Real Time: Understanding the Trade-offs Between Determinism and Throughput

Roland Westrelin, Java Real-Time Engineering,
Brian Doherty, Java Performance Engineering,
Sun Microsystems, Inc

TS-5609

Java

Sun microsystems

- Learn how response time determinism and throughput are related and how various Java™ technology implementations, including the Java Real-Time System (Java RTS) from Sun Microsystems, deliver performance according to these metrics.

Agenda

- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Agenda

- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Non Real-Time Performance

- "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

Real-Time Performance

- 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 is not real-fast
  - Predictability is the key
  - Hard Real-Time is not necessarily low latency
- Worst case defines performance
- Application scheduling can be proved feasible
  - Known Implementation/design flaw matters even if very unlikely

Real-Time Performance (continued)

- Hard vs soft Real-Time
- Measured with its own metrics:
  - Latency
  - Jitter
- Requires its own performance tools:
  - Tools that collect samples not sufficient

Example Temporal Constraints

- Deadline: a started task must complete by a given time
  - E.g., 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
  - E.g., stop button handler must respond within 500us of a press
- Jitter: Variance in the time interval between events
  - E.g., the input sensor must be sampled every 1ms +/- 100us

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
- Increased determinism typically comes at the expense of decreased throughput
- Some Implementation techniques used in OS or Java Virtual Machine (JVM™ machine):
  - Are incompatible with Real-Time
  - Or lead to higher latency/jitter

Agenda

- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Sun's Java Real-Time System

- Sun's implementation of the RTSJ (Real-Time Specification for Java platform):
  - JSR-001 compliant (version 1.0.2)
  - APIs and semantic enhancements to correctly reason about and control the temporal behavior of applications
  - High level portable abstractions
- Bring the advantages of Java platform to Real-Time
- Optimized Real-Time Java platform runtime based on Java HotSpot™ high performance virtual machine

Sun's Java Real-Time System (continued)

- Based on Java Platform, Standard Edition 5 (Java SE platform 5)
- Runs on Solaris™ Operating System, SPARC® technology, and x86/x64 platforms
- New: runs on Real-Time Linux on x86/x64 platforms (starting with 2.1)
- New: improved tools (2.1) including DTrace-based tools

Key RTSJ Features

- Scheduling and Dispatching
  - Managing schedulable objects
- Synchronization
  - Priority inversion avoidance
- Memory Management
  - Alternatives to the heap
- Asynchronous events and handlers
- Time, Clocks and Timers

Agenda

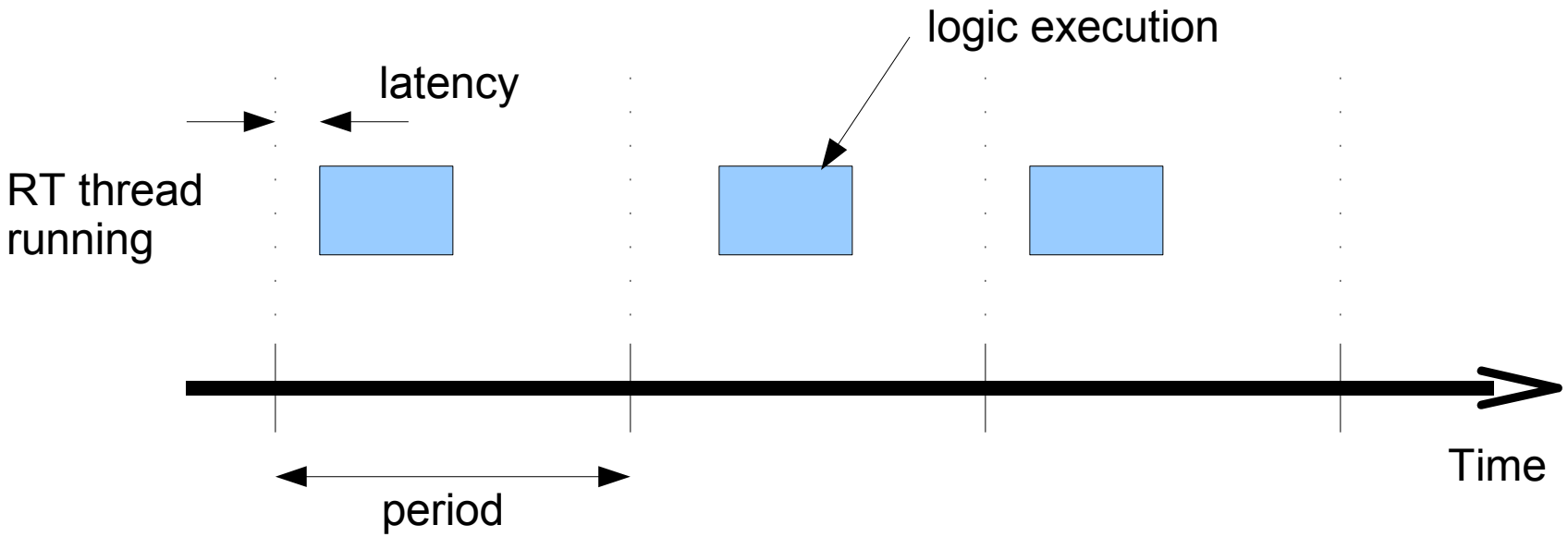
- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Response Time Benchmark

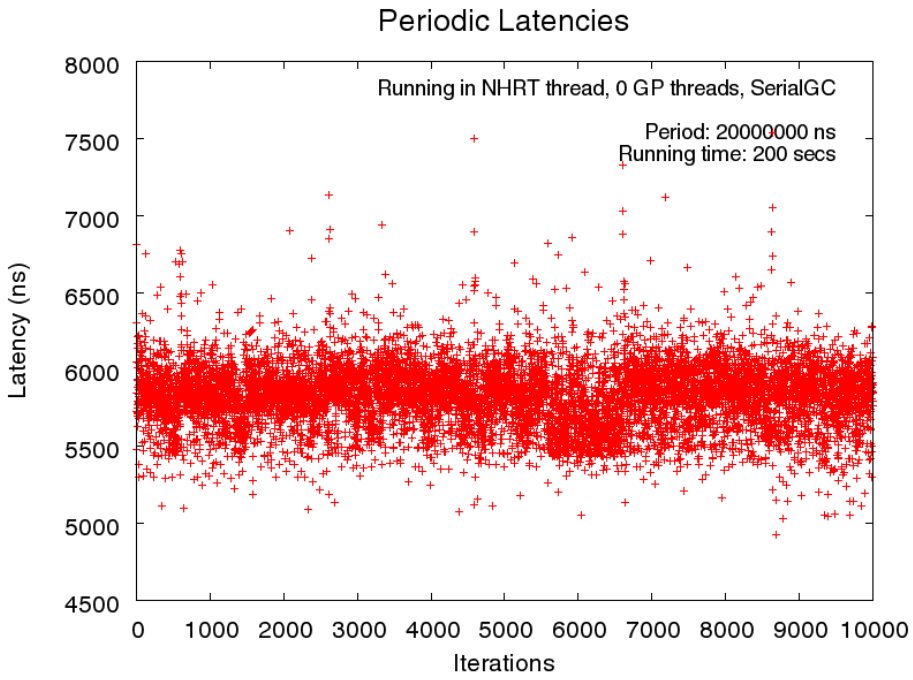
- Single Real-Time periodic thread
  - Wakes up at a pre-programmed fixed interval (absolute release dates)
  - Does not produce garbage in steady mode
- Simple but representative
  - Simple control loop: thread wakes up to read sensor and take some action
- At every release, wake-up latency is measured
- Benchmark measures:
  - Max difference between effective release time and expected release time is computed = max latency
  - Difference between smaller and larger latency = jitter

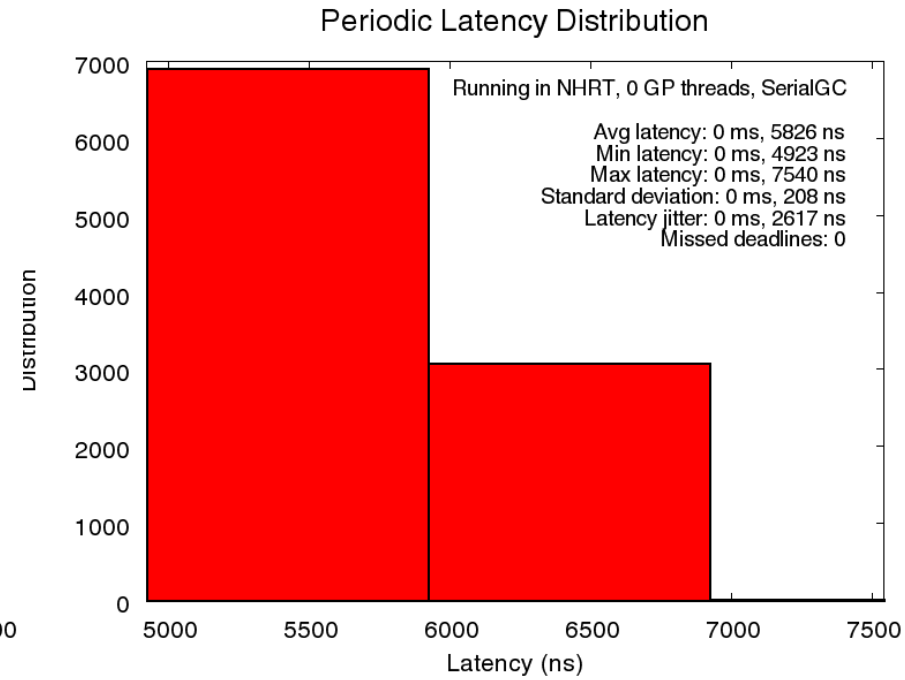
Response Time Benchmark (continued)

