Understanding the Connectivity of Heap Objects

Martin Hirzel, Johannes Henkel, Amer Diwan
University of Colorado at Boulder

Michael Hind
IBM T.J. Watson Research Center

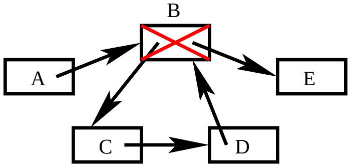
Motivation

- Connectivity often gets in the way of GC:
  - Pig-and-python problem
  - Write barrier overhead
- We investigate connectivity to see:
  - How GC can avoid problems with it
  - How GC can benefit from it

⇒ ***This is an empirical study of program behavior***

Key-object opportunism

- Hayes, *Using key object opportunism to collect old objects*, OOPSLA 1991
- Hypothesis: connected objects die together

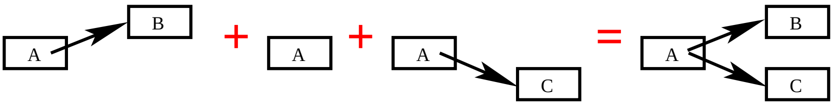


- Idea: when key object dies, collect connected objects
  - High benefit (reclaimed memory) at low cost
  - Opportunistic about what to collect

⇒ ***Need to understand connectivity to implement this***

Methodology

- We obtained traces from 22 Java programs
  - Traced events: allocation, pointer write, death
  - Infrastructure: Jikes RVM 1.1 aka Jalapeño
    (BaseBasenoncopyingGC, 1-processor PPC/Linux)
- We used the traces to construct and analyze the global object graph (Gog):
  - Nodes: all objects during whole run
  - Directed edges: all pointers during whole run

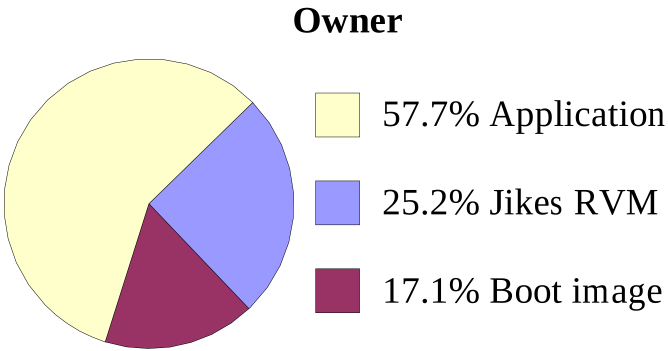


Benchmarks

Benchmark	Bytecode size	Total alloc.	Comments
mst	5KB	15,446KB	Java-Olden
bisort	4KB	16,085KB	
voronoi	13KB	17,712KB	
tsp	5KB	21,583KB	
em3d	7KB	22,101KB	
perimeter	9KB	31,528KB	
treeadd	3KB	35,751KB	
power	11KB	38,101KB	
health	9KB	38,618KB	
bh	17KB	42,900KB	
mpegaudio	56KB	35,870KB	SPECjvm98
db	9KB	97,899KB	
compress	17KB	132,931KB	
mtrt	56KB	173,683KB	
javac	1909KB	285,631KB	
jack	127KB	331,031KB	
jess	387KB	334,187KB	
ipsixql	1,986KB	99,908KB	XML database
xalan	4,200KB	123,412KB	XSLT processor
nfc	556KB	173,637KB	chat server
jigsaw	4,312KB	257,452KB	web server

Owner

- The Jikes RVM is implemented in Java
  - Allocates heap objects at runtime
  - Pre-allocates objects in bootimage

