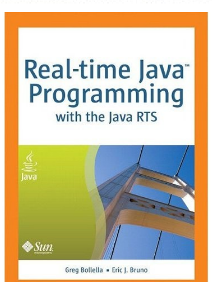
Java

Java is a trademark of Sun Microsystems, Inc.



JavaOne℠

Java™ RTS: Trade-Offs Between Throughput and Determinism

Roland Westrelin
Eric J. Bruno
Sun Microsystems

Goal

- Learn how response time determinism and throughput are related and how to balance them with real world application examples.

Agenda

- Real Time Performance
- Response Time Benchmark
- Customer case study
- Summary

Agenda

- **Real Time Performance**
- Response Time Benchmark
- Customer case study
- Summary

What is Real-Time?

- Simple definition: The addition of temporal constraints to the correctness conditions of a program
  - "When" is as important as "what"
  - "A late answer is a wrong answer"
- "real-time" does not mean "real-fast"
  - Going faster helps but ...
- Predictability is the key

Example Temporal Constraints

- Deadline: started task must complete by a given time
  - Once a request for a trade is received, it must execute within 5ms
- Latency: difference between when an event happens and when it is seen to have happened
  - Stop button handler must respond within 500us of a press
- Jitter: Variance in the time interval between events
  - The input sensor must be sampled every 1ms +/- 100us

Performance: non Real-Time

- "How fast is it?"
- Defined in terms of mean execution time
  - Over several iterations
  - Over several threads of execution
- = "throughput"
- Worst case execution time rarely matters as long as:
  - The application is always "reasonably" fast
  - A glitch "does not happen often"
- Observed/measured performance is what matters

Performance: Real-Time

- Worst case defines performance
- Hard Real-Time is not necessarily low latency
- Application scheduling can be proved feasible, guarantees are offered:
  - Known Implementation/design flaw matters even if very unlikely
- Measured with its own metrics:
  - Latency
  - Jitter
- Requires its own performance tools:
  - Tools that collect samples not sufficient

But...

- Execution time still matters...
- Users don't want to totally give up on throughput
  - Mix of Real-Time and non/soft Real-Time workload in a single app
- Some Implementation techniques used in OS or Java Virtual Machine (JVM™ machine):
  - Are incompatible with Real-Time
  - Or lead to higher latency/jitter
- Increased determinism typically comes at the expense of decreased throughput

Sun's Java Real-Time System

- Sun's implementation of the RTSJ
- Optimized Real-Time Java platform runtime based on Java HotSpot™ high performance virtual machine
- Based on Java Platform, Standard Edition 5 (Java SE platform 5)
- Runs on:
  - Solaris™ Operating System, SPARC® technology, and x86/x64 platforms
  - Real-Time Linux on x86/x64 platforms

Sun's Java Real-Time System (cont'd)

> Coming soon: 64 bit support (version 2.2)
> dtrace based tools offer full view of running system

Throughput vs determinism: Scheduling and resource usage

> For better throughput (Java SE):
  - Time sharing, let the system balance resources
  - Aggressive lock optimization (biased locking)

> For better determinism (Java RT):
  - Fixed priority, run to block scheduling, static cpu partitioning: not necessarily “best” overall usage of resources
  - Mandates Priority inversion protection scheme: more expensive than standard locks
  - Restricts use of lock-free algorithms

Throughput vs determinism: Native code generation (JIT)

- For better throughput (Java SE):
  - Compile hot methods: enable profile driven optimizations.
  - Be optimistic: allow recompilation if optimization needs to be undone.
- For better determinism (Java RT):
  - Compile everything in the critical path early: limited knowledge at compilation time
  - Make the code as steady as possible
  - Java RTS only supports hotspot's client compiler

Throughput vs determinism: Garbage Collection

- For better throughput (Java SE):
  - Stop the world GC
  - Generational GC to efficiently recycle short-lived objects
- For better determinism (Java RT):
  - Guarantee max pause time
  - Offer control over pause time frequency
  - Guarantee progress (recycling on time)
  - Concurrent or incremental GC

Throughput vs determinism: RTGC in Java RTS

- For better determinism:
  - GC runs concurrently with mutator threads
  - Critical threads always preempt the RTGC threads and allocate from reserved space
  - No stop the world phase (Pause one thread at a time)

Throughput vs determinism: RTGC in Java RTS (cont'd)

- Impact on throughput:
  - Objects may be split (heavy heap fragmentation). Compiled code need to accommodate for that.
  - Write barrier is more complex than standard hotspot
  - Not generational, non moving: higher cpu usage
    - Cost can be paid by extra CPUs
    - Cost of RTGC impacts only non-critical threads

Configuring Java RTS for best determinism

- Java RTS has a “soft real-time” behavior by default and need to be configured for hard realtime:
  - Configure garbage Collector
  - Preload/preinitialize classes, precompile methods: Init Time Compilation configuration
  - Use processor partitioning

