What's Hot in BEA JRockit

Marcus Lagergren and Staffan Larsen

Java Runtime Products Group
BEA Systems

TS-3484

Goal

Understanding and being able to take advantage of some of the key technical innovations in JRockit.

Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

Profiling and Management Tools

Q&A

Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

Profiling and Management Tools

Q&A

Quick JRockit Facts

- A Java™ VM for enterprise-wide usage
- 100% compatible with all applicable Java technology standards
- Available for J2SE™ 1.4.2 and Java EE 5
  - Windows (IA32, x64, IA64)
  - Linux (IA32, x64, IA64)
  - Solaris (SPARC64)
- Java SE 6 coming
- Fast, manageable, and free (as in beer)

JRockit: Optimizing Java Technology

- Challenge and opportunity: Java technology is a runtime system, not a static environment
- The keyword is adaptivity
  - The entire runtime system does lots of data collection for free. Use it!
- Adaptive Optimization and GC
- Creative use of data that is collected “for free”
  - Near zero sampling overhead
  - Memory leak detection tools
  - JRA recordings/runtime analyses

JRockit: Optimizing Java Technology

- How to optimize an object-oriented language
  - Getters and setters
  - Virtual methods
  - Exceptions
- Need to make aggressive assumptions and "gamble" that they are correct
  - Take performance hits if assumptions are invalidated
  - e.g., Revirtualization, undoing optimizations
- Don't hand optimize code, leave it to the Java VM

JRockit: Optimizing Java Technology

- Optimizing a garbage collected language
- Adaptive garbage collection
  - Runtime strategy changes
- Need concurrent garbage collection
- Might even need real-time demands
  - Deterministic GC
  - Service Level Agreements
- Good out-of-the-box behavior
  - “Type ‘java’ and it works”

Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

Profiling and Management Tools

Q&A

A Deterministic Java VM

Definition

- In this presentation, we use the term "Deterministic GC"
  - This is means a GC with guaranteed upper bound for pause times
- "Deterministic GC" should not be confused with the behavior of a pure real-time system where no randomness exists

A Deterministic Java VM

- Java platform is a runtime system
  - This is inherently non-deterministic
  - There isn't (and shouldn't be) an exact way to control when GC happens
  - The hard part: The runtime system needs to handle GC optimally
  - The easy part: Well, it has all the data
- Throughput vs. response time
  - Keeping reponse times down
- Deterministic behavior
  - "No interruptions"

