Towards Memory Management for Service-Oriented RTS

Tom Richardson
Overview

- Introduce service-oriented architecture (SOA)
- Motivation for integrating SOA with RTS (RT-SOA)
  - Dynamic reconfiguration
- Issue of memory management in RT-SOA
  - Issue of preconfigured GC
  - Issue of scoped memory and 3rd party services
- Dynamically reconfigurable GC
  - Reconfiguration analysis
  - Admission control
  - Reconfigurable GC
- Evaluation and conclusions
Service-Oriented Architecture (SOA)

- A service is an act or performance offered by one party to another
- Similar to objects, modules, and components etc, but:
  - Dynamically discoverable
  - Dynamically available
- Service-Oriented Architecture (SOA) is a way of designing a software system to provide services:
  - To end-user applications
  - To other services
  - Achieved by using published and discoverable interfaces (Publish-Find-Bind)
- SOA enables application dynamic reconfiguration
Dynamic Reconfiguration

- Dynamic reconfiguration
  - Enables the application architecture to be modified at run-time
  - Without shutting the application down

- Different SOA technologies offer different levels of application dynamic reconfiguration
  - The most basic level is service substitutability – i.e. the ability to bind with different service providers at run-time
  - OSGi Framework provides more powerful reconfiguration

- OSGi Framework
  - SOA dynamism – Can acquire new services and release services at run-time
  - CBSE with dynamism – Provides the ability to install, uninstall, and update components at run-time
Dynamic Reconfiguration Example

Service Provider

OSGi Framework

Service Registry

Service Requester

Service

Service

T1 T3 T2

JVM
Motivation for SOA in Real-Time Systems

- Dynamic reconfiguration improves the system availability
  - System does not need to be taken offline to be maintained/reconfigured
  - Improves application availability
  - Important in RTS in particular as they have high availability requirements

- Dynamic reconfiguration minimises memory resource requirements
  - Only require the components and services comprising the current mode of operation to be deployed
  - Important in embedded systems (resource constrained)

- Remote controllability- evolving RTS from a remote location
  - RTS deployed in harsh environment- danger in being physically present in deployment environment for updating software
  - Software updates in mass produced embedded devices such as consumer electronics
Current GCs with Dynamically Reconfigurable Systems

Runtime

- Application reconfigured at run-time
- GC NOT reconfigured accordingly
- Risk of memory exhaustion
Scoped Memory with Dynamically Reconfigurable Systems

- Scoped Memory (SM)
  - Avoids overheads of garbage collection (GC) and therefore suited to harder RTS

- Two approaches to using SM in SOA
  - Threads can enter scoped memory before calling services
    - IllegalAssignmentErrors if service method breaks RTSJ memory assignment rules
  - Services handle scoped memory
    - ScopedCycleExceptions depending on the scope stack of calling threads
    - Blocking – ensuring only one thread in SM at any one time

- We focus on GC not SM
  - RT-GC advancing, adequate for RT-SOA
Application Reconfiguration Example

<table>
<thead>
<tr>
<th>Thread</th>
<th>C (ms)</th>
<th>T (ms)</th>
<th>D (ms)</th>
<th>A (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0.1</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>12</td>
<td>12</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>T5</td>
<td>5</td>
<td>30</td>
<td>30</td>
<td>0.4</td>
</tr>
<tr>
<td>T6</td>
<td>1</td>
<td>18</td>
<td>18</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[ T_{GC} = 8 \text{ ms} \]
\[ C_{GC} = 2.5 \text{ ms} \]
\[ M = 24.5 \text{ MB} \]

- Perform application reconfiguration:
  - GC reconfiguration analysis
  - Application reconfiguration admission control
  - GC reconfiguration
Example – GC Analysis

- **Estimate GC cycle work** ($W_{GC}$)
  - Computation time to complete a GC cycle
    - Cost of reference traversal (root-set, live objects)
    - Cost of object evacuation (copying)
    - Cost of memory initialisation

\[
W_{GC} = c_1 \left( \frac{R + \sum_{i=1}^{n} A_i}{\text{sizeof}(\text{word})} \right) + c_2 \sum_{i=1}^{n} A_i + c_3 \left( \frac{H}{2} \right)
\]

