Controlling Fragmentation and Space Consumption in the Metronome

David F. Bacon
Perry Cheng
V.T. Rajan
IBM T.J. Watson Research Center
Problem Domain

- Hard real-time garbage collection
  - Implemented for Java
- Uniprocessor
  - Multiprocessors rare in real-time systems
  - Complication: collector must be finely interleaved
  - Simplification: memory model easier to program
    - No truly concurrent operations
    - Sequentially consistent
Metronome Project Goals

- Make GC feasible for hard real-time systems
- Provide simple application interface
- Develop technology that is efficient:
  - Throughput, Space comparable to stop-the-world
- **BREAK THE MILLISECOND BARRIER**
  - While providing even CPU utilization
Outline

- Overview of the Metronome
- Empirical Results
- What is Fragmentation?
  - Static and dynamic measures
- How do we control space consumption?
- Conclusions
The Metronome Collector
Real-time GC Approaches

- **Mark-sweep (non-copying)**
  - Fragmentation avoidance, coalescing
  - Subject to space explosion

- **Semi-space Copying**
  - Concurrently copies between spaces
  - Cost of copying, consistency, 2x space

- **The Metronome**
  - Mark-sweep, selective defragmentation
  - Best of both, adaptively adjusts
Components of the Metronome

- Incremental mark-sweep collector
  - Mark phase fixes stale pointers
- Selective incremental defragmentation
  - Moves < 2% of traced objects
- Time-based scheduling
- Segregated free list allocator
  - Geometric size progression limits internal fragmentation
Support Technologies

• Read barrier: to-space invariant [Brooks]
  – New techniques with only 4% overhead
• Write barrier: snapshot-at-the-beginning [Yuasa]
• Arraylets: bound fragmentation, large object ops
Empirical Results
Pause time distribution: javac

Time-based Scheduling

![Pause time distribution chart for javac.time.opt](chart1)

Count

Pause Time(ms)

Work-based Scheduling

![Pause time distribution chart for javac.work.opt](chart2)

Count

Pause Time(ms)

12 ms
Utilization vs. Time: javac

Time-based Scheduling

Work-based Scheduling
Space Usage: javac

javac.work.opt: Space usage vs time

- In Use (Time-Based)
- In Use (Work-Based)
- Maximum Live Data
- GC trigger
Parameterization

Tuner

\[ \Delta t \]

\[ s \]

\[ u \]

Mutator

\[ a^*(\Delta GC) \]

\[ m \]

Collector

\[ R \]

Real Time Interval

Maximum Used Memory

CPU Utilization

Allocation Rate

Maximum Live Memory

Collection Rate
Is it real-time? Yes

- Maximum pause time < 4 ms  [currently]
- MMU      > 50% ±2%
- Memory requirement < 2 X max live
Static Fragmentation
Segregated Free List Allocator

- Heap divided into fixed-size pages
- Each page divided into fixed-size blocks
- Objects allocated in smallest block that fits
Fragmentation on a Page

- **Internal**: wasted space at end of object
- **Page-internal**: wasted space at end of page
- **External**: blocks *needed* for other size
Fragmentation: $\rho=1/8$ vs. $\rho=1/2$
Dynamic Fragmentation
Locality of Size: $\lambda$

- Measures reuse
- Normalized: $0 \leq \lambda \leq 1$
- Segregated by size

$$\lambda = \sum_i \min (f_i / f, a_i / a)$$
\( \lambda \) in Real-time GC Context

- Mark-sweep (non-copying)
  \[ \text{Assumes } \lambda=1 \]

- Semi-space Copying
  \[ \text{Assumes } \lambda=0 \]

- The Metronome
  Adapts as \( \lambda \) varies
\( \lambda \) in Practice

![Bar chart showing \( \lambda \) and % Defrag for various applications: javac, db, jack, mtrt, jess, and fragger.](chart.png)
Controlling Space Consumption
Triggering Collection

- Defrag
- Free

TIME

MS1  DF1  MS2  DF2  MS3

PAGES
Factors in Space Consumption

\[ \lambda \]

\[ \begin{align*} 
MS1 & \quad DF1 & \quad MS2 & \quad DF2 & \quad MS3 \\
R & \end{align*} \]
Reducing Space Consumption

- **Collection Rate \( R \)**
  - Higher rate: less memory reserve needed
  - Speed up collector
  - Specify rate more precisely (bound worst-case)

- **Locality of Size \( \lambda \)**
  - Higher locality: less free page reserve needed
  - Specify page requirement more precisely
Conclusions
Conclusions

- **Contributions**
  - Time-based scheduling (Tunable)
  - Mostly non-copying collection ($\lambda$ varies)
  - Efficient software read barrier
  - Precise definition of fragmentation
  - Precise specification of collection triggers

- **The Metronome provides true real-time GC**
  - First collector to do so without major sacrifice
    - Short pauses (4 ms)
    - High MMU during collection (50%)
    - Low memory consumption (2x max live)