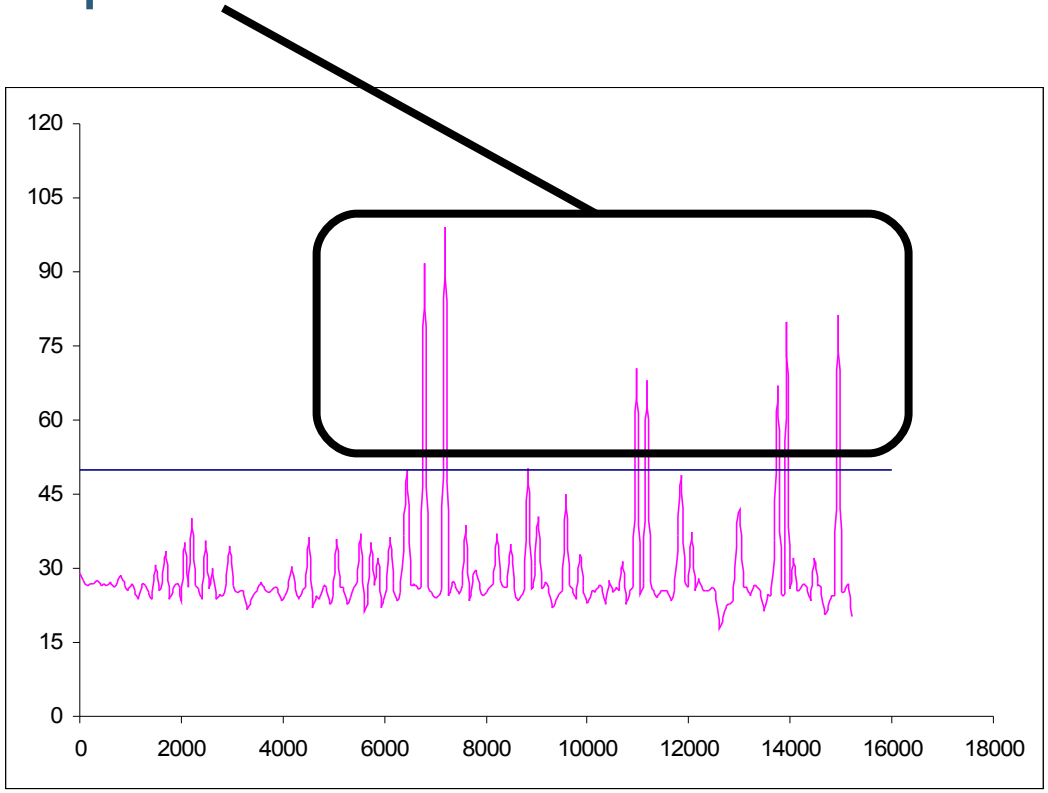
A Deterministic Java VM

Java technology is moving towards “real-time” applications

- SIP Server—Telecom (VOIP)
  - 50–100 ms response times
  - Maximize # calls set up per second
  - Longer response times means dropped calls (busy signal)
- Trading Processing—Financial Services
  - 10–20 ms response times
  - Maximize trades per seconds
  - Lower response times means more trade wins

