Age-Based Garbage Collection

Stefanovic, McKinley and Moss

Older-first Garbage Collection in Practice: Evaluation in a Java Virtual Machine

Stefanovic, Hertz, Blackburn, McKinley and Moss

Presented by Jin Yu

Mar 20, 2012

- [Paper 1]: Age-Based Garbage Collection
  - New GC algorithm: Older-First algorithm (Age-Based)
  - Comparison of several garbage collectors by simulation
- [Paper 2]: Older-First Garbage Collection in Practice: Evaluation in a Java Virtual Machine
  - Implementation of Older-First algorithm
  - Comparison of several garbage collectors

Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Introduction

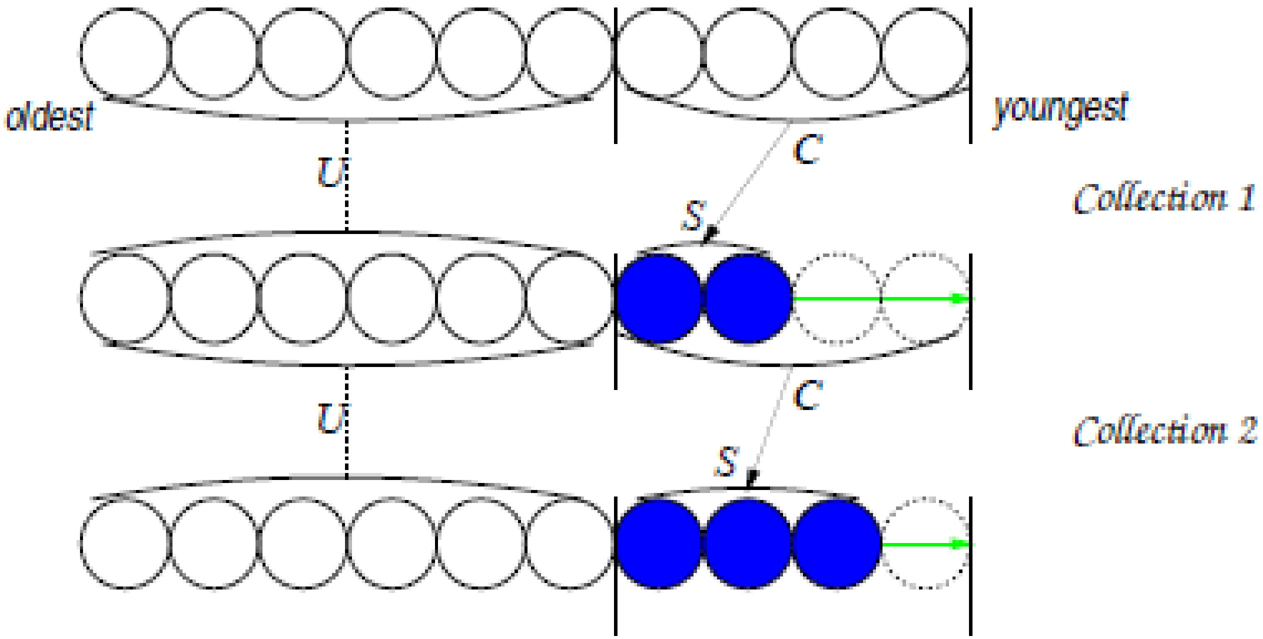
- Generational Collector (*traditional*)
  - Younger generations are frequently examined
  - *What if objects do not have enough time to die? (e.g., the very youngest objects)*
  - *Copying cost!*
- Older-First Collector (*proposed in [paper 1]*)
  - Older generations are frequently examined
  - *Lower copying cost, but higher pointer tracking cost!*
- If total cost is *copying* + *pointer tracking*, Older-First performs better!

Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Age-Based Garbage Collection

- Youngest-Only Collection (YO)



C : collected region

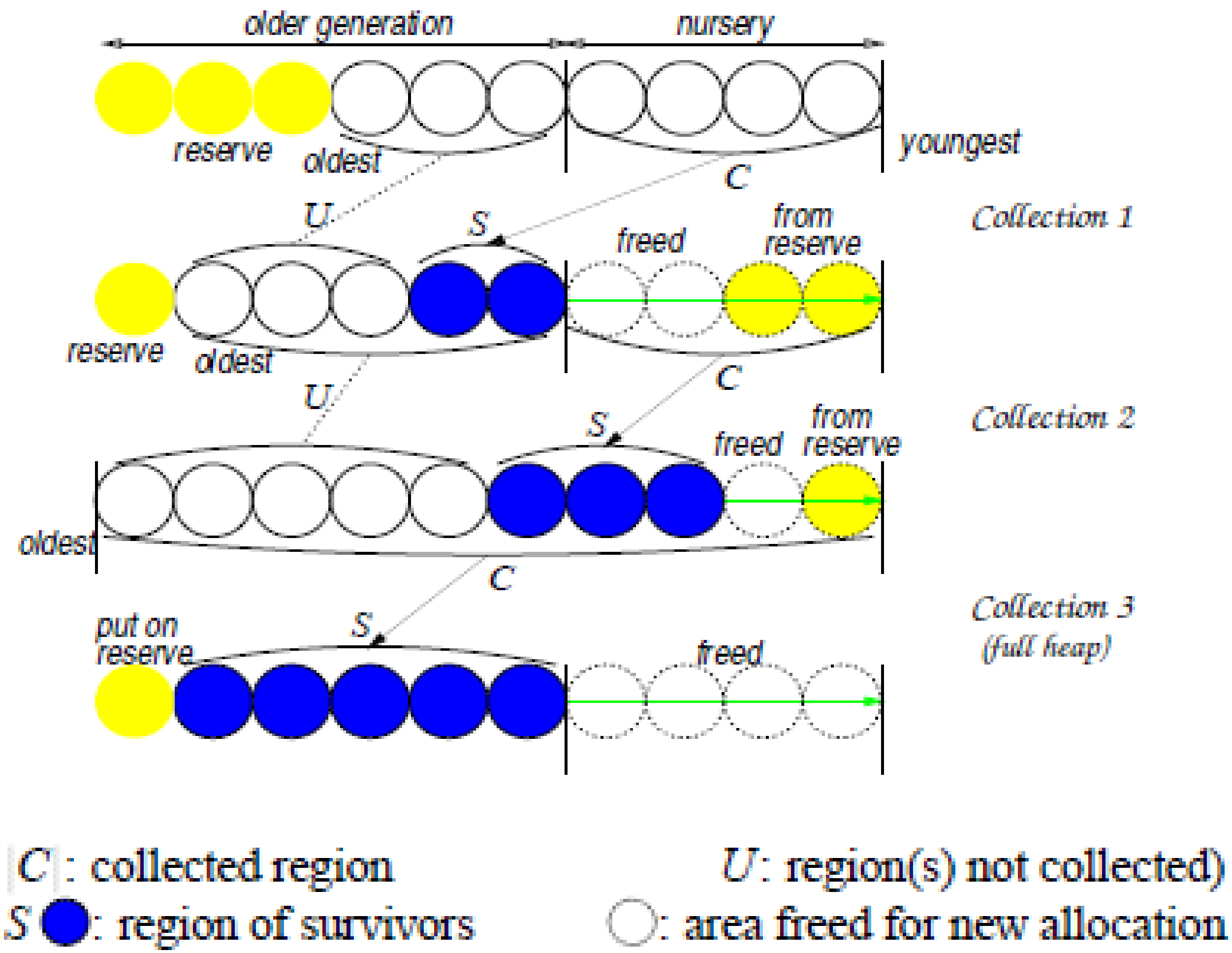
U: region(s) not collected)

