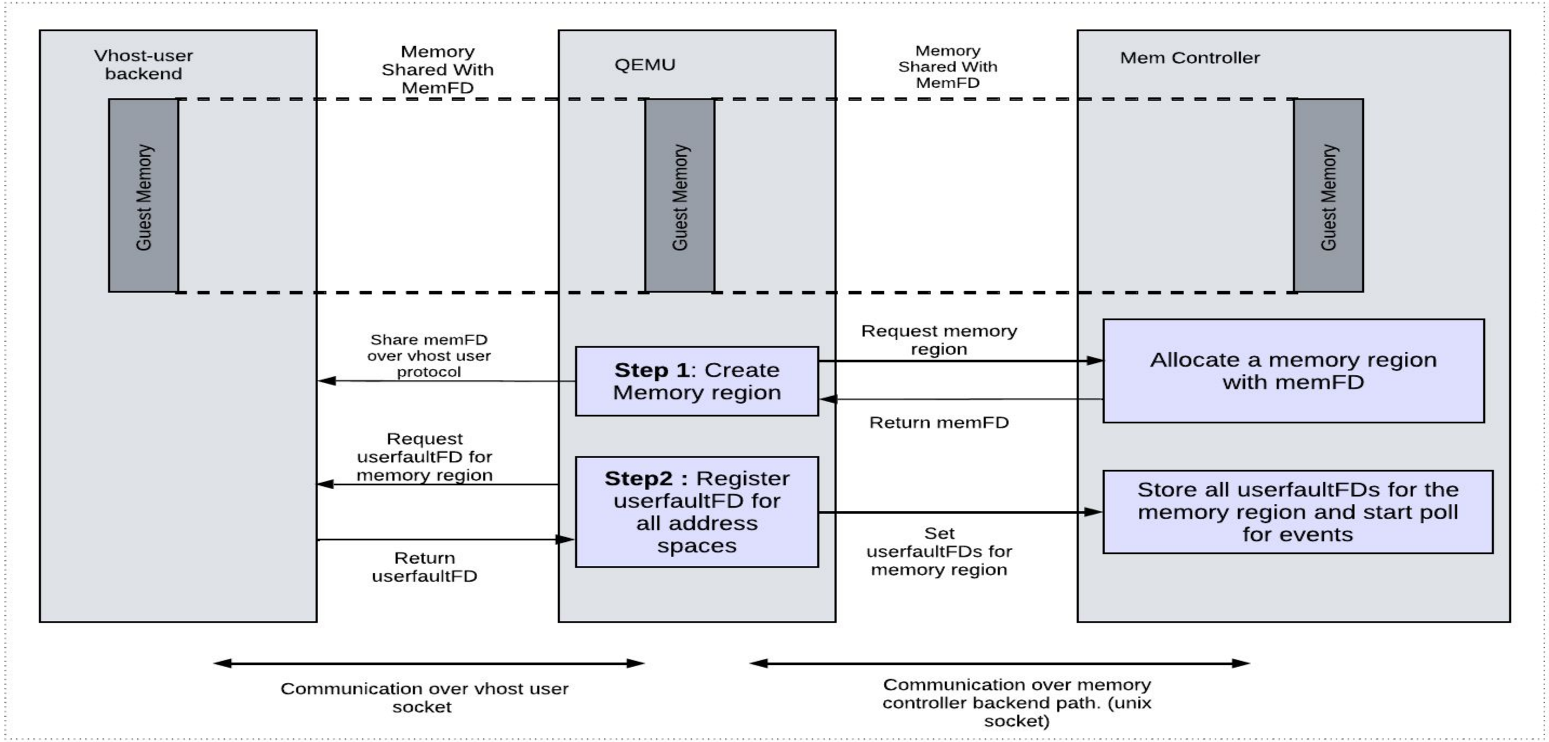
How UserfaultFD helps?

