Generation scavenging: A non-disruptive high-performance storage reclamation algorithm
By David Ungar

Presented by Donald Nguyen

February 9, 2009

Context

- The year was 1984...
  - Microcomputers, interactive programming environments
- Design interactive Smalltalk-80 system for Sun workstation
  - Main technical hurdles:
    - Limit pause times
    - Play well with virtual memory

Players

- Virtual memory
  - Segmentation
  - Paging
- Popular GC algorithms
  - Reference counting
  - Mark-sweep [MKSMW]
  - Copy-collector [Baker-78]
  - Generational collection [Lieberman-Hewitt-83]

Virtual Memory

- Segmentation
  - Main memory divided into blocks of potentially unequal lengths
  - Typically typed into program and data segments
  - When a segment is swapped in, it can only replace a segment of the same size or larger (fragmentation)
- Paging
  - Fixed-size blocks

Reference counting

- High overhead because additional action must be done for all objects
- Immediate reference counting
- Deferred reference counting [Deutsch-Bobrow]
  - Ignore references from local variables
  - Precludes reclamation during execution
- Question: Reference counting today?
  - Different overheads today?
  - What about William's anecdote about only using reference countable data structures?

Tracing Algorithms

- Mark-sweep
  - Mark phase inspects all live objects, sweep phase modifies all dead objects
  - Large pause times
- Scavenging algorithms
  - Psuedo real-time; overhead to install and check forwarding pointers, maintain remembered set
  - Baker's semispace algorithm
  - Ballard's modification of Baker's algorithm
    - Create separate area for old objects
    - 600 KB for old objects, 512 KB object table, 2 x 1 MB semispaces
  - Lieberman-Hewitt's generational algorithm, not implemented

Straw Man

- OS already has virtual memory, leverage abstraction of "infinite memory"
  - One object per segment/page
  - Use VM to keep live objects in main memory
  - Dead objects will be moved to secondary storage
- Problems?

Straw Man

- OS already has virtual memory, leverage abstraction of "infinite memory"
  - One object per segment/page
  - Use VM to keep live objects in main memory
  - Dead objects will be moved to secondary storage
- Problems?
  - For segmentation, object variance: [24, 128,000] bytes (mean: 50) and number: 32,000–64,000 objects
  - Object fragmentation

Straw Man

- OS already has virtual memory, leverage abstraction of "infinite memory"
  - One object per segment/page
  - Use VM to keep live objects in main memory
  - Dead objects will be moved to secondary storage
- Problems?
  - For segmentation, object variance: [24, 128,000] bytes (mean: 50) and number: 32,000–64,000 objects
  - Object fragmentation
  - Allocation rates:

$$\frac{70 \text{ bytes}}{1 \text{ object}} \times \frac{1 \text{ object}}{80 \text{ instructions}^{1}} \times \frac{9000 \text{ bytecodes}}{\text{second}} = \frac{7800 \text{ bytes}}{\text{second}}$$

