MFence: Defending Against Memory Access Interference in a Disaggregated Cloud Memory Platform

Jinhoon Lee, Yeonwoo Jung, Suyeon Lee, Safdar Jamil, Sungyong Park, Kwangwon Koh, Hongyeon Kim, Kangho Kim, Youngjae Kim

**Presenter: Yeonwoo Jeong**

Department of Computer Science and Engineering
Sogang University, Seoul
South Korea
Agenda

• Introduction & Background
• Motivation
• Design
• Evaluation
Memory-Intensive Applications

- Demand for processing memory-intensive applications is high
  ✓ Machine learning, graph processing, KV Store
- Big data applications require a high memory and cause memory shortages
Memory-Intensive Applications

• Limited memory capacity per machine
  ✓ OOM (Out of memory), application failures..

• Traditional solution for big memory
  ✓ Disk swap
Memory-Intensive Applications

• Limited memory capacity per machine
  ✓ OOM (Out of memory), application failures..

• Traditional solution for big memory
  ✓ Disk swap

Higher access latency in disk swap!
Disaggregated Memory Platform

- Memory of a remote server as an extension of limited local memory
  - One machine **borrows** memory from remote machine with high-speed network
- VM-based remote memory solution
  - **Client machine** (borrows the memory)
  - **Donor machine** (provides the memory)
Disaggregated Memory Platform

• Memory of a remote server as an extension of limited local memory
  ✓ One machine **borrows** memory from remote machine with high-speed network
• VM-based remote memory solution
  ✓ **Client machine** (borrows the memory)
  ✓ **Donor machine** (provides the memory)
Disaggregated Memory Platform

• Memory of a remote server as an extension of limited local memory
  ✓ One machine **borrows** memory from remote machine with high-speed network

• VM-based remote memory solution
  ✓ **Client machine** (borrows the memory)
  ✓ **Donor machine** (provides the memory)

**Memory Access Time**
- From local: 10~256 ns
- From remote: **2.8 us**
Disaggregated Memory Platform

- DCM[TC’19]* uses **local memory** as an **inclusive cache** and maximizes the **hit rate** to reduce fetching from remote memory
- Using a VM, donor’s memory can be perceived as its own memory space

Agenda

• Introduction & Background

• Motivation

• Design

• Evaluation
Cache Contention on DCM

• Processes in VM use the shared cache concurrently, a memory race condition occurs

Single run
Cache Contention on DCM

• Processes in VM use the shared cache concurrently, a memory race condition occurs

![Diagram showing cache contention in single and shared runs]
Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

  ✓ Workload: 1 linear/random memory-access

  ✓ All process memory footprint: 8GB

  ✓ Local memory capacity in VM: 4GB

---

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

\[\text{Cache Occupancy Ratio (\%)}\]

- Linear
- Random

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Cache Contention on DCM

- Evaluation with micro memory benchmark (PmBench*) on DCM

Recently, KV-Store became a key component in the infrastructure of large-scale data centers.

- A set of data is indexed to Key field.


Can we ensure the fairness per process?

Different memory usage per process

sharing local memory affects the performance
Opportunity with Cache Partitioning

• **Cache partitioning** can overcome the unfair utilization of local cache between memory-greedy processes

• **Challenges**
  - ✓ Host kernel cannot directly know page information of the process running on guest OS/VM
  - ✓ Overhead of page identification per process between host/guest area is not trivial
Agenda

• Introduction & Background

• Motivation

• **Design**

• Evaluation
MFence

User space

VM (Guest OS)

Kernel space

Cache Management Module

Page Owner Identification Module

Cache Control Module

Memory Donor Node

App1

App2
M Fence

**Cache Management Module**
- MFence manages a per-process LRU queue, providing local memory partitions of varying sizes
- The split queue (cache) has a unique ID[group ID (GID)] is assigned

![Diagram of MFence](image-url)
MFence

• **Cache Management Module**
  - MFence manages a per-process LRU queue, providing local memory partitions of varying sizes
  - The split queue (cache) has a unique ID [group ID (GID)] is assigned

