Flexible automatic memory management
for real-time and embedded systems

Sven Gestegård Robertz

Department of Computer Science

Lund University
Outline

- Problem statement
- Background
- Time-triggered GC
- Adaptive GC scheduling
- Priorities for memory allocation
- Summary
Problem statement

- Adding flexibility to hard real-time systems
  - The need for flexibility is increasing
  - Not all hard RT systems are safety critical
  - The gap between theory and practice

- Hard RT memory management in practice
  - Non-intrusiveness
  - GC work metrics
  - GC tuning
Problem statement

Write once — run anywhere
for hard real-time systems
Problem statement

*Treat scheduling and schedulability analysis separately*
Feedback scheduling

A simple model

\[ U_{sp} \rightarrow \text{Scheduler} \rightarrow \{T_i\} \rightarrow \text{Tasks} \rightarrow \{jobs\} \rightarrow \text{Dispatcher} \rightarrow c_i, U \]
Garbage Collection

- Batch GC
- Incremental GC
- Real-time GC
- Non-intrusiveness
- Practically feasible
Incremental GC

- GC work scheduled at allocations
- Increment size proportional to object size
- Ensuring sufficient progress

\[ w \geq W_{\text{max}} \cdot \frac{a}{F_{\text{min}}} \]

GC performed in-line with application code may cause long delays
Semi-concurrent GC scheduling

Presented in [Henriksson 98]

- Only suitable for fixed-priority scheduling
- Requires detailed tuning
GC work metrics

How to express GC work

- Based on known quantities
- Model the temporal behaviour of the GC
- Feasible to calculate at run-time
The evacuation pointer metric

\[ W = \Delta B \]

\[ W_{max} = E_{max} \]

Problem: A small increment of the metric may take very long time to perform.
An improved metric

\[ W = \alpha \cdot \text{roots} + \beta \cdot \Delta S + \Delta B + \gamma \cdot \Delta P \]

\[ W_{\text{max}} = \alpha \cdot \text{roots}_{\text{max}} + \beta \cdot E_{\text{max}} + E_{\text{max}} + \gamma \cdot M_{HP} \]

Requires tuning of \( \alpha, \beta \) and \( \gamma \)
Allocation-triggered GC

Issues:

- Bursty allocation
- Concurrent GC in EDF systems
- GC work metric concerns
Time-triggered GC

- Use time instead of allocation to trigger GC work
- Calculate GC cycle time that ensures sufficient progress
- \[ T_{GC} = f(H, L_{max}, \{a_p\}) \]
Time-triggered GC

Properties:

- GC rate independent of application behaviour
- GC can be scheduled as a normal thread
- GC scheduling independent of work metric
Adaptive GC scheduling

Manual tuning of GC scheduling parameters

- requires detailed analysis of both GC and application
- is based on worst case analysis
- is not possible if the run-time configuration or platform is unknown
Adaptive GC scheduling

Two orthogonal problems

- Tune GC cycle time
- Estimate GC work
Feedback scheduling

Scheduler

Memory manager and GC auto-tuner

Tasks

Dispatcher

\[ U_{sp}, T_{GC}, C_{GC} \]

\[ C_{GC} \]

\[ U, C_i \]

\[ \{T_i\} \]

\[ \{jobs\} \]
GC cycle time auto-tuning

A simple model:

\[ T_{\text{remaining this cycle}} = \frac{F}{\dot{a}} \]

\[ T_{GC} = \frac{F}{\dot{a}} + T_{\text{elapsed}} \]
Experiment
Priorities for memory allocations

Memory is a global resource

- Great responsibility on programmers
- Out-of-memory errors have serious consequences
Background

Not all of the code is critical

- Critical parts must always be executed
- Non-critical parts may be skipped if there is not enough memory to run them safely
- Critical and non-critical "aspects"
The basic idea

Prevent system from running out of memory by limiting the amount of non-critical allocations.

- Traditionally done manually
- Run-time system support

Priorities for memory allocations!
Non-critical limit

Keep the amount of live, non-critically allocated memory below a safe limit

or

Keep the amount of allocatable memory above the safe level
Example: logging

Simple control application

- Control – critical
- Logging – non-critical

```java
void control()
{
    calculateControlSignal();
    setState();
    try{
        deliverLogData();
    } catch(NoNonCriticalMemoryException e) {
        // not enough memory to safely allocate log data
    }
}
```
Example: all allocations critical
Example: log data is NC
Example: closeup
Example: Performance

a) log data objects are always allocated

b) allocation of log data is non-critical
Priorities for memory allocations

- Memory requirements can be separated into “critical” and “non-critical”
- Separate memory and CPU time priorities
  Not all of the allocations in a HP process are critical
- Run-time system support
- Improves robustness and performance
- Possibility to control the memory behaviour
- Worst case analysis only needed for critical parts
Summary

Time-triggered GC scheduling

- Cycle-level view on GC scheduling
- Non-intrusive GC with guaranteed progress under EDF
- Explicit scheduling parameters fits well into feedback scheduling and auto tuning systems

Priorities for memory allocation

- Increased robustness and performance
- Possible to control allocation rate
Future Work

- Integrated prototype
- Feedback scheduling
- Distributed systems, composability