University of Glasgow | School of Computing Science

ForgetMeNot

Jeremy.Singer@glasgow.ac.uk

University of Glasgow | School of Computing Science

ForgetMeNot

a new memory management scheme

University of Glasgow | School of Computing Science

ForgetMeNot

a new memory *mis* management scheme


1.Typical Application Behaviour
2.Memory Hierarchy
3.Crazy Ideas

Typical Java application behaviour

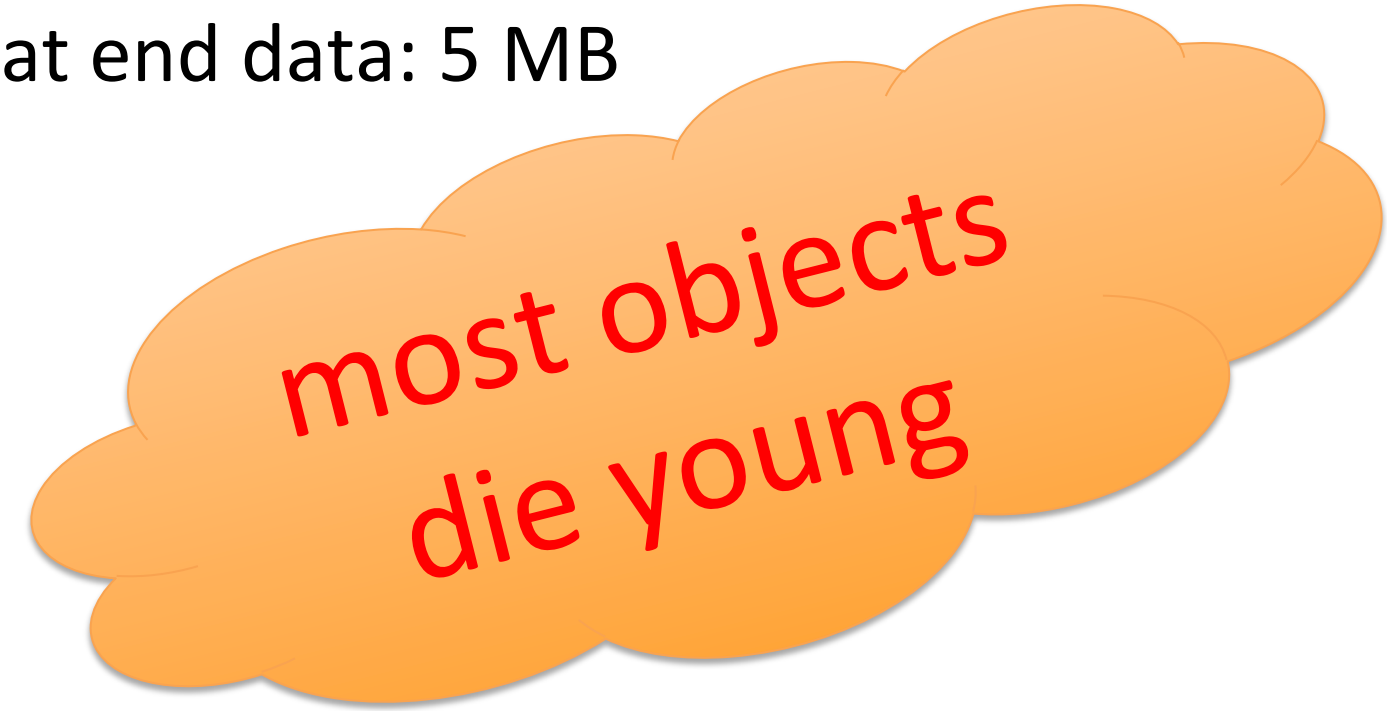
- xalan from DaCapo benchmark
- XML processing application
- run in 1GB heap with large input data