Traditional VM—Non-Deterministic

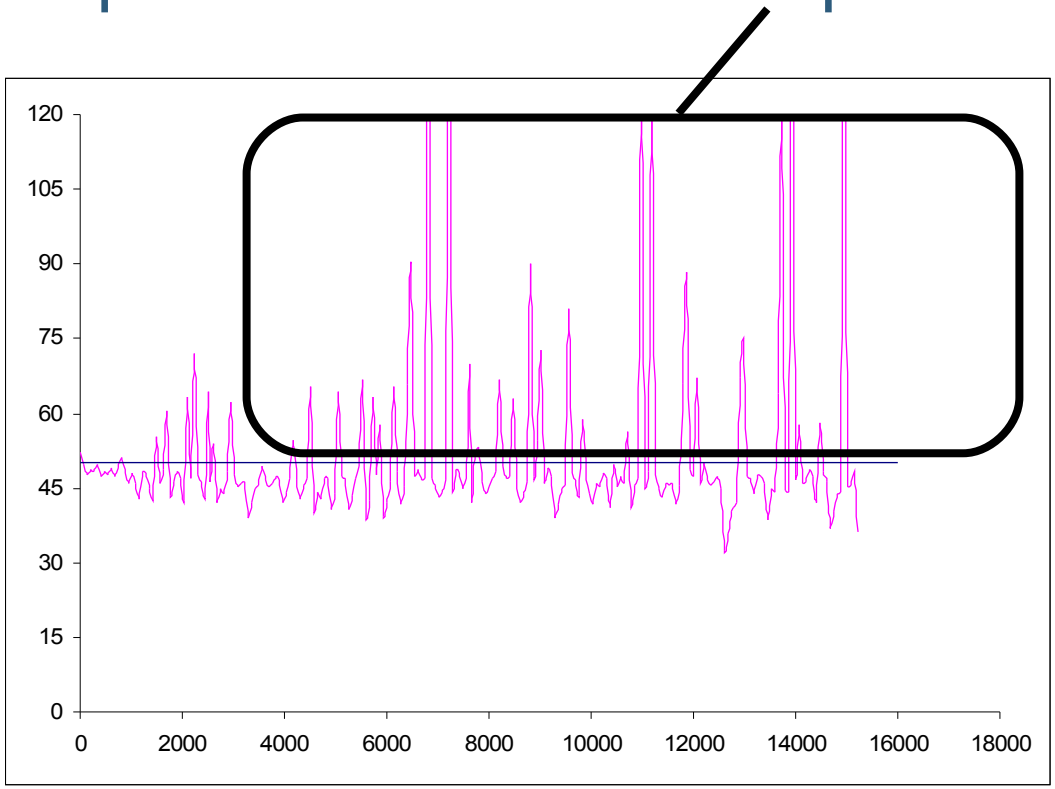
GC spikes cause occasional timeouts



Traditional VM—Non-Deterministic

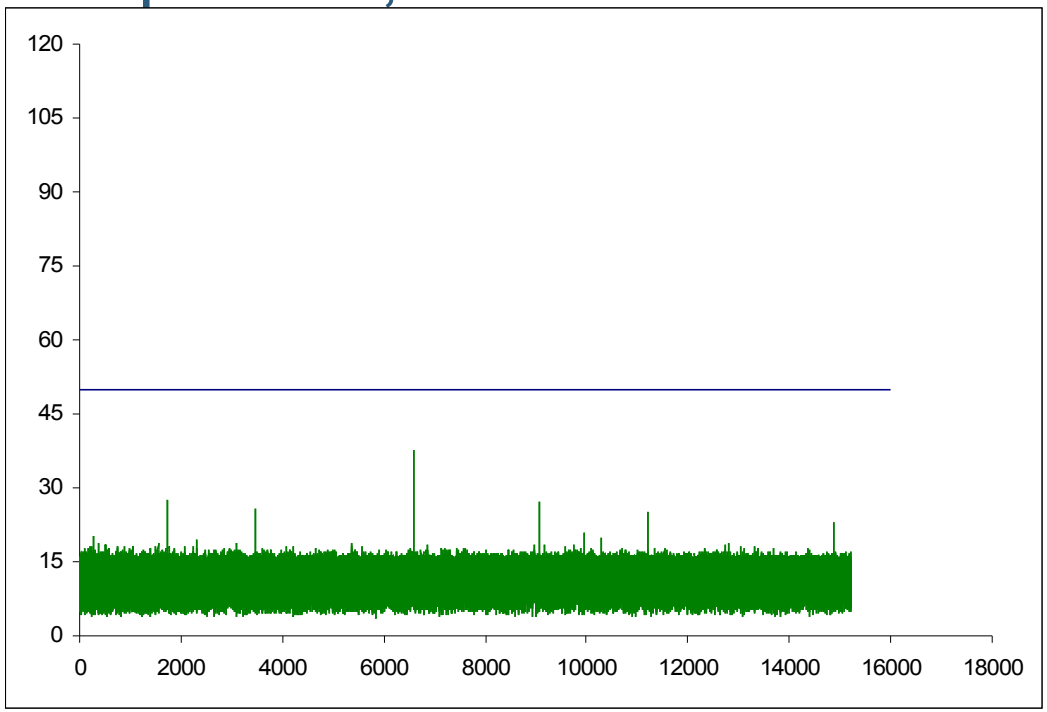
Collapses under strain when load increases

GC pauses cause unacceptable response times



JRockit—Deterministic GC

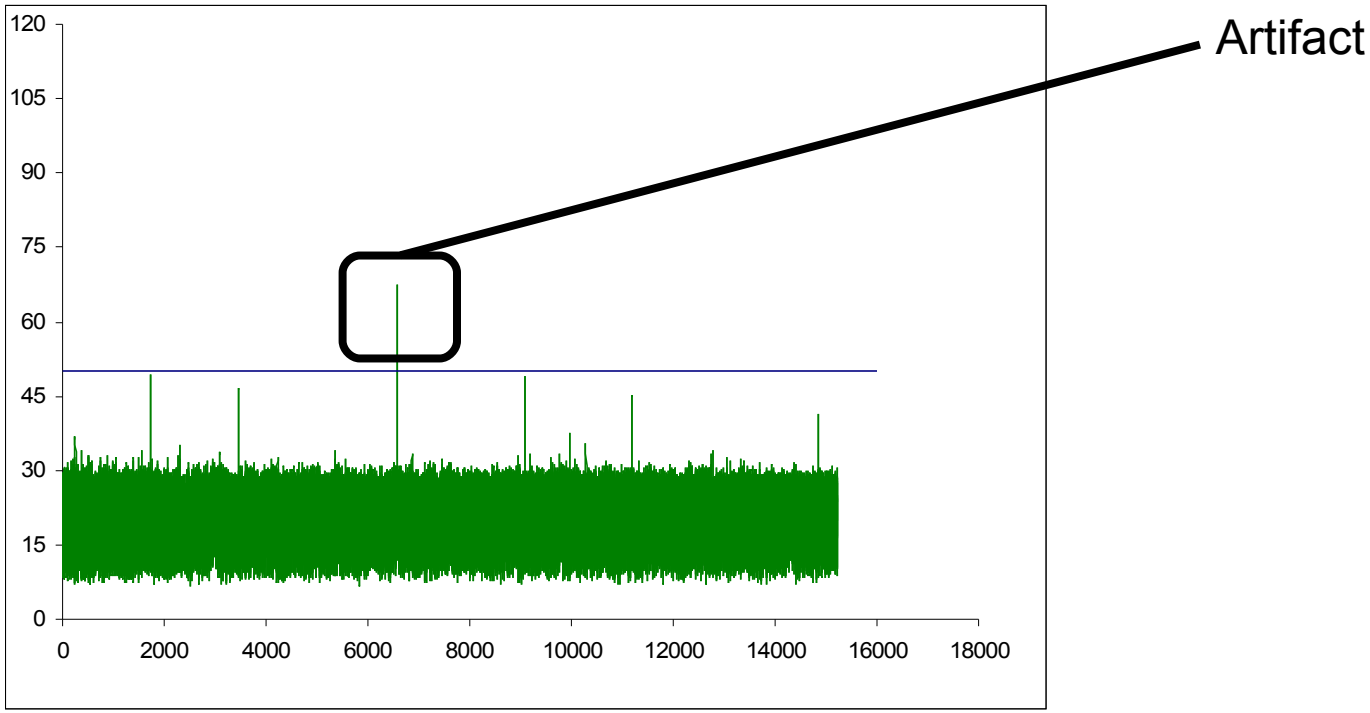
Low load: More frequent, but very short GC pauses, no timeouts