Agenda

- Real Time Performance
- **Response Time Benchmark**
- Customer case study
- Summary

Response Time Benchmark

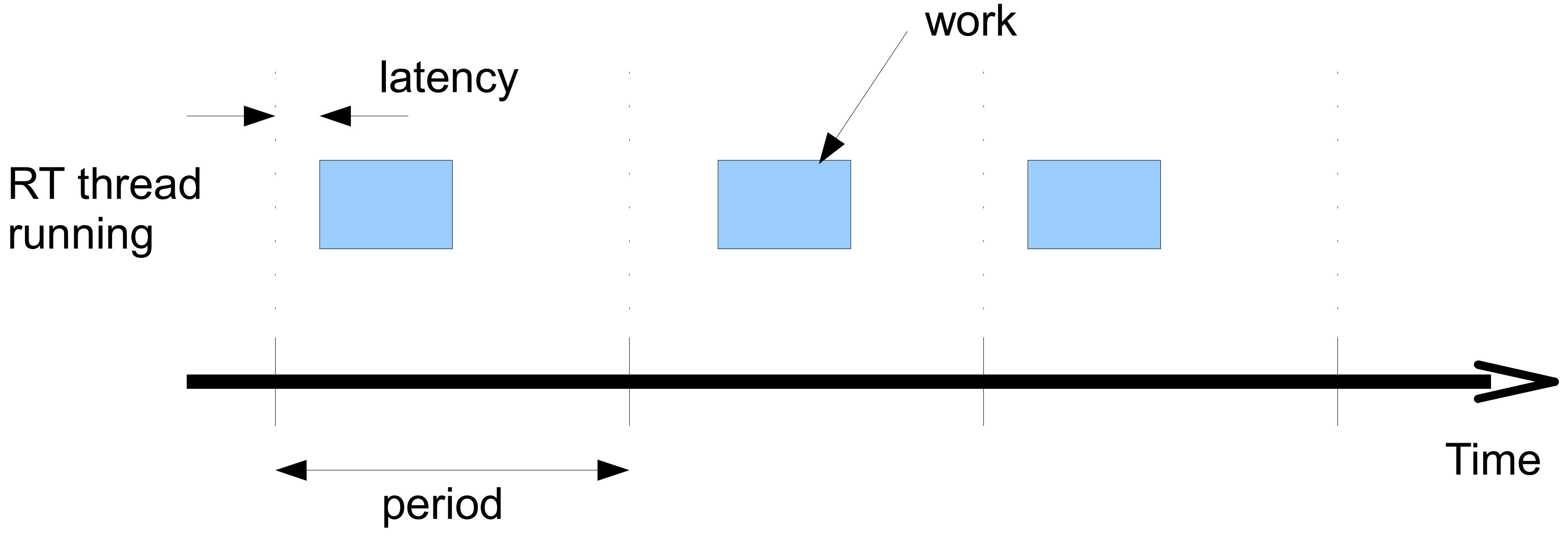
- Single Real-Time periodic thread
  - Wakes up at a pre-programmed fixed interval (absolute release dates)
  - Does not produce garbage in steady mode
- Simple but representative
  - Simple control loop: thread wakes up to read sensor and take some action
- At every release, wake-up latency is measured

Response Time Benchmark (cont'd)

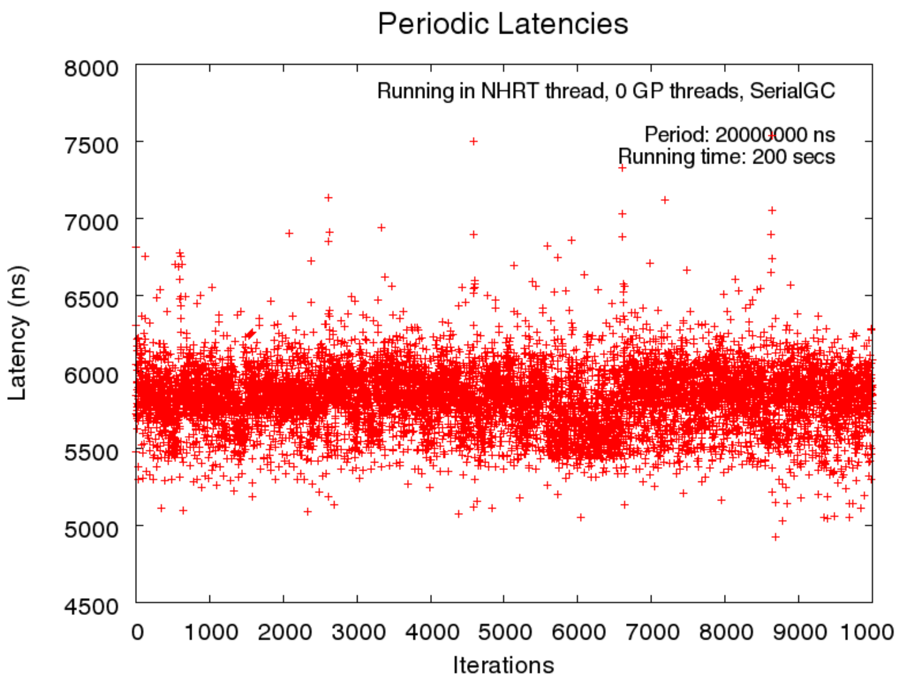
- Benchmark measures:
  - Max difference between effective release time and expected release time is computed = max latency
  - Difference between smaller and larger latency = jitter
- Runs with RT thread in a single processor partition and Init Time Compilation