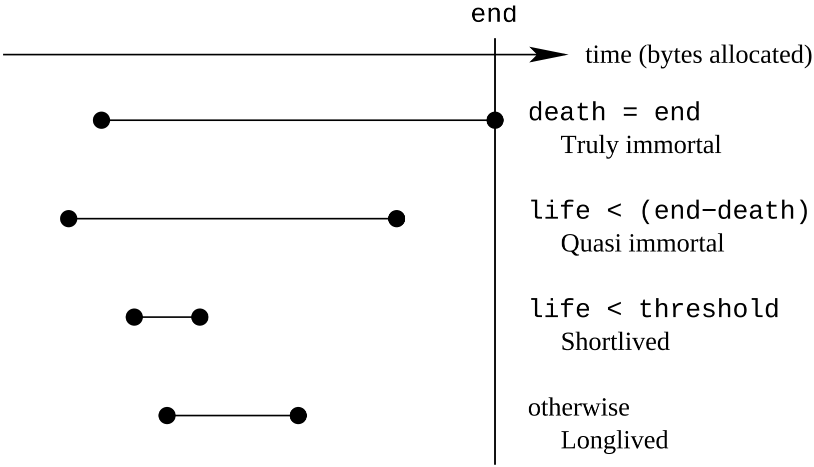


⇒ ***Jikes RVM objects put additional pressure on GC***

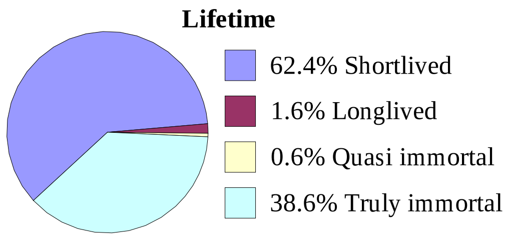
⇒ ***Good GC can speed up application and Jikes RVM***

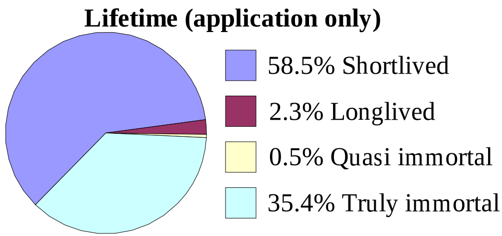
Lifetime definitions

- Classification of objects into four bins
- Slightly modified from [Blackburn *et al.* 2001]



Lifetime data





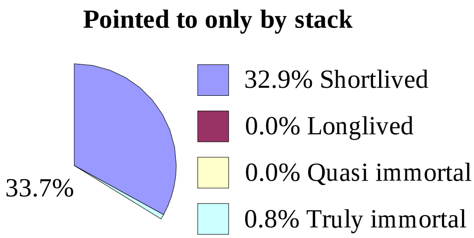
⇒ ***Almost no objects are longlived or quasi immortal***

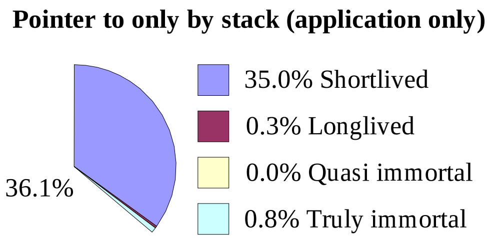
⇒ ***Rule of thumb: 60% shortlived, 40% truly immortal***

⇒ ***GC should avoid wasting effort on immortal objects***

Connectivity from stack

- Objects pointed to only by local variables



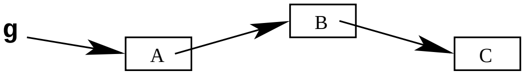


⇒ ***Most of these objects are shortlived***

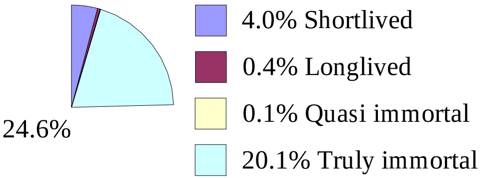
⇒ ***Stack allocation and regions can reclaim these cheaply***

Connectivity from globals

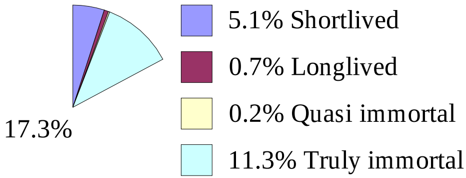
- Global variable = static field in Java
- Reachable = transitively pointed to



Reachable from globals



Reachable from globals (application only)

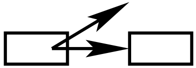
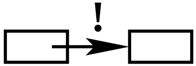
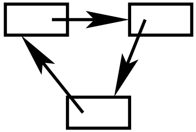
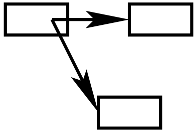


$\Rightarrow$ ***Many of these objects are truly immortal***

$\Rightarrow$ ***This could be used for pretenuring***

Do connected objects die together?

