Hard Real-Time Garbage Collection in Java Virtual Machines

Towards unrestricted real-time programming in Java

Jamaica Systems

Fridtjof Siebert, IPD, University of Karlsruhe
Structure

• Existing GC Techniques
• Definition of Real-Time Garbage Collection
• Jamaica’s Garbage Collector
• Example
• Conclusion

Harm Real-Time Garbage Collection in Java Virtual Machines
Garbage Collection

Automatic reclamation of unused memory

Mark

Sweep

Compact

Root Scanning

GARBAGE COLLECTION
Why is Real-Time GC so difficult in Java?

- Allocation is an integral part of Java programming
- Java is Multi-Threaded
- Java Objects and Arrays may be of very different sizes

Why is Real-Time GC
Existing Implementations

Real-Time Threads ↔ Real-Time Code uses subset of Java language

Generational, Age Based GC as separate thread

Concurrent ↔

Generational, Age Based →

Blocking ↔

Hard Real-Time Garbage Collection in Java Virtual Machines
Definition of Real-Time GC

Real-Time Garbage Collection means:

- Upper bounds for execution time must be hard upper bounds for pre-emption delay.
- Hard upper bound for pre-emption delay.
- Any operation must succeed in the desired way.
- Hard upper bounds for execution time of any operation.
- Upper bounds for execution time must be short (in the order of µsec/nsec).

For the implementation to be useful:

Real-Time Garbage Collection in Java Virtual Machines
Definition of Real-Time GC

Operations that are affected by GC are

- Memory reads (accesses to objects, arrays)
- Memory writes
- Allocation of objects
- (read-barrier code?)
- (write-barrier code?)

Operations that are affected by GC are

Hard Real-Time Garbage Collection in Java Virtual Machines
Real Time Garbage Collection in the Jamaica Virtual Machine
Root scanning typically causes the biggest problems in real-time garbage collection:

- Need to stop threads for a long time!
- But, additional overhead of writing 2-3 refs on every call (using write-barrier!)
- In the Jamaica Virtual Machine:
  - All life refs are also stored on the heap
  - There is only one single root pointer

Root scanning is a real-time operation, but writing 2-3 refs on every call (using write-barrier) adds additional overhead.

In the Jamaica Virtual Machine:

- Need to stop threads for a long time!
- Problems in real-time garbage collection:

Root scanning typically causes the biggest performance issues.
Avoid fragmentation without using handles:

- Heap is array of blocks of the same size
- Larger objects are represented as a graph
- At least one such block used for every Java
- Object allocated

Fragementation
Object Layout

Hard Real-Time Garbage Collection in Java Virtual Machines
GC Algorithm

• Simple incremental mark & sweep garbage collector
• Does not know about Java objects, just works with blocks
• Tagging is used to identify references
• One word per object

Garbage collector mark & sweep

Simple incremental

Root Scanning

Mark

Sweep

Compact
Heap Layout

13

Jamaica Systems

Heap Layout

Blocks

Tag Bits

Colours
Free memory dynamically as function of amount of application threads.

Amount of GC work is determined as function of amount of free memory.

Amount of GC work is determined as function of amount of free memory.

Couple GC activity with allocation in application threads.

Couple GC activity with allocation in application threads.

CPU time.

Recycle enough memory or use too much information on the application might not run GC running as a separate thread with no GC Activation.
heap size is possible

\[ \text{GC Activation} \]

With this

\[ \text{Amount of GC work is adjusted dynamically as needed by application} \]

\[ \text{No allocation means no GC work at all} \]

\[ \text{GC can guarantee an upper bound of GC real-time garbage collection in JVMs} \]

\[ \text{Hard-real time garbage collection in JVMs} \]
```java
public class HelloWorld {
    public static void main(String[] args) {
        int n, s, c;
        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        } else {
            n = 30;
        }
        s = 0; c = 14;
        for (int i = 0; i < n; i++) {
            String s1 = "                    ".substring(s+14);
            String s2 = "                    ".substring(s/2+7);
            String s3 = "                    ".substring(s+14);
            System.out.println(s1 + "Hello " + s2 + "World!");
            s = s + c / 4;
            c = c - s / 4;
        }
    }
}
```

Example
Example

Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Example

Determination of worst-case execution time of new StringBuffer()

Determine number of blocks:

\[ \text{numblocks} = 1 \times \text{max gc-unit} \times \text{num gc-unit} \]

Worst-case execution time:

\[ \text{wcet} = \text{numblocks} \times \text{max gc-unit} \times \text{wcet gc-unit} \]

\[ \text{wcet}\_373\_k = 1 \times 21 \times 2 \times 10^6 \text{ ms} = 42 \text{ ms} \]

\[ \text{wcet}\_373\_k = 1 \times 7 \times 2 \times 10^6 \text{ ms} = 14 \text{ ms} \]

Worst-case execution time:

1 numbblocks java.lang.StringBuffer

Determine number of blocks:

new StringBuffer()

Determination of worst-case execution time of

Hara Real-time Garbage Collection in Java Virtual Machines
Conclusion

JamaicaVM allows real-time programming using the full Java language. Having hard real-time guarantees on all parts of the programming language, including allocation, will ease software development for more complex real-time systems.

Having hard real-time guarantees on all using the full Java language.

JamaicaVM allows real-time programming.
3-Colour Marking

Colours:
- white: not marked yet
- grey: known to be reachable, but not scanned yet
- black: known to be reachable and scanned

GC-Invariant during Mark-Phase:
- black: known to be reachable and scanned
- grey: known to be reachable, but not scanned yet
- white: not marked yet

No grey objects left $\iff$ white objects are free

There is no reference from a black object to a white object.

Hard Real-time Garbage Collection in Java Virtual Machines
Colour Encoding

- black: -1
- white: 0
- grey: any other value

Grey objects form a linked list, the colour value is used to refer to the next element.

\[ \text{GC cycle completed in } O(\text{allocated}) \]

- constant time retrieval of a grey object
- constant time addition of grey objects
For three large applications:
Average number of references stored on a call:
Cost of exact root scanning:

**Experimental Data**

- Turbobol: 2.02
- Swingset: 2.51
- HotJava: 2.73

Hera Heali. Llime Harbage Collection in Java Virtual Machines
Experimental Data

Cost for object accesses using handles (scenario 1) and blocks (scenario 2)

Application: HotJava (106)
SwingSet (107)
TurboJ (109)

Objects

Cars

Arrays

Scenario

1 2 1 2 1

23

Application: HotJava (106)
SwingSet (107)
TurboJ (109)

Objects

Cars

Arrays

Scenario

1 2 1 2 1

23
Array Layout

Blocks 0-3 (data):
- data[0]
- data[1]
- data[2]
- data[3]

Blocks 4-5 (data):