Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory

Chenxi Wang*, Yifan Qiao* (co-first author), Haoran Ma, Shi Liu, Yiyi Zhang, Wenguang Chen, Ravi Netravali, Miryung Kim, Guoqing Harry Xu
Memory Challenge in Datacenters

“Memory capacity wall”

Memory underutilization in datacenters
Remote Memory Systems

**Efficiency:** Fastswap [EuroSys’20], AIFM [OSDI’20]

**Reliability:** Hydra [Fast’22], Carbink [OSDI’22]

**Multi-Tenancy?**
Multi-Tenant Cloud on Remote Memory

- Local Memory
  - Snappy
  - XGBoost

- Kernel Swap System
  - Host Server
  - NIC

- Fast Network (e.g., RDMA, CXL)

- Remote Memory
  - NIC
  - Remote Server
Kernel Swap Performs Poorly in Shared Settings

Experiment with four real-world cloud applications.

State of the art: Fastswap [EuroSys’20]

- **snappy**: Google’s file compression service
- **Memcached**: In memory key-value cache
- **XGBoost**: CPU-based ML framework
- **Spark**: Data-processing framework

Network: Mellanox ConnectX-3 IB (40 Gbps, 2 µs)
Kernel Swap Performs Poorly in Shared Settings

Run each application alone with 25% of their working sets cached in local memory.

Normalized Slowdown

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|---|---|---|
| snappy|   |   |   |   |   |   |   |   |   |
| Memcached | | | | | | | | | |
| XGBoost | | | | | | | | | |
| Spark  | | | | | | | | | |

applications run alone
Kernel Swap Performs Poorly in Shared Settings

Co-run four real-world applications with 25% of their working sets cached in local memory.
Where Does The Overhead Come From?

User

Kernel

Local Server

App. 1

App. 2

App. 3

Kernel Swap System

Networking Layer

RDMA

Remote Memory (swap partition)

Shared remote memory datapath
Interference #1: Shared Swap Resources

- Lock contention when alloc./free
- Swap throughput drops by 56%

Diagram showing the interaction between local memory, swap cache, networking layer, RDMA, and remote memory (swap partition) with applications 1, 2, and 3.
Interference #2: Mixed Access Patterns

Prefetch contribution drops by 33%
Interference #3: RDMA Bandwidth Competition

Competition occurs:
- among applications
- between application and prefetcher

P99 round-trip latency increases by 2.1x
Canvas Design: Holistic Isolation

RDMA Networking Layer

Local Memory

App. 1
Prefetcher
Swap Cache
Networking Layer

App. 2
Prefetcher
Swap Cache

App. 3
Prefetcher
Swap Cache

Remote Memory (swap partition)

Swap Partition
Swap Partition
Swap Partition

Canvas Design: Holistic Isolation
Isolation Enabled Adaptive Optimizations

Single-thread Prefetcher

Multi-thread Per-thread prefetcher

Local

Swap Cache

Network

Swap Partition

Remote

Remote

Local

Swap Cache

Network

Swap Partition

Remote

Local

Swap Cache

Network

Swap Partition

Remote

Per-thread prefetcher
Isolation Enabled Adaptive Optimizations

Local

\[
\begin{array}{c}
\text{snappy} \\
\text{Prefetcher} \\
\text{Swap Cache} \\
\text{Network} \\
\text{Swap Partition} \\
\text{Remote}
\end{array}
\]

Remote

Low BW Usage

High BW Usage

Two-tier prefetcher

Two-dimensional RDMA scheduler

Remote

Swap Partition

Swap Cache

Network

Prefetcher

Local

\[
\begin{array}{c}
\text{Spark} \\
\text{Prefetcher} \\
\text{Swap Cache} \\
\text{Network} \\
\text{Swap Partition} \\
\text{Remote}
\end{array}
\]
Isolation Enabled Adaptive Optimizations

Local
- snappy
- Prefetcher
- Swap Cache
- Network
- Swap Partition
- Remote

Remote

Heavily Swap

Two-tier prefetcher

Two-dimensional RDMA scheduler

Adaptive entry allocator
Remote Memory Management

Local Memory

Physical Pages

P1  P2  P3  P4  P5  …

4KB

Remote Memory (Swap Partition)

Swap Entries

E1  E2  E3  E4  E5  …

4KB

in-use entry

free entry
Remote Memory Management: Swap Out

Local Memory

Physical Pages

P1 P2 P3 P4 P5 ...

Swap out

Remote Memory

E1 E2 E3 E4 E5 ...

Scan & Entry Allocation

allocated entry
in-use entry
free entry
Remote Memory Management: Swap In

Local Memory

Physical Pages

Remote Memory

Swap in

E3 P2 P3 P4 P5

E1 E2 E3 E4 E5

Entry Free

in-use entry

free entry
Efficient When Swap is Rare

Local Memory

Remote Memory

Works well

P1 P2 P3 P4 P5

E1 E2 E3 E4 E5

snappy
Inefficient When Swap Is Intensive

Local Memory

Physical Pages

E1 E2 E3 E4 E5

Repetitive entry allocation & free causes heavy lock contention!
Adaptive Entry Allocator

Local Memory

Physical Pages

Remote Memory (swap partition)

Swap Entries

P1 P2 P3 P4 P5

E1 E2 E3 E4 E5

Sufficient remote memory: *Trade-off space for efficiency*

Memorize ID of allocated entry to avoid repetitive allocation
Adaptive Entry Allocator: Swap In & Reserve

Local Memory

Physical Pages

Remote Memory

Swap in & Reserve the entry
Adaptive Entry Allocator: Lock-Free Swap Out

Local Memory

Physical Pages

Remote Memory

P1 P2 P3 P4 P5 ...

#3

Lock-free swap out

E1 E2 E3 E4 E5 ...

E1 E2 E3 E4 E5
Adaptive Entry Allocator: Intensive Swap

Local Memory

Physical Pages

Remote Memory

Almost lock-free swapping

P1 P2 P3 P4 P5
#3 #1 #4

E1 E2 E3 E4 E5

...
Adaptive Entry Allocator: Free Mappings

Local Memory

Physical Pages

Free reserved mappings

Remote Memory Pressure!

Remote memory pressure: *Trade-off time for space*
Evaluation

Evaluated on 6 real-world cloud applications with 11 workload combinations

State of the art: Linux cgroup on Fastswap [EuroSys’20]

- How does Canvas improve throughput for co-running applications?
- How does Canvas reduce performance variation?
Results: Improved Throughput

Co-run Snappy, Memcached, XGBoost with Spark, Cassandra, and Neo4j, respectively

Normalized Slowdown

|             | Fastswap | Canvas |
|-------------|----------|--------|
| 25% Local Memory | 3.96     | 1.13   |
| Normalized Slowdown | 3.5X     | 1.13   |

Normalized Slowdown

|             | Fastswap | Canvas |
|-------------|----------|--------|
| 50% Local Memory | 2.01     | 1.06   |
| Normalized Slowdown | 2.01     | 1.06   |
Results: Reduced Performance Variation

Fix Snappy, Memcached, and XGBoost and co-run them with another Java application

- 11 workload combinations in total
- Under 25% local memory
- On average 7.4x variation reduction
Conclusion

Canvas: holistic isolation + adaptive optimizations

- Isolation is necessary for real deployment of remote memory
- Isolation improves performance and QoS for shared remote memory
- Isolation enables adaptive optimizations for further performance boosts

- Canvas offers co-running applications 6.2x speedup and 7.4x variation improvement

https://github.com/uclasystem/canvas
Thank You!