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



Response Time Benchmark: Sample Result



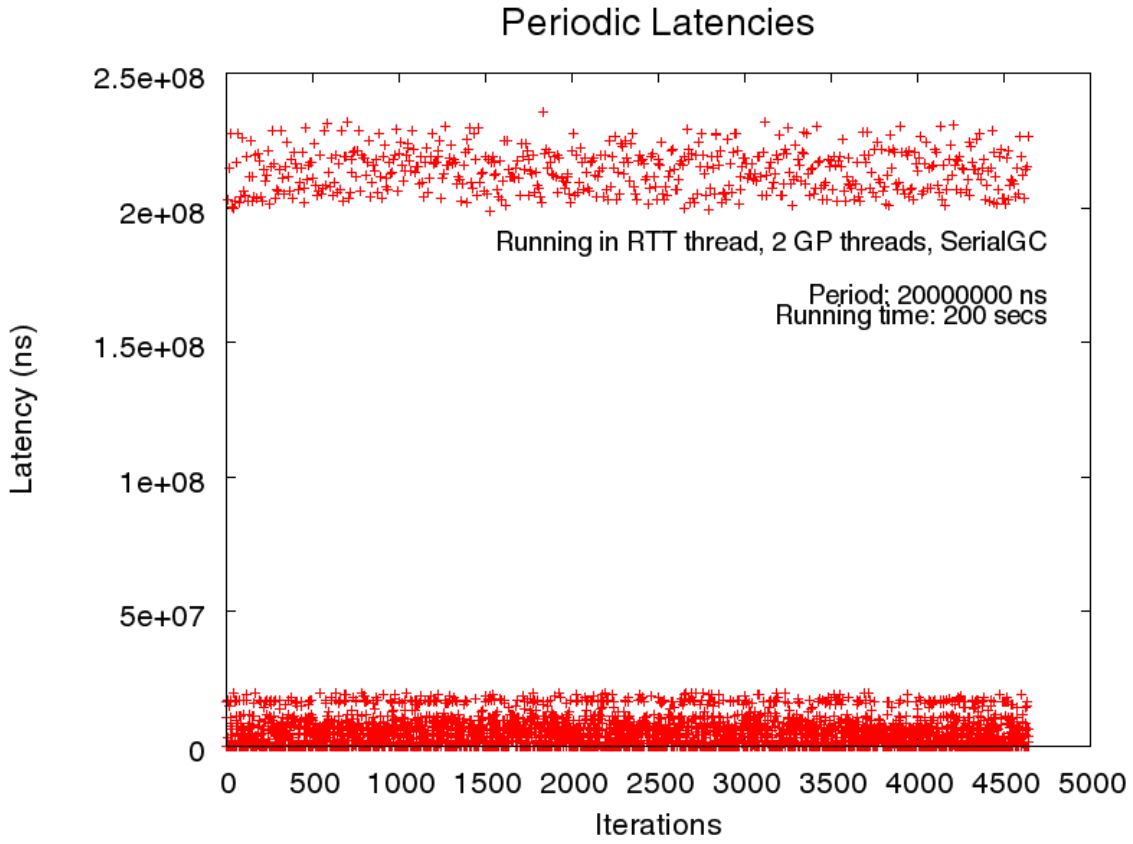


Java RTS 2.1 dev build, NHRT/Serial GC on Solaris/quad-core 2.8Ghz Opteron

Response Time: Load

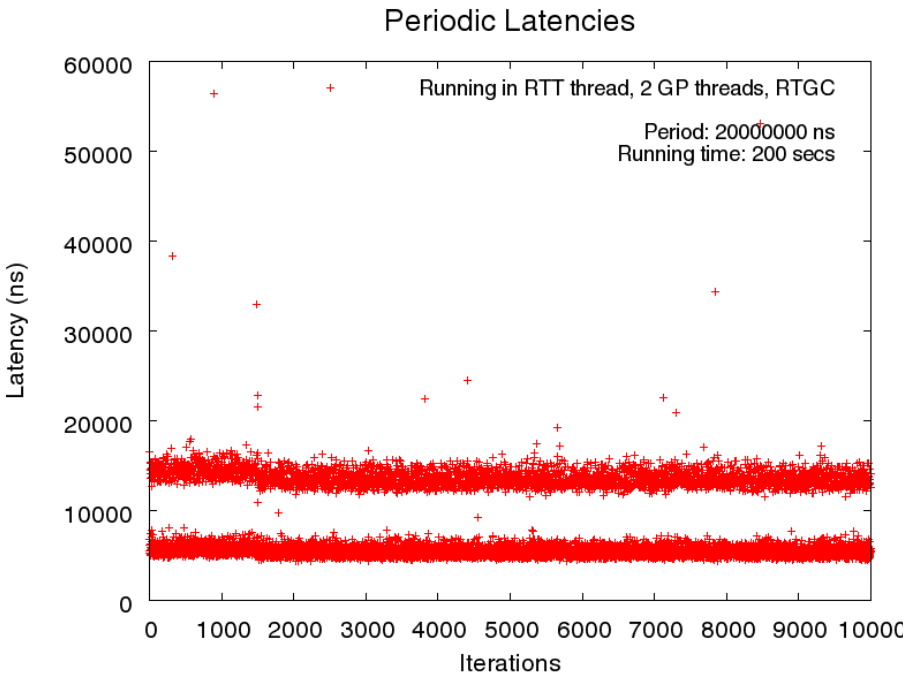
- Non Real-Time garbage producer threads
- Load the CPUs, exercise the system scheduler
- Triggers GC cycles: observe GC pauses
- Plan to add other types of load: I/O, network

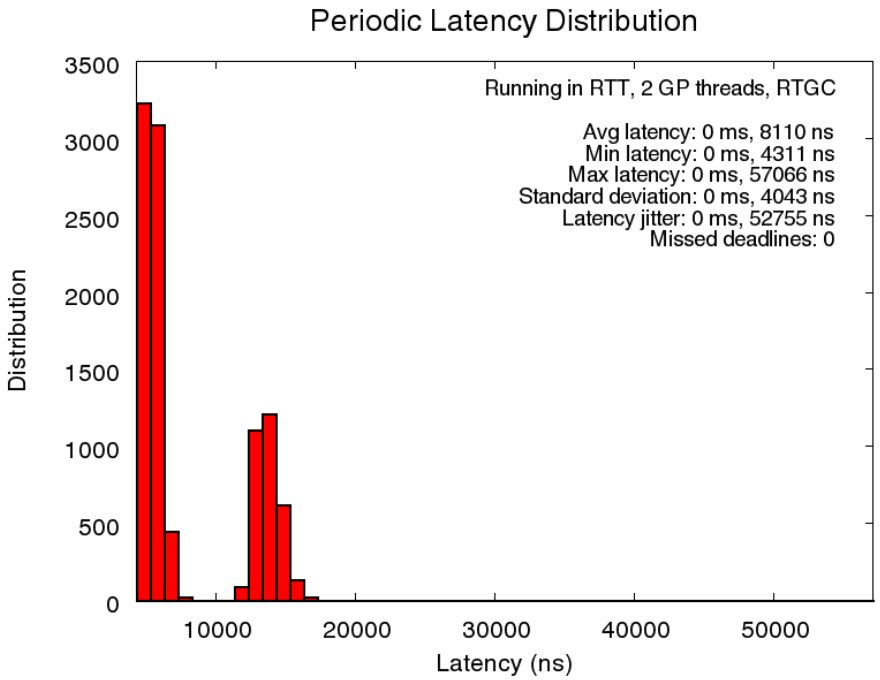
Response Time Benchmark: Non Real-Time GC