- **Userspace memory manager**
- What is userfaultFD?
  - Userspace **page fault handler**
  - Process waits until page fault is resolved by userspace
  - Supports **async page fault** for guests
  - No control over page swap out path

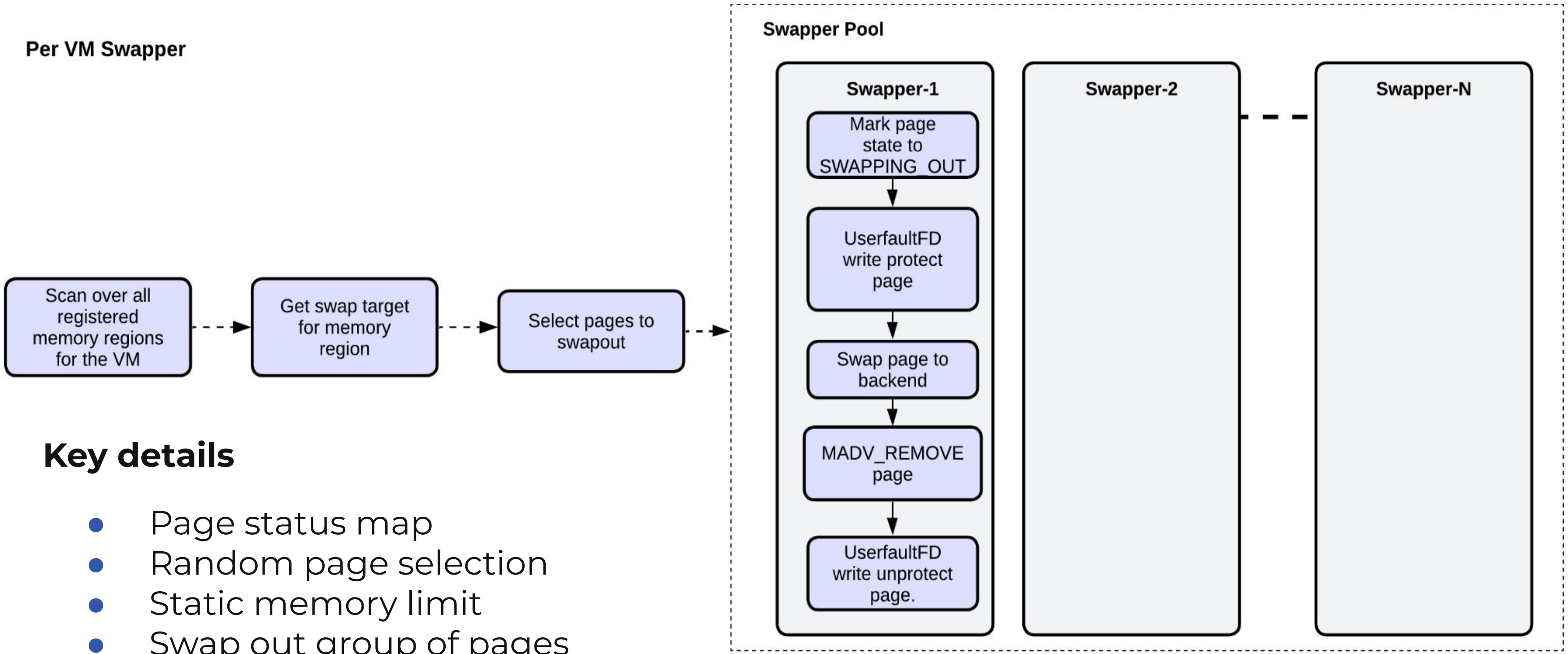
High Level Summary

- **Disable Native Swapping** for QEMU processes
- Introduce **common external service (Mem-controller)**
  - Handles and manages memory for all QEMU processes
- QEMU uses **shared memory** allocated by the service.
- Communicates with QEMU over a **UNIX socket**
- Takes full control over the VM’s address space using **userfaultfd**
  - All swapping and page faults decisions are managed by mem-controller