GC log output

```
0.422: [GC 30813K->3748K(992256K), 0.0034510 secs]
0.426: [Full GC 3748K->3502K(992256K), 0.0198550 secs]
1.795: [GC 196014K->6473K(992256K), 0.0036810 secs]
2.414: [GC 198985K->6047K(992256K), 0.0021170 secs]
2.841: [GC 198559K->5727K(992256K), 0.0023560 secs]
3.205: [GC 198239K->5759K(992256K), 0.0022000 secs]
3.452: [GC 198271K->5807K(1019904K), 0.0020840 secs]
3.746: [GC 225967K->6191K(992256K), 0.0029830 secs]
3.968: [GC 226351K->6139K(1019904K), 0.0033360 secs]
4.238: [GC 253947K->5867K(1019904K), 0.0019180 secs]
4.454: [GC 253675K->6011K(1019904K), 0.0017530 secs]
4.619: [GC 253819K->6107K(1019904K), 0.0015970 secs]
4.801: [GC 253915K->6091K(1019904K), 0.0015730 secs]
4.953: [GC 253899K->6051K(1019904K), 0.0014930 secs]
5.132: [GC 253859K->6235K(1019904K), 0.0014950 secs]
5.322: [GC 254043K->6347K(1019904K), 0.0017450 secs]
```

Summary of run

- Allocated data that died: 47 GB
- Old-gen data that died: 8 MB
- Alive at end data: 5 MB



GC causes overhead

- stop the world – pauses

http://gun.io/blog/how-to-hire-android-developers/

Android is plagued by what we developers colloquially call "jank." When an iPhone app scrolls smoothly and the Android counterpart stutters, that's jank. When an Android app overrides the back button inappropriately, has an ugly title bar, or crashes unexpectedly and unrepeatability, that's jank. Jank sucks.

Bring up this concept with your developer. They'll know what you're talking about, and if they're up to snuff, they'll immediately start getting defensive about their beloved platform and start offering their own anti-jank patterns.

They should know about how to properly use database-backed ScrollViews, reusable ViewHolders, and how to only redraw the parts of the screen that have changed. Ideally, they'll have seen the Google I/O talks talks on how to avoid jank,

GC causes overhead

- stop the world activities
- eat up parallel threads
- barriers in application code
- space in object headers

1. Typical Application Behaviour
2. **Memory Hierarchy**
3. Crazy Ideas

Broad Spectrum of Storage

Memory type	cost / GB	latency
L1 cache	£ 5760	1 ns
DRAM	£ 10	50 ns
HDD	£ 0.05	4 ms
Google Drive	£ 0.05 per month	2.5 ms + …
Amazon Glacier Store	£ 0.01 per month	5 hours

1. Typical Application Behaviour
2. Memory Hierarchy
3. **Crazy Ideas**

Cache instead of Collect

- locate infrequently accessed data further from compute elements
  - handles for objects, rather than direct pointers
  - prefetching
  - predicting access frequency

Overwrite instead of Collect

- [Rinard, 2007]

[Rinard, 2007]

Detecting and Eliminating Memory Leaks Using Cyclic Memory Allocation

Huu Hai Nguyen and Martin Rinard

Department of Electrical Engineering and Computer Science
Computer Science and Artificial Intelligence Laboratory
Singapore-MIT Alliance
Massachusetts Institute of Technology
Cambridge, MA 021139
nguyen@nus.edu.sg,rinard@csail.mit.edu

Abstract

We present and evaluate a new technique for detecting and eliminating memory leaks in programs with dynamic memory allocation. This technique observes the execution of the program on a sequence of training inputs to find m-bounded allocation sites, which have the property that at any time

to overlay live objects in memory. Our results indicate th
our bounds estimation technique is quite accurate in practic
providing incorrect results for only two of the 152 suitab
m-bounded sites that it identifies. To evaluate the potent
impact of overlaying live objects, we artificially reduce t
bounds at m-bounded sites and observe the resulting b
havior. The resulting overlaying of live objects often do

Overwrite instead of Collect

- [Rinard, 2007]
- bounded circular allocation buffers
- store at most n objects at each alloc site
- run pine fairly successfully
- dealing with crashes
  - restart app
  - checkpointing / runtime fault tolerance
  - try/catch, weak pointers / programmer fault tolerance

Reachability approximates liveness

- GC uses conservative approximation for liveness
- If I can access an object, will I?
- empirical study
- can we predict object death
  - are there better proxies than reachability?

Conclusions

- Lots of changes in system memory configurations and costs
- let's rethink the way we do memory management
- Three ideas here
  - cache, don't collect
  - overwrite, don't collect
  - speculative collection