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

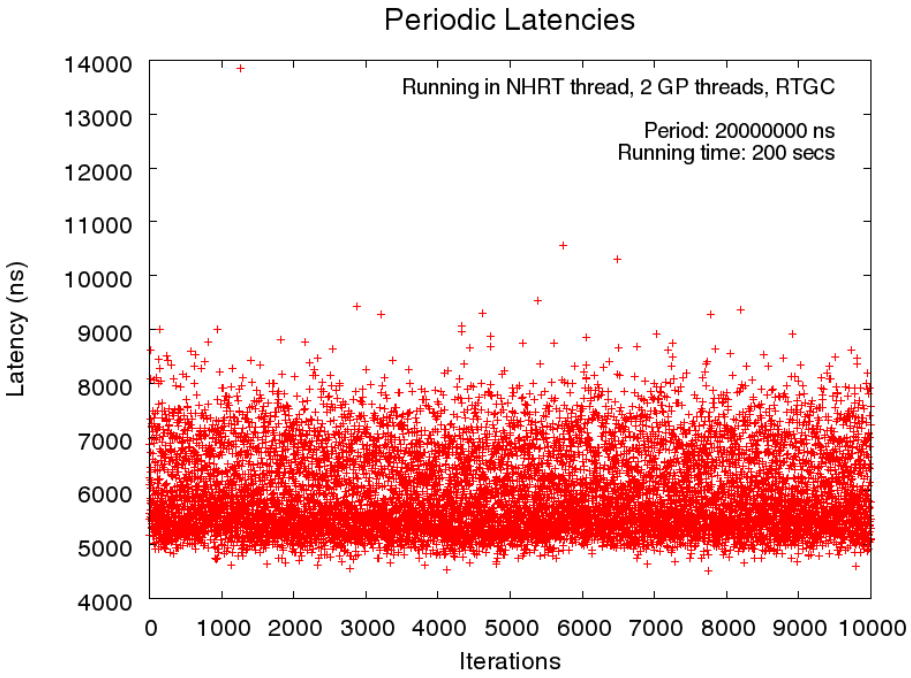
Response Time Benchmark: Real-Time GC

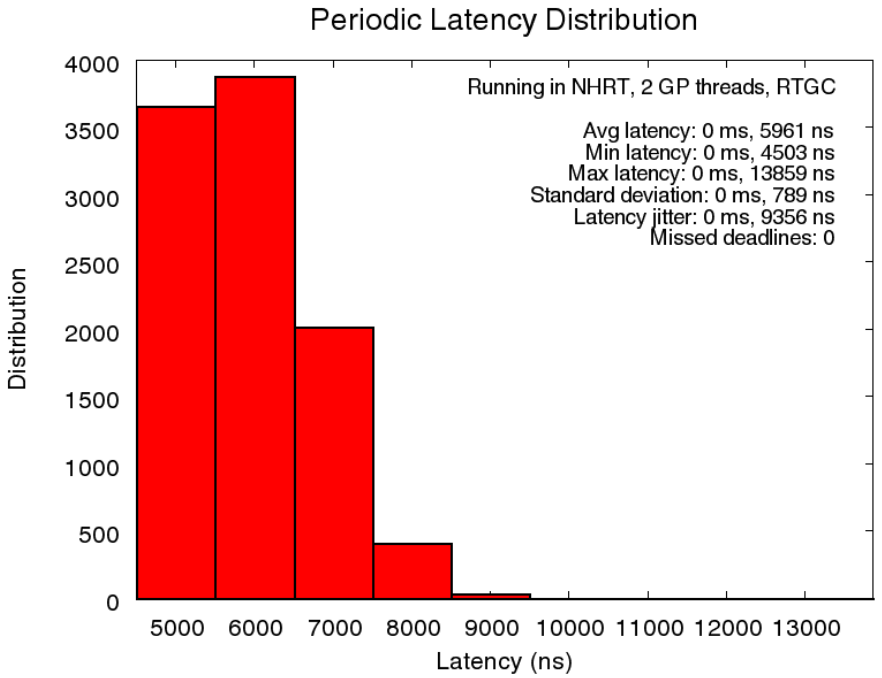




Java RTS 2.1 dev build, RTT/RTGC/2GP on Solaris/quad-core 2.8Ghz Opteron

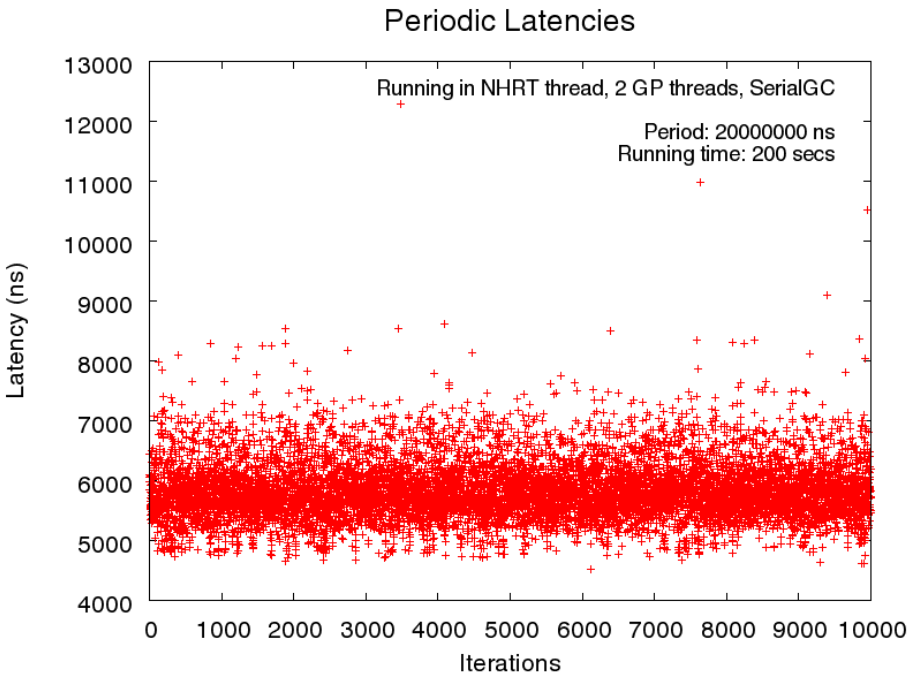
Response Time Benchmark: Real-Time GC (cont'd)

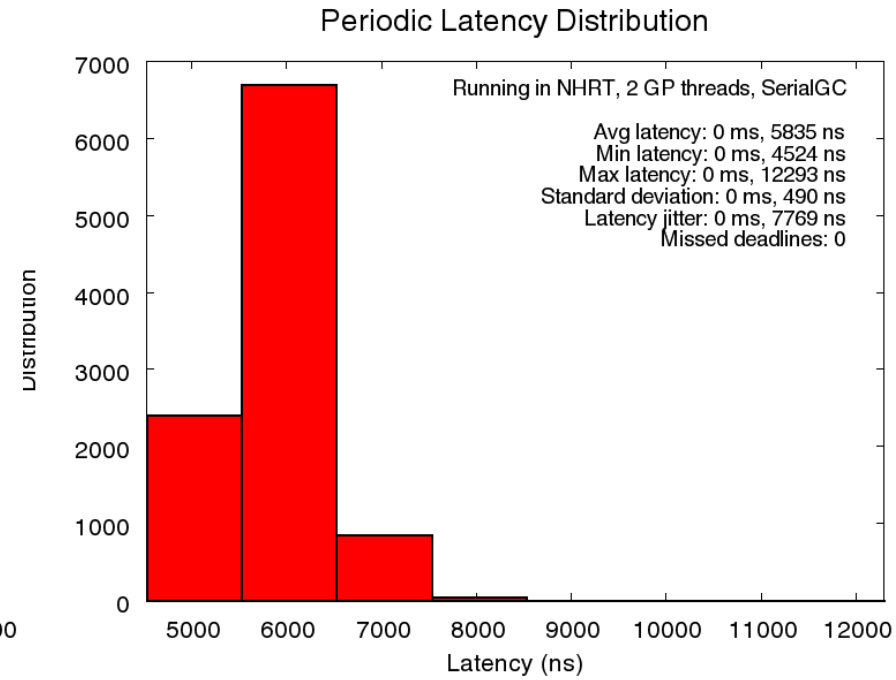




Java RTS 2.1 dev build, NHRT/RTGC/2GP on Solaris/quad-core 2.8Ghz Opteron

Response Time Benchmark: NHRT/serialGC





Java RTS 2.1 dev build, NHRT/serialGC/2GP on Solaris/quad-core 2.8Ghz Opteron

Agenda

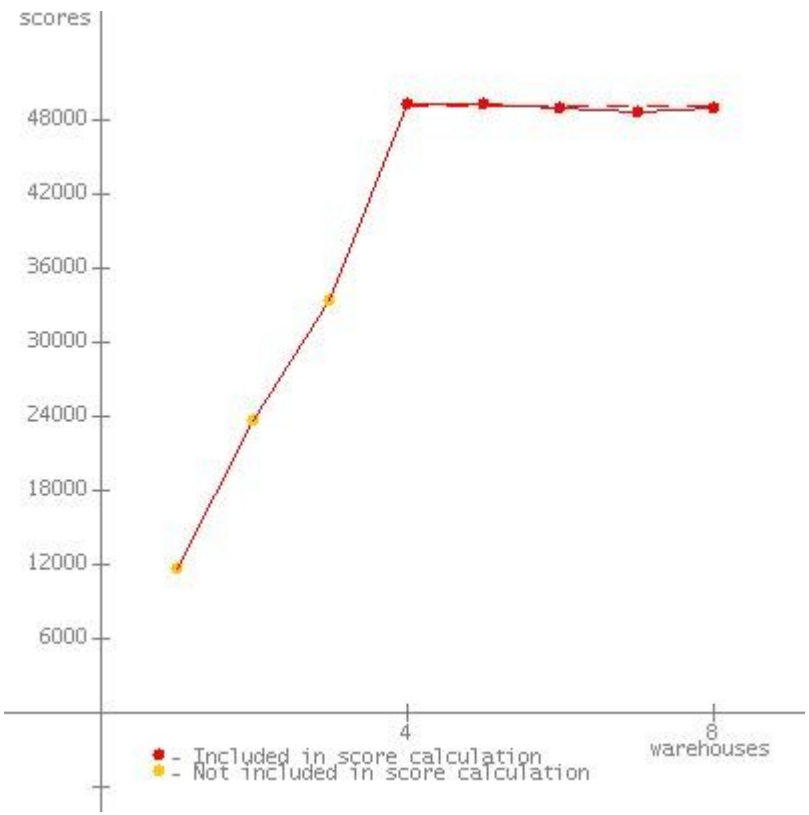
- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Throughput Benchmark