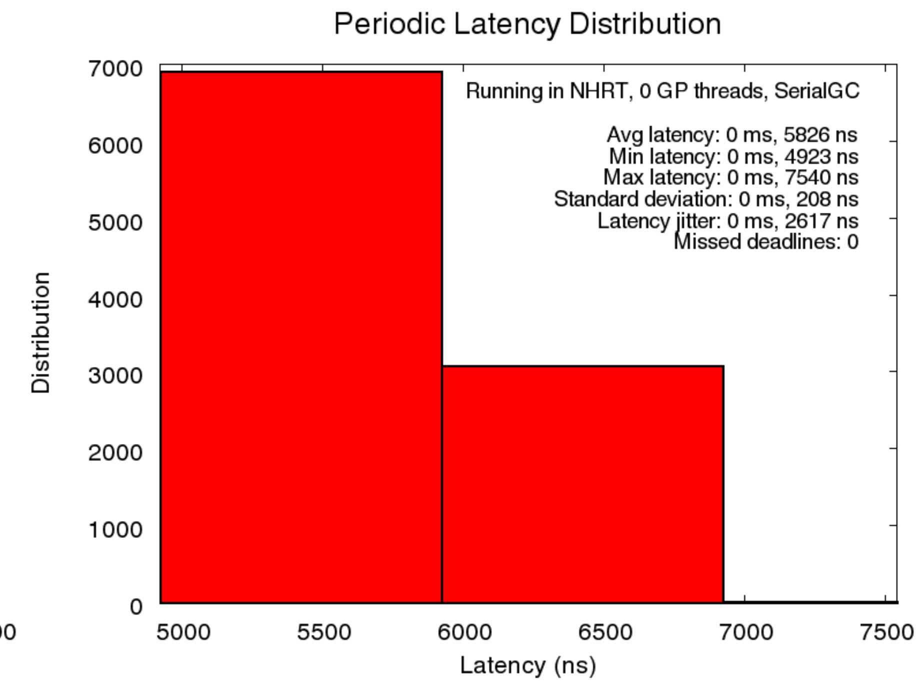
Response Time Benchmark (cont'd)

```
public void run(){
  for(int i = 0; i < iterations; i++) {
    Clock.getRealtimeClock().getTime(actualStartTimes[i]);
    waitForNextPeriod();
  }
}
```



