M3: End-to-End Memory Management in Elastic System Software Stacks

David Lion, Adrian Chiu, Ding Yuan
University of Toronto
Traditional Memory Provisioning

- Traditional working set model [Denning, 1968]
  - Requires a set of pages in memory to avoid thrashing

- Forced to provision for peak memory usage
  - Memory wasted outside the peak usage
Elastic Applications

- Performance continuously improves with additional memory
Provisioning Elastic Applications

- Potential to improve throughput by utilizing more memory
- Dynamically adjust application memory usage to availability
- Adjust memory usage in the OS?
  - OS lacks domain knowledge of memory usage
- Adjust memory directly in the application?
  - Memory abstractions hinder application memory management
Abstraction Problems

- Static settings forced to control physical memory usage
  - i.e. JVM heap size

<table>
<thead>
<tr>
<th>Layer</th>
<th>Abstracts</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Physical Memory</td>
<td>Require Static Settings</td>
</tr>
</tbody>
</table>
Abstraction Problems

- Physical memory optimistically retained
  - Not returned to the OS

<table>
<thead>
<tr>
<th>Layer</th>
<th>Abstracts</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Physical Memory</td>
<td>Require Static Settings</td>
</tr>
<tr>
<td>JVM</td>
<td>Allocation + Reclamation</td>
<td>Retain Physical Memory</td>
</tr>
</tbody>
</table>
Abstraction Problems

- Evicted memory sits unusable as garbage

<table>
<thead>
<tr>
<th>Layer</th>
<th>Abstracts</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Physical Memory</td>
<td>Require Static Settings</td>
</tr>
<tr>
<td>JVM</td>
<td>Allocation + Reclamation</td>
<td>Retain Physical Memory</td>
</tr>
<tr>
<td>Spark</td>
<td>Partition Input Data + Eviction</td>
<td>Uncoordinated Reclamation</td>
</tr>
</tbody>
</table>
Abstraction Problems: Example

- Cannot adapt to changes in memory demand
- Must provision for peak usage
Summary

- Memory abstractions unnecessarily limit elastic applications
  - Static settings unable to adapt
  - Problem made worse by multiple layers of abstraction
Summary

- Memory abstractions unnecessarily limit elastic applications
  - Static settings unable to adapt
  - Problem made worse by multiple layers of abstraction
- **M3 Goal:** improve throughput and memory utilization
  - Bridge memory abstractions
  - React to memory demand and availability
Outline

- Memory Management Issues in Elastic System Stacks
- M3 Design
  - Signal handling
  - Adaptive allocation protocol
  - Monitor
- Evaluation
- Conclusion
M3 Design Overview

- Remove static memory settings
- Monitor: send signals when memory reaches a threshold
  - Low threshold: early warning, return what you can
  - High threshold: nearing memory exhaustion
- Signal handler: reclaim and return memory
- Adaptive allocation protocol: slow application memory growth
## Practical Approach

- End-to-end argument [Saltzer, 1984]
  - Applications implement all policy
- Utilize existing memory reclamation mechanisms
  - GC, cache eviction, etc

<table>
<thead>
<tr>
<th></th>
<th>JVM</th>
<th>Go Runtime</th>
<th>Spark</th>
<th>Memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC Modified</td>
<td>220</td>
<td>50</td>
<td>250</td>
<td>170</td>
</tr>
</tbody>
</table>
Signal Handling

- Applications register signal handler
Signal Handling

- Monitor sends out signal
Signal Handling

- Application reclaims memory
- Spark evicts data blocks
Signal Handling

- Notify lower layer when memory reclamation completes
  - Spark calls JVM GC API
Signal Handling

- Lower layer reclaims memory
  - JVM runs a GC cycle
Signal Handling

- Return memory to OS
  - JVM uses `madvise` syscall
Adaptive Allocation Protocol

- Goal: slow memory growth to prevent exhaustion
- Dynamically adjust **allowed allocation rate**
  - Allowed allocations grow memory
- Disallowed allocations reclaim memory before continuing
  - No correctness issues or complex modifications

```c
int alloc(size_t size) {
  if (!allow()) {
    evict(size);
  }
  // continue allocation path
}
```
Adaptive Allocation Protocol

- Upon receiving a high signal allow rate is set to 0%
- Epoch: time taken to handle a signal
- $NUM_{epochs}$: user parameter to control recovery

![Diagram showing the relationship between epochs since high threshold signal received and allow rate.](attachment:image.png)
Adaptive Allocation Protocol

- Reward efficient reclamation
  - Fast reclamation $\rightarrow$ small epoch $\rightarrow$ more allocations allowed
- Memory demand is respected
  - More allocations $\rightarrow$ more allowed allocations $\rightarrow$ memory growth

![Graph showing relationship between epochs and allow rate](image-url)
Monitor

- Goal: maximize memory utilization while avoiding exhaustion
- Dynamically adjust high and low thresholds
  - Lower when usage stays high
    - Slow reclamation requires earlier signals
  - Raise when usage stays low
    - Later signals allow for higher memory utilization
Outline

- Memory Management Issues in Elastic System Stacks
- M3 Design
  - Signal handling
  - Adaptive allocation protocol
  - Monitor

Evaluation

- Conclusion
Evaluation Methodology

- Cluster of 9 servers, each with 64GB memory and 16 cores
- Spark + JVM: PageRank, n-weight, k-means
  - HiBench [Huang et al. 10]
- Go Cache: runs a mixed read and write benchmark
- 16 Workloads
  - Combination of jobs with different scheduling
- Measure average speedup of each application’s completion time
Configurations

- Default
  - All parameters set to default values

- Global Optimal
  - Minimize average throughput of all workloads
  - Knowledge and tuning of all possible job combinations

- Oracle
  - Optimize jobs individually for each workload
  - Individual tuning and reconfiguration for each job scheduling

- Oracle with Spark Parameters
  - Further tune unadvised parameters
Results

- **Oracle with Spark Conf**: 1.6x average, 3.1x best case
- **Global Optimal**: 1.9x average, 3.4x best case
Unmodified Execution
M3 Execution
Worst Case

- **Oracle with Spark Configuration**
  - 3.8% slower on average, 7.0% worst case
Limitations

- Cannot guarantee optimal memory distribution
  - Distribution is decentralized

- Requires cooperative applications
  - Faithful implementation of policies
  - Not withholding memory after receiving a signal
Related Work

- Memory Distribution among Virtual Machines
  - Ballooning [Waldspurger 02], App. Ballooning [Salomie et al. 13]
  - Resource deflation [Sharma et al. 19]
  - MemOpLight: among containers [Laniel et al. 20]

- MM coordination
  - JVM resizing: [Alonso & Appel 90], CRAMM [Yang et al. 06]
  - GC coordination: Taurus [Maas et al. 16]

- Uniqueness of M3:
  - Cross all memory abstractions in elastic applications
  - End-to-end design
Concluding Remarks

- Data center applications suffer due to abstractions
  - Multiple uncoordinated layers
  - Fall back to static settings
- M3 bridges memory abstractions
  - Coordinates memory management
  - Dynamically adjusts to changes in memory demand
- Open sourced at: https://github.com/dsrg-uoft