JRockit—Deterministic GC

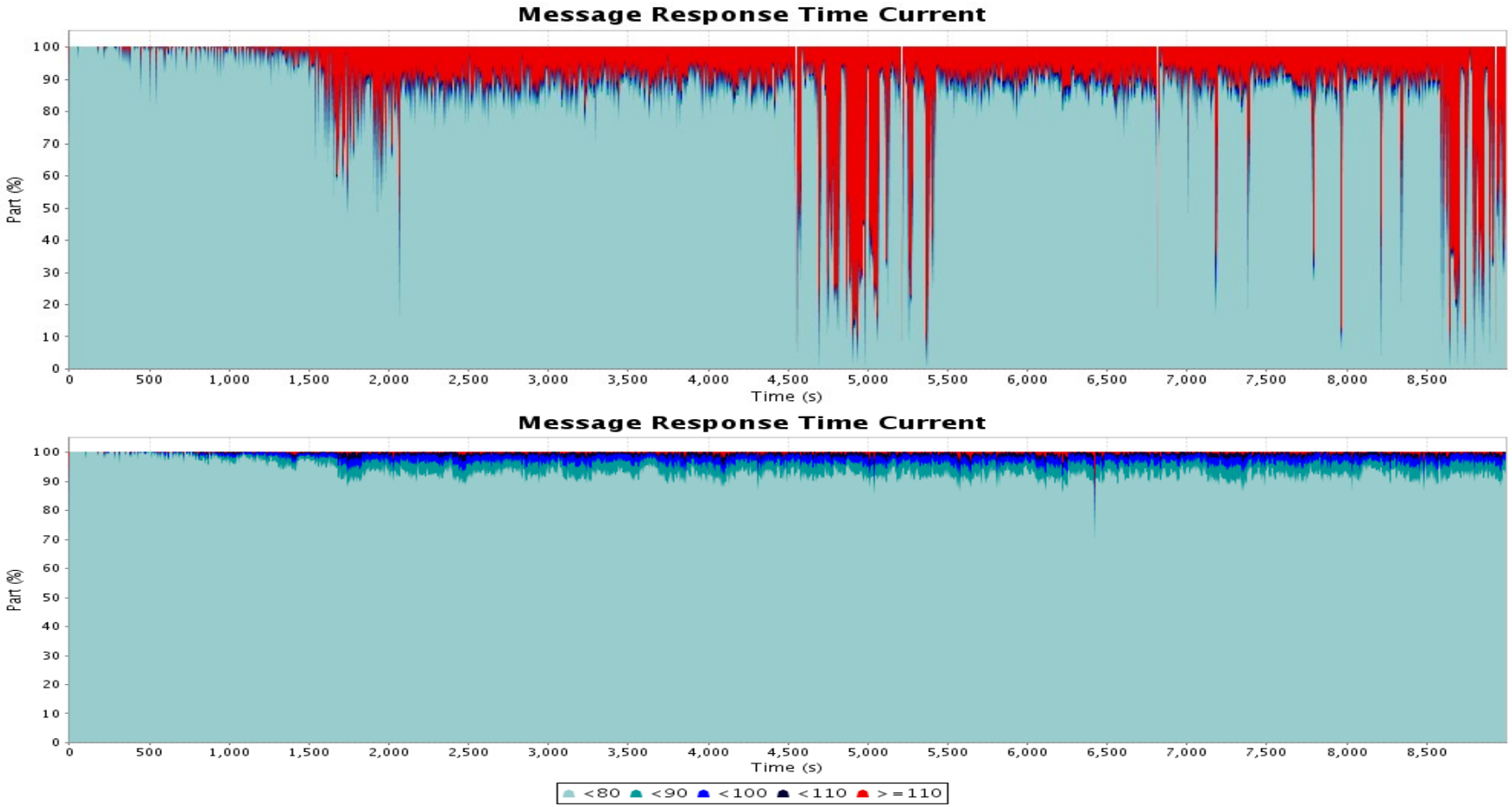
Handles the increased load just fine

Frequent, slightly longer GC pauses, very few timeouts



Deterministic Java VM

Response time histograms



Deterministic Java VM

Can it really be this good?

- Yes!
- Previous results are data from a real SIP Server
- Even a fraction of this gain means large cost savings
- Predictability is much better, giving better QoS (less busy calls)

Deterministic Java VM

How does it work?

- Any modern GC is mostly concurrent
  - But sometimes it needs to stop the world
- Basic idea: postpone stopping the world until we know that it will be a very short pause
- How?
  - Runtime analysis, collect data
  - Load measurements
  - Where is the GC activity?

Deterministic Java VM

How does it work?

- Continuously free up resources
- Interrupt jobs that take too long, e.g., compaction
- Load balancing
  - Mutating threads assist GC
  - Again: sampling based

What About Near Real-Time Java Technology?