- $W_{GC} = 29.2$ ms
Example – GC Analysis

Calculate GC parameters

- Find the maximum CPU utilisation for GC
- GC period ($T_{GC}$) – equal to the application thread with the smallest period
- GC budget ($C_{GC}$) – find maximum value of $x$ such that all threads remain schedulable

$$C_{GC} = \max \left\{ x \mid \forall t \in \tau_{i=0\ldots n}: \left[ \frac{T_i}{T_{GC}} \right] x + \sum_{j=1}^{i} \left[ \frac{T_i}{T_j} \right] C_j \leq D_i \right\}$$

- $T_{GC} = 5\text{ms}$, $C_{GC} = 0.5\text{ms}$
Example – GC Analysis

Calculate GC cycle time

- During a GC period, the GC thread can only perform an amount of work equal to $C_{GC}$
- Therefore a number of GC periods will be required to complete a GC cycle

$$R_{GC} = (T_{GC} - C_{GC}) + \left( \frac{W_{GC}}{C_{GC}} - 1 \right) (T_{GC} - C_{GC}) + W_{GC}$$

- $R_{GC} = 294.7$ ms
Example – Admission Control

- Application reconfiguration admission control
  - Application’s memory requirement is 2 * (worst case live memory plus garbage allocation in a GC cycle of worst case length)
  - As at end of a GC cycle, before semispace flip, both semispaces will have copies of live memory, and one GC cycle’s worth of garbage alloc

\[
M = 2 \left( \sum_{i=1}^{n} \left( \left[ \frac{R_{GC}}{T_i} \right] + 1 \right) A_i \right) + \sum_{i=1}^{n} A_i
\]

- Guarantees threads will not experience memory exhaustion
  - If free mem \(\leq M\) then accept application reconfiguration and reconfigure GC
  - Else, reject application reconfiguration as it would cause memory exhaustion

- \(M = 93.9 \text{ MB}\)
Example – GC Reconfiguration

- Reconfiguration analysis determines new C,T,D parameters for the GC
- Admission control controls application reconfiguration
- Need to reconfigure the GC with these parameters
- Only GC available that can be reconfigured is Sun’s GC
  - Only the GC thread’s priority can be modified, but not its C,T,D

Solve by:
  - Setting GC thread’s priority to a background priority
  - Creating a GC controller thread to manipulate the GC thread’s priority such that it appears like a time-based GC
  - GC controller thread period = $T_{GC}$
  - Time GC runs at high priority = $C_{GC}$
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Expected result</th>
<th>Actual result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single thread</td>
<td>Memory not exhausted (Reconfig analysis)</td>
<td>Garbage is collected. No memory exhaustion.</td>
</tr>
<tr>
<td>Several threads</td>
<td>Memory not exhausted (Reconfig analysis, Admission control)</td>
<td>Garbage is collected. No memory exhaustion Admission control functions correctly.</td>
</tr>
<tr>
<td>Dynamic unbounded structures</td>
<td>Eventually memory exhausted</td>
<td>Any garbage is collected, but memory is eventually exhausted.</td>
</tr>
<tr>
<td>Misbehaving thread</td>
<td>Memory not exhausted (Memory allocation enforced)</td>
<td>Misbehaviour is detected and the thread is blocked. No memory exhaustion</td>
</tr>
</tbody>
</table>
Conclusions

- Can develop dynamically reconfigurable RTS applications with:
  - GC Reconfiguration analysis
  - Admission control
  - Reconfigurable GC
  - Memory allocation enforcement

- No risk of garbage related memory exhaustion

- Dynamic reconfiguration:
  - Improves system availability
  - Reduces the memory requirement of application

- Beneficial to RTS as they typically have:
  - high availability requirements
  - resource constrained
Questions?