Using Page Residency to Balance Tradeoffs in Tracing Garbage Collection

Daniel Spoonhower

Carnegie Mellon University
(Joint work with Guy Blelloch and Robert Harper)

11 June 2005
Motivation: Balancing Tradeoffs

Between copying and non-copying collection:
- use of space and time (i.e. overhead)
- semantic constraints (e.g. pinning)
- locality, traversal order, etc.

Flexibility is critical:
- not all objects should be treated the same
- application behavior changes over time
Adaptive Hybrid Collection

Our goal:

To build a tracing collector that \textit{dynamically} balances these tradeoffs

Must determine collection technique... 
- separately for each part of the heap
- at the beginning of each collection

\textit{Use resideny to adapt to runtime conditions}
Outline of Talk

Adaptive Hybrid Collection
- use runtime information to combine copying and non-copying collection

Predicting Residency
- runtime property directly related to costs of these techniques

Experimental Results
- show that adaptive collection performs well in many environments
Adaptive Hybrid Collection

Extend *mostly* copying collection (Bartlett)
- reduce fragmentation in constrained GC

Use a similar mechanism to uniformly account for:
- pinned objects
- large objects
- densely populated pages

By generalizing this mechanism, we will develop a family of hybrid collectors.
Mostly Copying Collection

“Copy objects... most of the time.”

Key ideas:

- divide heap into pages
- determine strategy for each page dynamically: objects may be evacuated or promoted in place

Gain benefits of copying and non-copying collection
divide heap into pages...
Mostly Copying Collection

... allocate new objects...
Mostly Copying Collection

... begin collection...
Mostly Copying Collection

\[ \ldots \text{flip} \ldots \]
Mostly Copying Collection

...some objects are unreachable...
 Mostly Copying Collection

...evacuate reachable objects...
Mostly Copying Collection

...pinned...
Mostly Copying Collection

... in-place promotion!
Mostly Copying Collection

... and reclaim unused space.
Large objects often relegated to separate heap

- ...because they are expensive to copy

We use in-place promotion instead.

- avoids usual penalties of copying large objects
- no additional heap space required
- use small objects to fill in gaps
Large Objects Shouldn’t Be Relocated

They don’t cause (much) fragmentation

- little benefit to moving them around

Similarly for densely populated pages:

- expensive to evacuate

- little fragmentation
Page Residency as a Guiding Principle

Residency = density of reachable objects
- avoid evacuating objects from dense pages
- focus on fragmented parts of the heap

Large objects occupy pages with high residency.
- therefore, they are promoted in place
Revisiting Allocation

Copying collectors use a simple space check.
- constant time when there is sufficient space
- requires contiguous unused space

In-place promotion requires more sophisticated management of free space.
- as in mark-sweep collection
- need an unused portion of appropriate size
Revisiting Allocation

Use a list of free gaps on pages promoted in place.
- we use a modified first-fit algorithm

Use residency to avoid allocating on dense pages.
- throw away gaps if residency is too high
- trading space for time
Overview of Our Hybrid Algorithm

Collection strategy is determined for each page.

Two data structures for heap traversal:
- Cheney queues for evacuated objects
- mark stack for objects promoted in place

Allocation using a modified first-fit:
- favor free pages (treated as large gaps)
- other free list algorithms also applicable
Two Mechanisms to Reclaim Space

Reclaim space both...

- via evacuation
  - use residency to determine which objects should be evacuated

- during allocation
  - use residency to determine which pages have enough space to make allocation worthwhile
Collector Configurations

- Two thresholds determine when residency is low enough for evacuation or allocation.
  - $r_{\text{evac}}$: evacuation threshold, maximum residency at which objects are evacuated
  - $r_{\text{alloc}}$: allocation threshold, maximum residency at which free space is reused (both measured as % of page)

- Classic algorithms fall out as special cases
  - semi-space: $r_{\text{evac}} = 100\%, r_{\text{alloc}} = 0\%$
  - mark-sweep: $r_{\text{evac}} = 0\%, r_{\text{alloc}} = 100\%$
A Continuum of Configurations

- **Mark-sweep**
  - $r_{alloc}$ when free space is reused
- **Semi-space**
  - $r_{revac}$ when objects are evacuated
- **Hybrid**

Graph showing the continuum of configurations with $r_{alloc}$ and $r_{revac}$ axes.
Progress in Talk

So far...