S ●: region of survivors

○: area freed for new allocation

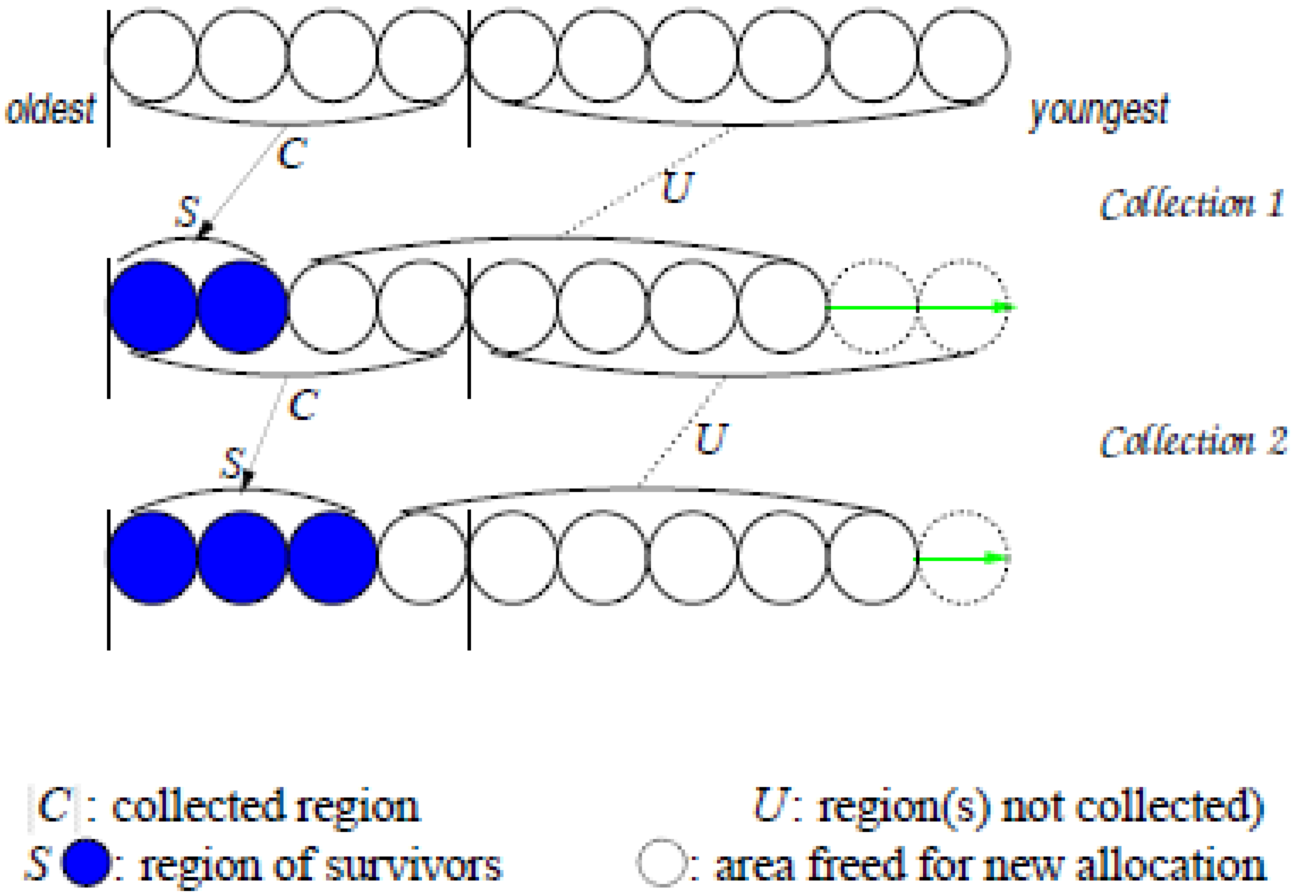
Age-Based Garbage Collection

- Generational (Youngest-Only) Collection



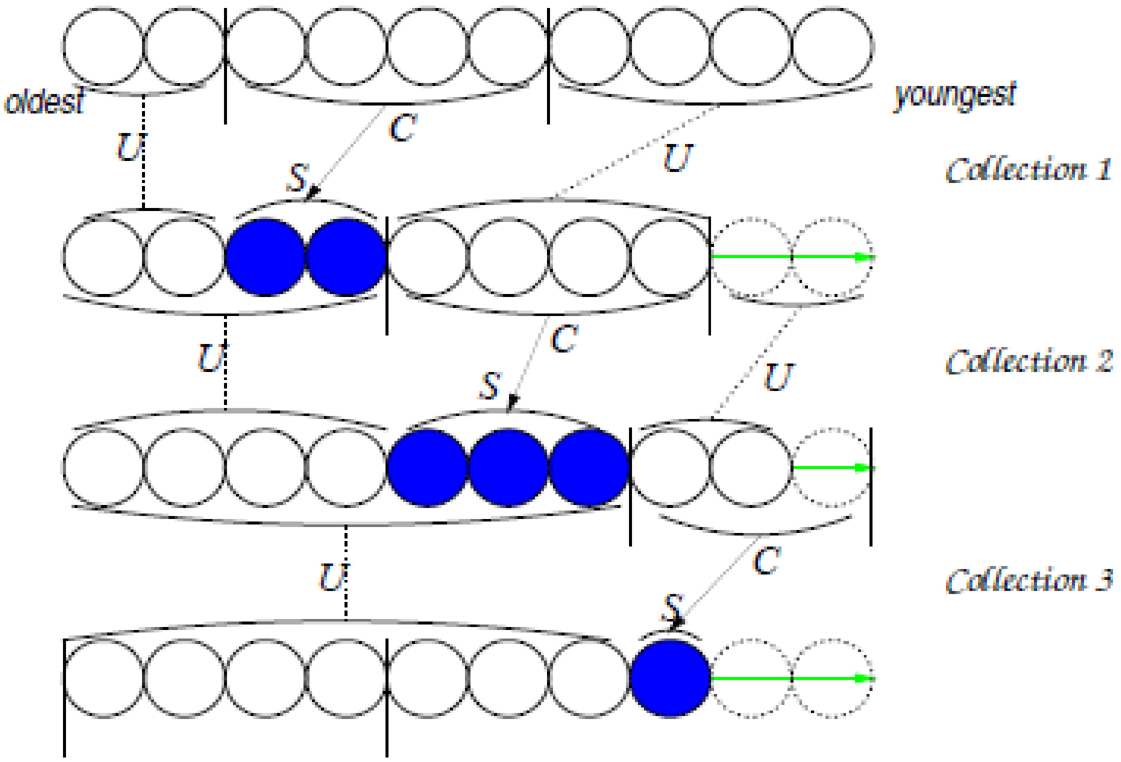
Age-Based Garbage Collection

- Oldest-Only Collection (OO)



Age-Based Garbage Collection

- Older-First Collection (OF)



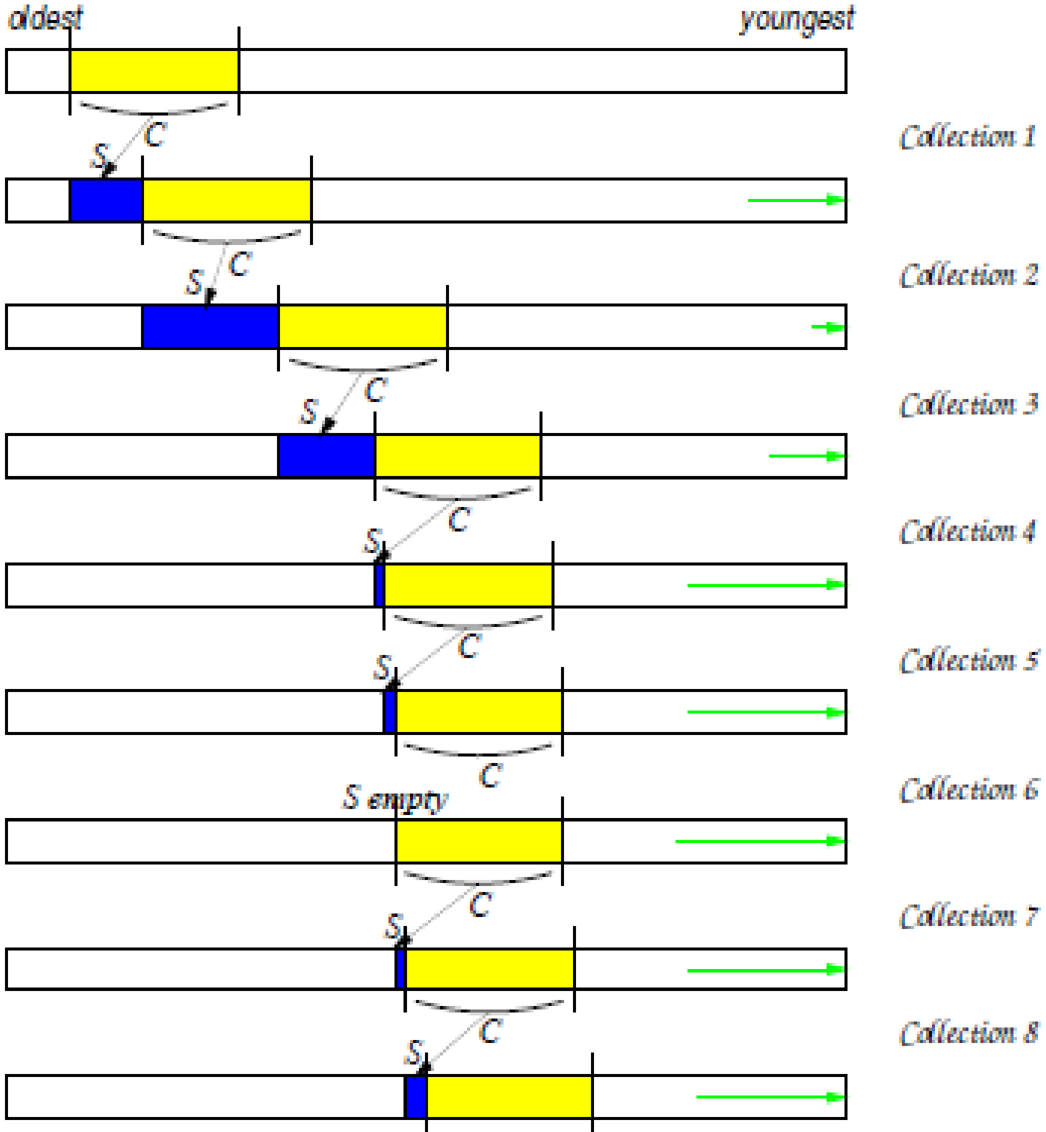
C : collected region

U: region(s) not collected)

S ●: region of survivors

○: area freed for new allocation

Older-First Window Motion Example



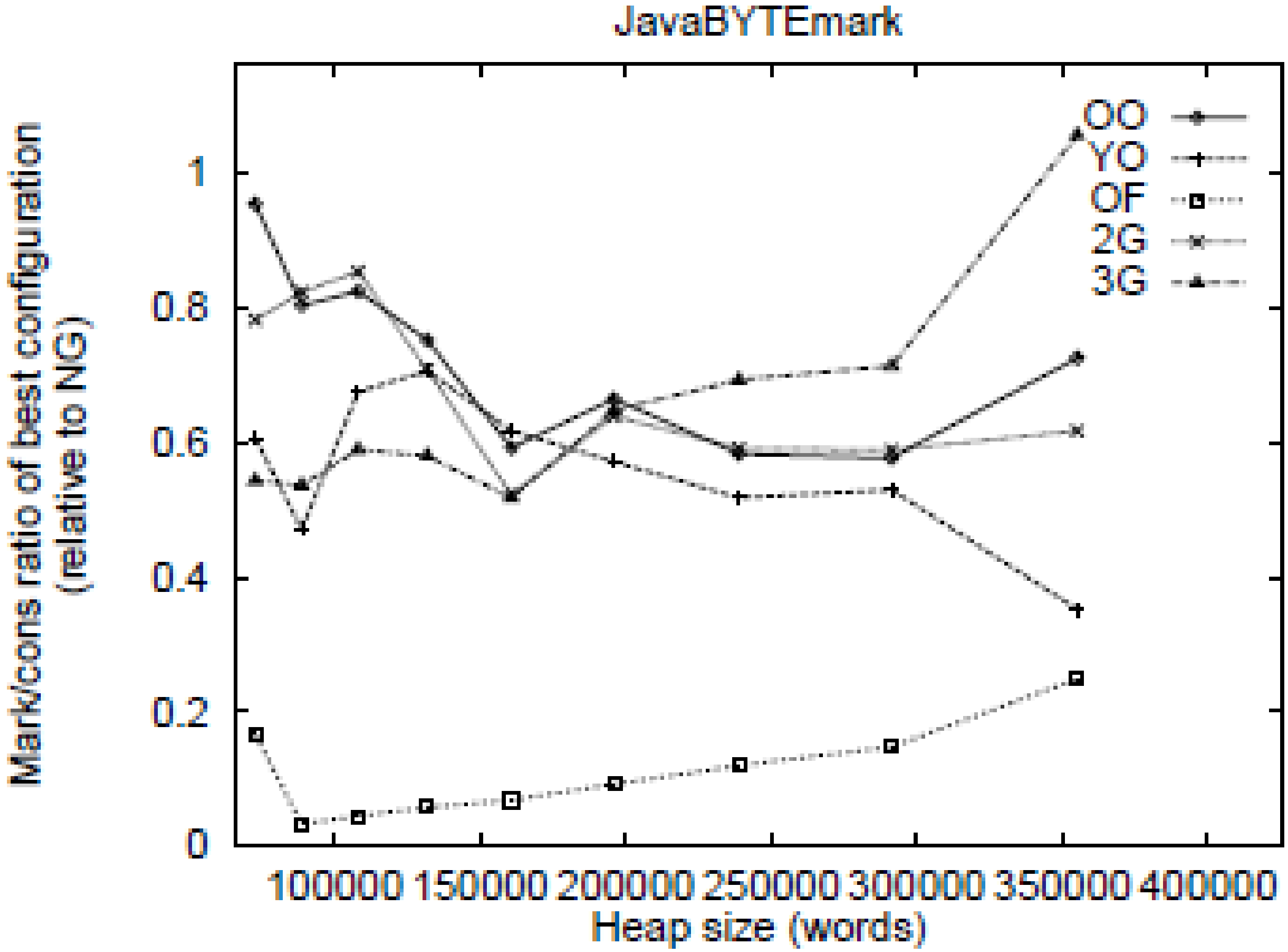
Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Benchmarks in [Paper 1]

