
ORACLE®

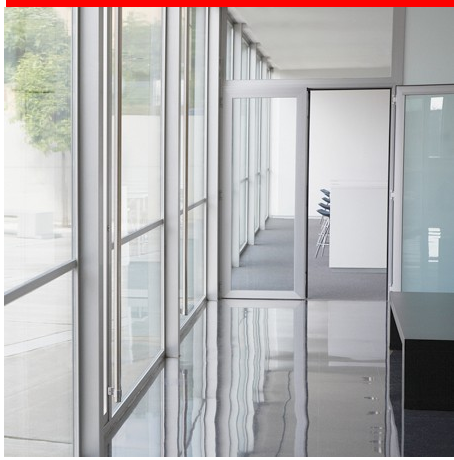
ORACLE®

Java Virtual Machine Strategy

Henrik Ståhl, Sr Director Product Management
Paul Hohensee, JVM Technical Lead

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

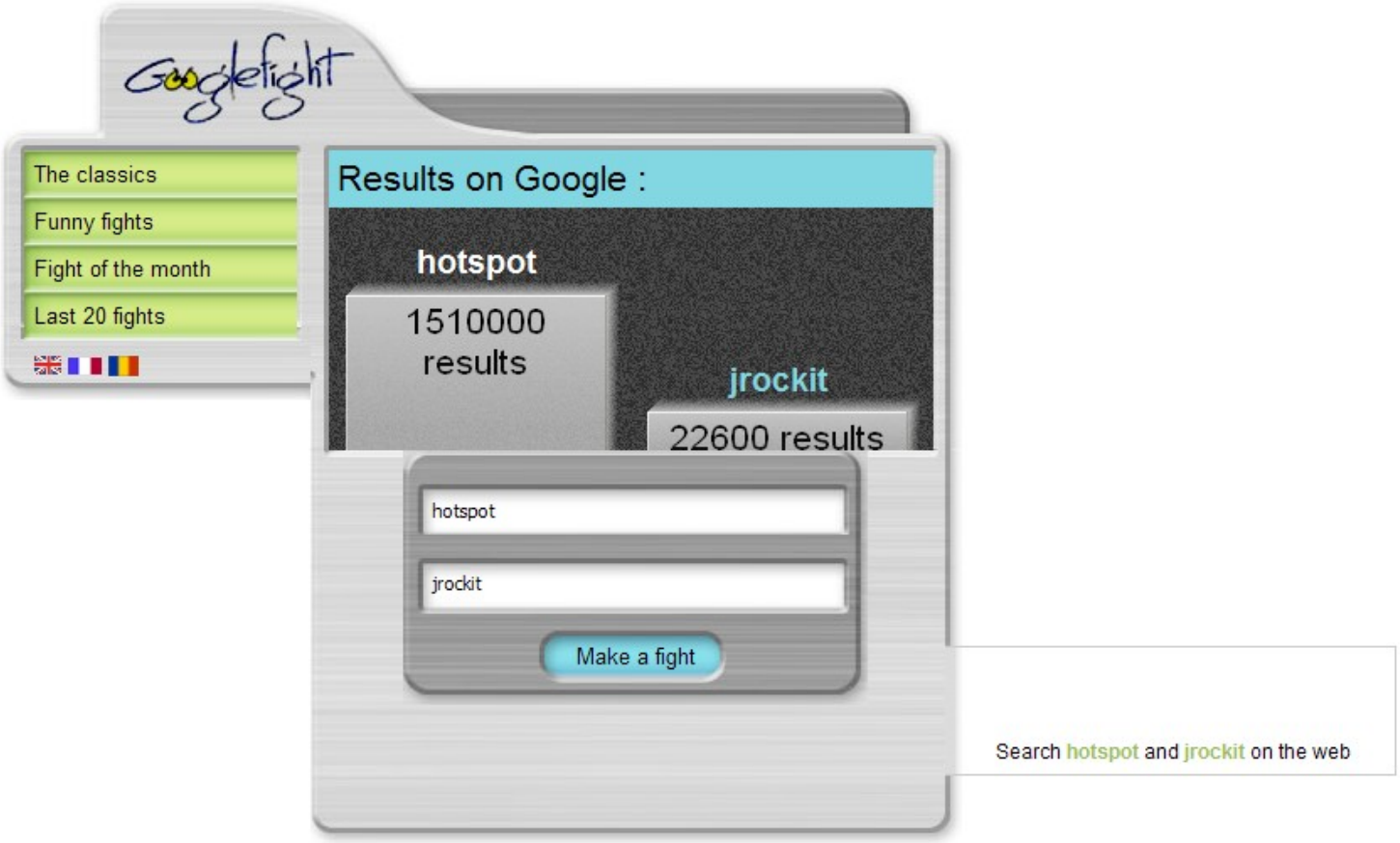


- Oracle's JVM Strategy
- Product Update
  - HotSpot
  - JRockit
- JVM Roadmap
  - Convergence Projects
  - Futures

Our Dilemma

- Two mainstream JVMs
- HotSpot
  - Versatile (client and server), market share leader, high quality and performance
- JRockit
  - Specialized for the Oracle server stack, basis of value-adds like Mission Control, Virtual Edition and Soft Real-Time
- Wasteful to continue development on both, so
- Goal: Single JVM incorporating best features
  - Pick one code base, port features from the other to make the result better than *both* parents everywhere it counts!
  - After much deliberation and business and technical due diligence over several months, it came down to ...