- SPECjbb®2005
  - Java Technology-based Business Benchmark
  - Simulates a 3-tier business system
  - Primary entity is the Warehouse
    - One thread per warehouse
    - Transactions manipulate and report on warehouse inventory
  - Two Phases
    - Warm up – warehouses increment from 1 to number of CPUs
    - Measurement – warehouses increment from number of CPUs to 2x number of CPUs
  - Goal – maximize throughput
    - Measures SPECjbb operations/sec (JBOPS)
    - Result is the average JBOPS per warehouse during the measurement phase
    - Response times are recorded, but not considered in the reported metric

SPECjbb2005 Example Output

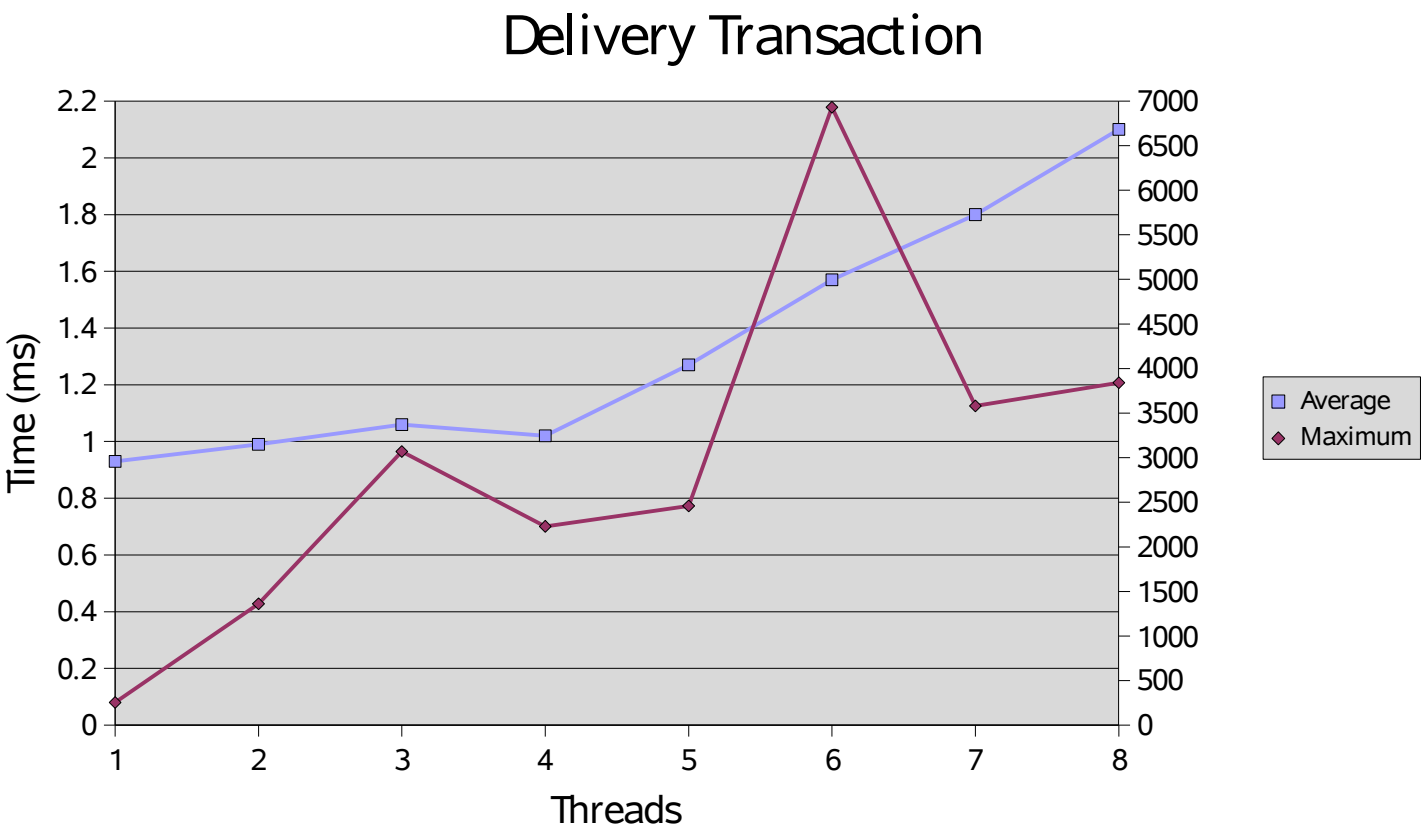
Warehouses	SPECjbb2005 Bops	Ind. In Metric
1	11716	
2	23688	
3	33393	
4	49384	*
5	49376	*
6	48998	*
7	48734	*
8	48984	*
SPECjbb2005	(from 4 to 8)	49097 SPECjbb2005 bops



Sun Fire X4200 (2 chips, 4 cores, 4 threads, 2.6 GHz) 49097 SPECjbb2005 bops, 49097 SPECjbb2005 bops/JVM.

SPEC and the benchmark name SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation. Results as of 12/05 on www.spec.org. For the latest SPECjbb2005 benchmark results, visit http://www.spec.org/osg/jbb2005

SPECjbb2005 Transaction Times



Sun Fire X4200 (2 chips, 4 cores, 4 threads, 2.6 GHz) 49097 SPECjbb2005 bops, 49097 SPECjbb2005 bops/JVM.

SPEC and the benchmark name SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation. Results as of 12/05 on www.spec.org. For the latest SPECjbb2005 benchmark results, visit http://www.spec.org/osg/jbb2005

Agenda

- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

SPECjbb2005®rt[1] Overview

- A research modification of SPECjbb2005
  - 'rt' == 'response time'
  - Throughput results ***can't*** be directly compared to SPECjbb2005 results
- Adds Injector threads
  - Asynchronously injects transactions at a prescribed rate
  - Injection rate automatically calibrated during warm-up or specified in the configuration file
- Adds a shared transaction queue between Injector threads and Warehouse threads
  - Transaction response time metric computed as completion time - expected enqueue time
  - Expected enqueue time != actual enqueue time
    - Injector thread scheduling delays will not reduce transaction rate and scheduling delays are not ignored

1. SPECjbb2005rt is not an official SPEC® benchmark. It's simply a name for this research modification that reflects both the lineage of the benchmark, SPECjbb2005, and how it differs from that lineage. Should this modified benchmark be accepted by the SPEC organization, it may not be given this name.

SPECjbb2005®rt Overview (continued)

- RTSJ RealtimeThread support
  - Injector and Warehouse threads
  - Priority levels
  - Configured declaratively
- Restructured to step up load level incrementally rather than adding threads incrementally
  - Uses a fixed number of warehouses, typically fewer than the number of CPUs
- Adds per-transaction type response time histograms

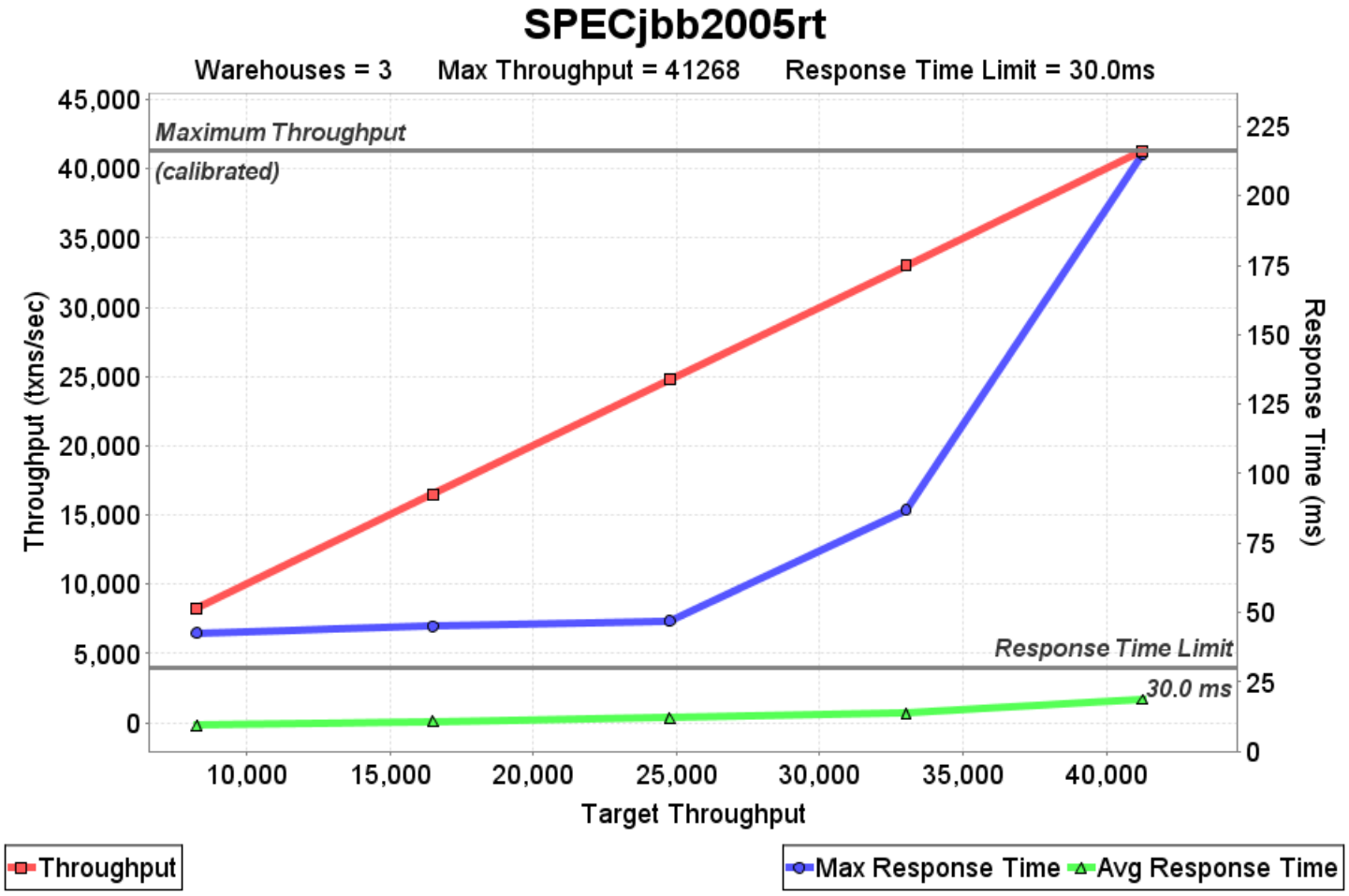
SPECjbb2005®rt Overview (continued)