Response Time Benchmark: Sample Result



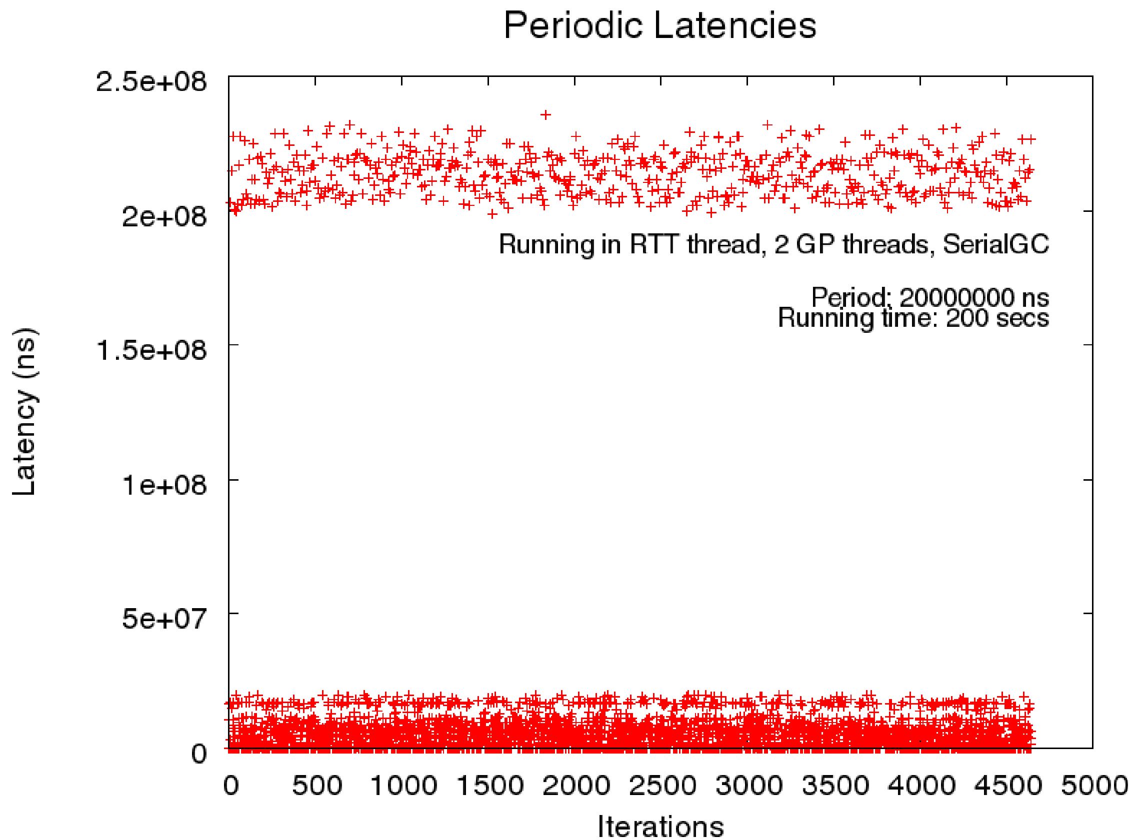


Java RTS 2.1, NHRT/Serial GC on Solaris/quad-core 2.8Ghz Opteron

Response Time: Load

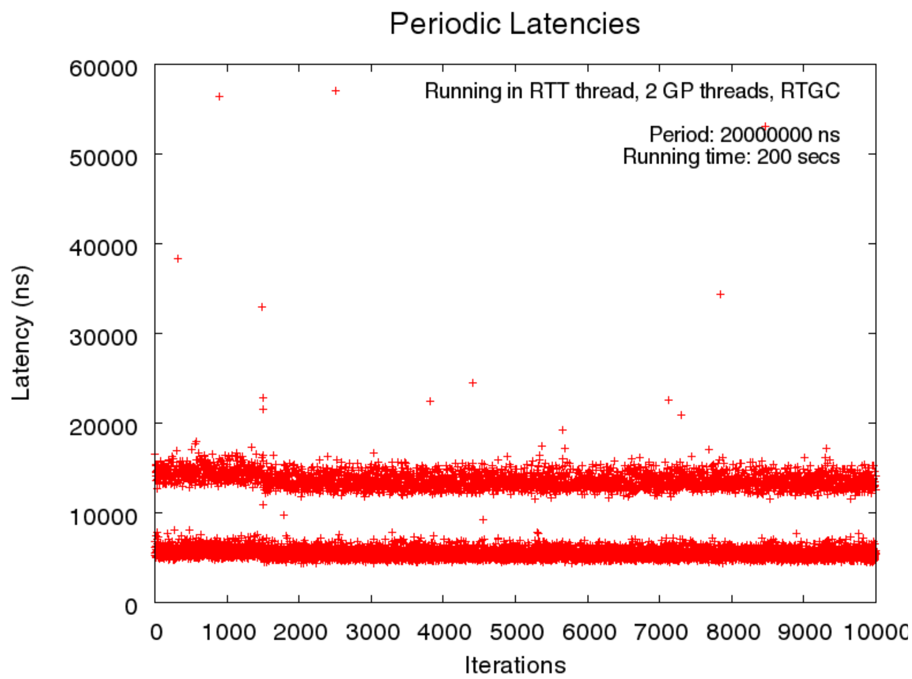
- Non Real-Time garbage producer threads
- Load the CPUs, exercise the system scheduler
- Triggers GC cycles: observe GC pauses
- Plan to add other types of load: I/O, network

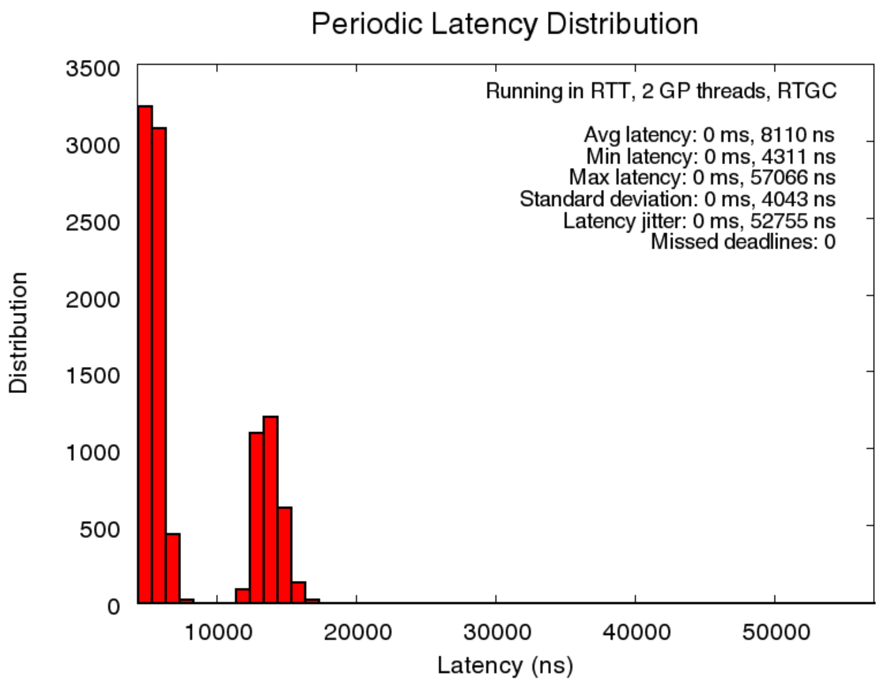
Response Time Benchmark: Non Real-Time GC



