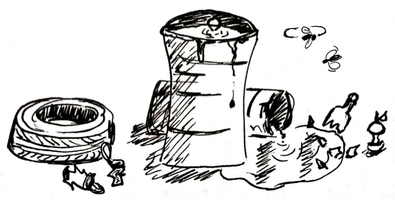
Issues of Java Garbage Collection



University of New Brunswick - Faculty of Computer Science

Markus C. Goffart
markus.goffart@unb.ca

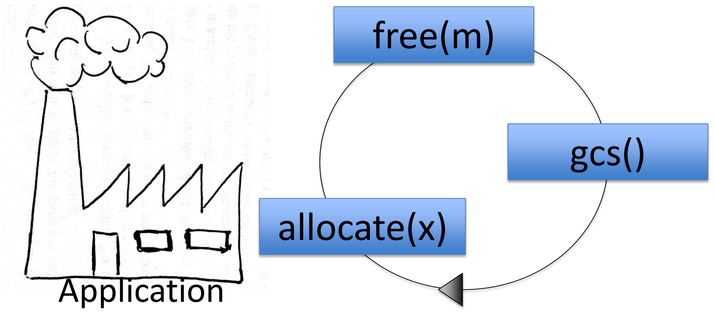
Gerhard W. Dueck
dueck@unb.ca

Motivation

Get an understanding of different Garbage Collector (GC) types and their parameters to improve the throughput of the application with the help of an own Garbage Collection Simulator.

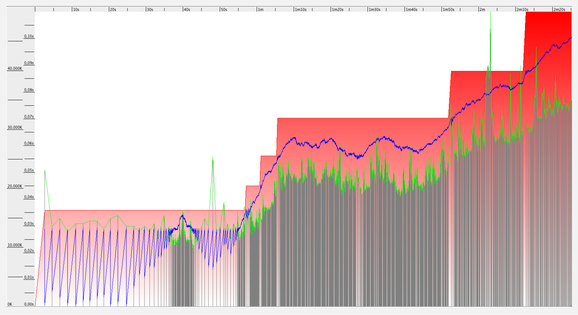
Methology

- Create an Application:
  - Allocates objects - *allocate(x)*
  - Set delete flag on objects – *free(m)*
  - Clean with own Garbage Collection Simulator - *gcs()*
- Test and compare with different GCs of Java VM



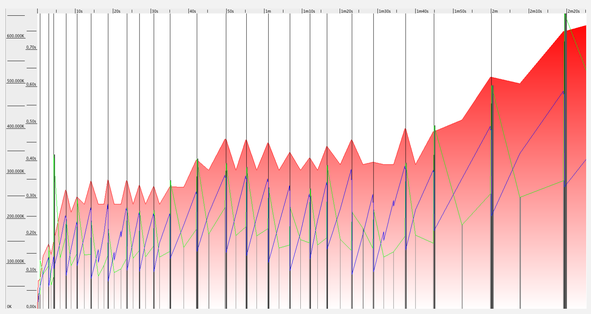
- One total heap
- Only theoretical
- Freed objects with DeleteFlag

- Too high frequency
- No realistic values in amplitude
- Strong overhead if more than 80% of heap

