Hermit: Low-Latency, High-Throughput, and Transparent Remote Memory via Feedback-Directed Asynchrony

Yifan Qiao, Chenxi Wang, Zhenyuan Ruan, Adam Belay, Qingda Lu, Yiying Zhang, Miryung Kim, Guoqing Harry Xu
Datacenter Must Balance Latency With Utilization

Latency-Critical Applications

Remote Memory Systems
Remote Memory Systems

- **Host Server**
  - **CPU**
  - **Local Memory**
  - **Kernel Swap System**

- **Fast Network** (e.g., RDMA, CXL)
  - **Remote Memory Pool**

- **80 ns**

Network: Mellanox ConnectX-5 Ex (100 GbE, 7 µs for a 4 KB page)
Kernel Swap Can Incur High Tail Latency

**1914x** higher tail latency!

Latency (µs)

Network: Mellanox ConnectX-5 Ex (100 GbE, 7 µs for a 4 KB page)
Where Does the Latency Come From?

Application

- Page Reclamation
- Deduplicate Swap-ins
- Metadata Bookkeeping
- Fetch & Prefetch
- Remote Memory

User Kernel

0.2 μs

remote access Fault
Where Does the Latency Come From?

Application

User
Kernel

Page Reclamtion
(Occasionally)

Deduplicate Swap-ins

Metadata Bookkeeping

Fetch & Prefetch

Remote Memory

remote access

1180 μs (32 pages)

Network: Mellanox ConnectX-5 Ex (100 GbE)
Where Does the Latency Come From?

Application

- Page Reclamation: 1180 μs
- Deduplicate Swap-ins: 2.8 μs
- Remote Memory
- Metadata Bookkeeping
- Fetch & Prefetch

Network: Mellanox ConnectX-5 Ex (100 GbE)
Where Does the Latency Come From?

- Application
  - Page Reclamation: 1180 μs
  - Deduplicate Swap-ins: 2.8 μs
  - Metadata Bookkeeping: remote access, 1.3 μs
  - Fetch & Prefetch
  - Remote Memory

Network: Mellanox ConnectX-5 Ex (100 GbE)
Where Does the Latency Come From?

- Page Reclamation: 1180 μs
- Deduplicate Swap-ins: 2.8 μs
- Metadata Bookkeeping: 1.3 μs
- Fetch & Prefetch: 0.2 μs
- Remote Memory: remote access

Network: Mellanox ConnectX-5 Ex (100 GbE)
Where Does the Latency Come From?

![Diagram showing latency components]

- **Page Reclamation**: 1180 μs
- **Deduplicate Swap-ins**: 2.8 μs
- **Metadata Bookkeeping**: 1.3 μs
- **Fetch & Prefetch**: 0.2 μs
- **Remote Memory (remote access)**: 9.1 μs

Network: Mellanox ConnectX-5 Ex (100 GbE)
Where Does the Latency Come From?

- **Application**
  - Page Reclamation (Occasionally) 1180 μs
  - Deduplicate Swap-ins 2.8 μs
  - Metadata Bookkeeping 1.3 μs
  - Fetch & Prefetch 0.2 μs
  - Remote Memory 9.1 μs

- **Kernel Swap (>32%)**
  - Remote access

- **Hardware**
  - Network: Mellanox ConnectX-5 Ex (100 GbE)
How To Reduce Latency?

Application

User

Kernel

Page Reclamation

Deduplicate Swap-ins

Metadata Bookkeeping

Fetch & Prefetch

Remote Memory
Kernel Bypassing Is Not a Panacea

Can we eliminate performance bottlenecks in the kernel directly?

- Remote Memory
- User Managed
- Page Reclamation
- Deduplicate Swap-ins
- Metadata Bookkeeping
- Prefetch

Lose Transparency
Lose kernel protection & isolation
Can Asynchrony Reduce Latency?

No data dependency! Make it asynchronous!

Network: Mellanox ConnectX-5 Ex (100 GbE)
Naive Asynchrony Is Not Enough

- Linux: kswapd
- Fastswap [EuroSys’20]: dedicated core

User

Kernel

Application

- Deduplicate Swap-ins
- Metadata Bookkeeping
- Fetch & Prefetch
- Remote Memory

Reclaim Thread

Page Reclamation
Naive Asynchrony Is Not Enough

User

Kernel

Application

Page Reclamation

Deduplicate Swap-ins

Metadata Bookkeeping

Fetch & Prefetch

Remote Memory

Statically Controlled

Reclaim Thread

Page Reclamation
Must Have Controlled Asynchrony

- **When** to start reclamation?
- **How** many cores for reclamation?
Challenge #1: When To Start Reclamation

When to start reclamation?