Java RTS 2.1, RTT/Serial GC/2GP on Solaris/quad-core 2.8Ghz Opteron

Response Time Benchmark: Real-Time GC





Java RTS 2.1, RTT/RTGC/2GP on Solaris/quad-core 2.8Ghz Opteron

Agenda

> Real Time Performance
> Response Time Benchmark
> **Customer case study**
> Summary

Customer Case Study

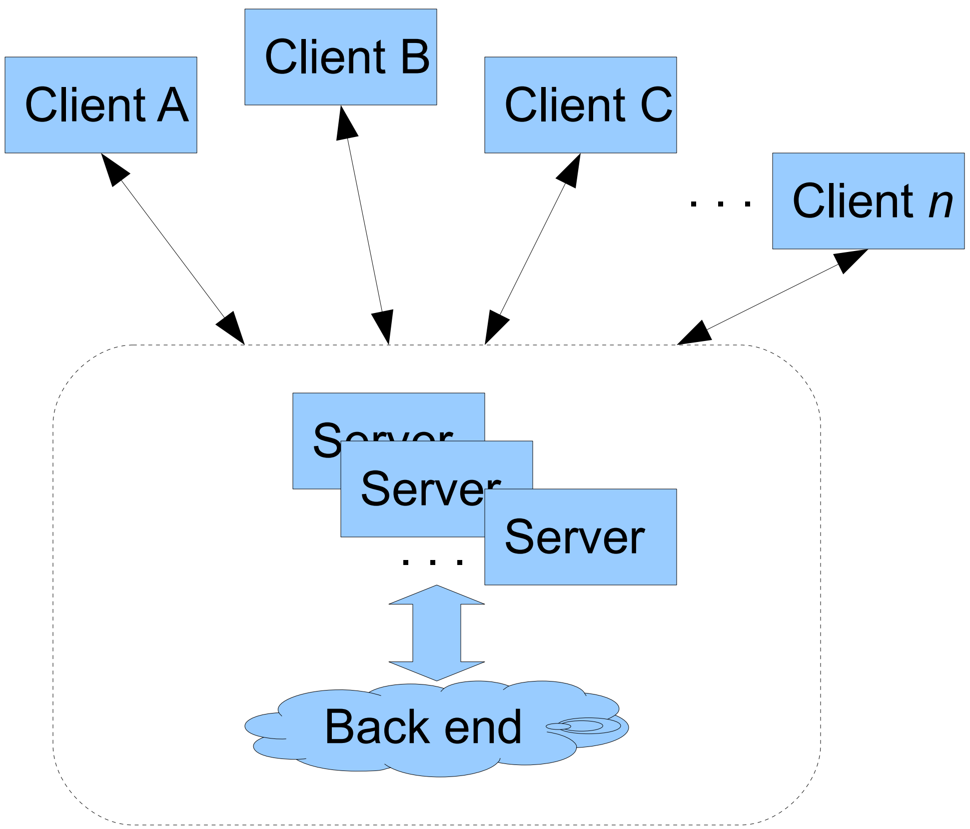
- Lots of work in the financial services space
  - Exchanges
  - Investment banks
  - Trading floors
  - And so on...
- Most are message-processing applications
- Throughput is important, but so is latency

Customer Case Study

- For example:
  - Order requests from distributed clients
  - Order execution
  - Response sent back to client
  - Important factors:
    - Round-trip response time subject to quality of service agreement
    - Requests per second

Customer Case Study

> Architecture:



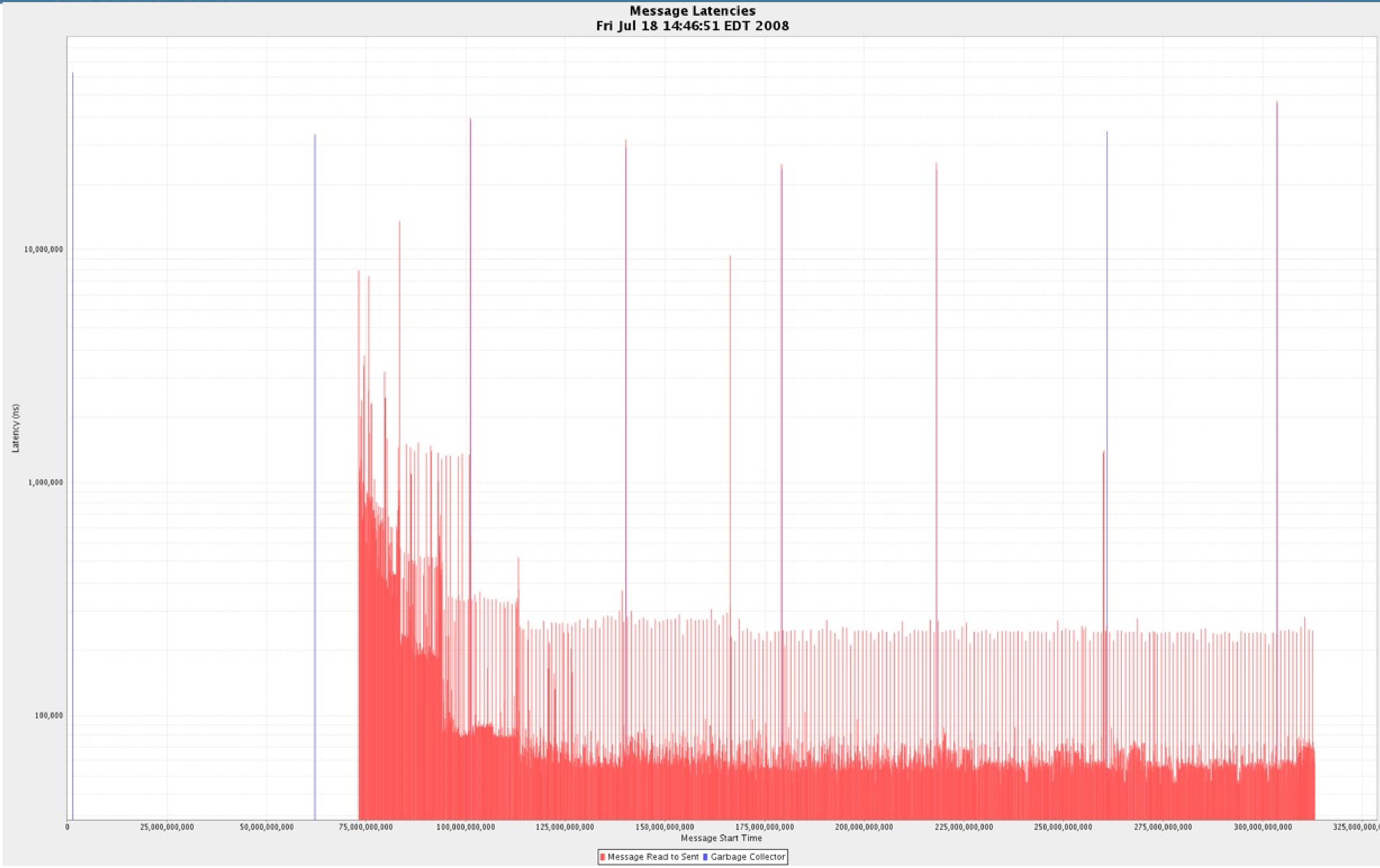
Customer Case Study
The Goal

- Reduce latency
- Remove all outliers
- Maintain throughput as much as possible

Customer Case Study

One Example

- Common Java SE issues:
  - GC pauses
  - Lack of thread priorities
  - Lack of priority inversion control
- Let's examine a graph on Java SE...

