IBM Software Group

Toward Deterministic Java Performance

Mark Stoodley, Mike Fulton
Toronto Lab, IBM Canada Ltd.

The Real-Time World

- **Responsive in “real time”**
  - Often keyed to real world events
  - Performing work on a regular basis
  - Asynchronous events
  - Graceful handling of truly exceptional conditions
- **Deterministic performance key to meet response time requirements**
- **Java performance not really responsive as-is**
  - But it’s a nice development environment
  - Motivates the Real-Time Specification for Java

Overview

- Real-Time Java
- Java performance isn't really deterministic ✪
- Mitigating the Chaos
- Summary

Real-Time Java

- **JSR #1: Real-Time Specification for Java**
- **Facilities to support Real-Time programming**
  - Make performance more controllable & predictable
  - Large-scale enhancements to Java
    - Threading, scheduling
    - Memory management
    - Asynchronous event handling, control transfer, termination
    - Physical memory access

Example 1: Memory Management

- **SPEC assumes that managed memory (garbage collection) is incompatible with real-time needs**
- **New memory areas that are not collected**
  - Immortal memory
  - Memory scopes
- **New thread type "No Heap Realtime Thread"**
  - Not permitted to even see a heap reference
  - No need to stop for any reason when GC occurs

Java performance isn't really deterministic

- **Chaos lurks everywhere:**
  - Thread scheduling is at the whim of the operating system
  - Garbage collection occurs "whenever" for "however long"
  - JIT compilations occur "whenever" for "however long"
  - Aggressive JITs recompile methods that seem "hot"
  - JIT compilers employ many speculative optimizations
  - Class loading occurs on demand

Mitigating the JIT Chaos: stop doing “bad” stuff

- **JIT compiling delays are unacceptable**
  - Also derivative effects: profiling, sampling
  - Could run at low priority BUT risk priority inversion
- **Ahead-of-Time (AOT) compilation a better option**
  - Takes compiler out of the run-time performance equation
  - Possibly lower performance to deal with resolution order
  - Derivative effects also removed
  - BUT maybe more difficult to achieve high performance

Mitigating the JIT Chaos: stop doing “bad” stuff

- **Stop doing speculative optimizations**
  - No flat-word monitors
    - Also simplifies priority-inversion support
  - No monitor coarsening
  - Profiling-based optimizations
  - Not easy because JIT compilers speculate a LOT
- **Devirtualization ok if all classes are pre-loaded**

Mitigating the JIT Chaos: stop doing “bad” stuff

- **Class loading is a trouble spot**
  - Loading one class often requires loading other classes
  - Once class is loaded, devirtualizations may be invalid
    - Lots of call sites may need to be patched for correctness
  - Updates many VM data structures also accessed by GC
    - Particularly a problem for NoHeapRealtimeThreads
- **Application-level pre-loading is one option**
  - Collect list of loaded classes in one execution
  - “Force” class to load before application begins executing

Summary

- **Java not suitable as-is for Real-Time workloads**
- **Real-Time Specification enhances Java for RT**
- **Java VMs have many sources of nondeterminism**
  - GC, thread scheduling, JIT compiler
- **These problems can be largely mitigated**
  - Ahead-of-Time compiles, class preloading, stop doing speculative optimizations
  - Lower sustained performance but more deterministic

Contact Information and Acknowledgments

Mark Stoodley

Toronto Lab, IBM Canada Ltd.

mstoodle@ca.ibm.com

With thanks to:

Mike Fulton

Backup Slides

Example 2: Asynchronous Transfer of Control (ATC)

- **RT programs need to respond to truly exceptional conditions quickly and drastically**
- **Thread that detects condition may need to interrupt other threads actively working**
- **ATC provides facilities to mark methods that can be safely interrupted**
  - More draconian exception semantics in such methods
- **Also mechanisms to initiate such interruptions**