Initial Setup



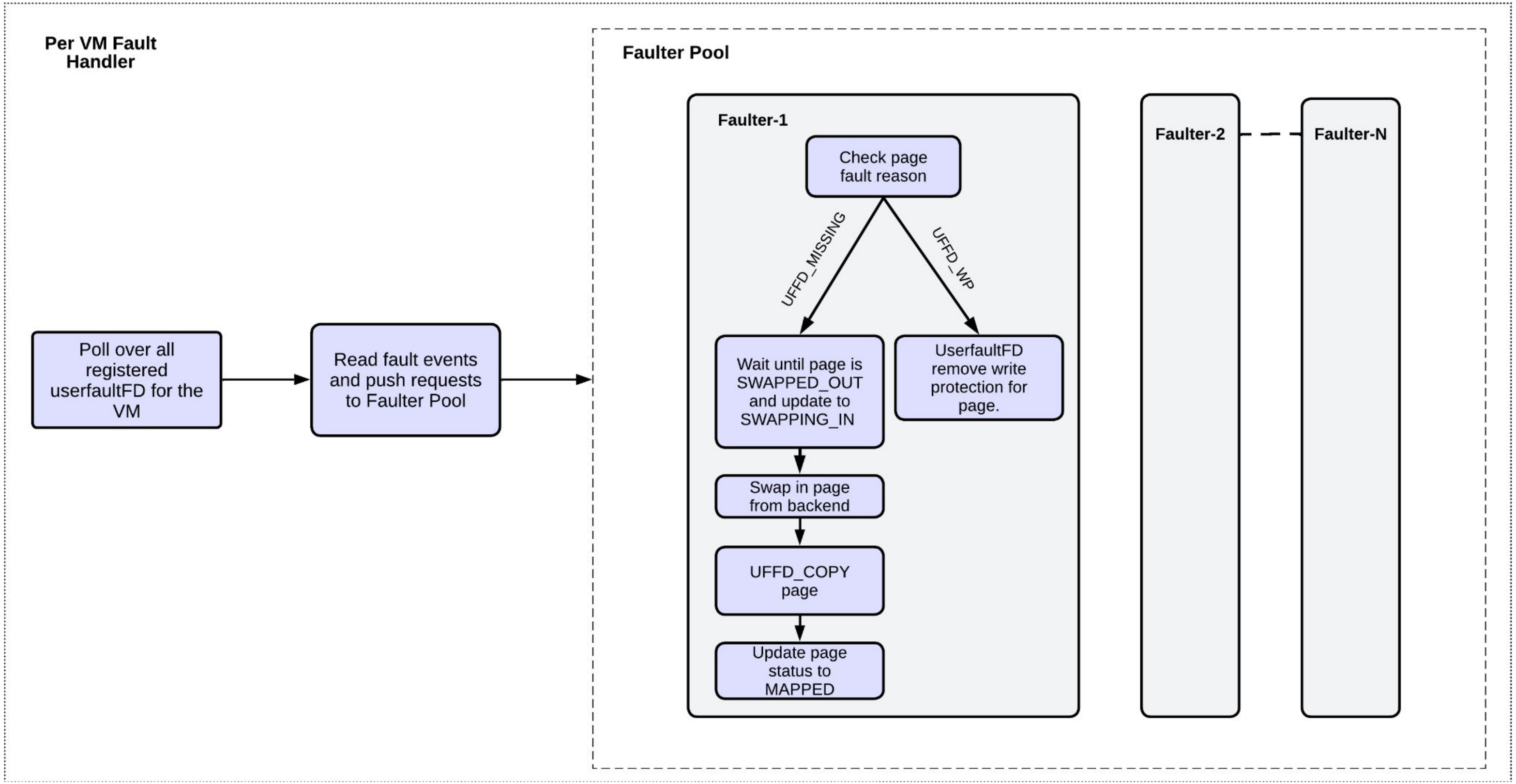
Swap Out



Key details

- Page status map
- Random page selection
- Static memory limit
- Swap out group of pages

