Mako: A Low-Pause, High-Throughput Evacuating Collector for Memory-Disaggregated Datacenters

Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, Guoqing Harry Xu
Memory Capacity Bottleneck in Datacenters

Growing imbalance between processor computation and memory capacity

Memory underutilization in datacenters
Memory Disaggregation

- CPU Server
- Small Local Memory

**Network**
e.g. RDMA over InfiniBand
Java Heap Structure

Memory Servers
Java heap

CPU Server
Java heap

RDMA over InfiniBand

Virtual Address

Swap System

Physical Memory

Addresses aligned

Local RAM (cache)

Virtual Address

Page-level Data Communication

RDMA messages

Page faults

60 ns

~ 10 μs

Java Process

Memory Server #1

Memory Server #2
Concurrent GC

STW Mark Start
STW Mark End
STW Evac End

Concurrent Marking
Concurrent Evacuation
Concurrent Update References

Normalized Execution Time

Local Memory Ratio

100%  50%  25%  13%

Normalized Execution Time

App Thread
GC Thread
Concurrent GC

No Locality

Resource Contention

Process

App Threads  GC Threads

Local DRAM

Swap

InfiniBand

Memory Servers
Can we move concurrent marking and evacuation to memory servers?
Can we move concurrent marking and evacuation to memory servers?

Reduced Interference  Near-data Computing
Challenges

No efficient way to enforce memory coherence between the CPU and memory servers

• Concurrent Updating Reference
  • Problem: Overwritten updated references

• Concurrent Evacuation
  • Problem: Lost forwarding address
Heap Indirection Table

Region #1

- obj B
- header

Region #2

- obj A
- field f

allocated entry
free entry
Problem #1: Concurrent Updating References

CPU Server

Memory Server

- Region 0
  - Header
  - 1024
  - Stale ObjB
  - ObjA

- Region 1

- Region 2
  - Stale ObjB

- Region 3
  - ObjB
Problem #1: Concurrent Updating References

- CPU Server
  - Header
  - 1024
  - Stale ObjB
  - ObjA

- Memory Server
  - Region 0
    - Header
    - 1024
    - Stale ObjB
    - ObjA
  - Region 1
  - Region 2
    - Stale ObjB
  - Region 3
    - ObjB
Problem #1: Concurrent Updating References

CPU Server

Memory Server

Region 0
- Header
- 1024
- Stale ObjB
- ObjA

Region 1
- Header
- 1024
- Stale ObjB
- ObjA

Region 2
- Stale ObjB

Region 3
- ObjB
Problem #1: Concurrent Updating References

CPU Server

Memory Server

Region 0
Header
0
Stale ObjB
ObjA

Region 1
Header
1024
Stale ObjB
ObjA

Region 2
Stale ObjB

Region 3
ObjB
Problem #1: Concurrent Updating References

CPU Server

Memory Server

Region 0
Header
0
Stale ObjB
ObjA
Region 1
Header
1024
Stale ObjB
ObjA
Region 2
Stale ObjB
ObjB
Region 3
ObjB
Problem #1: Concurrent Updating References

CPU Server

Memory Server

- **ObjA**: Not updated.
- **ObjB**: Updated.

Region 0
- **Header**: 0
- **Stale ObjB**

Region 1
- **Header**: 1024
- **ObjB**
- **ObjA**

Region 2
- **Stale ObjB**

Region 3
- **ObjB**
Problem #1: Concurrent Updating References

- CPU Server
- Memory Server

Diagram:
- Region 0
  - Header
  - 0
  - Stale ObjB
  - ObjA
- Region 1
- Region 2
- Region 3

Stale ObjB

ObjB
Problem #1: Concurrent Updating References

CPU Server

Memory Server

- Region 0
  - Header
  - 0
  - Stale ObjB
  - ObjA

- Region 1
- Region 2
- Region 3
  - Stale ObjB
  - ObjB
Solution #1

- Eliminate the need to directly update pointers at both the CPU and memory servers
Solution #1

- Eliminate the need to directly update pointers at both the CPU and memory servers
Solution #1