- Definitions vary, but usually ~10 ms is upper bound
- Current implementation
  - Low response times
  - Average pause time much shorter than for existing solutions
  - Good enough for most applications (80/20 rule)
- Planned improvements
  - Lower, even more predictable pause times
  - Less severe GC spikes
  - Higher throughput

Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

Profiling and Management Tools

Q&A

The 64-bit World

Why is the world going to 64-bits?

- Performance, performance, performance
  - The need for performance drives the change
- It's all about data sets
  - 4 GB is your average heap size nowadays
  - A science fiction number in 1998, when JRockit started
- ...And data bandwidth
  - More data processed at the same time, 64-bit and 128-bit registers
- BEA has been doing 64-bit Java VM research since 2001, resulting in an excellent 64-bit Java VM

The 64-bit World

Challenges

- Data set size is the most generic problem
  - Java technology is particularly sensitive, since it's a garbage collecting language
  - We need to maintain fast GC for huge heaps and keep pause times down
  - Access time to objects on heap is also critical
  - Even if GC throughput is good when "stopping the world", we can't let pause times get too long
  - Real-time systems—QoS

The 64-bit World

Challenges

- Pointer size also matters
  - A 64-bit pointer is 2x a 32-bit pointer
  - An app with a certain size on a 32-bit system will automatically get bigger on a 64-bit system
    - Larger data structures
    - We need to optimize data structure size, considering different types of pointers
  - Larger amounts of data are shuffled
    - Cache misses
  - Pointer loads and store are slower, generally speaking, than on 32-bit systems

The 64-bit World

Solutions

- Intelligent address management
  - Common structures always use 4-byte pointers
  - Compressed references
    - 32-bit objects
    - Objects are always aligned, just ignore low zero bits
    - Use offsets from heap start as references instead of absolute pointers
  - Address space isn't necessarily limited to 32-bits, but intelligently used

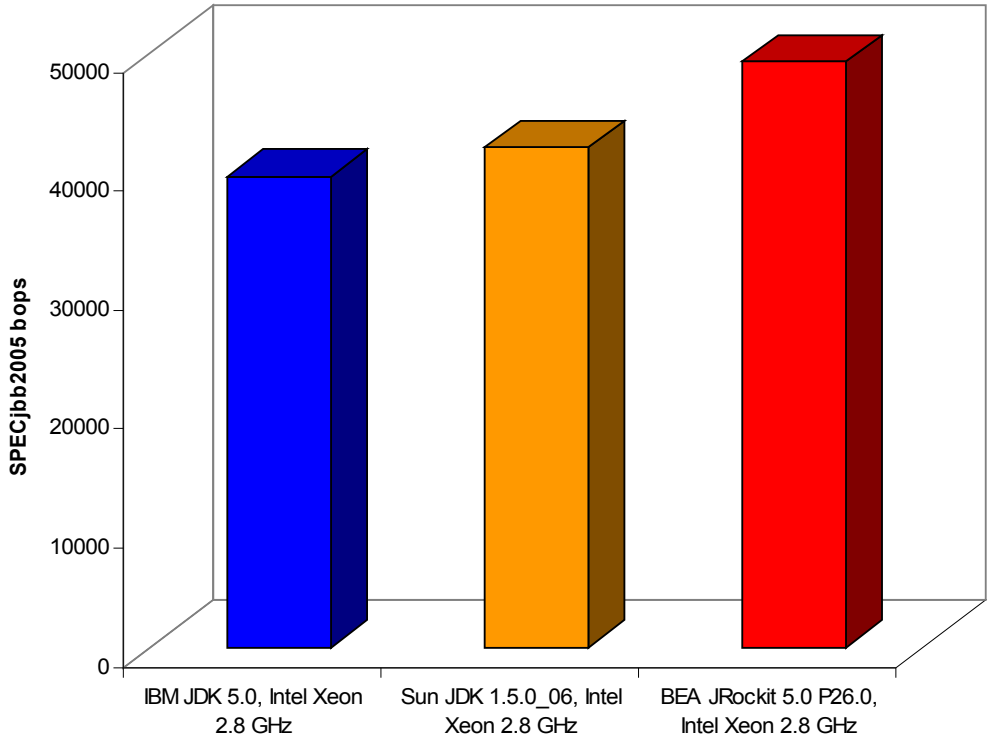
The 64-bit World

Challenges

- Porting
  - Your 32-bit Java-based app should ideally need no porting effort
    - Issues with native code and Java Native Interface
  - Solutions
    - JRockit uses Mixed Mode Execution (MME)
      - Enables 32-bit Java Native Interface libs on a 64-bit system
      - Good for transition