Swap In


Per VM Fault Handler
Poll over all registered userfaultFD for the VM
Read fault events and push requests to Faulter Pool
Faulter Pool
Faulter-1
Check page fault reason
UFFD_MISSING
UFFD_WP
Wait until page is SWAPPED_OUT and update to SWAPPING_IN
UserfaultFD remove write protection for page.
Swap in page from backend
UFFD_COPY page
Update page status to MAPPED
Faulter-2
Faulter-N

Live Migration Overview

- Sending pages which are swapped out requires costly page faults.
  - **Disturbs guest WSS and causes thrashing**
- Pages on remote swap target can be **directly mapped on destination**
- Page faults on destination side still possible
  - Page data received from source is always the latest
  - Page faults can **mapped with zero page**, avoiding costly swapins
  - Keep source and destination **swap state in sync** by sending frequent hints

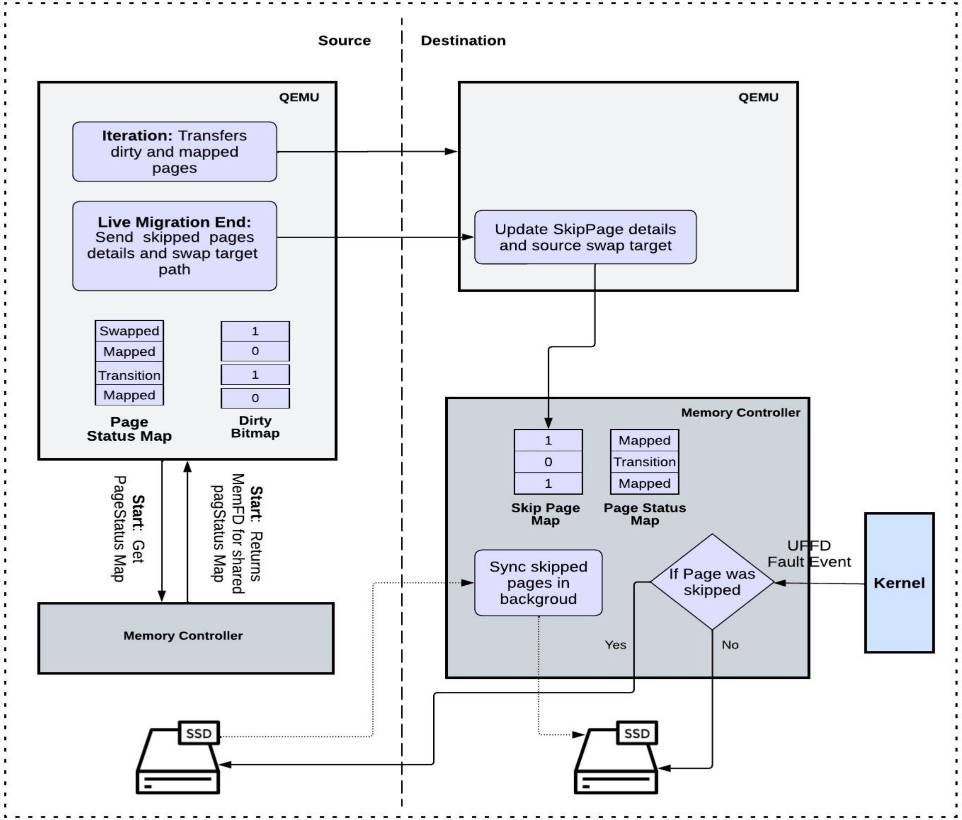
Live Migration Workflow

Steps on Source:

- Mem-controller shares **page status map** with QEMU
- QEMU **skips transferring** swapped out pages
- QEMU sends **skipped pages details** to destination in final stage of live migration