- Three Phases
  - Warm up
    - Single warehouse, synchronous injection
  - Calibration
    - Find the theoretical maximum throughput from one warehouse
    - Synchronous injection
    - Expected maximum throughput computed as the average over 3 measurements.
    - Can be skipped by specifying a fixed maximum
  - Measurement
    - Up to 1 warehouse per CPU, typically less
    - Throughput stepped up from 20% to 100% of maximum at 20% increments
      - Increment amount and number of increments are tunable

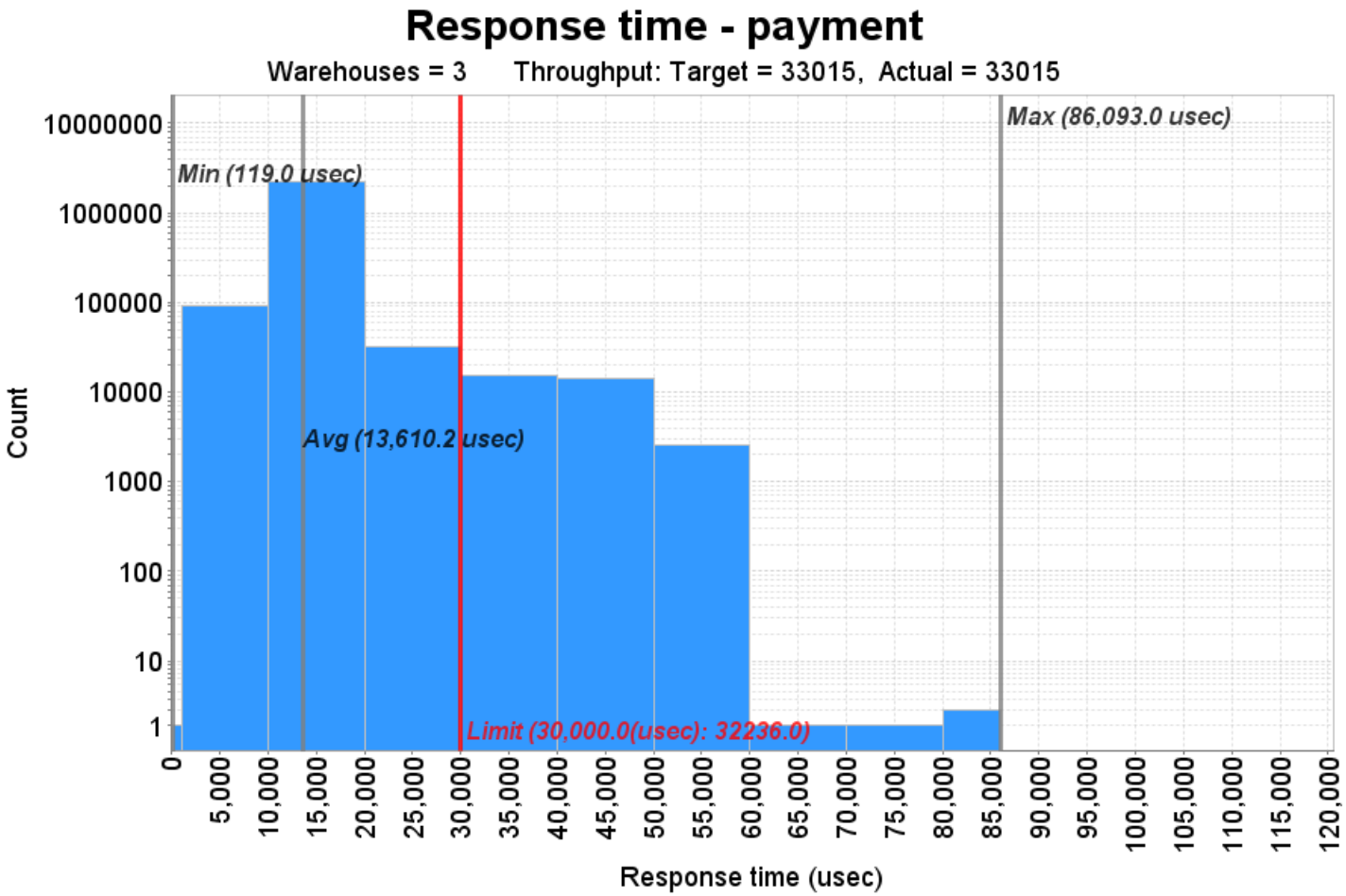
SPECjbb2005®rt Overview (continued)

- Result: JrtBB Ops/sec (JrtBOPS) @ <= max response time
  - Maximum throughput for which the maximum response time requirement is met.
  - Can be used to compare throughput levels at a specific maximum response time of interest
  - Can be used to compare response time distributions at a specific throughput of interest
  - Can be used to compare real-time qualities of different deployment platforms
    - Hardware platforms
    - Operating Systems
    - Java implementations
  - Queuing statistics recorded, but not used in result metric

SPECjbb2005rt Example Output



SPECjbb2005rt Example Output



SPECjbb2005rt Example Output

Transaction Type	Txn Count	Txn Exceeds 30000 (us)	Total Time (sec)	Min Time (us)	Max Time (us)	Avg Time (us)	StdDev Time (us)
Payment	2401207	32236	32681	119	86093	13610	3922