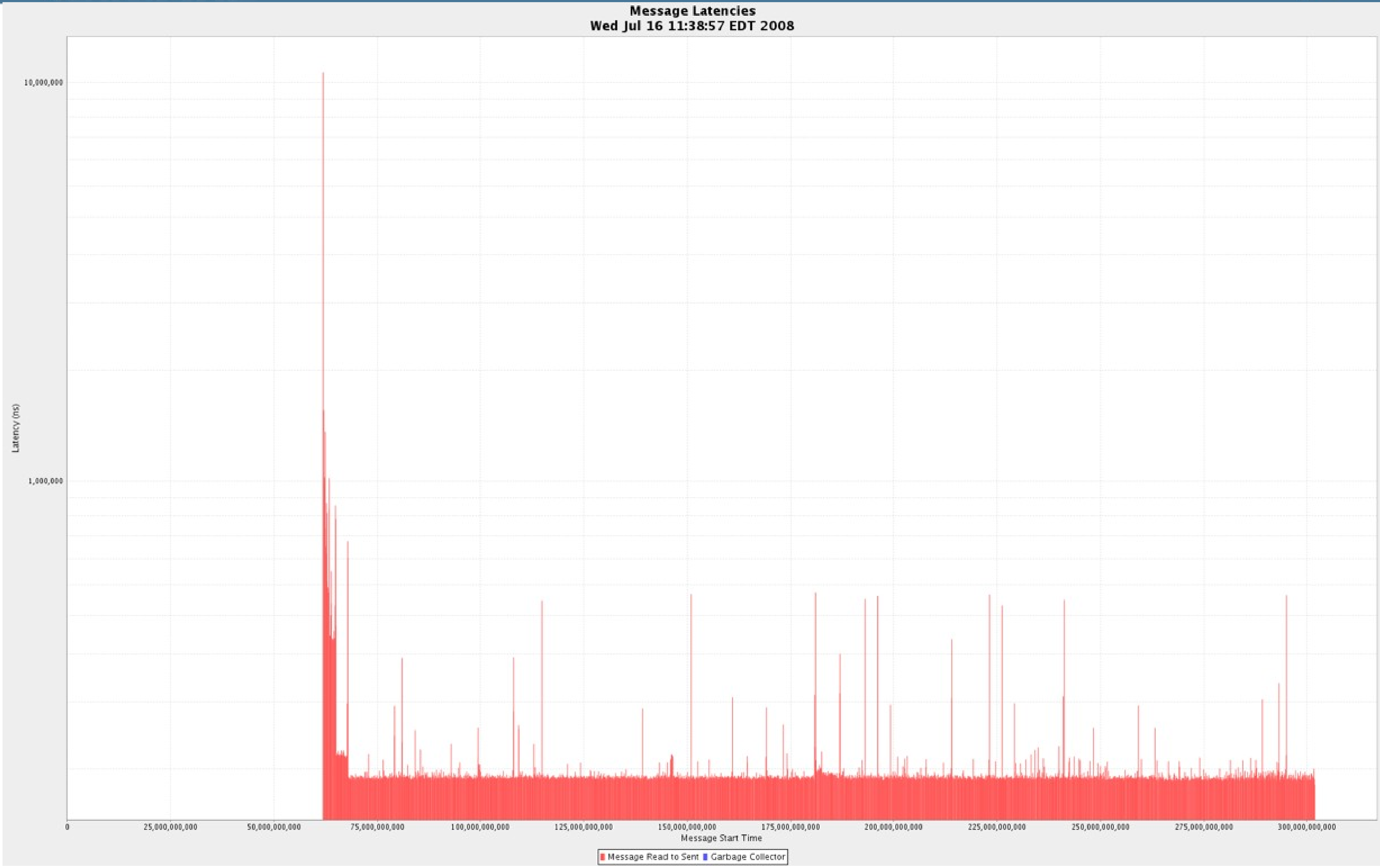

Message Latencies
Fri Jul 18 14:46:51 EDT 2008
10,000,000
1,000,000
100,000
Latency (ns)
0
25,000,000,000
50,000,000,000
75,000,000,000
100,000,000,000
125,000,000,000
150,000,000,000
175,000,000,000
200,000,000,000
225,000,000,000
250,000,000,000
275,000,000,000
300,000,000,000
Message Start Time
Message Read to Sent
Garbage Collector

Customer Case Study
Real-time Approach

- Common approach:
  - Quantify memory requirements and allocation rate
  - Identify time-critical code path
  - Use `javax.realtime.RealtimeThread` (RTT)
  - Apply appropriate priority to each RTT
  - Methodical testing with careful measurement
  - Use DTrace and TSV