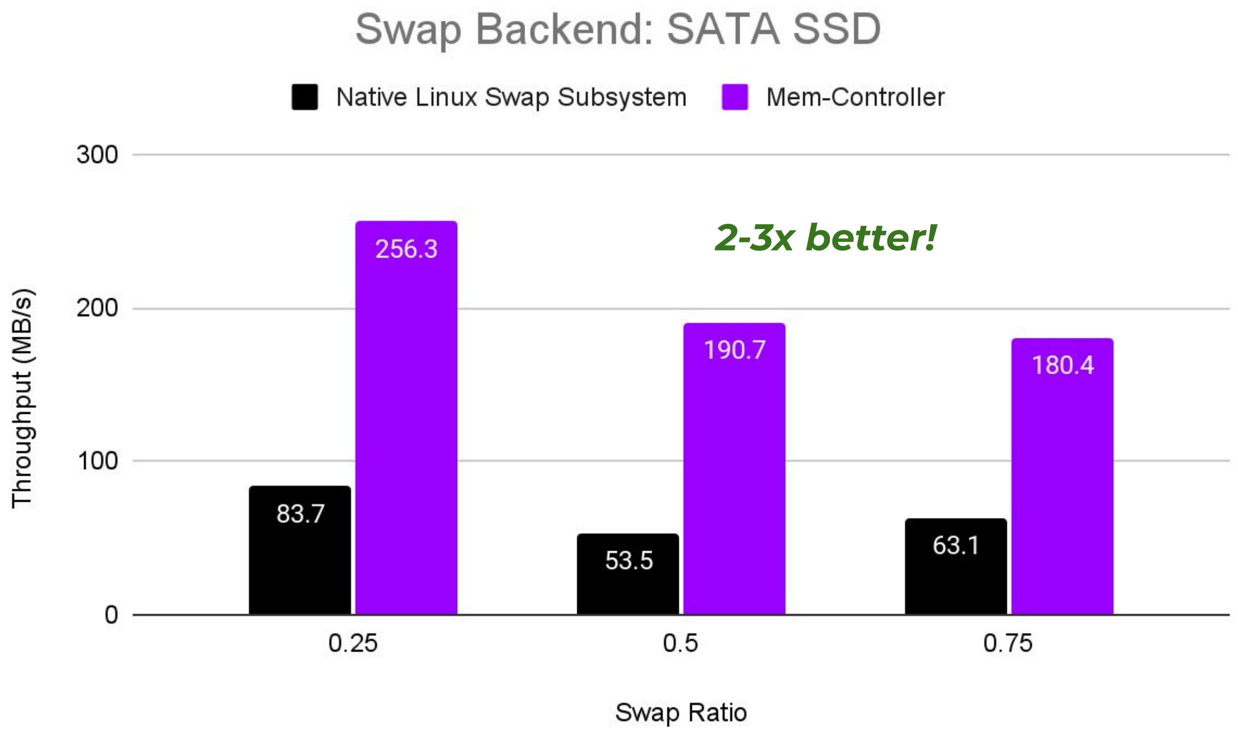
Steps on Destination:

- QEMU updates skipped pages details to mem-controller
- Skipped pages faults are **resolved from source target**
- Skipped pages **synced in background**