Results

64-bit vs. 32-bit Java VMs, SPECjbb2005



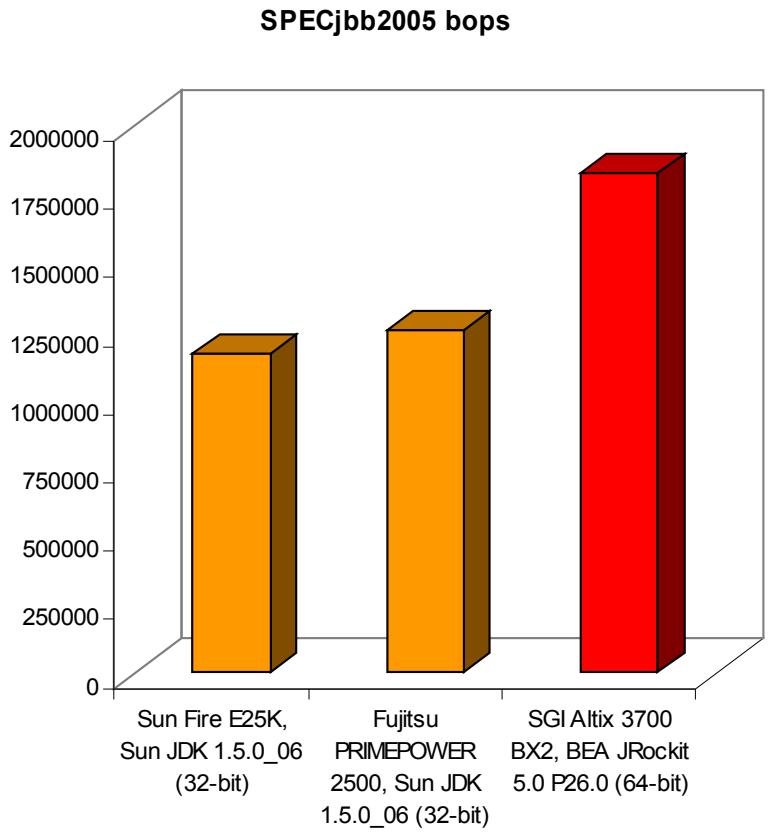
- Sun and IBM runs use 32-bit JVMs
- JRockit runs use 64-bit Java VM
- All runs use 2x DualCore Intel Xeon 2.8 GHz
- All runs use a single Java VM

Disclaimer: SPEC and the benchmark name SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation. Competitive benchmark results stated above reflect results published on http://www.spec.org as of April 19, 2006. For the latest SPECjbb2005 benchmark results, visit http://www.spec.org/osg/jbb2005.

Results

SPECjbb2005—1,828,349 bops—World Record!

- Sun and Fujitsu runs use 32-bit JVMs
- JRockit run uses 64-bit JVM
- SGI: 128 x Itanium 2
- Sun: 72-chip/144-core UltraSPARC IV+
- Fujitsu: 128 x SPARC64V
- All runs use multi-Java VM configurations



Disclaimer: Sun Fire E25K 1164995 SPECjbb2005 bops, 32361 SPECjbb200 bops/Java VM, Fujitsu PRIMEPOWER 2500 1251024 SPECjbb2005 bops, 39095 SPECjbb2005 bops/Java VM, SGI Altix 3700 BX2 1828349 SPECjbb2005 bops, 28568 SPECjbb2005 bops/JVM. SPEC and the benchma name SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation. Competitive benchmark results stated above reflect results published on http://www.spec.org as of April 19, 2006. For the latest SPECjbb2005 benchmark results, visit http://www.spec.org/osg/jbb2005.

Summary—64-bit

- There is some overhead involved in moving to a 64-bit system
- We have minimized that overhead
- Now it's your job to utilize the benefits of a much larger address space
- All Future JRockit ports will be 64-bit

Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

Profiling and Management Tools

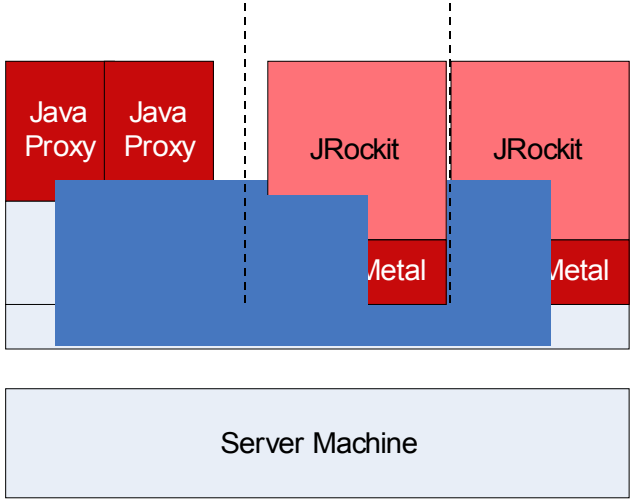
Q&A

Resource Management and Java VMs

- Resource Management has been poor
  - Java EE 5: Ability to measure how much resources the Java VM is using
- JRockit is extending Resource Management
  - To control how much resources that are used
  - To measure resources usage at the thread-level
  - To make sure JRockit works well with hypervisors (VMware and Xen)
- JSR 284 will standardize resource consumption