Customer Case Study
Real-time Approach, continued

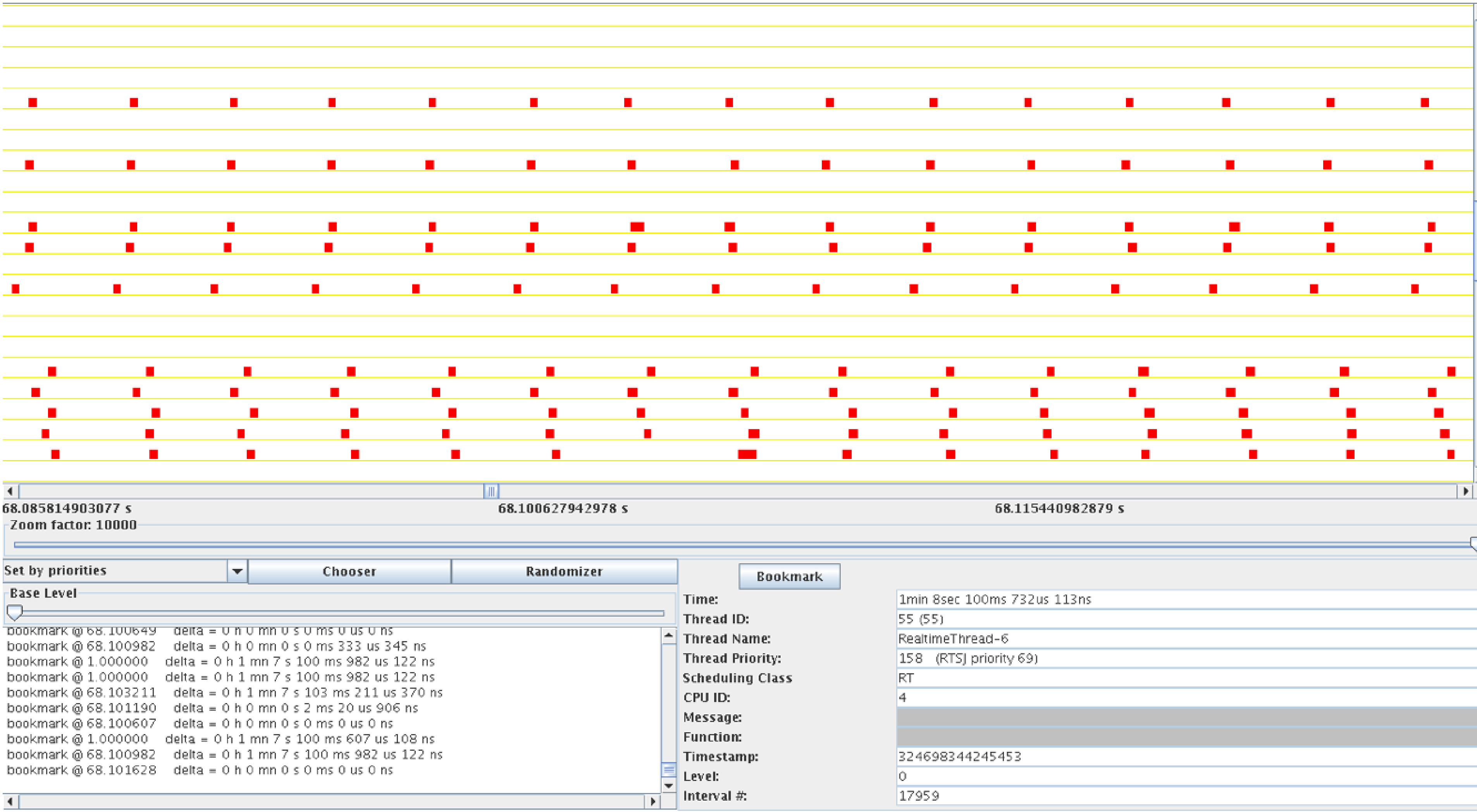
- Some other considerations:
  - Use the RTSJ clock API
  - Possible RTGC tuning
  - Processor sets and RTT binding
  - Initialization-Time Compilation
  - Possible use of critical reserved bytes
- Same application tuned for Java RTS...


Message Latencies
Wed Jul 16 11:38:57 EDT 2008
10,000,000
1,000,000
Latency (ns)
0
25,000,000,000
50,000,000,000
75,000,000,000
100,000,000,000
125,000,000,000
150,000,000,000
175,000,000,000
200,000,000,000
225,000,000,000
250,000,000,000
275,000,000,000
300,000,000,000
Message Start Time
Message Read to Sent
Garbage Collector

Customer Case Study

- Most satisfied with
  - RTGC – removes GC interference
  - Ability to measure and tune with Java RTS
  - Control:
    - Java RTS gives control to the developer for many run-time factors
  - Tools
    - DTrace and TSV, in particular...

The Thread Scheduling Visualizer (TSV):

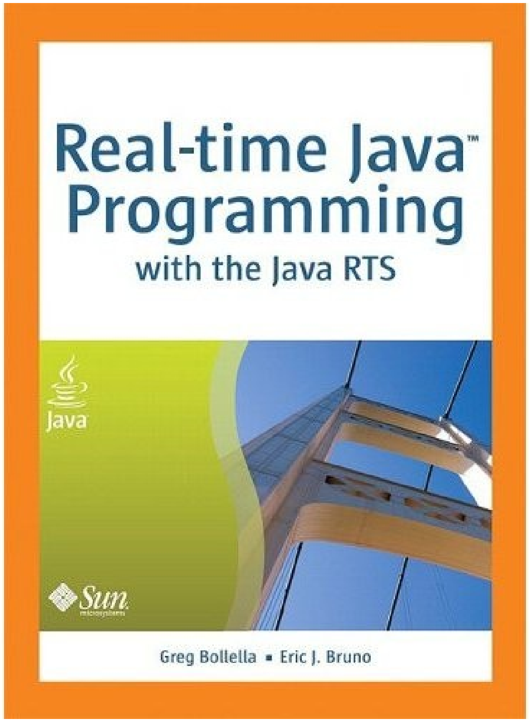


Summary

- SE implementations can be tuned for low latency to some extent...
- … But can't offer guaranteed determinism.
- Need to balance throughput and determinism/latency

Java RTS Book

- "Real-Time Java Programming: With Java RTS"
  - By Eric J. Bruno and Greg Bollella
  - http://my.safaribooksonline.com/9780137153626



JavaOne

Thank You

Roland Westrelin
Roland.Westrelin@Sun.COM

Eric Bruno
eric.bruno@sun.com
www.ericbruno.com