**Diagram:**
- User space
- Kernel space
- Memory
- App1
- App2
- DCM
- Global LRU queue
- Queue 1
- Mgmt module
- C1
- C2
MFence

• **Cache Management Module**
  ✓ MFence manages a per-process LRU queue, providing local memory partitions of varying sizes
  ✓ The split queue (cache) has a unique ID[group ID (GID)] is assigned

![Diagram of MFence architecture]
MFence

• **Cache Management Module**
  ✓ MFence manages a per-process LRU queue, providing local memory partitions of varying sizes
  ✓ The split queue (cache) has a unique ID[group ID (GID)] is assigned

[Diagram of DCM (Data Center Memory) showing Global LRU queue, Queue 1, Queue 2, and Mgmt module]
MFence

• **Cache Control Module**
  ✓ MFence allows the user to manage the cache of the process
  ✓ Cache group allocation, cache area creation, cache area release via user API
MFence

• **Cache Control Module**
  ✓ MFence allows the user to manage the cache of the process
  ✓ Cache group allocation, cache area creation, cache area release via user API

![Diagram of MFence architecture with Cache Control Module (DCM), Cache area, Queue 1 and Queue 2, and Management module (Mgmt)]
MFence

- **Cache Control Module**
  - MFence allows the user to manage the cache of the process
  - Cache group allocation, cache area creation, cache area release via user API

Add_Process_CRegion(PID,GID)
MFence

• **Cache Control Module**
  ✓ MFence allows the user to manage the cache of the process
  ✓ Cache group allocation, cache area creation, cache area release via user API

  ```python
  Destroy_CRegion(GID)
  ```

  ![Diagram showing cache control module with App1 and App2, C1 and C2, DCM, and control module with queues and management module]

  **User space**
  **Kernel space**
  **Memory**
**MFence**

- **Page Owner Identification Module**
  - Each LRU queue caches only pages that belong to the process
  - Determine the queue into which to insert a page fetched from a remote server
MFence

• **Page Owner Identification Module**
  - Each LRU queue caches only pages that belong to the process
  - Determine the queue into which to insert a page fetched from a remote server

✓ **Kernel cannot be aware of upper layer’s processes**
✓ **Kernel cannot confirm page's owner and perform memory separation between processes**

1) Check vCPU
2) Fetch faulted page
3) Check owner of page
4) Add to queue

Client Node

Memory Donor Node

Owner ID module

owner of page

4) Add to queue
MFence

- **Page Owner Identification Module**
  - Hypercall method
  - gCR3 method
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

Guest User
Guest Kernel

App1

Host Kernel

KVM

Queue 1

Cache is initially partitioned for the process

Comm Buffer

Transfer *comm buffer’s address* with Hypercall to host kernel

Memory Donor Node

Swap Space
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence architecture]

- Guest User
- Guest Kernel
- App1
- Return to App
- Page Hit
- Page fault
- Event occur!
- Comm Buffer
- KVM
- VM-exit & Context switch to host kernel
- Host Kernel
- Transfer **comm buffer's address** with Hypercall to host kernel
- Memory Donor
- Node
- Swap Space
**MFence**

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence](image)

- **Guest User**
  - **Guest Kernel**
    - **Return to App**
    - **Page Hit**
    - **Comm Buffer**
    - **Page fault Event occur!**
- **Host Kernel**
  - **VM-exit & Context switch to host kernel**
  - **KVM**
    - **Host kernel reads a faulted GPA(Guest Page Address) from vCPU in KVM**

![Memory Donor Node](image)
MFence

- **Page Owner Identification Module** *(Hypercall)*
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence](image-url)
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of page fault handling]

- Host kernel **delegates** page fault handling to DCM module
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence architecture](image-url)
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer
MFence

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence](image)
**MFence**

- **Page Owner Identification Module (Hypercall)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

- Overhead involved in Hypercall through comm buffer is too heavy
- How to minimize the overhead of page owner identification?