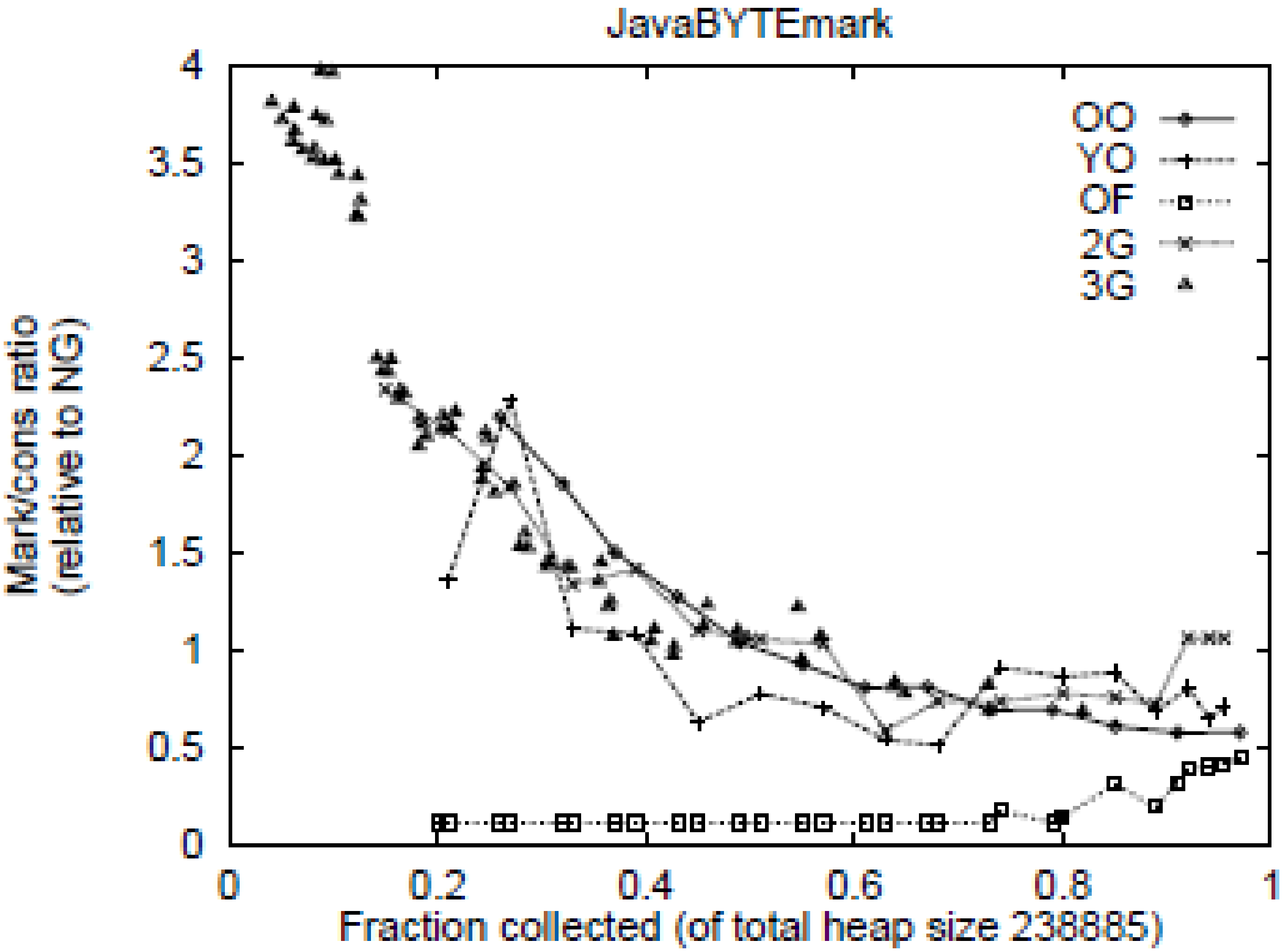
- Refer to Table 1 for benchmark properties
- Based on Object-Oriented languages
- Java
  - JavaBYTEmark, Bloat-Bloat, and Toba
- Smalltalk
  - StandardNonInteractive, HeapSim, Lambda-Fact5, Lambda-Fact6, Swim, Tomcatv, Tree-Replace-Binary, Tree-Replace-Random, and Richards

Estimating Copying Costs



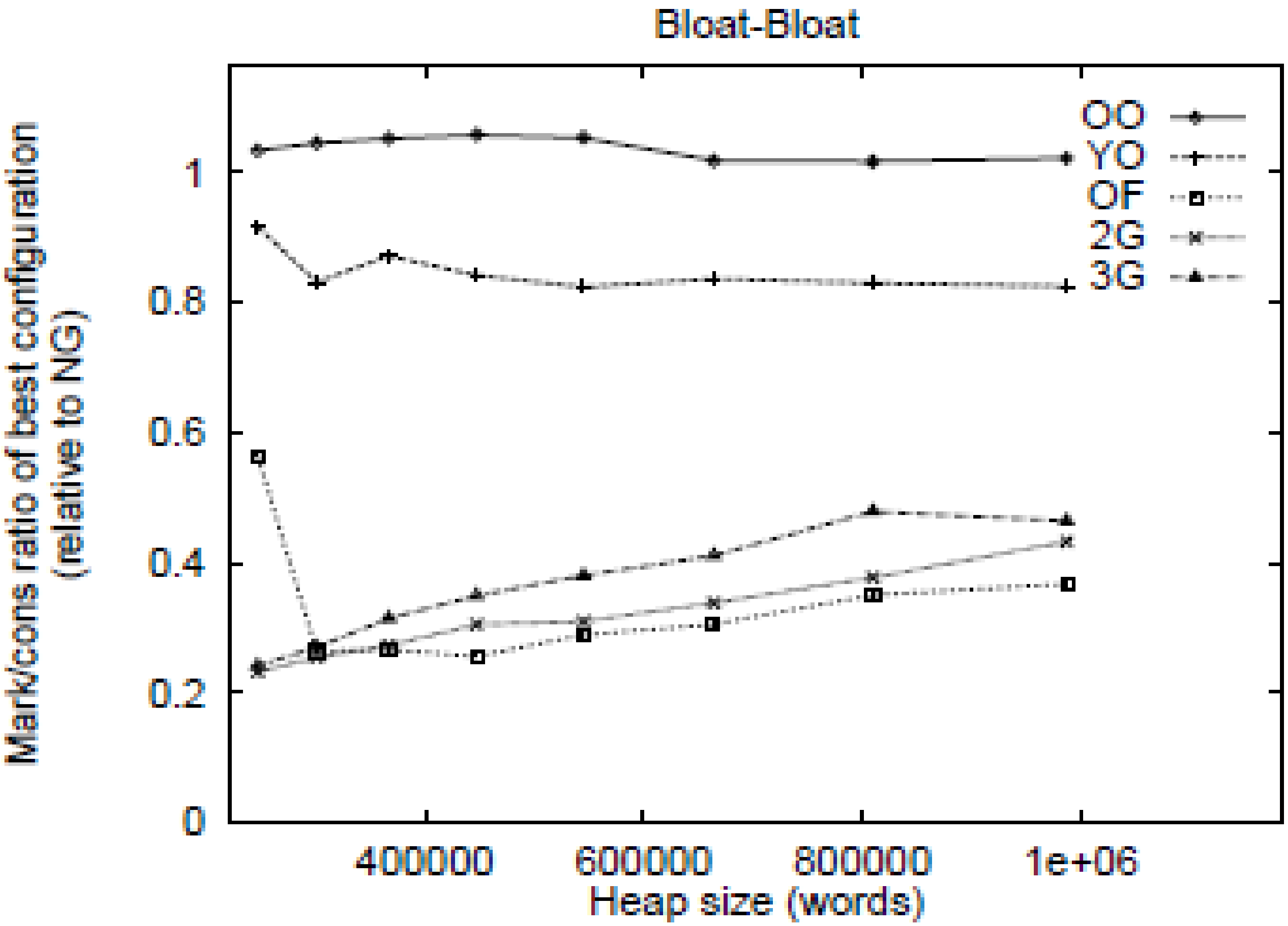
(a) Best configuration.

Estimating Copying Costs



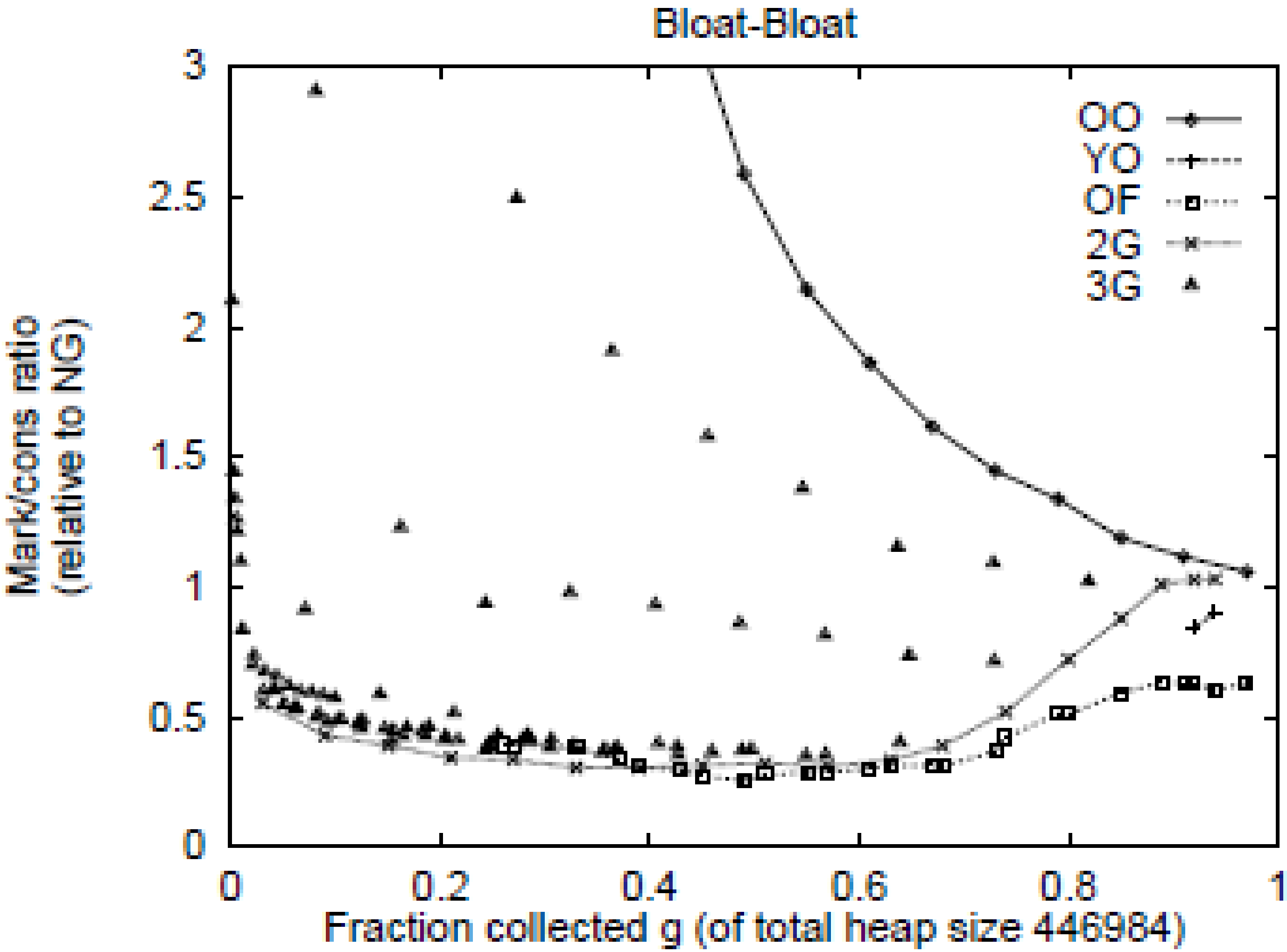
(b) Representative heap size.