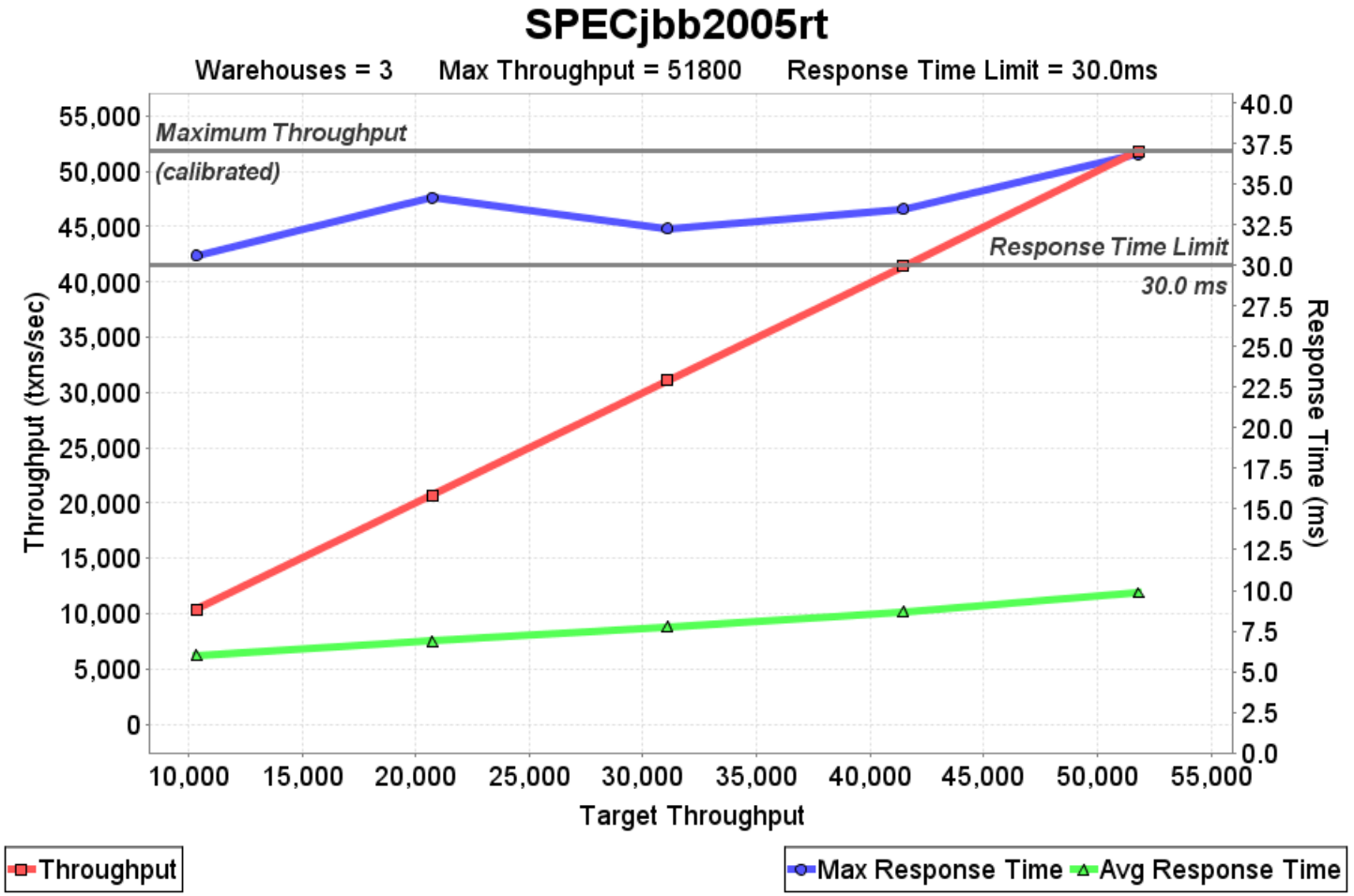
Comparisons

- Three different Java platform implementations
  - Sun's Java SE platform implementation with the throughput collector on 64-bit Solaris
  - Third party Java SE platform implementation with a deterministic pause time collector on 64-bit Windows
  - Sun's RTSJ Implementation with the real time garbage collector on 64-bit Solaris
    - Sun's Java RTS is not offered on Windows at this time
- Fixed injection rate of 17000 tx/sec
  - Overload throughput for Java RTS
- 60 minute measurement interval per throughput step
- 30ms maximum pause time
  - Artificial maximum, could be set to any desirable limit
  - Application dependent based on transaction execution times and other factors

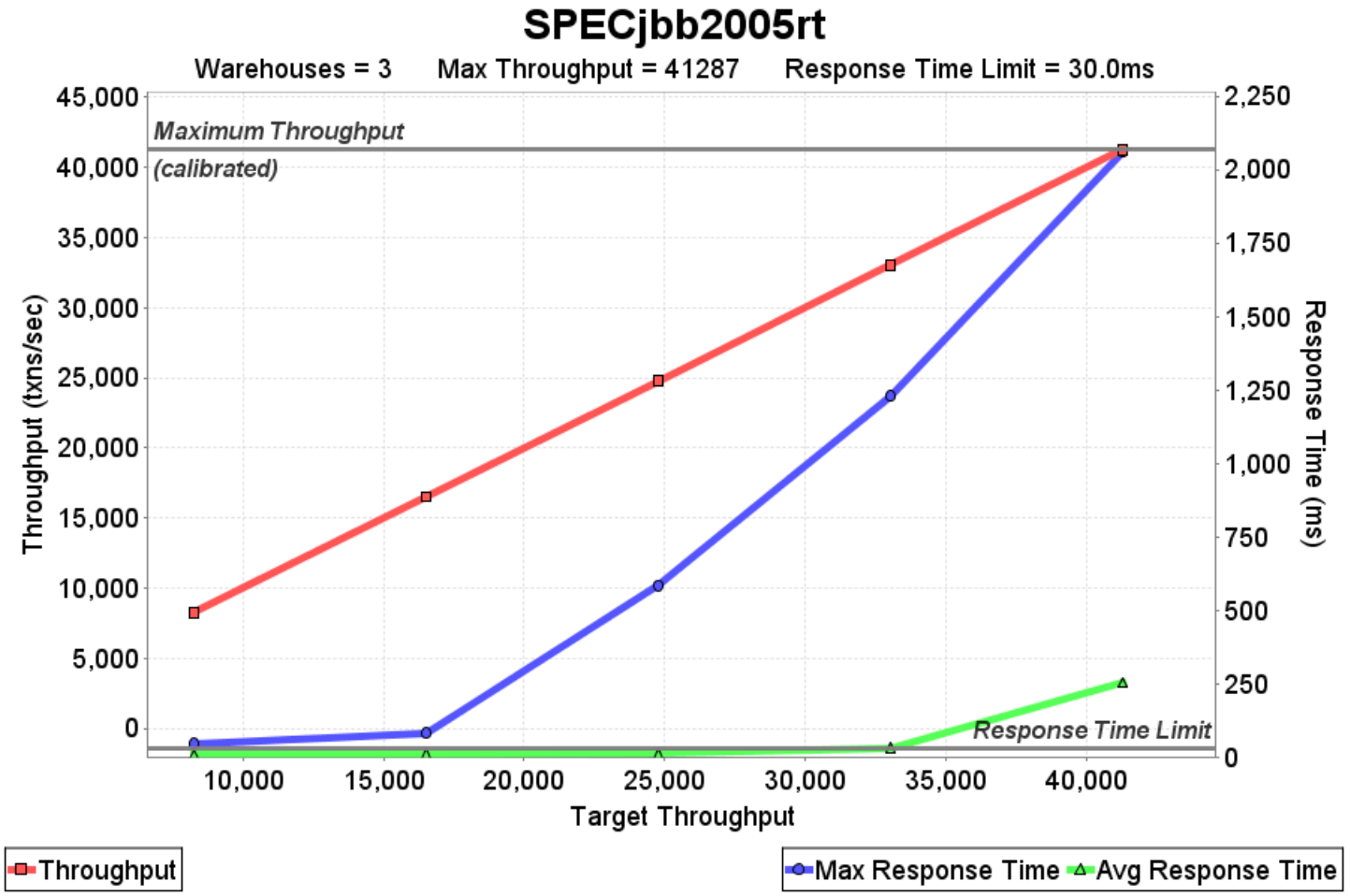
Comparisons (continued)

- Hardware
  - Sun Ultra™ 24 Workstation
    - Intel Core 2 Extreme Processor Q6850 @ 3.0GHz
      - 1 Chip, 4 Cores
    - 8GB RAM
- Software
  - Solaris Operating System 10 8/07 (Update 4)
    - Sun Java RTS 2.0 Update 1
    - Sun Java SE platform 1.7.0 b26
  - Windows XP – 64-bit, SP2
    - 3rd Party Java SE platform with a soft real-time garbage collector