Application

CPU

Core 1  Core 2  Core 3  Core 4

Memory Usage
Challenge #1: When To Start Reclamation

Reclaim too early:
- Memory underutilization

Start reclamation
Challenge #1: When To Start Reclamation

- Reclaim too early:
  - Memory underutilization
- Reclaim too late:
  - Memory exhaustion
Challenge #2: How Many Cores For Reclamation

Too few cores:
- Memory exhaustion

Taken by reclaim threads

Application

CPU
- Core 1
- Core 2
- Core 3
- Core 4

Memory
Challenge #2: How Many Cores For Reclamation

Too few cores:
- Memory exhaustion

Too many cores:
- Interfere user threads
Hermit Design: Feedback-Directed Asynchrony

Application

- Deduplicate Swap-ins
- Metadata Bookkeeping
- Fetch & Prefetch
- Remote Memory

Feedback

Reclaim Scheduler

User
Kernel
Hermit’s Adaptive Reclaim Scheduling

Local Memory Usage

# Cores for Reclamation

Max # Cores

0 1

Low-Water Mark
High-Water Mark

Limit

Local Memory Usage
Hermit’s Adaptive Reclaim Scheduling

- **When** to start reclamation?

![Diagram showing the relationship between cores for reclamation, local memory usage, and reclamation thresholds.](image)

- Local Memory Usage
- # Cores for Reclamation
- Max # Cores
- Limit
- Low-Water Mark
- High-Water Mark
Hermit’s Adaptive Reclaim Scheduling

- **When** to start reclamation?

![Diagram showing the relationship between cores for reclamation and local memory usage. The diagram includes a graph with axes for maximum number of cores and local memory usage, with markers indicating low-water and high-water marks.]

- # Cores for Reclamation
- Local Memory Usage
- Max # Cores
- Limit

- Low-Water Mark
- High-Water Mark
Hermit’s Adaptive Reclaim Scheduling

How many cores for reclamation?

- Local Memory Usage
- # Cores for Reclamation
- Max # Cores
- Limit
- Low-Water Mark
- High-Water Mark
Hermit’s Adaptive Reclaim Scheduling

# Cores for Reclamation

Dynamically Adjust

Local Memory Usage
Hermit’s Adaptive Reclaim Scheduling

# Cores for Reclamation

- Max # Cores
- Low-Water Mark
- High-Water Mark
- Limit

\[ \propto \text{Swap in throughput} \]
Hermit’s Adaptive Reclaim Scheduling

*Page Turnaround (PT)*: how long a swapped-out page remains untouched

![Graph](image)

- **# Cores for Reclamation**
- **Max # Cores**
- **Low-Water Mark**
- **High-Water Mark**
- **Limit**

**When PT remains high**
Hermit Achieves Low Latency

- Deduplicate Swap-ins
- Metadata Bookkeeping
- Fetch & Prefetch
- Remote Memory

Overlapped with I/O

Reclaim Scheduler

User
Kernel

Application

Remote Memory

Deduplicate Swap-ins

Overlapped with I/O

Thread

Reclaim Threads
How To Improve Throughput?

Latency-critical path

- Application
  - Deduplicate Swap-ins
  - Metadata Bookkeeping
  - Fetch & Prefetch
  - Remote Memory

Non-critical path

- Reclaim Thread
  - Page Reclamation
  - Reclaim Scheduler

User
Kernel
Aggressive Batching For Async. Reclamation

Batched operations for a group of pages:
- TLB Shootdown
- Page I/O Write
- cgroup Accounting
- etc.

2.9x more CPU efficient
Evaluation

Evaluated 6 real-world cloud applications with varying local memory ratios

- **Latency-Critical**: Memcached, SocialNet, Gdnsd
- **Batch-Processing**: Spark, XGBoost, Cassandra

State of the art: Fastswap [EuroSys’20]

- Offload page reclamation to a single dedicated core

- How does Hermit maintain low end-to-end tail latency?
- How does Hermit improve application throughput?
Low Tail Latency

Memcached with Facebook USR workload.
Low Tail Latency

Memcached with Facebook USR workload.

![Graph showing 99% Latency (μs) vs. Offered load (Mops) with different load profiles. The y-axis represents 99% Latency in microseconds, ranging from 0 to 500. The x-axis represents Offered load in Mops, ranging from 0 to 4. The graph includes lines labeled 'All local.'
Low Tail Latency

Memcached with Facebook USR workload.

- Cache 70% of working set in local memory (i.e., 30% in remote memory).

4x throughput drop!

99% Latency (μs)

Offered load (Mops)

All local
Fastswap
(state of the art)
Low Tail Latency

Memcached with Facebook USR workload.

- Cache 70% of working set in local memory (i.e., 30% in remote memory).

`99% Latency (μs)`

- All local
- Fastswap
- Hermit

99.7% lower

Offered load (Mops)
High Throughput

Memcached with Facebook USR workload.

- Cache 70% of working set in local memory (i.e., 30% in remote memory).
Memcached with Facebook USR workload.

- Cache 70% of working set in local memory (i.e., 30% in remote memory).
High Throughput For Batch Applications

Hermit offers 1.24x higher throughput (up to 1.87x)
Low latency, high throughput, and transparency can be achieved simultaneously!

- Asynchrony reduces latency and improves throughput
- Feedback loop is critical to the effect of asynchrony
- Design can be generalized to other kernel components such as page migration for CXL-attached memory

- Hermit offers up to 99.7% lower latency and 1.24x higher throughput without changing a single line of user code

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