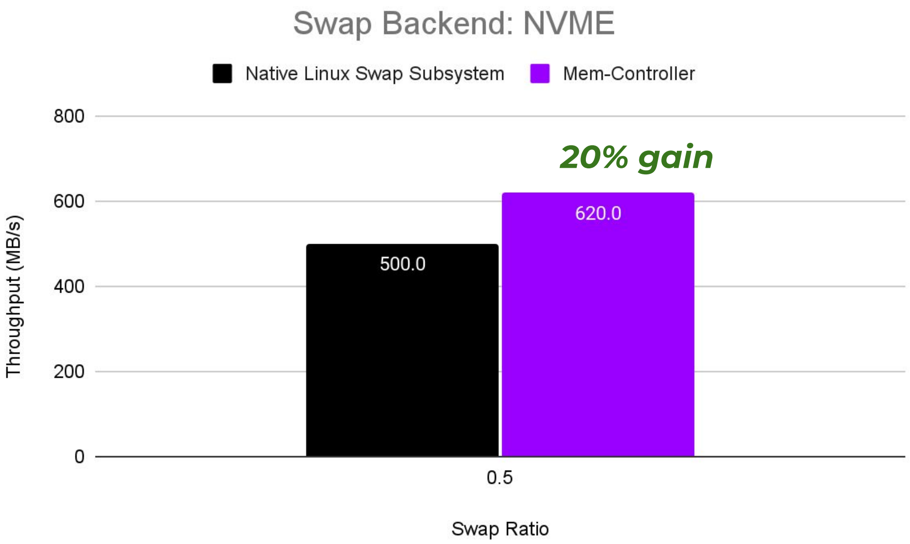
Results

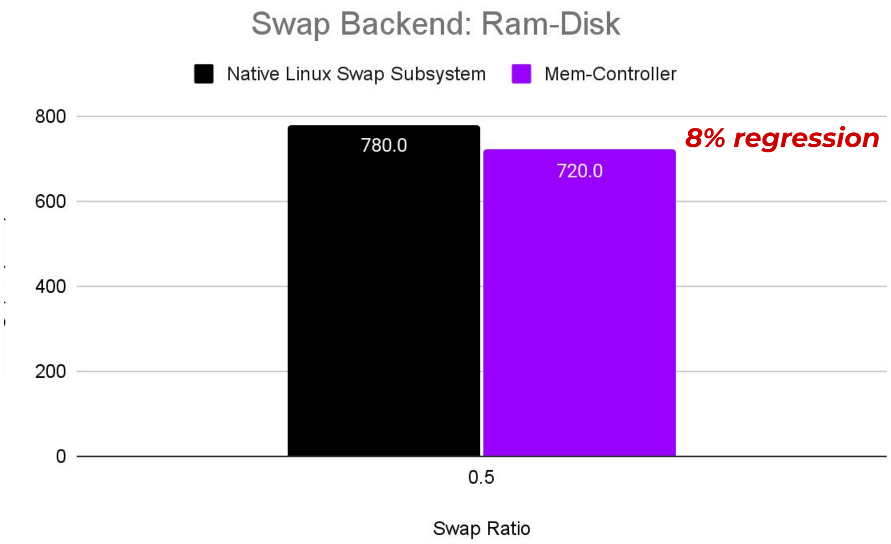
Throughput Comparison



VM Config - Memory: 12GB, 12vCPUs
Workload Config: Synthetic, 10GB WSS, Single threaded