- outlined a hybrid algorithm for garbage collection
- shown how residency thresholds give rise to a continuum of configurations

Next: how to determine residency...
Residency can be computed during a heap traversal at little additional cost.

However, we need to know residency at the beginning of collection...

And to conservatively estimate amount of data to be copied by the end of the previous cycle.
Use Past Measurements

One possibility: explicitly measure residency during each collection (using an extra traversal).

Instead, use measurements to predict future values

- historical measurements avoid a second pass over heap
- measured residency still accounts for changes in program behavior over time
Other hybrids fix residency estimates in advance.

- **syntactic** properties to distinguish objects
- *e.g.* size, type, allocation point

**Age:** “Most objects die young.”

⇒ “Nursery is sparsely populated.”

- *i.e.* using age to predict residency

- however, generational collectors are often inconsistent in using age (*e.g.* large objects)
Recovering From Poor Predictions

Adaptive GC recovers naturally from poor predictions.

- at most two collections to recover space
- *without* changes to algorithm

Residency is used in two ways:

- to reserve sufficient space for replicas
- to determine how objects are promoted

Overestimation in latter is not a problem.
Recovering From Poor Predictions

Consider the impact of overestimation:
- many sparse pages remain – high fragmentation
- measurements give true residency for each page
- next collection will relocate objects to reclaim space

GC with poor predictions *behaves* like an explicit measurement phase.
- in effect, only corrects predictions when they would be wrong
Allocation uses true residency, not predictions.

- actual residency is known at the end of collection

Recovers space immediately, no need to wait for another collection

- more overestimation leads to cheaper allocation
Experimental Results

If we know usage patterns of an application in advance, we choose an appropriate algorithm.

- adaptive hybrid GC relieves us of this choice

Consider the impact of heap size on GC overhead

- caveat: other effects of GC not shown here
Impact of Heap Size

Heap size = total memory available to collector

We expect copying collectors to perform well when there is plenty of space.

However, for small heaps, semi-space collectors may not satisfy all allocation requests.
Impact of Heap Size

![Graph showing the impact of heap size on total garbage collection time for different garbage collection algorithms: Mark-sweep, Semi-space, and Hybrid. The graph plots the total gc time (s) against the heap size (MB). The algorithms show different patterns and performance as the heap size increases.]
Impact of Heap Size

![Graph showing the impact of heap size on total GC time. The graph compares Mark-sweep, Semi-space, and Hybrid collections. The x-axis represents heap size (MB) ranging from 40 to 140, and the y-axis represents total GC time (s) ranging from 0.1 to 0.7. The graph shows a decreasing trend in total GC time as heap size increases.]
Summary of Other Measurements

Same configuration performs well in many different environments

- different applications
- small and large footprints (100 kB to 100 MB)

Insensitive to changes in configuration thresholds

- 90%/90% hybrid is a good choice for our implementation
- small variations don’t dramatically change performance
Related Work

- Mostly Copying – Bartlett; Smith & Morrisett
- Dynamic Selection – Soman, Krintz, & Bacon (changes collection technique for entire heap)
- Garbage First – Detlefs, Flood, Heller, & Printezis (also uses residency as a measure of cost)
- Metronome – Bacon, Cheng, & Rajan (different point in the “mostly” design space)
Summary

Adaptive hybrid collection
- combines (non-)copying GC dynamically
- use page residency to guide runtime decisions

Continuous spectrum of tracing collectors
- trade off space for cost of copying objects

Use measurements to adapt to application behavior
- predict residency based on past measurements
- naturally recovers from poor estimates