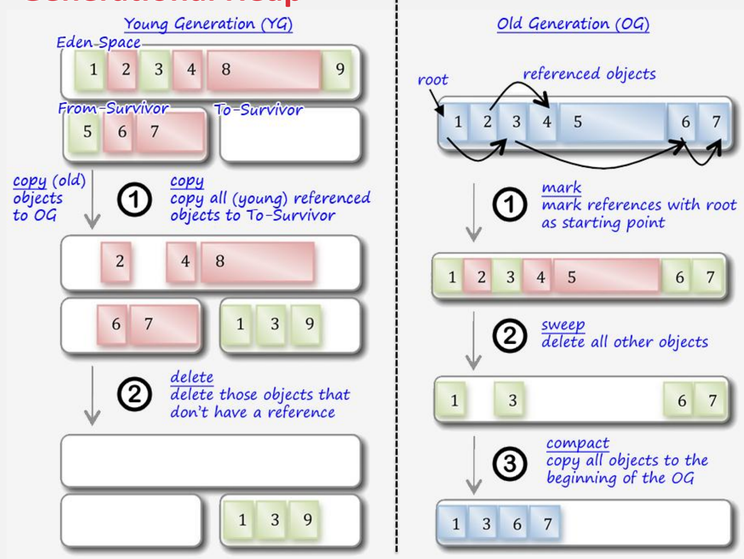


- YG: Serial
- OG: Mark&Sweep-algorithm
- Stop-The-World-Phase (STW)

- Several YG-Collections before Full Collection appears
- High Frequency → many pauses → high overhead
- ~0.8s pauses

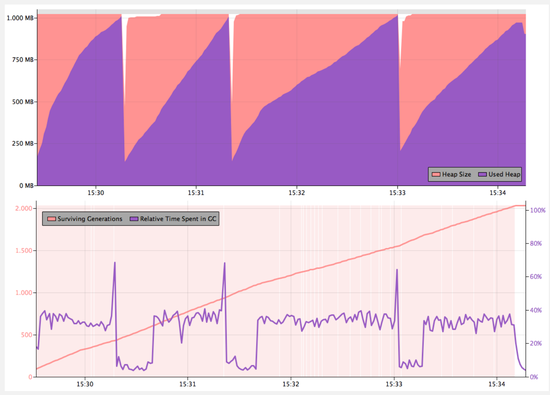


Generational Heap



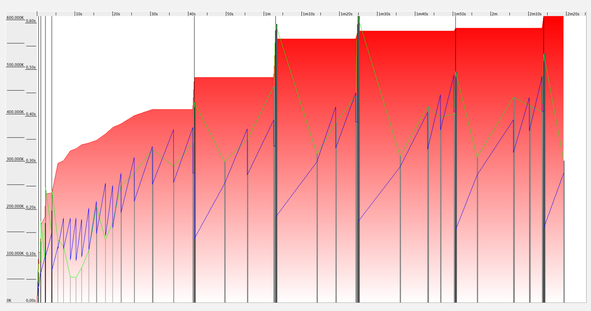
- Complete heap divided in sub regions of same size
- Card Table → Remembered set (→ trash density)
- Regions with most **garbage** will be cleaned **first**
- Regions can be cleaned in parallel (→ fast)

- In general overhead of 30%
- Complete heap in use (cause of regions)
- Complete cleaning causes 60% overhead



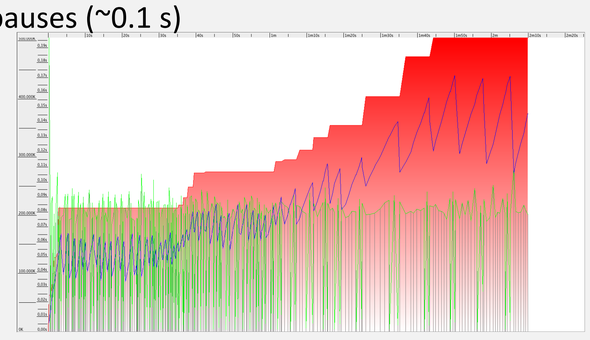
- Like Serial M&S Collector
- BUT: YG on Parallel Hardware
- → faster in YG

- Less pauses → less overhead
- ~0.6 s pauses



- Changes of M&S-Alg. in OG
- Short STW phases
- 2 phases are concurrent

- Concurrent marking (only STW to detect roots and remark)
- Low pauses (~0.1 s)



Conlusion

- Choose GC regarding field of operation
- Increasing maximum heap size will not solve problem but defer it
- No well chosen parameters for own implementation (too high GC frequency + overhead)
- Testing new implementations with different GC parameters can increase throughput