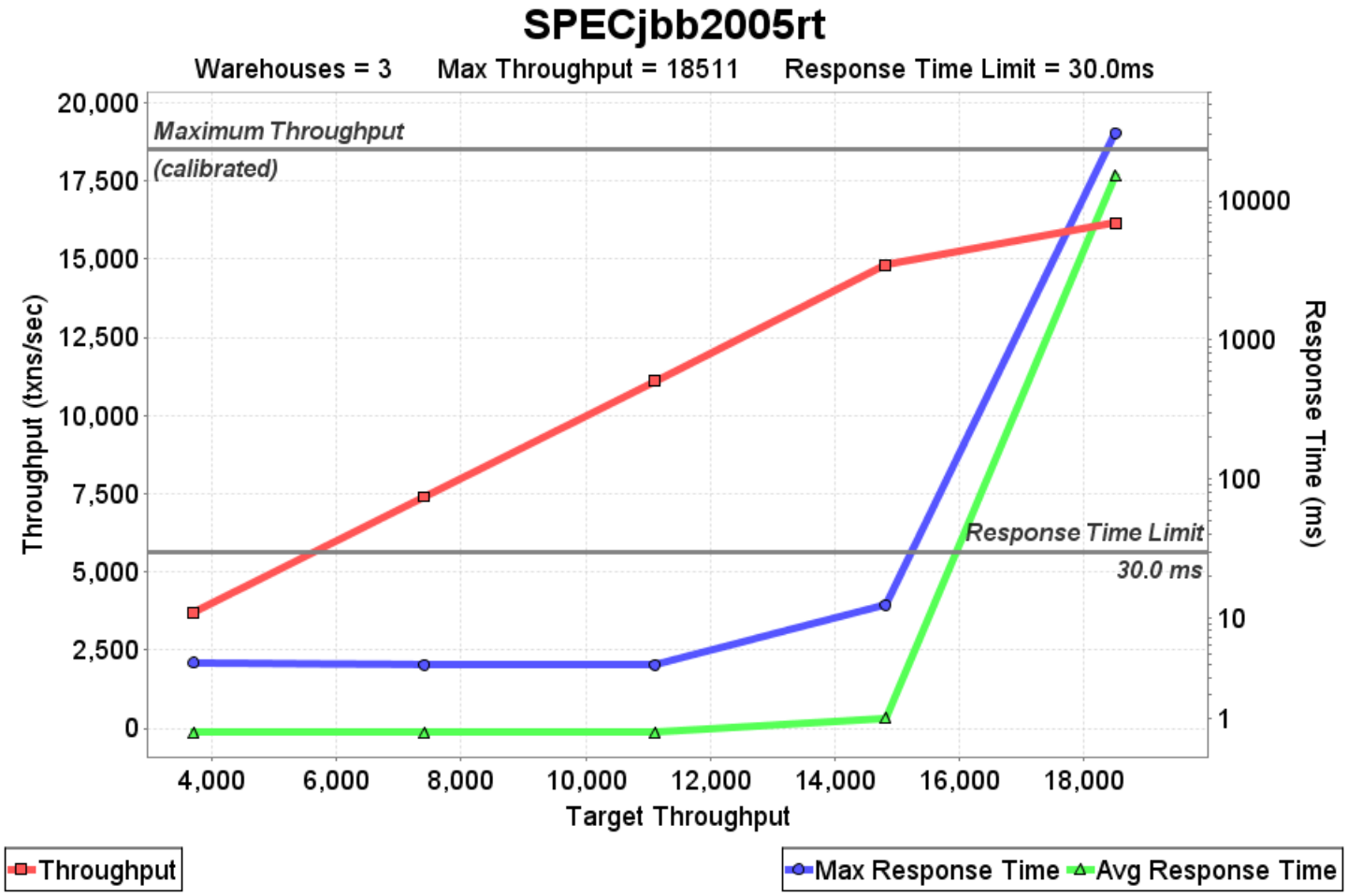
Sun's Java SE – self calibrated



Third party Java SE – self calibrated



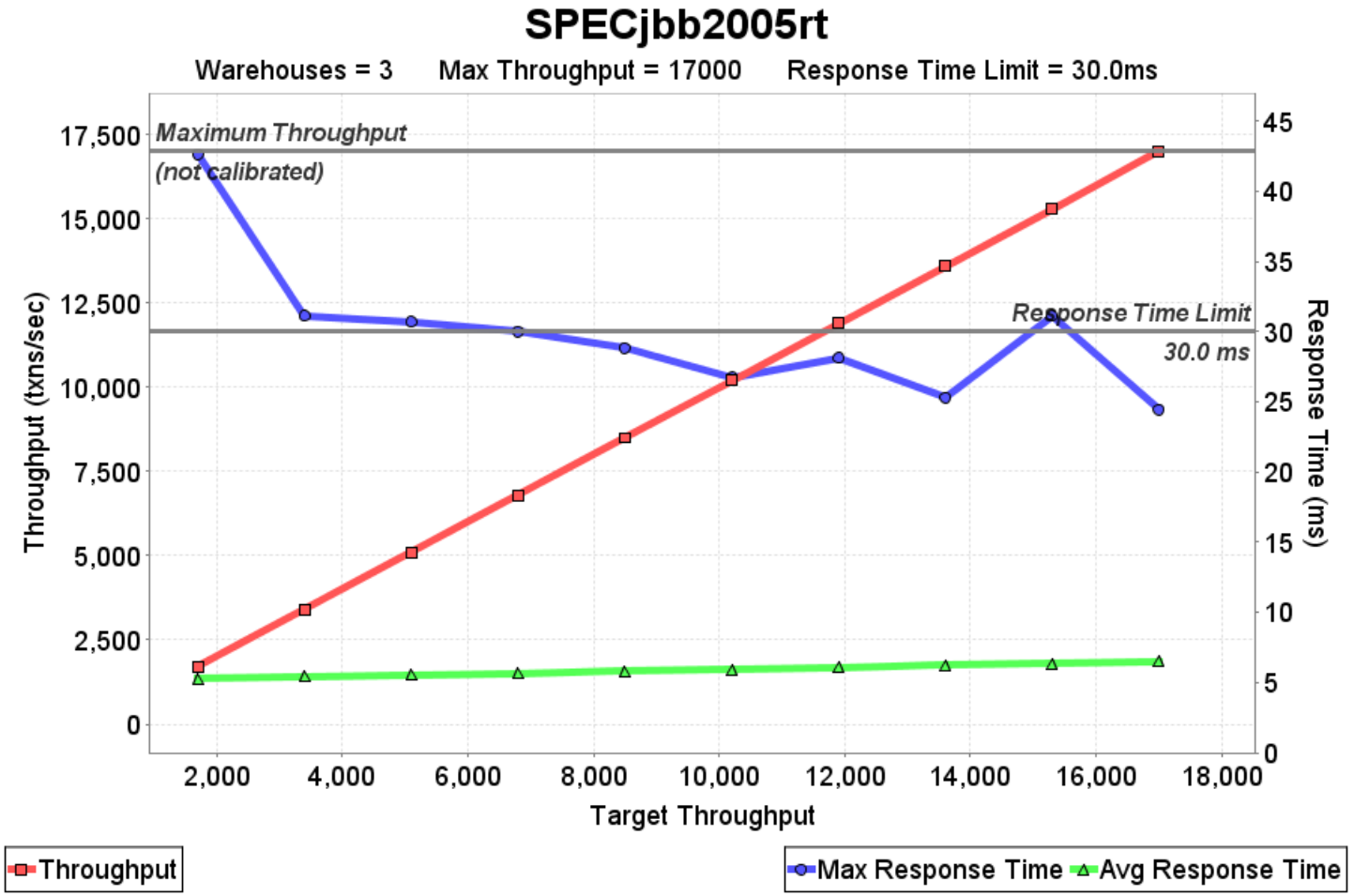
Sun's Java RTS – self calibrated



Calibration results

- As expected, each implementation is driven to overload
- Java RTS shows a significantly lower overload throughput than the Java SE implementations
  - Or does it?
- Rerun the benchmark with a fixed throughput of 17000 txn/sec – the Java RTS overload point.

Sun's Java SE – fixed throughput



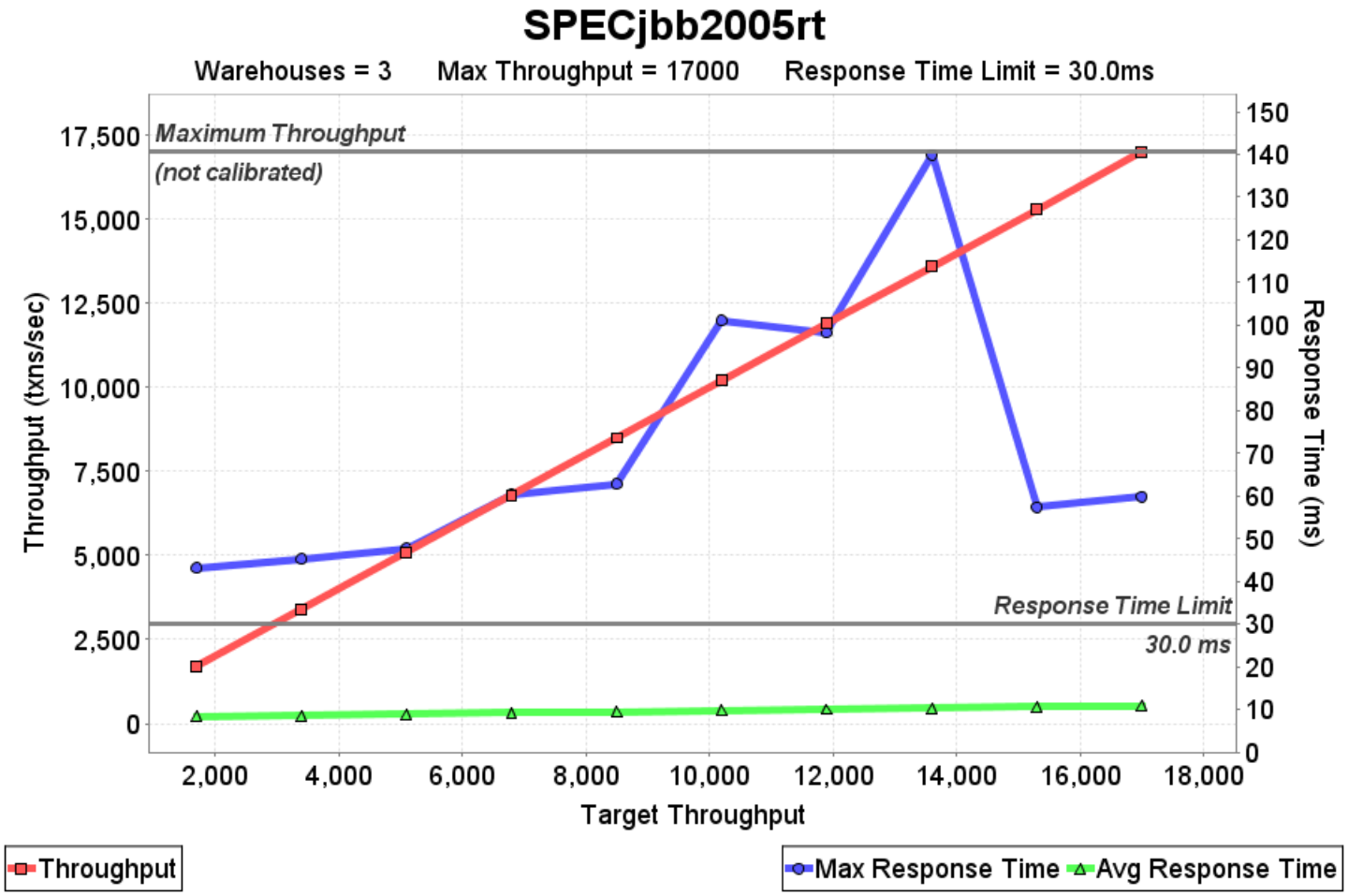
Sun's Java SE

Target Bops	Actual Bops	Avg Response (ms)	Max Response (ms)	Stdev
1700	1700	5.30	42.66	2.83
3400	3400	5.42	31.17	2.79
5100	5100	5.55	30.70	2.71
6800	6800	5.68	30.00	2.64
8500	8500	5.82	28.84	2.57
10200	10198	5.94	26.74	2.50
11900	11907	6.09	28.16	2.42
13600	13600	6.23	25.30	2.37
15300	15300	6.36	31.14	2.32
17000	17000	6.50	24.47	2.27

Sun's Java SE Results

- Average response times are good
- Maximum response times exceed the 30ms threshold at some throughput levels
  - Generational garbage collection delivers reasonable results
  - Assumes Full GC events are avoided
- Implementation is capable of delivering 51800 tx/sec, but can't deliver even 2000 tx/sec without exceeding the maximum response time criteria
- Some soft real-time applications might be satisfied with this level of performance
  - Assumes periodic response time spikes are tolerable
  - Assumes that sustained throughput doesn't achieve overload conditions
  - Assumes that Full GC events can always be avoided.

