GateKeeper: Transparent Placement of Big Data Objects in Hybrid Managed Heaps

Iacovos G. Kolokasis
kolokasis@ics.forth.gr

Shoaib Akram
shoaib.akram@anu.edu.au

Foivos Zakkak
fzakkak@redhat.com

Polyvios Pratikakis
polyvios@ics.forth.gr

Angelos Bilas
bilas@ics.forth.gr
Analytics frameworks need large managed heaps

- Analytics frameworks use managed runtimes

- To process **large amounts of data** they need **large heaps**

- DRAM in a single server **scales slower** than data growth!

- Fast storage devices are desirable for processing
  - Provide higher capacity than DRAM

(Source: Micron’s Perspective on Impact of CXL on DRAM Bit Growth Rate Report)
Common practice: Move objects over fast storage devices

- Analytics frameworks offload objects on fast storage devices (off DRAM)
  - Transform objects to byte stream
  - High serialization/deserialization (S/D) overhead
Common practice: Move objects over fast storage devices

- Analytics frameworks offload objects on fast storage devices (off DRAM)
  - Transform objects to byte stream
  - High serialization/deserialization (S/D) overhead

- Recent work, extend managed heaps beyond DRAM (hybrid heaps)
  - Direct access to objects → No S/D
  - Two managed heaps → No GC scans over the device
Common practice: Move objects over fast storage devices

- Analytics frameworks offload objects on fast storage devices (off DRAM)
  - Transform objects to byte stream
  - High serialization/deserialization (S/D) overhead

- Recent work, extend managed heaps beyond DRAM (hybrid heaps)
  - Direct access to objects → No S/D
  - No GC scans over the storage device

- **Challenge:** Find objects for moving to the device
  - Cope with slow device accesses
Existing object selection approaches

Application modification

Application agnostic
Existing object selection approaches

Application modification

Programming models
- Provide application specific knowledge
- Significant effort for application writing

Application agnostic

Code instrumentation via JIT compiler
- Extra instructions before each load/store operation
- Significant runtime overhead

Page faults
- Protect/unprotect pages in the virtual address space
- Signal handling and page faults overheads
Transparent placement of big data objects in hybrid heaps

- Decide which objects to move from H1 to H2
  - **Avoid code instrumentation** and **page fault** overheads

- Leverage storage capacity to reclaim objects lazily
  - **Reclaim** dead objects **without GC scans on H2**

- Fix wrong decisions (fallback mechanism)
  - **Identify** objects that **increase I/O traffic**
  - **Transfer** objects from H2 to H1 **without scanning H2**
Decide which objects to move from H1 to H2

- Goal: **Avoid** code instrumentation and page fault overhead
Decide which objects to move from H1 to H2

- Goal: **Avoid** code instrumentation and page fault overhead

- Fast heap is a generational heap
  - Young generation for newly created objects
  - Old generation for mature objects
Decide which objects to move from H1 to H2

- **Goal:** Avoid code instrumentation and page fault overhead

- **Fast heap is a generational heap**
  - Young generation for newly created objects
  - Old generation for mature objects

- **We identify during GC long-lived objects**
  - Increase the age of each object (epochs)

- **High memory pressure in H1**
  - Move objects from H1 to H2
  - Transfer objects with earliest epoch
Decide which objects to move from H1 to H2

- Goal: **Avoid** code instrumentation and page fault overhead

- Fast heap is a generational heap
  - Young generation for newly created objects
  - Old generation for mature objects

- We identify during GC long-lived objects
  - Increase the age of each object (epochs)

- High memory pressure in H1
  - Move objects from H1 to H2
  - Transfer objects with earliest epoch
Decide which objects to move from H1 to H2

Goal: **Avoid** code instrumentation and page fault overhead

- Fast heap is a generational heap
  - Young generation for newly created objects
  - Old generation for mature objects
- We identify during GC long-lived objects
  - Increase the age of each object (epochs)
- High memory pressure in H1
  - Move objects from H1 to H2
  - Transfer objects with earliest epoch
Leverage storage capacity to free objects lazily

Goal: **Reclaim** dead objects **without GC scans**

- GateKeeper organize H2 in fixed-sized regions
  - Objects from same root in the same region
  - Reclaim whole regions (**bulk free**)  

- Per region DRAM metadata (**no object access**)
  - Live bit $\rightarrow$ region liveness

- GC identifies H2 live regions
  - Free regions by zeroing regions metadata

---

**JVM**

<table>
<thead>
<tr>
<th>Fast Heap (H1)</th>
<th>Slow Heap (H2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Region 0</td>
</tr>
<tr>
<td></td>
<td>Region 1</td>
</tr>
</tbody>
</table>

**JVM metadata (DRAM)**

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Live</strong></td>
<td><strong>Live</strong></td>
</tr>
</tbody>
</table>
Goal: Identify objects that increase I/O traffic

- Portion of DRAM is a cache for H2 to reduce slow accesses
  - Require cache locality → workloads behavior changing

- We use a kernel module to track H2 active pages
  - Maintain metadata per region
  - Track dirty pages

- GateKeeper scans H2 page cache on every minor GC
  - Mutator threads are stopped
  - No synchronization interference with GC threads
Fix wrong decision placement

- **Goal:** *Transfer* objects from H2 to H1 *without scanning H2*
  - Transfers from H2 to H1 needs objects references update
    - Requires scans to H2 $\rightarrow$ high I/O traffic
  - Transfer primitive arrays and leaf objects to H2
    - Alleviate references between H2 objects
    - Only forward references (H1 to H2) exists
  - Moving primitive objects from H2 to H1 require only forward references update
    - GC marking phase: finds forward references

EuroDW ’24
Key Takeaway

- Data growth is higher than DRAM capacity scaling
- Analytics frameworks require large managed heaps to process very big datasets
- Fast storage devices (e.g., NVMe SSDs) provide higher capacity than DRAM
- Extend managed heaps over NVMe SSD to cope with data growth
- GateKeeper: Decide transparently what object to move from the fast to the slow tier
  - With low runtime overhead
  - Transfer objects from the slow to the fast tier efficiently
GateKeeper: Transparent Placement of Big Data Objects in Hybrid Managed Heaps

We thankfully acknowledge the support of the European Commision projects AERO (GA No 10048318)
Iacovos G. Kolokasis is supported by the Meta Research PhD Fellowship (2022 – 2024)

kolokasis@ics.forth.gr