Estimating Copying Costs



(a) Best configuration.

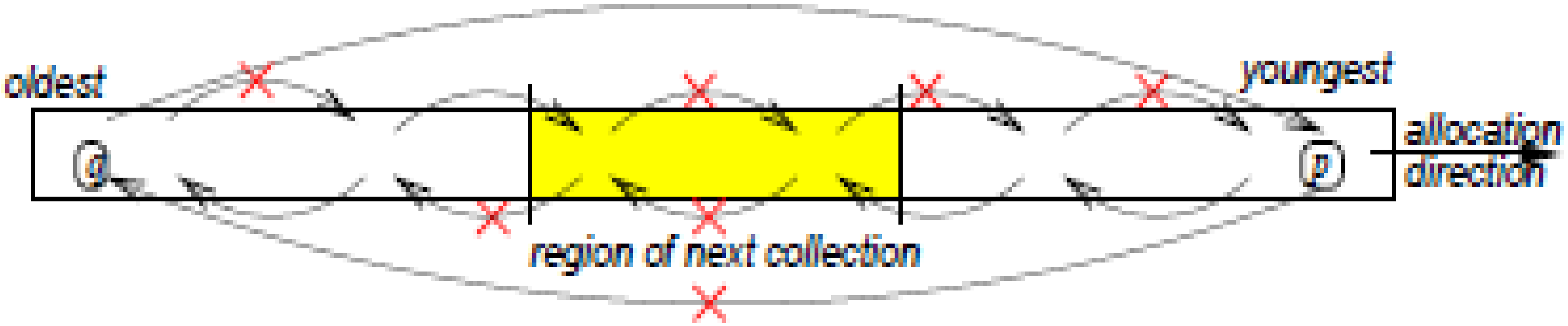
Estimating Copying Costs



(b) Representative heap size.

Write Barrier

- Rule: just remember a cross-block pointer whose target will fall into the collected region earlier than its source
- Directional filtering of pointer stores

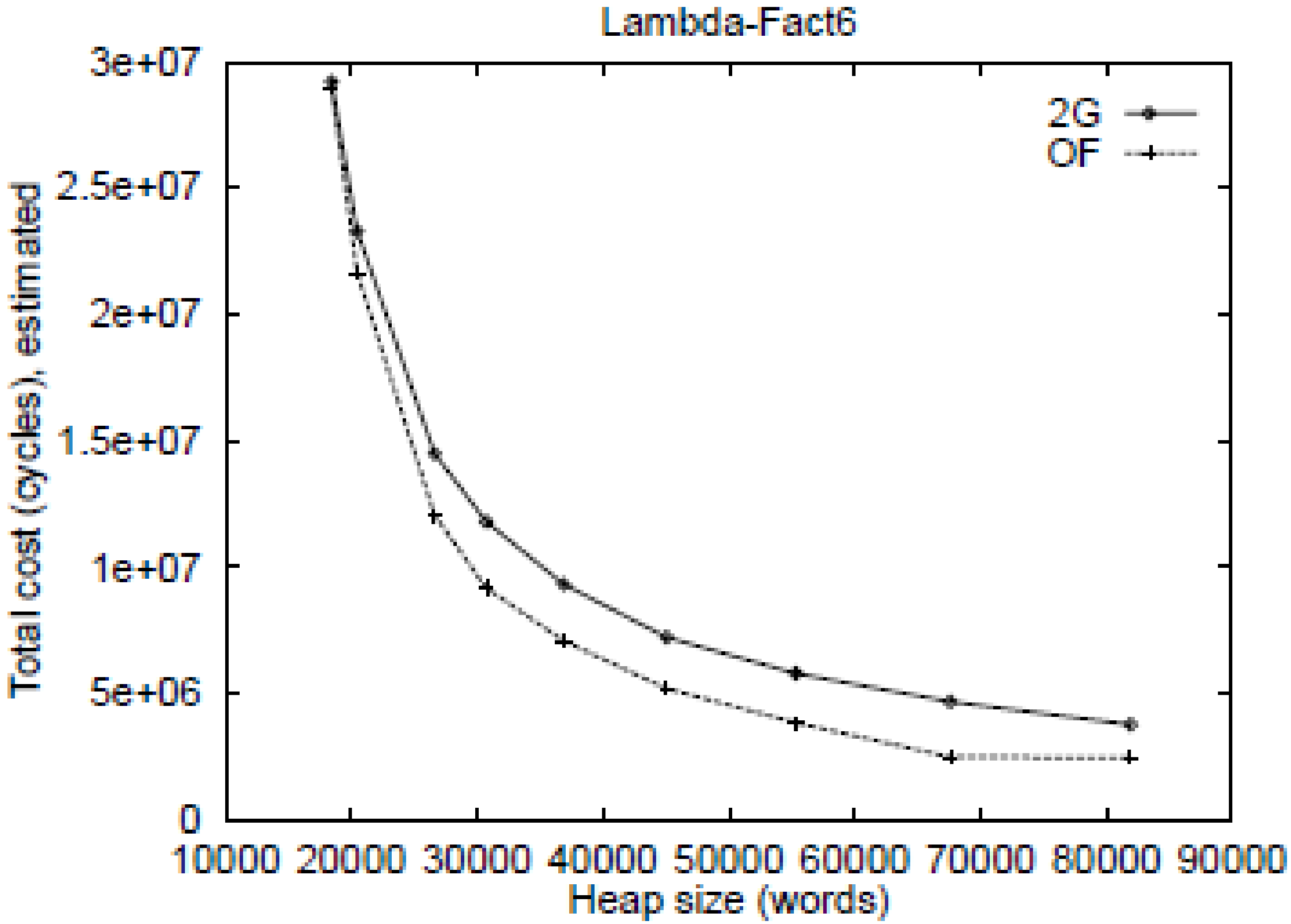


Estimating Total Costs: OF vs. 2G

- Total cost = Copying cost + Pointer-tracking cost
- OF outperforms on some benchmarks
  - JavaBYTEmark, StandardNonInteractive, HeapSim, Lambda-Fact5, Lambda-Fact6, and Richards
- OF and 2G have similar performance on the remaining benchmarks

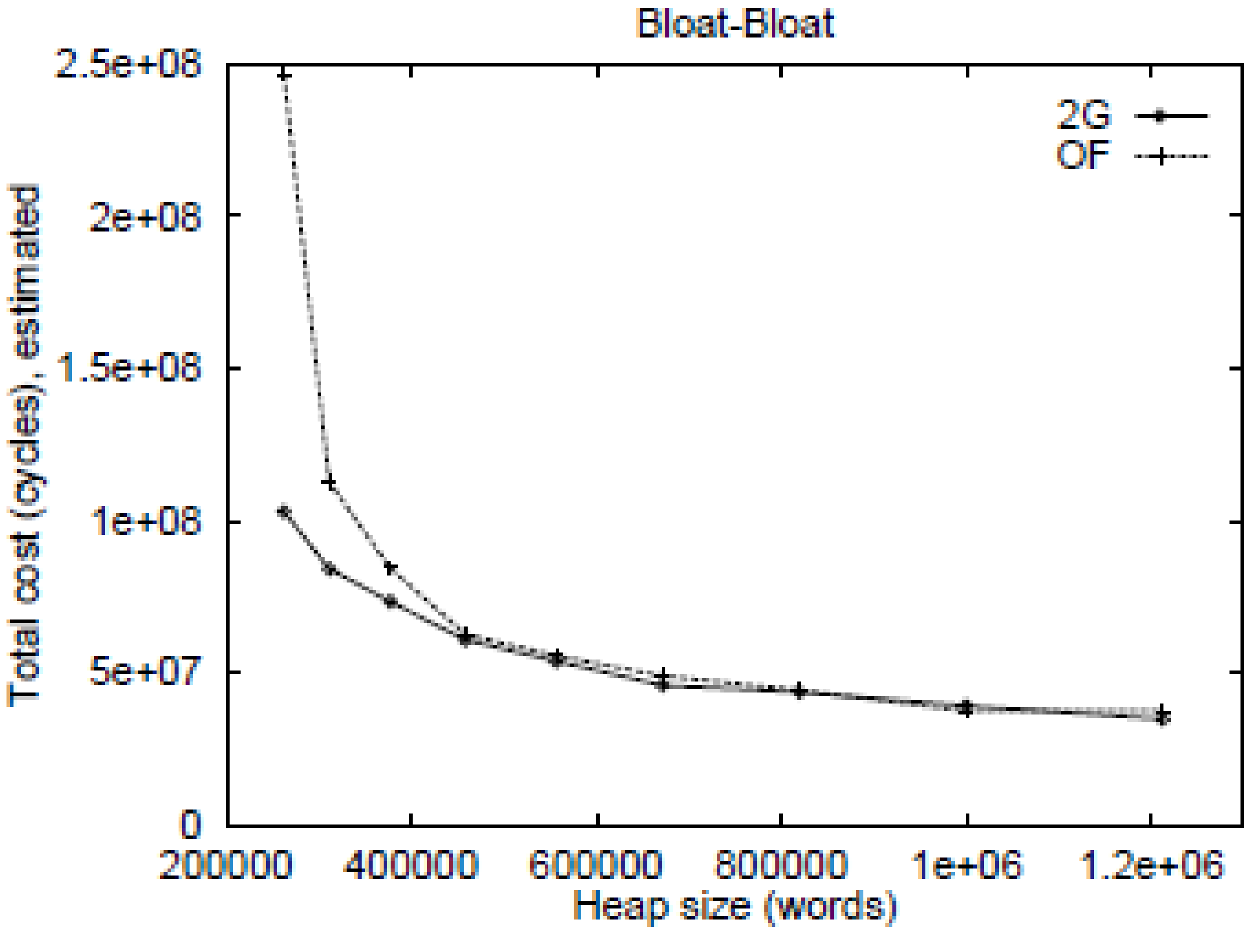
Example of Estimating Results

- Total collection cost: Lambda-Fact6



Example of Estimating Results

- Total collection cost: Bloat-Bloat



Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Discussion of [Paper 1]