- Eliminate the need to directly update pointers at both the CPU and memory servers

![Diagram showing the relationship between regions and objects](image_url)
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dead</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

Region 0
- ObjA
- Dead

Region 1
- ObjB

Region 2
- ObjB

Region 3
- ObjC
- ObjC
Problem #2: Concurrent Evacuation

**CPU Server**

- ObjB
- ObjC

**Memory Server**

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjB</td>
<td>ObjC</td>
</tr>
</tbody>
</table>
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjB</td>
<td></td>
</tr>
<tr>
<td>ObjB</td>
<td></td>
<td>ObjC</td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

Region 0
- ObjA
- ObjB (Dead)
- ObjC

Region 1
- ObjB

Region 2
- ObjB

Region 3
- ObjB
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>ObjB</td>
<td>ObjB</td>
<td>ObjC</td>
</tr>
<tr>
<td>Dead</td>
<td>ObjB</td>
<td>ObjB</td>
<td>ObjC</td>
</tr>
<tr>
<td></td>
<td>ObjC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem #2: Concurrent Evacuation
Problem #2: Concurrent Evacuation

CPU Server

Memory Server

Region 0
- ObjA
- Dead
- ObjB
- ObjC

Region 1
- ObjA
- ObjB
- ObjC

Region 2
- ObjB
- ObjA
- ObjB
- ObjC

Region 3
- ObjB
- ObjA
- ObjB
- ObjC
Solution #2

- Causes of Problem #2:
  - JVM accesses data at object-level, while the OS kernel manages data at page-level.
  - Synchronization on each object access would incur too much overhead

Tablet Lock

Region-level Synchronization
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dead</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tablet   Tablet
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjB</td>
<td>ObjC</td>
</tr>
</tbody>
</table>

Tablet Tablet
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjB</td>
<td>ObjB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tablet | Tablet
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjB</td>
<td></td>
</tr>
<tr>
<td>ObjB</td>
<td></td>
<td>ObjB</td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td>ObjC</td>
<td></td>
</tr>
</tbody>
</table>

Tablet

Tablet
Solution #2

CPU Server

Memory Server

Region 0 | Region 1 | Region 2 | Region 3
---------|---------|---------|---------
ObjA     | Dead    | ObjB    | ObjC    
ObjB     | ObjB    | ObjC    | Tablet  
ObjC     | Tablet  | Tablet  |         

ObjA

EntryB

ObjB

ObjC
Solution #2
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjA</td>
<td>ObjC</td>
</tr>
<tr>
<td>ObjB</td>
<td>ObjB</td>
<td>ObjC</td>
<td></td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td>Tablet</td>
<td>Tablet</td>
</tr>
</tbody>
</table>

EntryB

ObjB

ObjC

ObjA

Dead

ObjB

ObjC

ObjC
Solution #2

CPU Server

Memory Server

<table>
<thead>
<tr>
<th>Region 0</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjA</td>
<td>Dead</td>
<td>ObjA</td>
<td>ObjA</td>
</tr>
<tr>
<td>ObjB</td>
<td>ObjB</td>
<td>ObjC</td>
<td>ObjC</td>
</tr>
<tr>
<td>ObjC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tablet | Tablet

EntryB

ObjB
Benchmarks

• We evaluate Mako on 7 workloads under three different local memory ratios: 50%, 25%, and 13%
  • Dacapo: Tradesoap, Tradebeans, H2
  • Apache Cassandra: Insert Intensive, Update Intensive
  • Apache Spark: PageRank, Transitive Closure

• We compare Mako with
  • Shenandoah: a modern concurrent collector in OpenJDK
  • Semeru: a G1-based generational GC for disaggregated memory
Results: Throughput

![Diagram showing speedup and local memory ratio]
Results: Pause Time

• Mako achieves $\sim 12$ms at the 90th-percentile pause time

• Semeru’s pauses are 2 to 3 orders of magnitude longer
Key Takeaways

• Under new hardware and system settings, it might be a good idea to bring back some old concepts and techniques. E.g. Heap Indirection Table

• Offloading GC to memory servers makes compute near data, which improves applications’ throughput
Q&A