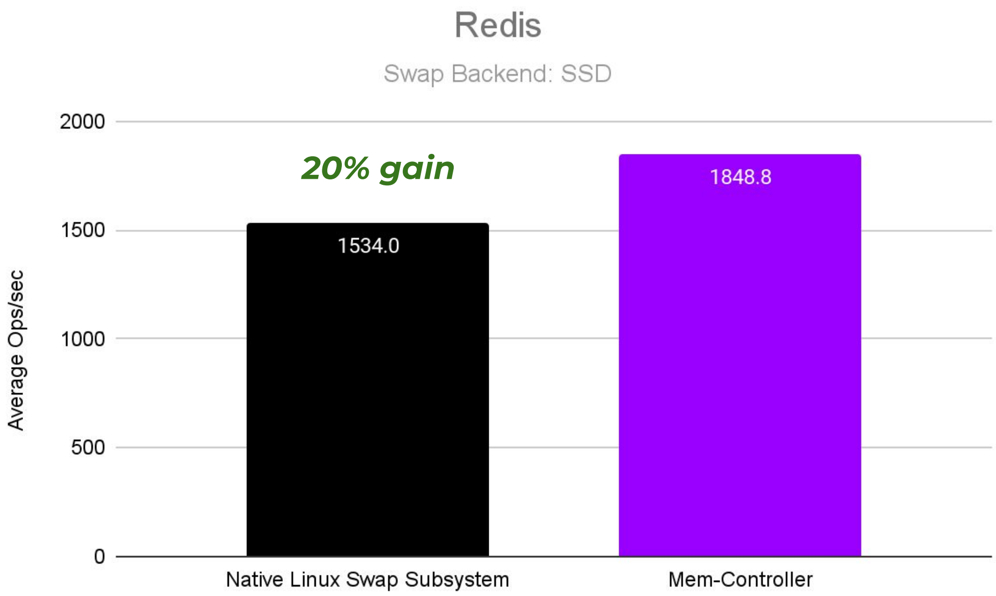
Throughput Comparison





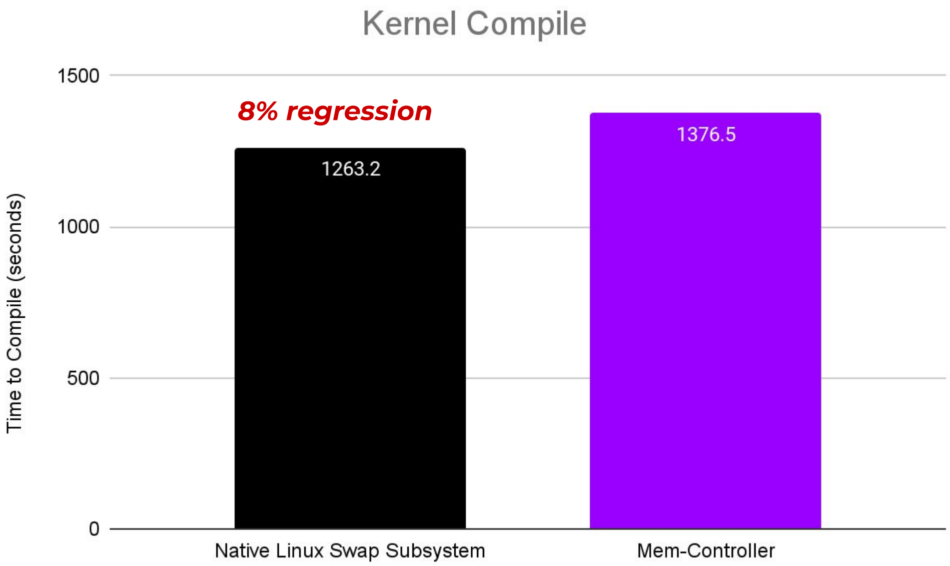
VM Config: Memory: 12GB, 12vCPUs
Workload Config: Synthetic, 10GB WSS, Single threaded

Real Workload: Redis



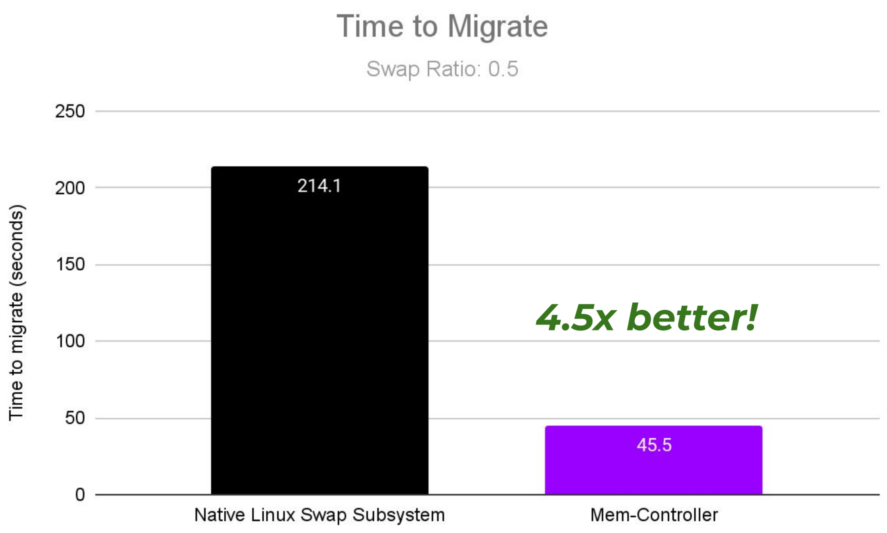
VM Config: Memory: 5GB, 4 vCPUs
Swap Ratio: 0.5
Workload WSS : ~2.5GB

