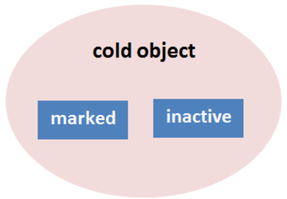
# Identifying cold objects in the IBM JVM

**Baoguo Zhou, Gerhard W. Dueck**
University of New Brunswick, IBM Canada
Faculty of Computer Science
Email: barry.zhou@unb.ca, gdueck@unb.ca

## Motivation

**Introduction**: in some specific Java application, there may be a number of objects which are alive but infrequently accessed. Those objects persist in heap memory, consuming limited internal memory resource and, even if paged out of physical memory, are frequently touched as they are traversed by the garbage collection. We refer to these objects which are marked and inactive as “cold objects”.

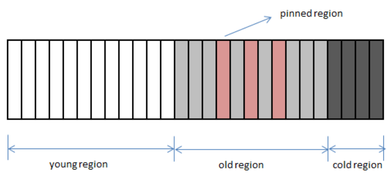
**Ideas**: Identify cold objects and move them to cold regions that are excluded from normal garbage collection operations (mark, sweep, compact, copy-forward).




## Background

**Heap:** All Java objects are stored on the heap.

**Region:** when Java virtual machine is running the balanced garbage collector, the heap is divided into multiple regions.




**Region age:** increases as a region survives successive garbage collection cycles.

**Region density:** the proportion of the region occupied by objects.

**Pinned region:** an older, dense region that has been selected for exclusion from copy-forward and compaction operations.

The Java VM is instrumented to track activity at the object level within pinned regions, enabling cold objects to be identified over time. Once activity has stabilized within a pinned region, the cold objects contained in the region can be harvested, that is, they can be moved into a cold region. Since cold regions are excluded from normal garbage collection operations and infrequently accessed by the mutator, they can be swapped out to a backing store, freeing physical memory for more active regions.

## Solutions:

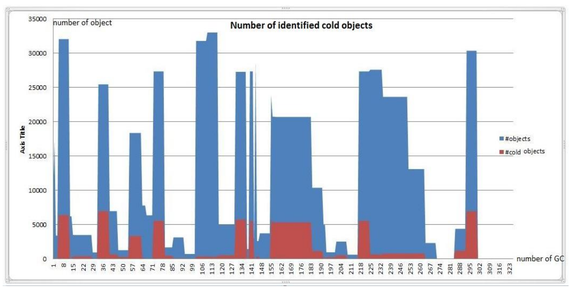
**The methods**: track objects activity within pinned regions by walking mutator stacks to collect active object references; objects that are marked but inactive are cold.

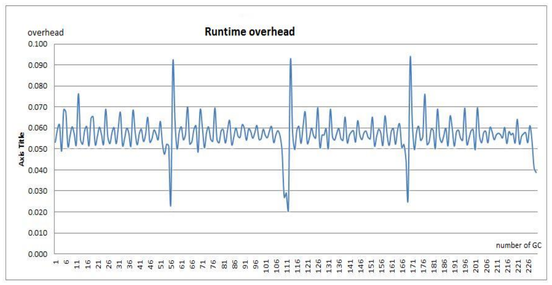
The process is as follows:

- Select dense, maximally aged regions for pinning.
- Walk mutator thread stacks frequently to collect active references.
- Collect cold objects from pinned regions that have received no new active references for a pre-set period of time.

## Experimental results

Using the SPECjvm2008 compiler.compiler and derby benchmarks, a significant number of cold objects were identified. The runtime overhead for activity tracking ranged from 5-6%. Each benchmark was run for 1 hour.




**Conclusion**: the results show the method of identifying cold objects is feasible. Future work will determine whether the reduction in memory pressure is a worthwhile benefit given the sampling overhead.