MemLiner: Lining Up Tracing and Application for a Far-Memory-Friendly Runtime

Chenxi Wang*, Haoran Ma* (co-first author), Shi Liu, Yifan Qiao, Jonathan Eyolfson, Christian Navasca, Shan Lu, Guoqing Harry Xu
Memory Capacity Bottleneck in Datacenters

Growing imbalance between processor computation and memory capacity

Memory underutilization in datacenters
Far-Memory System

Host Server

CPU

Small Local Memory

60 ns

~ 10 μs

Network

e.g. RDMA over InfiniBand

Remote Memory Pool
Applications written in high-level languages are dominant in datacenter workloads.
Garbage Collection

Tracing is done concurrently with applications

<table>
<thead>
<tr>
<th>Local Mem Ratio</th>
<th>25%</th>
<th>13%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slowdown</td>
<td>2.6x</td>
<td>3.4x</td>
</tr>
</tbody>
</table>
Resource Competition

Process

App Threads

GC Threads

Local DRAM

Swap

Memory Servers

InfiniBand
Ineffective Prefetching

Without Concurrent Tracing

With Concurrent Tracing

Page Fault Sequence

Faulty Page Index

Page Fault Sequence

Faulty Page Index
Can we disable concurrent tracing?

End-to-end Execution Time

Elapsed Time (s)

Without Concurrent Tracing  With Concurrent Tracing

GC Pause Time

Elapsed Time (s)

Without Concurrent Tracing  With Concurrent Tracing
Are application and garbage collection completely unrelated?
Observations

1. Application and GC are just temporally unaligned

2. Changing object access order in GC is possible
Key Design Idea

(a) Current runtime

(b) MemLiner runtime
Object Classification

1. **Local Objects**: Currently being accessed by application threads
   - GC threads should touch

2. **Incoming Objects**: In remote memory, will soon be accessed by app threads
   - GC threads should touch

3. **Distant Objects**: In remote memory, will **not** be accessed by app threads soon
   - GC threads should **delay** the access
Challenges in Classifying Objects

- **Local**
  - How to inform GC threads accessed objects

- **Incoming**
  - What kind of objects will be accessed by app threads soon

- **Distant**
  - How to estimate the location of objects
Barriers

Read Operation

\[ a = b.f \quad \text{or} \quad a = b[i] \]

Write Operation

\[ b.f = a \quad \text{or} \quad b[i] = a \]
Local Objects

\[ a = b \cdot f \]
\[ b \cdot f = a \]
Incoming Objects

- Currently being accessed by app
- In remote memory, used by app soon
Distant Objects

Counter maintained by kernel

Object reference maintained by runtime

\[ \text{Diff}(\text{Current Epoch}, \text{Timestamp}) < \delta \]
Benchmarks

- MemLiner is implemented in two widely-used garbage collectors:
  - G1 GC
  - Shenandoah GC
- Evaluated MemLiner on 12 workloads using a range of local memory ratios
- MemLiner is run on two swap systems: Fastswap and Leap
Results: Throughput

G1 GC

<table>
<thead>
<tr>
<th>Local Memory Ratio</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>1.48</td>
</tr>
<tr>
<td>13%</td>
<td>1.51</td>
</tr>
</tbody>
</table>

Shenandoah GC

<table>
<thead>
<tr>
<th>Local Memory Ratio</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>25%</td>
<td>2.16</td>
</tr>
<tr>
<td>13%</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Results: Prefetching Effectiveness

- An average of 1.6x speed up under 25% local memory on Leap
- Reduces 58% of on-demand swap-ins, and 53% of total swap-ins on average.
Key Takeaways

- Runtime should also be taken into consideration when hardware changes
- Runtime serves as a semantic bridge between application and underlying OS/hardware architecture

Thank you! Code at [https://github.com/uclasystem/MemLiner](https://github.com/uclasystem/MemLiner).
Q&A