[given connectivity, (equideath pairs) / (all pairs)]

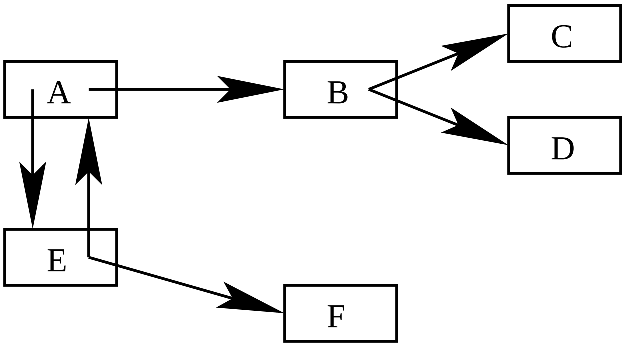
$Connectivity(O_1, O_2)$	$\Pr\{t_{death}(O_1) = t_{death}(O_2)\}$	
Any pair of objects	33.1%	
$pointsTo(O_1, O_2)$	76.4%	
$pointsTo(O_1, O_2) \wedge mutated(O_1)$	61.1%	
$pointsTo(O_1, O_2) \wedge \neg mutated(O_1)$	83.4%	
$\text{Scc}(O_1) = \text{Scc}(O_2)$	72.4%	
$\text{Wcc}(O_1) = \text{Wcc}(O_2)$	46.3%	

⇒ ***Yes for*** $pointsTo(O_1, O_2)$ ***or*** $\textbf{Scc}(O_1) = \textbf{Scc}(O_2)$

⇒ ***Connected objects should be garbage collected together***

reach

- $\mathit{reach}(X) = |\{Y \in \textsc{Gog} \mid Y \rightarrow^* X\}|$
- Number of objects in GOG that reach an object
- E.g. $\mathit{reach}(\mathrm{F}) = |\{\mathrm{A,E,F}\}| = 3$



$\Rightarrow$ ***Rough indication for how "difficult" an object is to collect***

reach

Percentile	25%	50%	75%	95%
Arithmetic mean without *ipsixql*				
Shortlived	1	1	1	2
Truly immortal	42,670	45,471	48,809	83,324
Only *ipsixql*				
Shortlived	1,066,692	1,066,692	1,066,693	1,066,693
Truly immortal	22,864	22,865	22,865	22,865

$\Rightarrow$ ***Shortlived objects tend to have reach*** ≤ 2

$\Rightarrow$ ***With connectivity information, shortlived objects should be easy to collect***

Connectivity-Based GC

- Ongoing work: new GC that expoits connectivity
- Partition objects by connectivity
- High intra-partition connectivity

  ⇒ ***Key object opportunism***
- Low inter-partition connectivity

  ⇒ ***Write barrier removal***

Related work

- Regions
- Escape analysis
- Fitzgerald and Tarditi, *The case for profile-directed selection of garbage collectors*, ISMM 2000
- Dieckmann and Hölzle, *A study of allocation behavior of the SPECjvm98 Java benchmarks*, ECOOP 1999
- Shuf et al., *Characterizing the memory behavior of Java workloads*, SIGMETRICS 2001

Conclusions

- Objects pointed to only from the stack are often shortlived, objects reachable from globals are often immortal

  $\Rightarrow$ ***Roots-connectivity is correlated with lifetime***

- Connected objects tend to have the same deathtime

  $\Rightarrow$ ***Connected objects should be garbage collected together***

- Shortlived objects tend to be reached by few objects in the GOG

  $\Rightarrow$ ***May be easy to collect with connectivity information***

- We are currently implementing a CBGC that does opportunistic partial GC without write barriers