- Comparing Collectors
  - OF achieves lower total costs than 2G in many cases
- Pointer Tracking
  - OF gets higher cost here, but not excessive
- Caching and memory effects
  - OF visits the entire heap more regularly
    - *Locality in cache?*
    - *Paging?*

Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Problem Statement of [Paper 2]

- Validate the simulation model in [paper 1]
- Compare execution times and copying ratios of OF and Generational collectors
- Explore pause times and the total collection time tradeoff

Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Design and Implementation of the OF Collector

- Frame: maximum object size, minimum unit of collection
- TOD (Time-of-Death): representing the age for each frame, indicating the frame’s position in the larger *logical* address
- Write Barrier: just remember a cross-block pointer whose target block’s frame has a smaller TOD value than its source block’s frame

Experimental Method

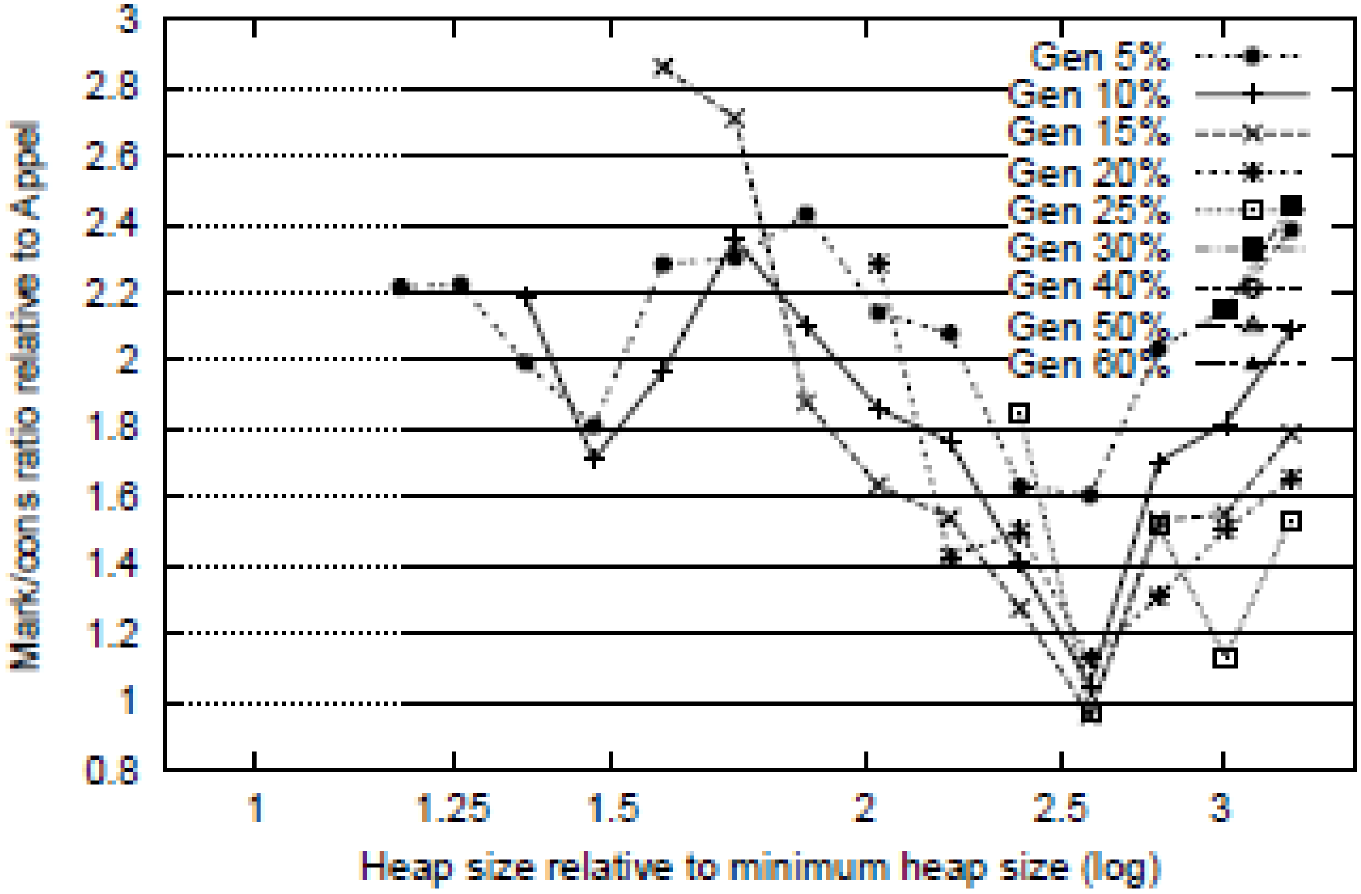
- Collector Families
  - Generational
    - *Appel*: 2G collector with variable nursery size
    - *Gen* (2G) and *OF* collectors with fixed window size
  - Non-generational
    - Semi-space (*SS*)
- Environment
  - Jikes RVM 2.0.3, Macintosh PowerMac G4 (PowerPC), 32KB L1 cache, 256KB L2 cache, 640MB memory, Yellow Dog Linux2.1 (kernel 2.4.10)
- Metrics: mark/cons ratio, execution time

Benchmarks in [Paper 2]

- Refer to Table 1 for benchmark properties
- Ten Benchmarks:
  - SPEC_201_compress, SPEC_202_jess, SPEC_205_raytrace, SPEC_209_db, SPEC_213_javac, SPEC_222_mpegaudio, SPEC_228_mtrt, SPEC_228_jack, pseudojbb, pseudojBYTEmark