---

**Diagram**

1. **App1**
2. **Host Kernel**
   - Terminate page fault handling
3. **KVM**
4. **Queue 1**
   - Insert the page fetched from remote memory **into the queue** corresponding PID
5. **Memory Donor Node**
   - Swap Space
MFence

- **Page Owner Identification Module (gCR3)**
  - It uses PGD (Page Global Directory), the address of the page table, as a PID
  - PGD can be obtained by reading the vCPU's guest CR3 register value

![Diagram of MFence](image.png)

- **Key** field
- **Guest User** → **Guest Kernel**
- **Host Kernel**
- **KVM** → **Queue 1** → **Cache** → **Memory Donor Node**

*Cache is initially partitioned for the process*
**MFence**

- **Page Owner Identification Module (gCR3)**
  - It uses PGD (Page Global Directory), the address of the page table, as a PID
  - PGD can be obtained by reading the vCPU's guest CR3 register value

![Diagram of Page Fault Handling](image)
MFence

• **Page Owner Identification Module (gCR3)**
  • It communicates with the guest kernel to find PID of a fetched page
  • Requires one Hypercall handshake to establish communication buffer

[Diagram showing the flow of data between Guest User, Guest Kernel, Return to App, Page Hit, N, Y, KVM, Host Kernel, and Memory Donor Node.]

Host kernel reads **gCR3 (Guest Control Register 3)**
From vCPU in KVM to get
**PGD (Page Global Directory)**
MFence

• **Page Owner Identification Module (gCR3)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of page fault handling process]

Host kernel delegates page fault handling to DCM module.
MFence

- **Page Owner Identification Module (gCR3)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer

![Diagram of MFence](image-url)
MFence

- **Page Owner Identification Module (gCR3)**
  - It communicates with the guest kernel to find PID of a fetched page
  - Requires one Hypercall handshake to establish communication buffer
Agenda

• Introduction & Background

• Motivation

• Design

• Evaluation
Evaluation

• Evaluation Setup

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel® Xeon Gold 6330, 2.00 GHz 28core * 2</td>
</tr>
<tr>
<td>Memory</td>
<td>16GB (DDR4, 3200MHz) * 8</td>
</tr>
<tr>
<td>Network</td>
<td>Mellanox ConnectX-5 100Gb/s EDR HCA</td>
</tr>
<tr>
<td>SSD</td>
<td>Intel SSD 750 (Read: 2.2 GB/s, Write: 900 MB/s)</td>
</tr>
<tr>
<td>OS</td>
<td>Linux kernel-4.18.0-240.10.1.el8</td>
</tr>
</tbody>
</table>

• Comparisons

✓ DCM(A): DCM where the workload is running alone
✓ DCM(S): DCM where the workload is executed in combination with other workload, which can cause interference between workload
✓ MFence(gCR3): DCM with cache partitioning adopting gCR3
✓ MFence(Hypercall): DCM with cache partitioning adopting the Hypercall
Evaluation

• Workloads (Bigdata kernel applications)
  ✓ Grep
  ✓ AG(Aggregation)
  ✓ GAG(Group By Aggregation)

• Evaluation Setting
  ✓ Memory footprint 8GB for each workload
  ✓ Local cache size is half the sum of the workload’s footprints for each comparison
Evaluation

• MFence performance of cache partitioning with workload combination
• MFence performance of cache partitioning with workload combination
Evaluation

• MFence performance of cache partitioning with workload combination

![Graph showing cache space (GB) vs AG and Grep latency (μs) with DCM(A), DCM(S), MFence (gCR3), and MFence (Hypercall) for both alone and shared scenarios.]
Evaluation

• MFence performance of cache partitioning with workload combination

![Graphs showing cache space and AG latency](image-url)
• MFence performance of cache partitioning with workload combination
Conclusion

• Memory races between processes on disaggregated memory platform causes memory imbalance and ultimately causes a difference in performance

• We proposed MFence, a mechanism to prevent memory race on a VM-based disaggregated memory platform (DCM)

• MFence devised a lightweight identification module (gCR3) to get the page information of the process running on guest OS from Host OS

• The experiment showed that there was no memory race when each workload combination provided half the memory of the local cache, and the corresponding change in performance was measured
Questions?

Yeonwoo Jeong
akssus12@sogang.ac.kr
DISCOS Lab, Sogang University