Hypervisor Aware Java VM

- Get resource control from the hypervisor
- Optimize performance on hypervisor
  - Communication hypervisor/Java VM to make Java technology operations faster
  - Communication Java VM/hypervisor to make hypervisor operations faster
  - Avoid the OS overhead
  - VMware/Xen



Agenda

Introduction to JRockit

Deterministic VM/“Real-Time” GC

Delivering 64-bit Performance

Resource Management

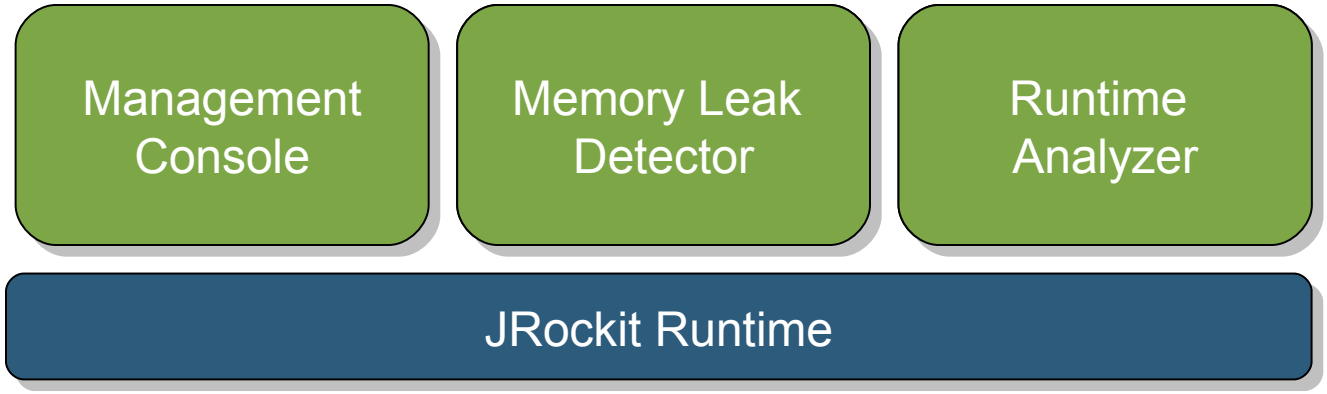
Profiling and Management Tools

Q&A

Profiling and Management Tools

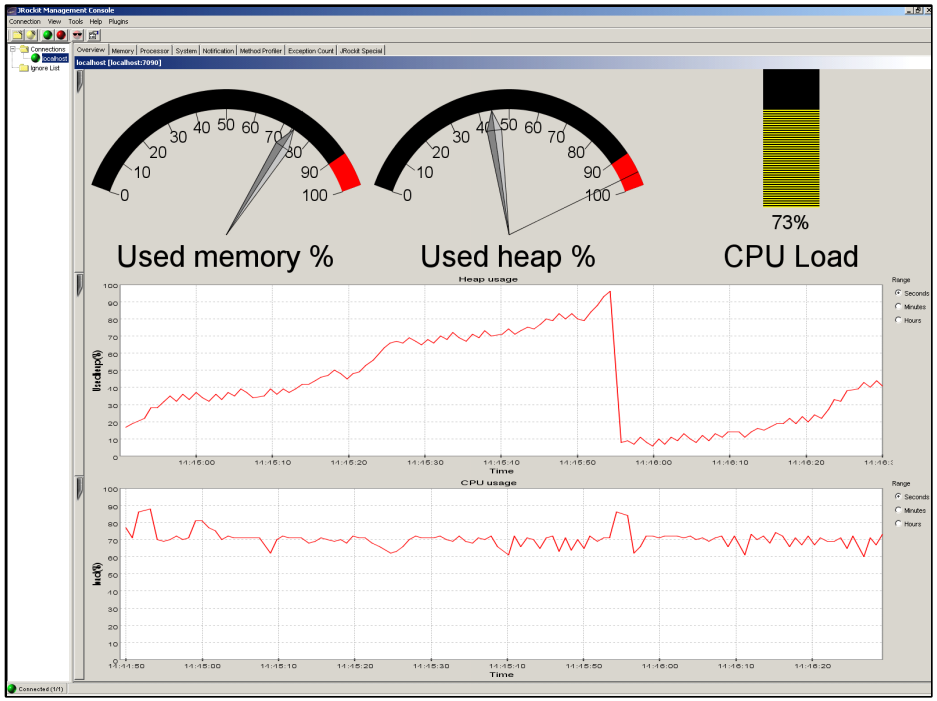
JRockit Mission Control

- Production-time monitoring
- Extremely low-performance cost
  - Unique JRockit architecture
  - Open to third parties