Experiment Results for *pseudojbb*

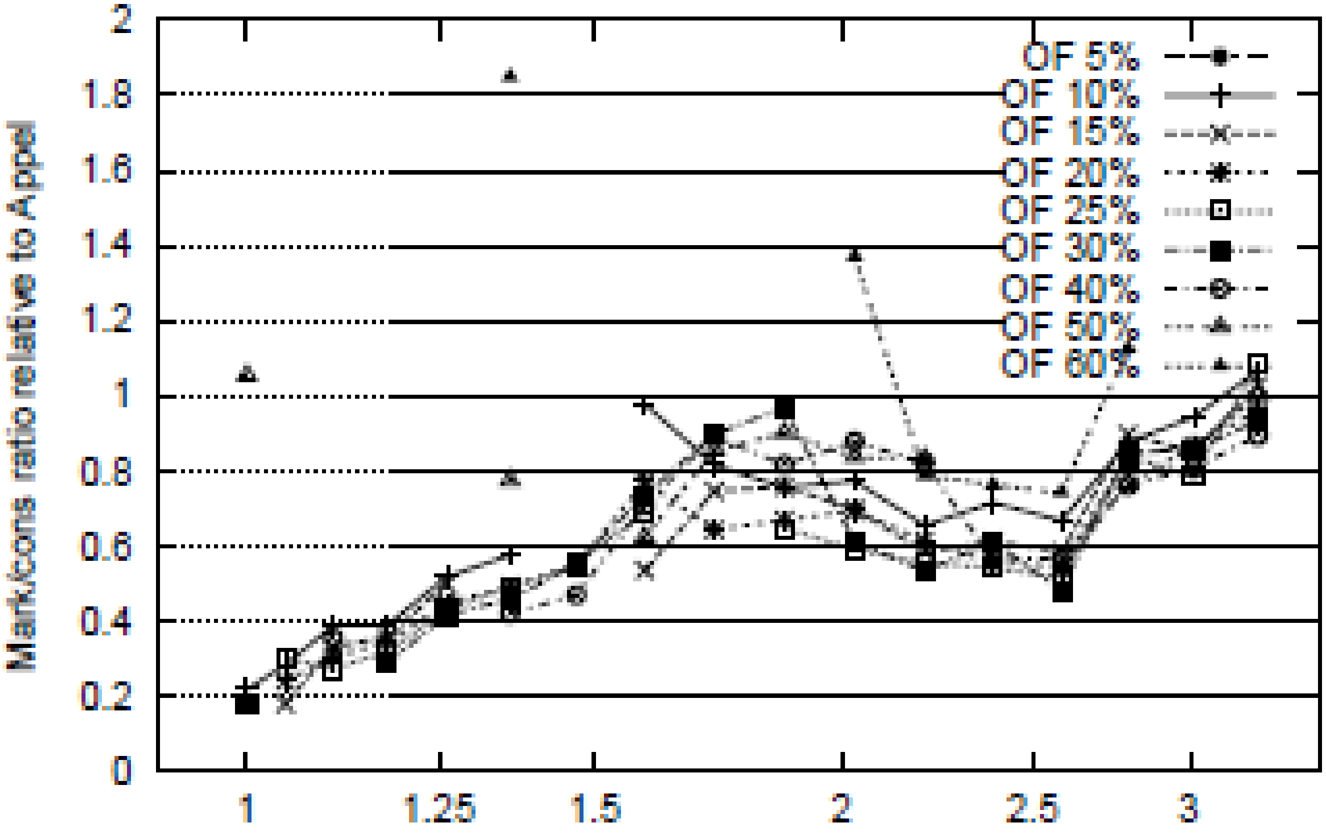
- Mark/cons ratio



(a) Generational collector

Experiment Results for *pseudojbb*

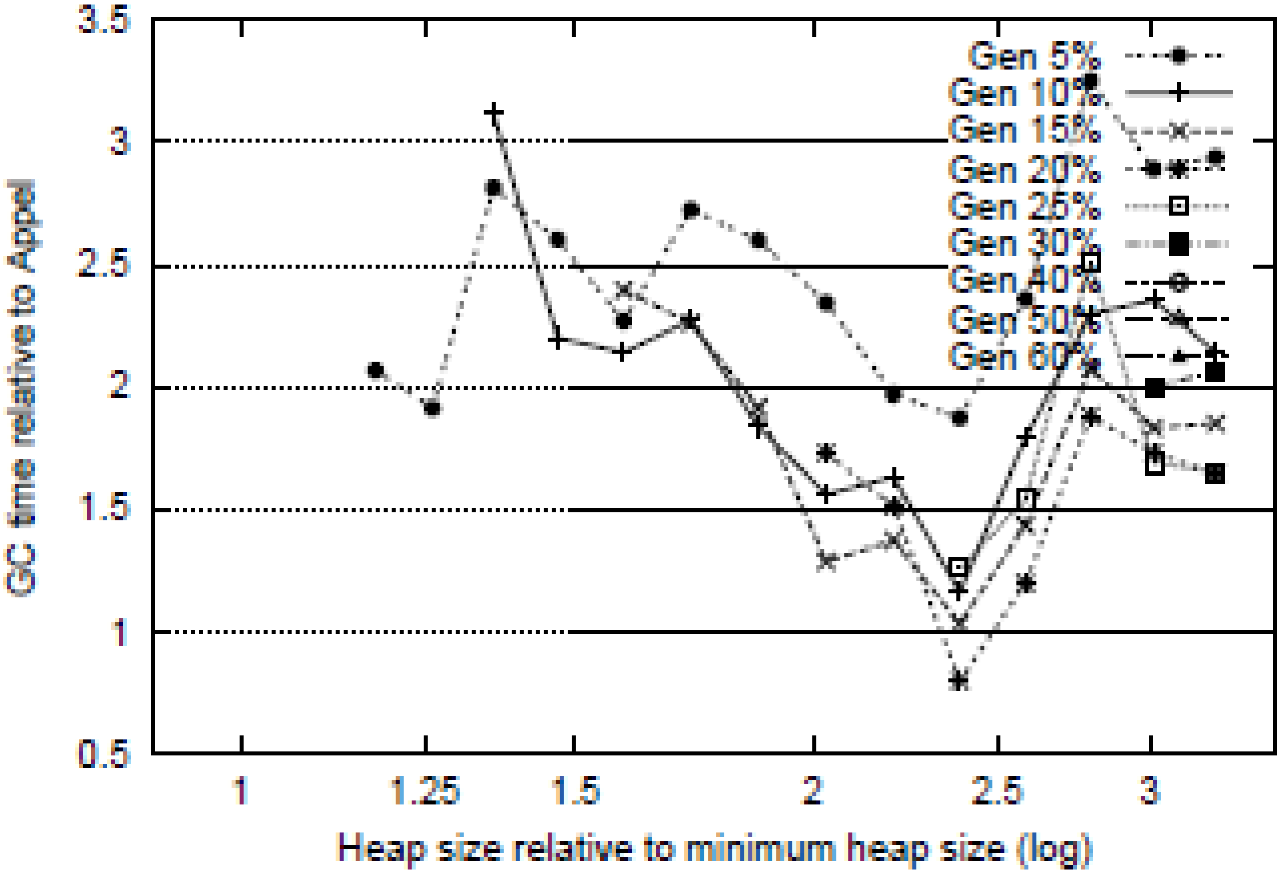
- Mark/cons ratio



(b) Older-First collector

Experiment Results for *pseudojbb*

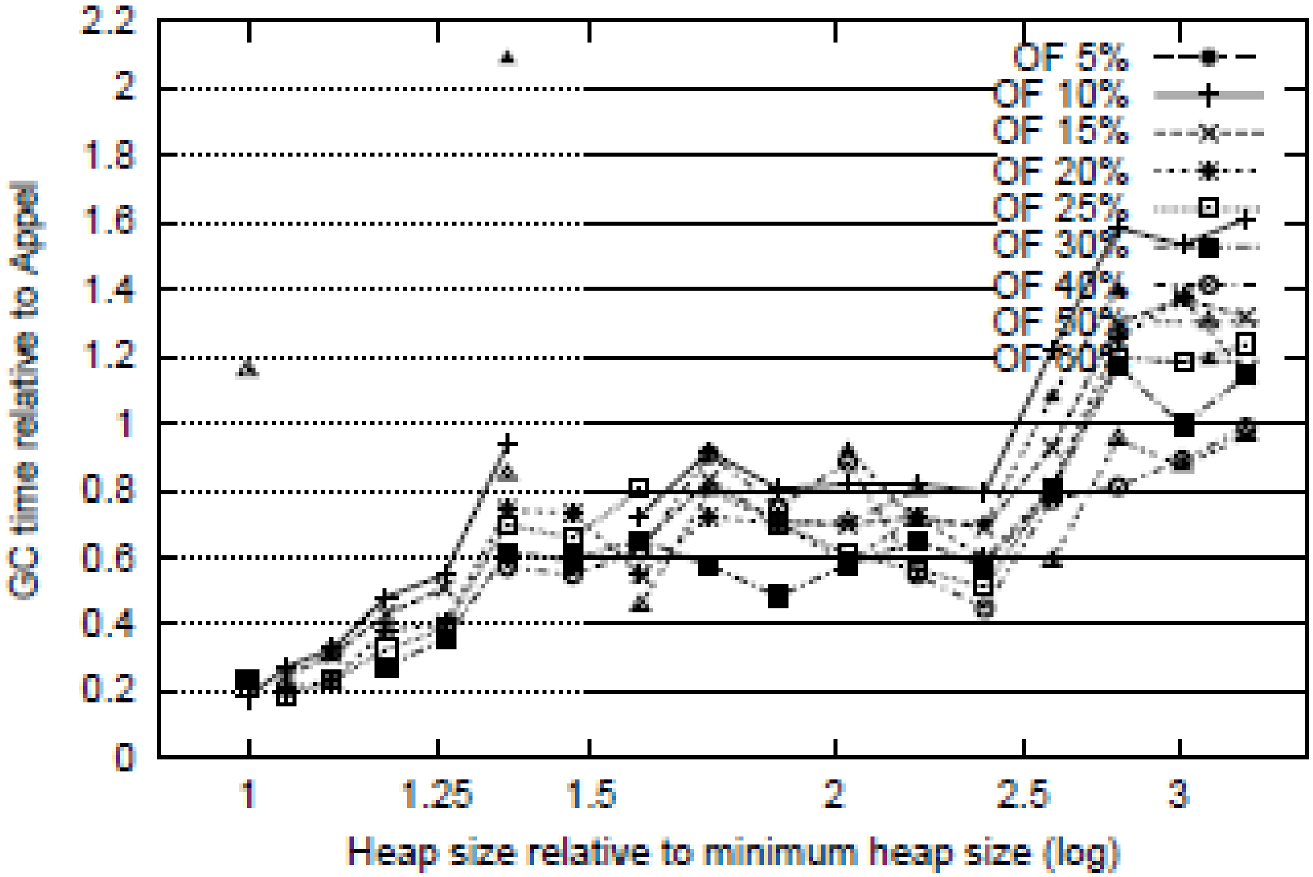
- Garbage collection time



(a) Generational collector

Experiment Results for *pseudojbb*

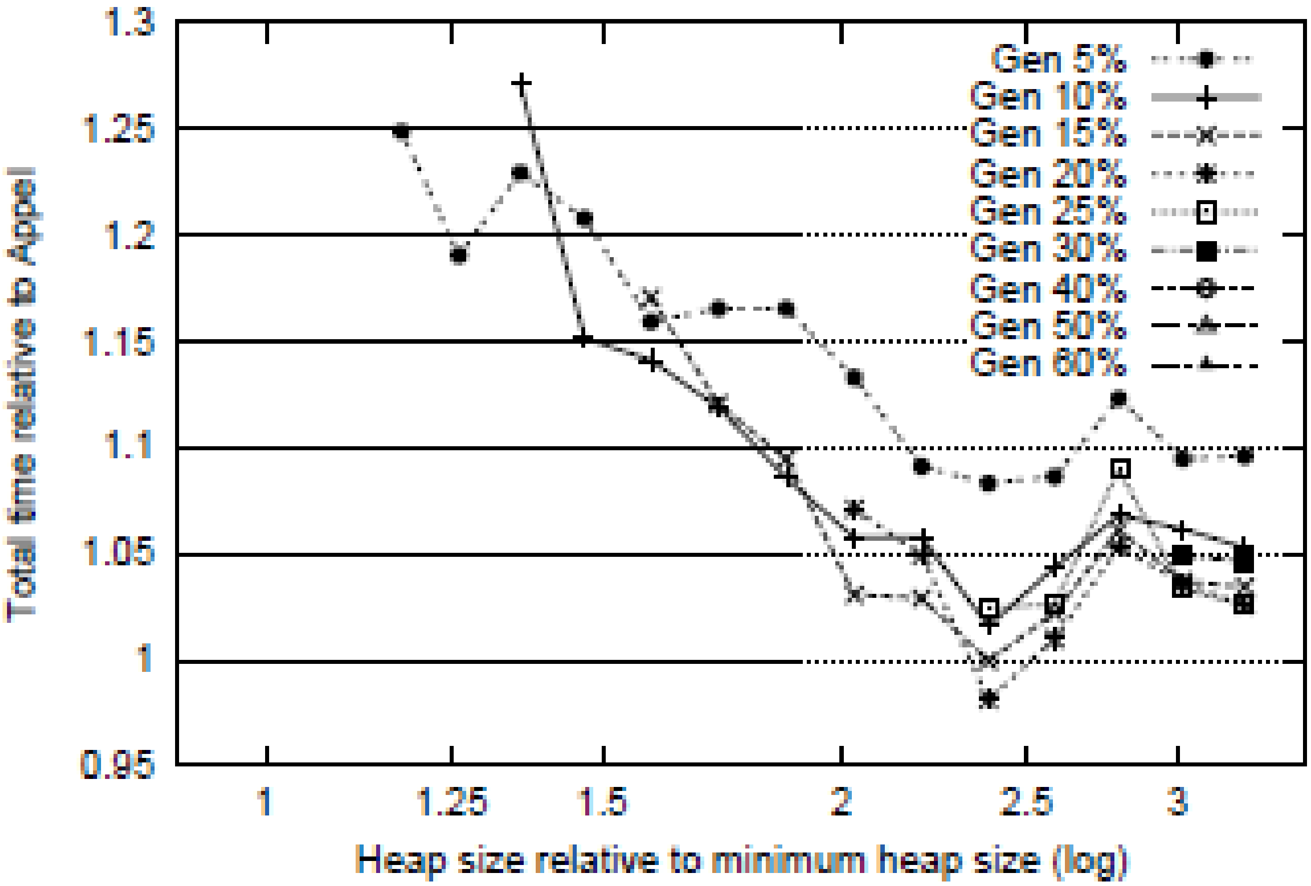