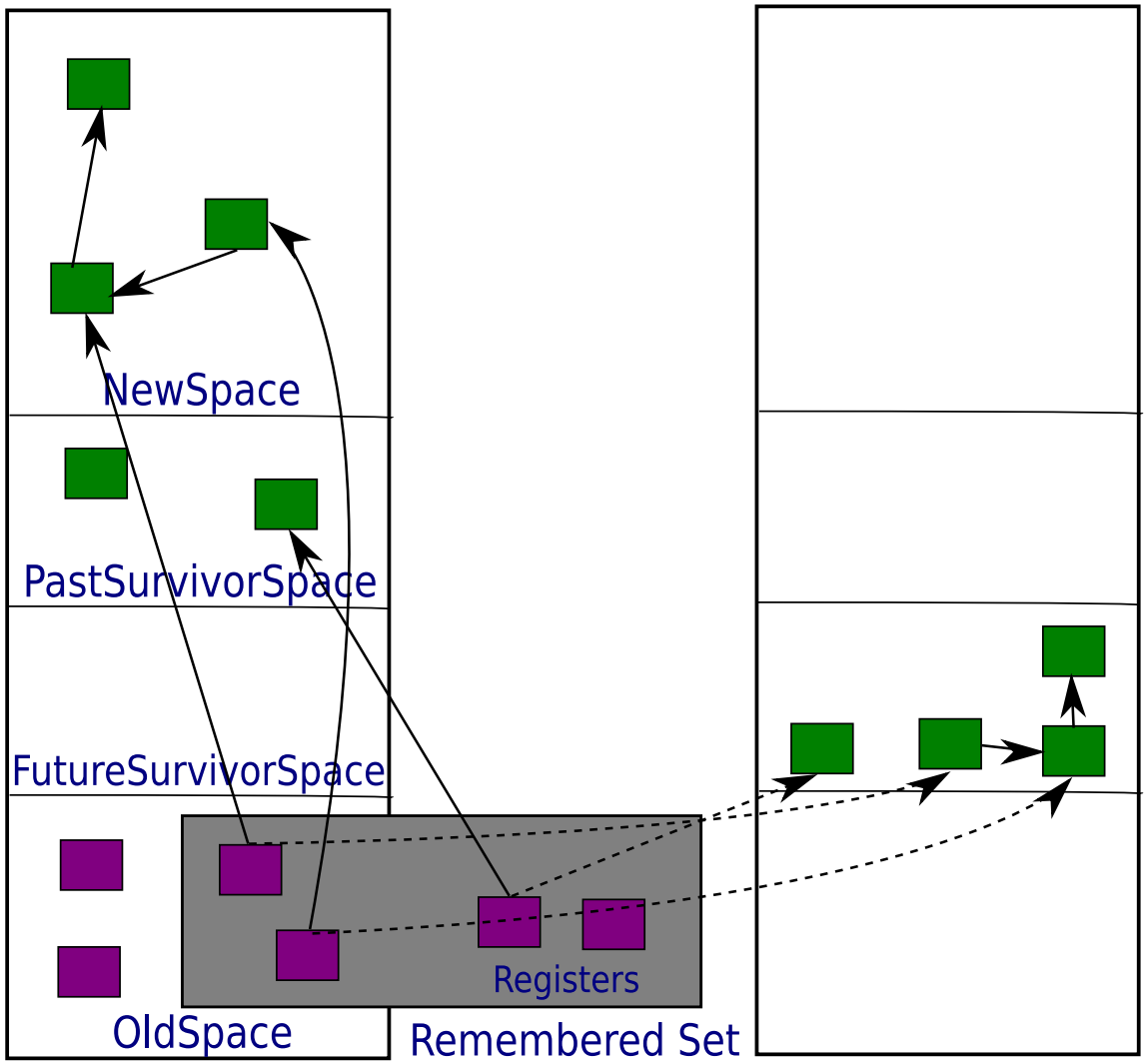
$$\frac{\text{BS-II second}}{7800 \text{ bytes}} \times \frac{\text{Dorado second}}{13 \text{ BS-II seconds}} \times \frac{100 \text{ MB}}{\text{disk}} \approx \frac{20 \text{ minutes}}{\text{disk}}$$

[1] Appel reports 1 object every 30 instructions

Design Space

- Keep main memory footprint low: 1.5–3 MB
- Low pause time, but not necessarily real-time
- Reduce page faults
- Reduce CPU overhead

Generational Scavenging



Generational Scavenging

- Novelties:
  - Regions of different sizes: NewSpace (140 KB), *SurvivorSpace (28 KB each)
  - OldSpace not resident in main memory
  - *Tenuring*: after a while a surviving object is promoted to OldSpace.
- Similarities with Ballard's algorithm:
  - Young and old generations
  - Copies live objects
  - Reclaims old objects offline
- Differences with other generational algorithms:
  - Divides new space into three regions instead of two
  - Not incremental, no checks needed on loads

Results

	CPU Time (%)	Main Memory (KB)	Paging I/O	Pause time (s)	Pause interval (s)
page it	?	15	$\approx 50/s$		
imm. ref. count	15–20	15	?	0	∞
compaction				1.3	60–1200
def. ref. count	11	40	?	0.030	0.30
compaction				1.3	60–1200
mark-sweep	25–40	1900	90/gc	4.5	74
Ballard	7	2000	0	0	∞
Gen. Scavenge	1.5–2.5	200	1.2/s	0.38	30

Table 1: Summary of reclamation strategies

Is This Still Relevant Now?

- Revisiting the claims:
  - Limit pause times to a fraction of a second
  - Requires no hardware support
  - Meshes well with virtual memory
  - Reclaims circular structures
  - Uses less than 2% of the CPU time
- Is the experimental methodology still sound?
- Should old objects be kept in secondary storage?

Breaking or Enforcing Abstractions

- If computer science is generally about making abstractions and extracting performance about breaking them, how should we view Ungar's work?
  - A case for a wider interface between runtime system and OS?
  - An isolated issue related to implementing a managed memory system?
- What memory management techniques can help or hinder virtual memory systems and vice versa?
  - E.g., Appel recommends mapping the memory just before the free space (assuming allocation goes from high to low) to an inaccessible page so that the out of memory test can be converted to a page fault