Management Console

- Monitor
  - CPU and memory usage
  - Real-time data feed
- Notify
  - OutOfMemory
  - CPU usage
- Manage
  - Heap size
  - CPU affinity

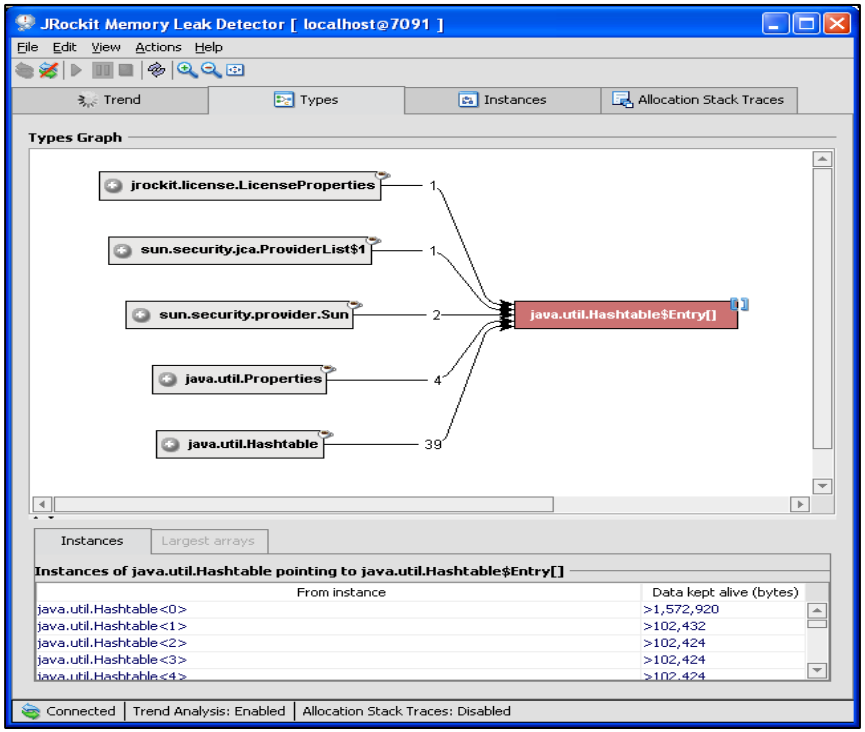


Memory Leak Detector

- Designed for use in production systems
  - Close to zero overhead (memory and performance)
  - Can be enabled at runtime, online
- Tight GC coupling
  - Use existing GC information
- GUI Tool + JRockit “server”

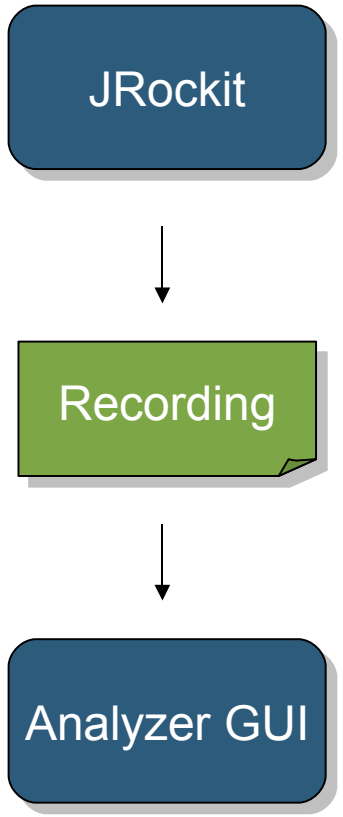
Memory Leak Detector

- Questions
  - Is this program leaking?
  - What is leaking?
  - Where/why is it leaking?
  - How do I fix it?
- Features
  - Trend Analysis
  - Referring Types
  - Referring Instances
  - Allocation Sites



Runtime Analyzer (JRA)

- Application and Java VM Profiler
  - Detailed heap information
  - Method profiler
  - Optimizations
  - Lock profiling
- Exposes data already collected
  - Low overhead
- Off-line analysis
  - Java VM built-in recorder
  - Separate analyzing tool



Summary of Tools

- Monitoring
- Memory leak detection
- Runtime profiling

- Available in production with low overhead!
- No overhead before and after usage

Summary

- Deterministic Java VM—near real time
- 64-bit computing—already here
- Resource management—utilize your hardware
- Monitor and profile—in production systems

For More Information

- "Bare Metal": No Need for an OS in a Virtualized Server Environment? An Alternative to MVM?
  - TS-3792

- http://dev2dev.bea.com
- http://forums.bea.com
- http://www.spec.org
  - SPECjbb2005
  - SPECjAppServer2004

Q&A

What's Hot in BEA JRockit

Marcus Lagergren and Staffan Larsen

Java Runtime Products Group
BEA Systems

TS-3484