- Garbage collection time



(b) Older-First collector

Experiment Results for *pseudojbb*

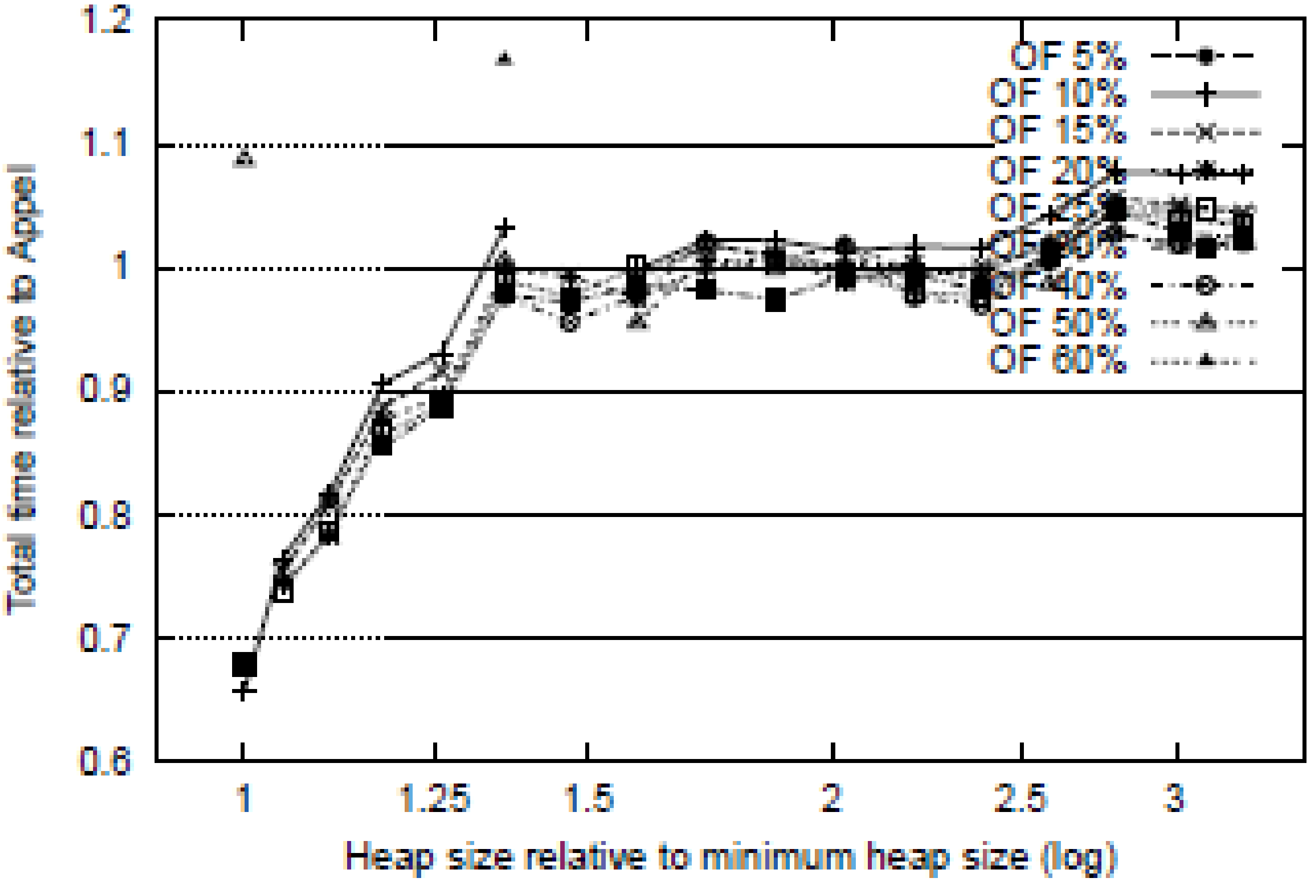
- Total execution time



(a) Generational collector

Experiment Results for *pseudojbb*

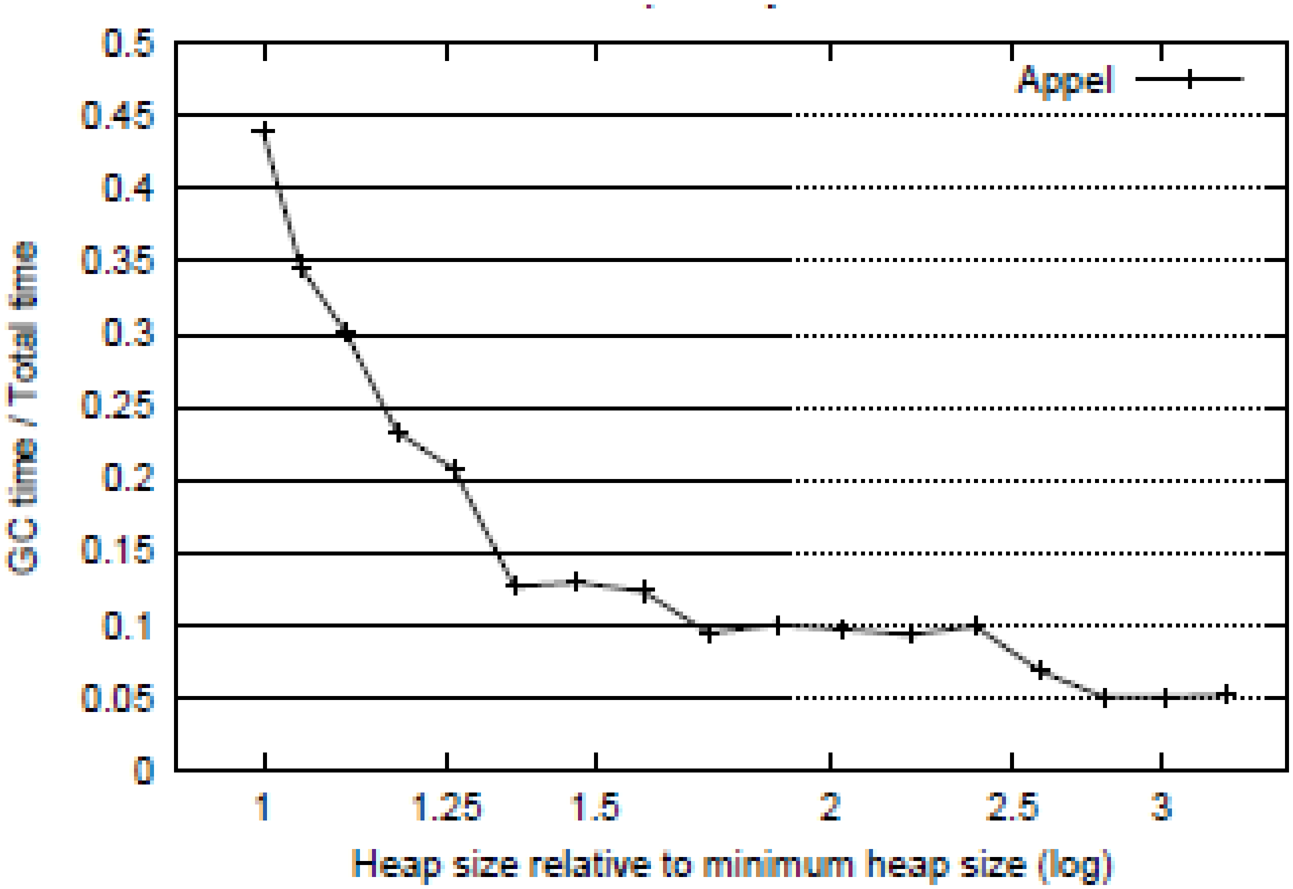
- Total execution time



(b) Older-First collector

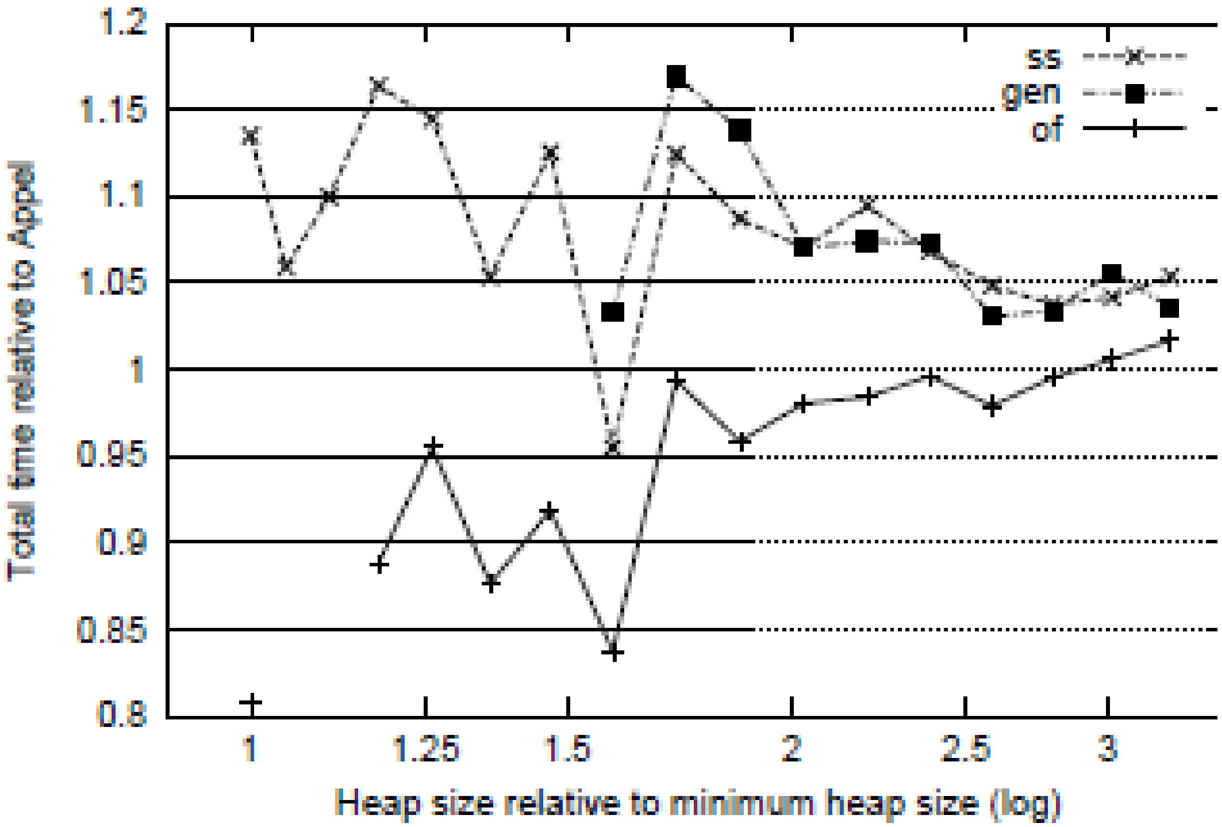
Experiment Results for *pseudojbb*

- GC time as fraction of total execution time (Appel collector)



