Garbage Collection: Simulation? Modeling?

Peter Libič
Department of Distributed and Dependable Systems
http://d3s.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
Last Time (March 2011)

- Motivation and use cases
- Lifetime tracing
- Results
  - Simple model
  - Generational model
Recall: Throughput Collector in HotSpot VM

- Generational collector
- Young generation
  - 3 spaces
  - Eden + 2 Survivors
- Allocations in Eden
- Young GC collects all young spaces
- Full GC empties young generation
- Card table records references from old to young
Last Time

- I asked if the imprecision is caused by:
  - Fragmentation?
  - Pre-tenuring?
  - Early GC triggering?
  - One of hundreds of other optimizations?

- And
  - What to do with that?
After Several Months

I asked if the imprecision is caused by:

- Fragmentation? (crossed out)
- Pretenuring? (crossed out)
- Early GC triggering? (crossed out)
- One of hundreds of other optimizations?

And

- What to do with that?
Today

• A road to modeling the GC
  • I would like to hear your opinion on my approach
• Report on experiments
  • Early GC triggering
  • Tracing path of object in the heap
    - Copies and promotions
• Collection order
• Latest ideas
GC Modeling: Approach

- Dependency identification
- Simulation
- Single Component Model
- Input / Model Composition
GC Modeling: Approach

- First two steps:
  - Identification of inputs to the simulator
  - Simulation of GC algorithm with all the needed data available
GC Modeling: Approach

- From simulation to a model
  - Simplify the algorithm
  - Simplify the data
    - e.g. by using statistical distributions instead of exact data
GC Modeling: Approach

- From a single component to composition
  - Composition of inputs or more sophisticated model?
  - Possibility to find another fundamental input
GC Modeling: But..

- Dependency identification
- Simulation
- Single Component Model
- Input / Model Composition

Still here...
Experiments

- Early GC triggering?
  - Example of an experiment
  - What if collector triggers full collections sooner than at the moment when the heap is full?

- Quick look into GC log:

  [Full GC [PSYoungGen: 2016K->0K(14336K)] [PSOldGen: 116106K->80938K(122880K)] 118123K->80938K
   (137216K) [PSPermGen: 16737K->16737K(34048K)], 0.2450910 secs] [Times: user=0.24 sys=0.00, real=0.25 secs]

- Thinking of possible causes:
  - Unreported object overhead
  - Safety margin (survivor space size reservation)
    - Differences are suspiciously close to that size
  - Different -> check the impact on simulation
    - Using measured differences to trigger simulated GC earlier
Early GC Triggering - Results

Y axis: old GCs (count)
X axis: heap configuration id
Tracing Object Movements

- Goal is to follow all object movements in the heap
  - Copying between the young generation spaces
  - Promotion from young to old generation
  - Allows to see detailed behavior of collector
  - Implementation uses JVMTI agent + small and dirty JVM hacks
  - Huge output

CTS for tag 654045, age: 1, size: 3; YG_cas:0, YG_flush:0, YG_lab:0, YG_FULL:0; OG:0, OG_nfull:0, OG_cas:0, OG_flush:0, OG_lab:0, OG_FULL:0
CTS for tag 654039, age: 1, size: 4; YG_cas:0, YG_flush:0, YG_lab:0, YG_FULL:0; OG:0, OG_nfull:0, OG_cas:0, OG_flush:0, OG_lab:0, OG_FULL:0
CTS for tag 654036, age: 1, size: 6; YG_cas:0, YG_flush:1, YG_lab:0, YG_FULL:1; OG:1, OG_nfull:1, OG_cas:0, OG_flush:1, OG_lab:1, OG_FULL:0
CTS for tag 654037, age: 1, size: 3; YG_cas:0, YG_flush:0, YG_lab:0, YG_FULL:0; OG:1, OG_nfull:0, OG_cas:0, OG_flush:0, OG_lab:0, OG_FULL:0
Object Movement Statistics

- Summary of important changes in the heap

```
>>>>>>> STATS: Allocs: 18344 of size 17236400; Deallocs: 14999 of size 0
Tenured objects after 0copies: 1209 of size 5729768
Tenured objects after 1copies: 144 of size 15032
Tenured objects after 2copies: 2484 of size 324560
Tenured objects after 3copies: 33852 of size 1607008
Copied to survivor - objects after 0copies: 3507 of size 2158080
Copied to survivor - objects after 1copies: 20 of size 456
Copied to survivor - objects after 2copies: 81 of size 18072
Copied to survivor - objects after 3copies: 12978 of size 547448
Copied to survivor - objects after 4copies: 18 of size 888
Copied to survivor - objects after 5copies: 85 of size 3224
Copied to survivor - objects after 6copies: 7607 of size 368016
```
What Did I Learn?

• Copying/Promotions work as I expected:
  • In young GC
    - Surviving objects are copied into survivor space
    - After several (7) copies the object is promoted
    - If there is no space left in the survivor, all the other objects are promoted
  • In full GC
    - At the end of GC, all surviving objects are in old generation
• No pre-tenuring
Collection Ordering Effects

- Object movement statistics are very different in simulator and in the measurement.

- Maybe the traversal order has a large effect?
  - Objects not fitting in survivor space are promoted.
  - If older objects that die later are visited (+copied) sooner then there will be more garbage in old generation quickly.
Experiments with Collection Ordering

- Different young generation collection orderings in simulation:
  - Random
  - Linear from the oldest to the youngest objects
  - Linear reversed
  - The worst: from object that will die the last to object that will die the first

- Results
  - Just tiny changes (in order of unit of per cents)
Deterministic Traces?

- Is it possible that non-determinism of benchmarks causes bad results because of imprecise lifetime trace?

- Experiment:
  - Own artificial benchmark
    - Single thread
    - Deterministic

- Experiment:
  - Forced synchronization
Artificial Workload - Results

X axis: Heap configuration ID
Artificial Workload - “Selected” Results

X axis: Heap configuration ID
Forced Synchronization

- Synchronization of threads each N allocations
- Almost no change in GC counts
Latest Idea (or Last Hope?)

- Simulator uses lifetimes and sizes of object
- All the results show the real GC has more work to do than the simulator expects
  - More objects in the old generation
  - If the object is garbage during young GC, simulator deletes that object

- What if the object become garbage because the object pointing to it became garbage and it is stored in the old generation?
Dead References

- Old Generation
- Card Table
  - Survivor 1
  - Survivor 2
  - Eden
  - Young Generation
Dead References

Old Generation

Card Table

Survivor 1  Survivor 2

Eden

Young Generation
Dead References

- Old Generation
- Card Table
  - Survivor 1
  - Survivor 2
- Eden
- Young Generation
- Garbage
Dead References

- Until now, simulator uses:
  - Lifetimes of all objects
  - Sizes of all objects
- Now, record of all references pointing from older to younger objects at all times needed
  - Really BIG
- Data measurement and simulator ready (+-bugs)
- First results expected on Thursday
Chaotic System?

- Small changes cause big effects
- Can removing several references (from otherwise dead objects) improve GC performance?
  - If 'Dead references' experiment shows simulation improvement then the answer is yes
  - Should programmers delete references that are not needed anymore?
Conclusion

• Measurements, experiments, ...

• Many results are still imprecise

• Do we need basically the complete application to be able to simulate its GC behavior?

• We should introduce delete/free() into Java!
Thank You for Your Attention

Questions?