Third party Java SE – fixed throughput



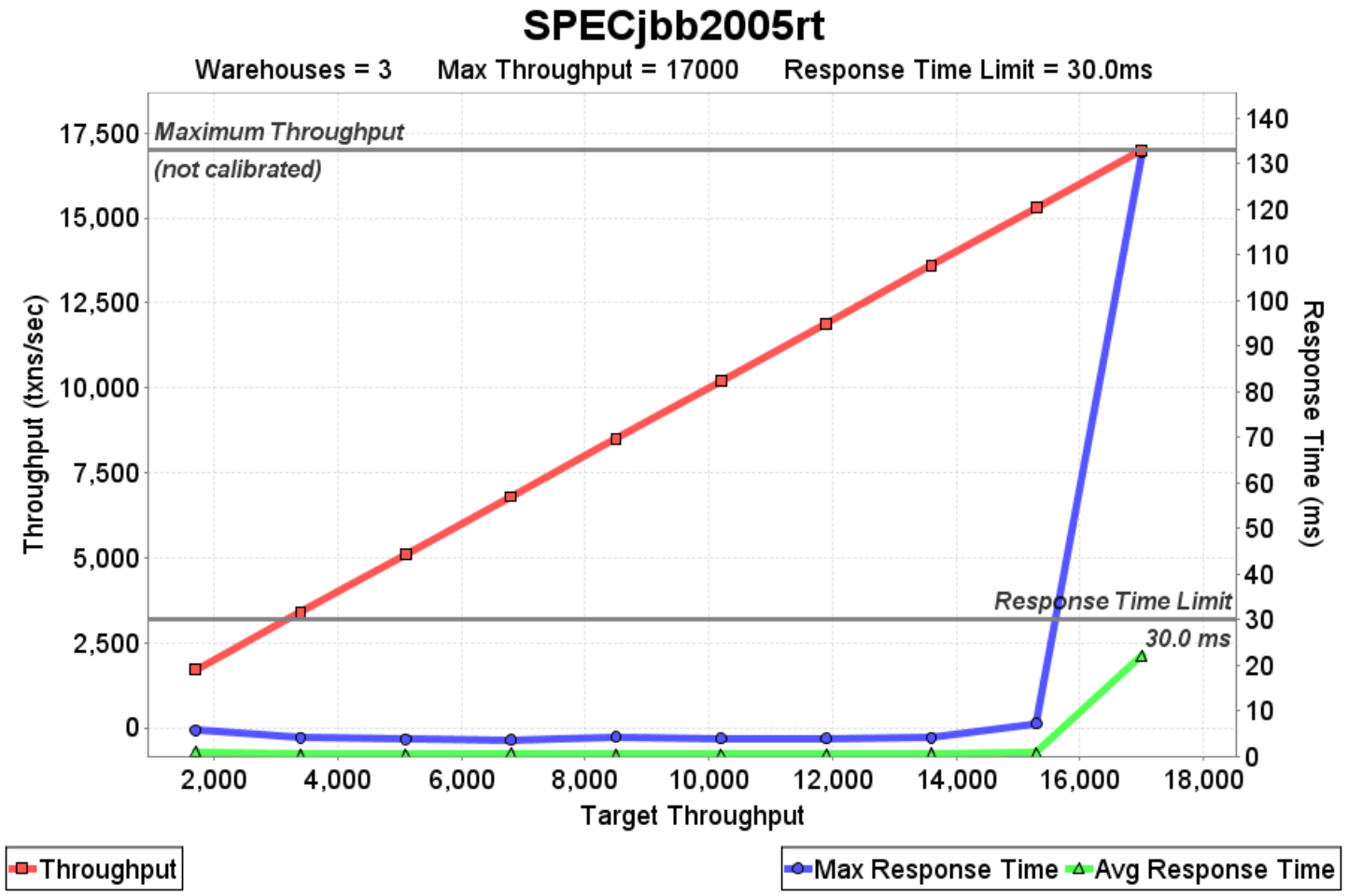
Third party Java SE implementation

Target Bops	Actual Bops	Avg Response (ms)	Max Response (ms)	Stdev
1700	1700	8.30	43.16	4.38
3400	3400	8.56	45.22	4.24
5100	5100	8.88	47.78	4.07
6800	6800	9.16	60.39	3.94
8500	8500	9.47	62.80	3.81
10200	10200	9.72	101.04	3.76
11900	11901	10.04	98.27	3.63
13600	13601	10.30	139.93	3.54
15300	15301	10.58	57.32	3.39
17000	17001	10.81	59.75	3.28

Third party Java SE results

- Average response times are good
- Maximum response times exceed the 30ms threshold at all throughput levels
  - Concurrent garbage collection delivers good average response times that meet the response time criteria but fails to deliver maximums within the expected limits
- Implementation is capable of delivering 41287 tx/sec, but can't deliver even 2000 tx/sec without exceeding the maximum response time criteria
- Some soft real-time applications might be satisfied with this level of performance
  - Assumes maximum response time spikes are tolerable or otherwise avoidable
  - Assumes that sustained throughput doesn't achieve overload conditions

Sun's Java RTS – fixed throughput



Sun's Java RTS

Target Bops	Actual Bops	Avg Response (ms)	Max Response (ms)	Stdev
1700	1700	1.01	5.74	0.14
3400	3400	0.56	4.21	0.19
5100	5100	0.57	3.85	0.15
6800	6800	0.65	3.72	0.10
8500	8500	0.57	4.28	0.12
10200	10200	0.60	3.94	0.13
11900	11900	0.60	3.94	0.16
13600	13600	0.64	4.07	0.22
15300	15300	0.81	7.25	0.33
17000	17000	22.18	132.44	19.87

Sun's Java RTS results

- Average response times are very good
- Maximum response time limit never exceeded
  - Real-time garbage collection delivers sub-millisecond results
  - At 17000 tx/sec the average response time spikes to 22.2ms
  - At 17000 tx/sec the maximum response time exceeds the response time criteria
- Implementation is capable of delivering 15300 tx/sec with 0.81ms average and 7.25 max response times
- Most soft real-time applications are likely to be satisfied with this response time performance.
  - Assumes that sustained throughput doesn't achieve overload conditions

Throughput/Response time comparisons

Implementation	Actual Bops	Avg Response (ms)	Max Response (ms)	Stdev
Sun Java SE	15300	6.36	31.14	2.32
3rd Party Java SE	15300	3.39	57.32	3.34
Sun Java RTS	15300	0.81	7.25	0.33

Throughput Sacrifice?

- Real-time responsiveness typically results in a throughput trade-off when compared to pure batch type throughput results
- To achieve soft real-time response time distributions with non-real-time Java platform implementations, throughput sacrifices are needed
- Real-time Java platform implementations can deliver superior response time distributions at similar throughput levels as non-real-time alternatives

Agenda

- Real Time Performance
- RTSJ and Sun's Java RTS Product Offering
- Response Time Benchmark
- Throughput Benchmark
- Throughput and Response Time Benchmark
- Summary

Summary

- Real-time performance is defined in terms of temporal behavior
- Correct qualification of real-time performance requires well-suited metrics and benchmarks
- We presented 2 such benchmarks:
  - A simple response time benchmark
    - defines basic characteristics of a real-time Java implementation
  - A complex benchmark
    - illustrates the throughput vs response time trade-off
- Call to action: utilize the appropriate metrics and understand the trade-offs when evaluating real-time offerings

THANK YOU

Roland Westrelin
roland.westrelin@sun.com

Brian Doherty
brian.doherty@sun.com

TS-5609

For More Information

- Sessions
  - TS-5767: Real-Time Specification for Java: The Revolution Continues
  - TS-5716: D-I-Y (Diagnose-it-Yourself): Adaptive Monitoring for Sun Java Real-Time System
  - TS-4797: Fully Time-Deterministic Java Technology
- BOFS
  - BOF-6353: Meet the Java Real-Time and Java SE Embedded Teams
- Labs
  - LAB-7420: The Real-Time Programming Challenge on the Java Platform: How to Build Real-Time Solutions for Real-World Devices
- RTSJ and Java RTS
  - http://java.sun.com/javase/technologies/realtime
- Standard Performance Evaluation Corporation (SPEC)
  - http://www.spec.org

What Impacts Determinism

- The system
  - Hardware
  - Operating System
- In the virtual machine
  - Garbage Collector
  - Class loading
  - Just-In Time compiler

How Java RTS Application Controls Latency

- Garbage Collector
  - RTSJ memory areas
  - Real-Time Garbage Collector
- Class loading
  - Class preloading
- Just-In Time compiler
  - Asynchronous JIT
  - Precompilation at initialization time