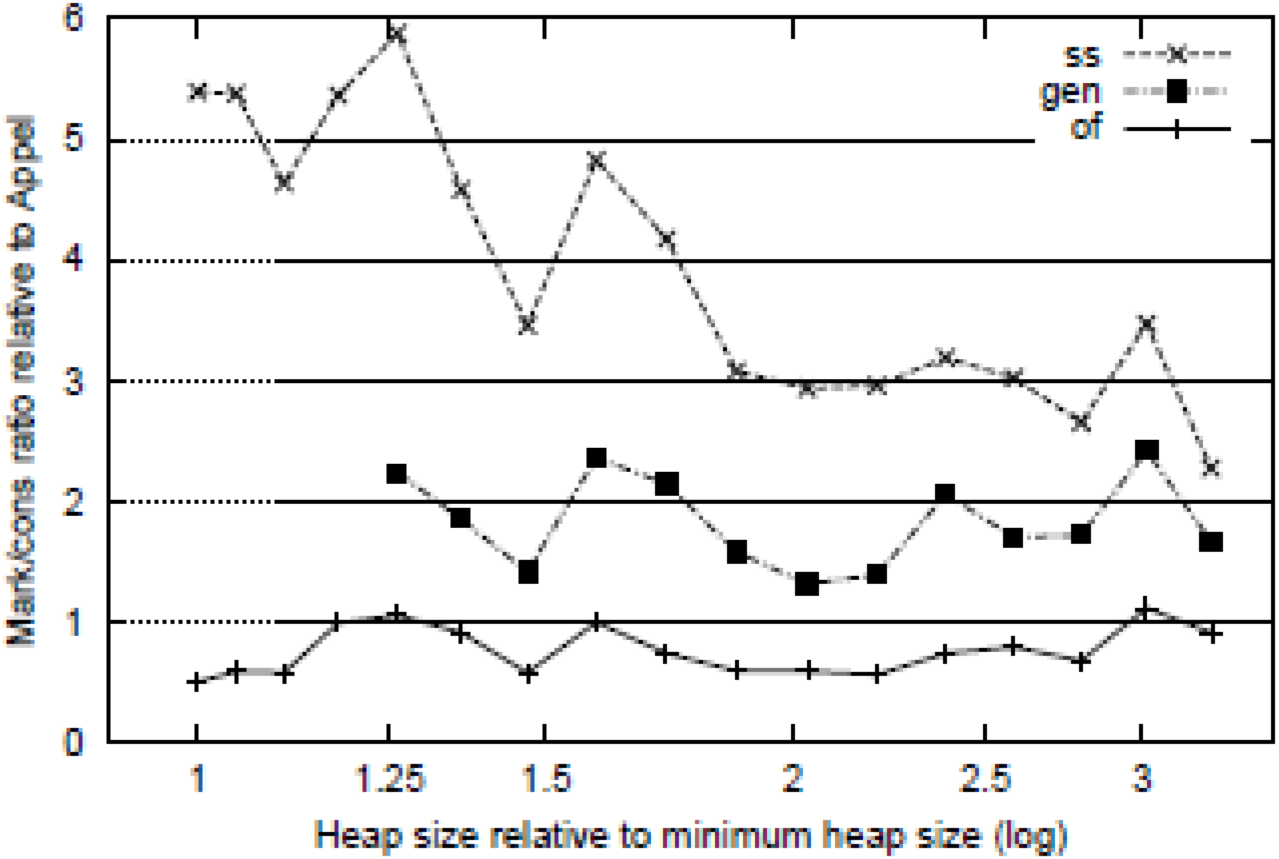
Experiment Results for All Benchmarks

- Total execution time: OF outperforms SS and Gen (e.g., *SPEC_201_compress*)



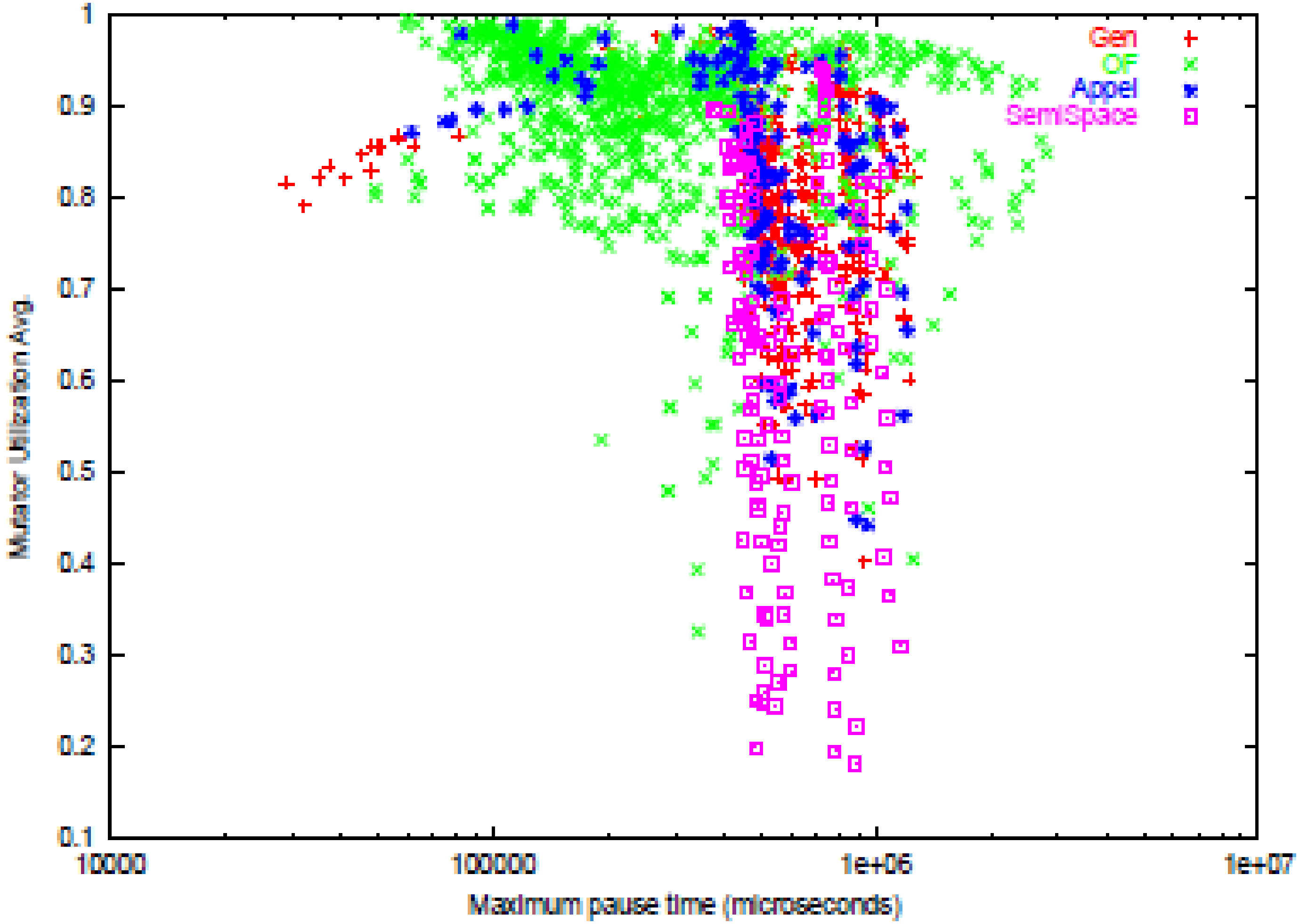
Experiment Results for All Benchmarks

- Mark/cons ratio: OF outperforms SS and Gen in 7 out of 10 benchmarks (e.g., *SPEC_209_db*)



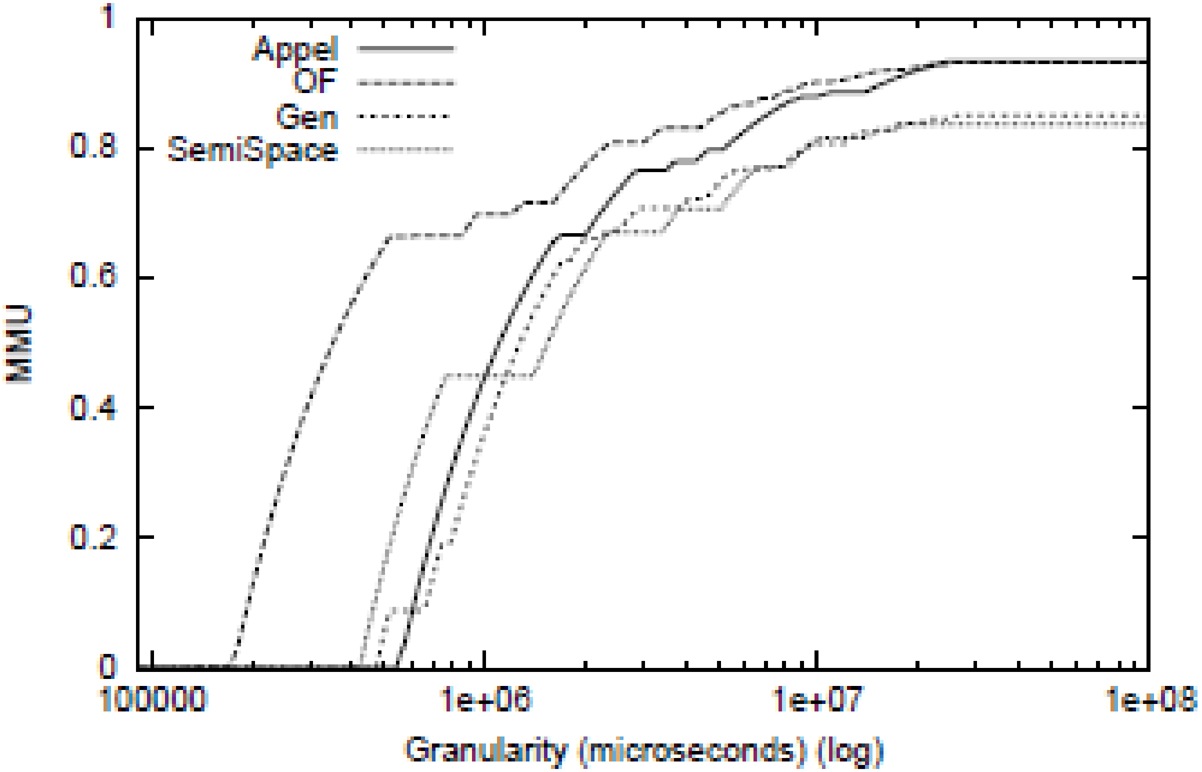
Experiment Results for All Benchmarks

- Mutator utilization vs. Maximum pause time



Experiment Results for All Benchmarks

- MMU (Minimum Mutator Utilization): OF outperforms SS, Gen, and Appel in 6 out of 10 benchmarks (e.g., *SPEC_201_compress*)



Outline

- Introduction
- Age-Based Garbage Collection
- Simulation Results of [Paper 1]
- Discussion of [Paper 1]
- Problem Statement of [Paper 2]
- Experimental Design and Results of [Paper 2]
- Conclusion

Conclusion

- OF is proved to improve GC performance (simulation and experimental results)
- Practical to avoid copying the very youngest objects
- Better implementations of OF are possible