Configuring the Heap and Garbage Collector for Real-Time Programming.

... A user’s perspective to garbage collection

Fridtjof Siebert, IPD, University of Karlsruhe

1

Structure

• What is the purpose of a garbage collector
• What knobs are there to turn?
• Different GCs: Blocking, Generational, Concurrent, Conservative, Mixed
• GCs that couple work with allocation
• Example
• Conclusion
What is the purpose of GC?

An automatic mechanism for memory management, to take the burden of
- memory reclamation
- memory defragmentation
- dangling references
- memory leaks / forgotten `free()`s from the user.

GC as a black box

The user does not understand what is happening inside the GC algorithm!

But: The user might configure the behaviour of the GC.

We have a black box with knobs to turn!

To be able to configure the GC seen as a black box, the user needs guidance and tools to make a good decision.

Else he will make bad decisions!
What are these knobs?

- Heap size (min/initial, max, ...)
- Object count
- Amount of GC work
  - GC priority
  - GC threshold
  - GC scanning rate
  - GC CPU-percentage
  - ...
- Control GC Algorithm
  - Select a GC implementation

In a blocking garbage collector

Effect of changing heap size $h$:

- GC pause time $p$ changes, ex. $p \sim h$
- GC pause frequency $f$ changes, ex. $f \sim 1/h$
- Application behaviour changes:
  - Runs fine for $h \geq h_{min}$
  - Crashes for $h < h_{min}$

What is the value of $h_{min}$?
In a generational GC

<table>
<thead>
<tr>
<th>Heap:</th>
<th>GCed area</th>
<th>‘old’ area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age:</td>
<td>young</td>
<td>old</td>
</tr>
</tbody>
</table>

Effect of changing heap size $h$:

- Same behaviour as blocking GC, but
  - Shorter pause time $p_{young}$ and higher frequency $f_{young}$ for collecting young
  - Pause time $p_{old}$ for collecting old as long as for blocking GC.
  - Application dependency: lower frequency $f_{old}$ of collections of old area?

Concurrent Garbage Collector

GC running e.g. as a separate thread.

Values to be adjusted

- heap size $h$
- GC threshold $t$ (when does GC start?)
- GC rate $r$ / priority / etc.

Minumum GC rate $r_{min}$ depends on application. We need a tool to determine this value!
Conservative Garbage Collector

Change of configuration or of input data has unpredictable effects on GC performance and effectiveness!

⇒ useless for nearly any serious application!

Mixed approaches

Example: HotSpot offering choice between

⇒ Generational GC + Blocking for old area

--or--

⇒ Generational GC + Incremental for old

What guides the user by his choice?

If one choice does not work, just try the other one and hope?
GC coupled with allocation

Values to be adjusted by user

- heap size $h$
- static GC work $w_{stat}$ on allocation

**Heap:**

| max. used $h_{max}$ (e.g. 60%) | GC1 | GC2 |

$\Rightarrow$ set $w_{stat}$ to 5 ($= \frac{2}{(1-h_{max})}$)

Requires tool to determine $h_{max}$ and $w_{stat}$.

GC work determined dynamically

Values to be adjusted by user

- heap size $h$

GC determines $w_{dyn}$ as function of amount of free memory.

$wcet_{alloc}$

Tool to select $h$ and determine $wcet_{alloc}$ needed.
Example

```java
public class HelloWorld {
    public static void main(String[] args) {
        int n, s, c;
        s = 0;
        c = 14;
        for (int i = 0; i < 30; i++) {
            String s1 = "            ".substring(s + 14);
            String s2 = "            ".substring(s / 2 + 7);
            System.out.println(s1 + "Hello " + s2 + "World!");
            s = s + c / 4;
            c = c - s / 4;
        }
    }
}
```

Example

```
> jamaica -analyse 5 HelloWorld
> HelloWorld
Hello   World!
Hello   World!
[...]
### Application used at most 117224 bytes for Java heap
###
#### heapSize  wcet  dynamic  wcet  static
#### 337k       7     3
#### 226k       7     4
#### 189k       10    5
#### 170k       14    6
#### 162k       16    7
#### 152k       21    8
#### 143k       28    10
#### 134k       40    14
#### 121k       138   40
#### 118k       286   80
```
Example

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

Determine number of blocks:

```
> numblocks java.lang.StringBuffer
1
```

Worst-case execution time:

\[ wcet = \text{numblocks} \cdot \max_{gc\_unit} \cdot wcet_{gc\_unit} \]

\[ wcet_{152k} = 1 \cdot 21 \cdot 2\mu s = 42\mu s \]

\[ wcet_{226k} = 1 \cdot 7 \cdot 2\mu s = 14\mu s \]
Conclusion

• The user can not be burdened with understanding the GC mechanism used by an implementation.

• Current implementations lack tools that guide the user in making a good choice for GC configuration

• Fewer knobs are better! A value that can’t be changed can’t be set wrong.