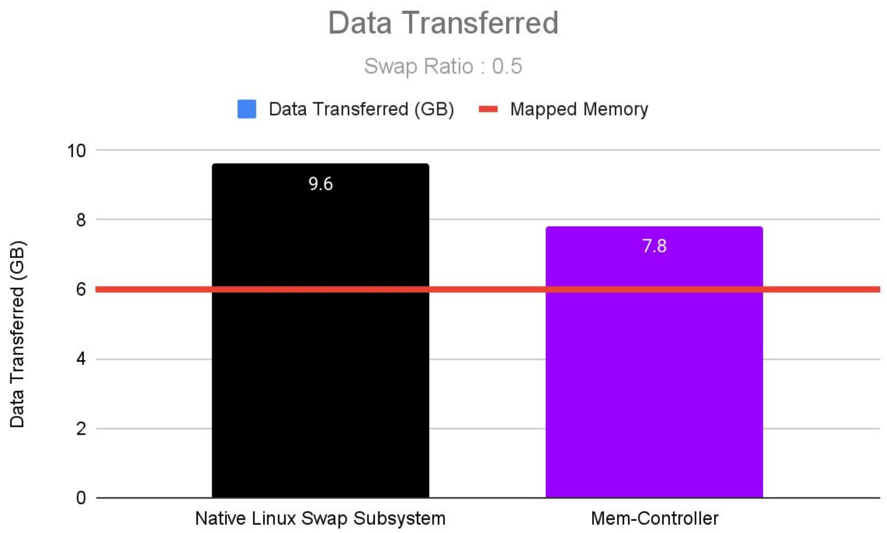
Real Workload: Kernel Compile



VM Config: Memory: 2GB, 4 vCPUs
Swap Ratio: 0.5
Swap Backend: SATA SSD

Live Migration Gains





VM Config: Memory: 12GB, 12vCPUs
Workload Config: 10GB WSS Single threaded
Swap Backend: SATA SSD

Conclusion

- A light weight **userfaultFD based memory-controller** approach performs really well and gives **significant improvement** in memory overcommitted VM's runtime performance.

- UserfaultFD based approach is performing well but it **bottlenecks for superfast swap backends**.
  - Will explore reducing userfaultFD bottlenecks as future work

- Control over swapped out pages, helps in **avoiding page thrashing** during live migrations, hence **faster live migrations**.

Future Work

NUTANIX

Reduce UserfaultFD Latencies

UserfaultFD based approach doesn’t scale for **super-fast swap devices**

- UserfaultFD operation cost increases with number of **shared memory address spaces**
    - Most userfaultFD operations are not completely synchronous, so can benefit from parallel swapper/faulter
    - Larger swap granularity (e.g. 16KB) significantly reduces average overheads
- Large latency due to **context switches**
    - Frequent user/kernel transitions
    - Investigate on new approaches to handle and acknowledge events
    - Consider using iorings mapped into both kernel and userspace

Reduce UserfaultFD Latencies

- Avoid extra **memory copies** by UFFD_COPY

- High latencies with **MADV_REMOVE** or fallocate **PUNCH_HOLE**
  - Most significant overhead
  - Exclusively locks memory address space, preventing parallel operations
  - Need further efforts to make it faster

Memory Balloon Hints

- Modifications on memory address not allowed by anything other than mem-controller.
  - Disturbs management and statistics managed by mem-controller.

- Ballooning is managed by QEMU, QEMU does following on balloon events.
  - **Inflate Balloon**: MADV_REMOVE or MADV_DONTNEED on pages.
  - **Deflate Balloon**: MADV_WILLNEED on pages.

- Balloon events need to be managed by mem-controller process.
  - QEMU processes virtio-balloon rings and **sends events to mem-controller**
  - or **shares virtio rings** with mem-controller for direct processing.

Thank You

NUTANIX