Hmm…



This site is not affiliated with or sponsored by Google

SE-Tools Network | SE-Inspector | SE-Keywords | SE-Check | SE-Rank | SE-Bid | Se-Spider | Se-Flash | Se-Fight | Meceoo.com

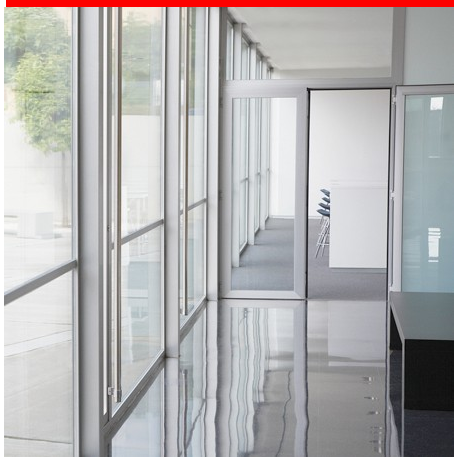
Ok, let's be serious

- Technical due diligence did not point to a clear winner
- Way more engineers working on HotSpot, so
- Converged JVM based on HotSpot with all goodness from JRockit
- Open-sourced via OpenJDK
  - Some value-add such as Mission Control will remain proprietary
- Free binary download includes closed source goodies
  - Same use license as before
- Oracle committed to continued investment
  - JVM teams have been merged, working on feature transfer
  - And moving forward on new projects
- Will be a multi-year process

Shocking Discovery

- “Great minds think alike.”
- HotSpot and JRockit were already converging
- Similar market requirements lead to similar features
- E.g., escape analysis
  - Optimize use of objects allocated and used by only a single thread
- E.g., to get incremental GC, divide and conquer
  - Deterministic GC: power-of-2 number of heap regions
  - Garbage First: arbitrary number of heap regions

Agenda



- Oracle's JVM Strategy
- Product Update
  - HotSpot
  - JRockit
- Converged JVM Roadmap
  - Convergence Projects
  - Futures

JVM Enhancements YTD

- Java may have been moving slowly (we're fixing that) but the JVMs have not!
- JVM decoupled from JDK since 2005 (JRockit) and 2007 (HotSpot)
  - Will remain decoupled
- Aggressive, incremental enhancements to both JRockit and HotSpot over the past year

HotSpot Recent Goodies (1)

- Up to 32 GB heap compressed pointer JVM
  - Three variations, fastest first: < 4gb, up to ~26gb, up to 32gb
  - x64 JVM now as fast or faster than 32-bit x86 JVM
- NUMA garbage collector enhancements
  - Good parallel collector scaling beyond a single socket
  - -XX:+UseNUMA with parallel collector chunks the young generation on a per-socket basis: assumes transient data is local to a socket
  - Survivor space and old generation interleaved across sockets: assumes surviving data is shared across sockets
- Optimized String concatenation
  - Use a single char[] temporary for multiple concatenations

HotSpot Recent Goodies (2)

- Garbage First collector
  - Consistent low pause times with very large heaps
  - Stop-the-world, region based, incrementally compacting, prefer to collect regions that are mostly garbage
  - Eventual CMS replacement
  - Still experimental but getting there!
- Optimized and intrinsified array copy and fill
  - SSE4.2 support on x86/x64
- Compressed Strings
  - String object points to either a byte[] or a char[]
  - Starts out as byte[] unless locale requires char[]
  - Can drastically reduce heap footprint
- Escape analysis, on by default
- Many, many, many bugfixes and small enhancements

JRockit R28 (1)

- Next-generation JRockit, released early 2010
- Themes: Robustness, Serviceability, Performance
- Robustness
  - Cooperative Thread Suspension
  - More robust JIT compiler (no more native OOM)
  - White box testing APIs
  - Refactored codebase for maintainability
- Serviceability
  - JRockit Flight Recorder
  - HPROF heap dump support
  - Enhanced JMX agent
  - Native memory tracking
  - Fine granular compiler directives

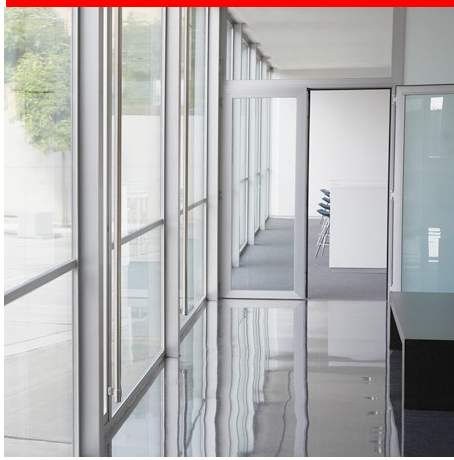
JRockit R28 Goodies (2)

- Performance
  - Up to 64 GB compressed references (was 4 GB)
  - Up to 30% lower GC pause times overall
- Flight Recorder
  - Major serviceability effort integrated into Mission Control
  - New event mechanism integrated with logging
  - Java event API: integrated app and JVM events
  - Event ring buffer can be saved and inspected, captures info when problems happen
  - User-specified overhead, default low enough for production use

JVM Recommendation for Oracle Customers

- HotSpot
  - Oracle Apps and Middleware on Solaris
  - Client and non-Oracle Apps on Solaris/Windows/Linux
- JRockit
  - Oracle Apps and Middleware on Windows/Linux
- Recommendation only; use what works best for you

Agenda



- Oracle's JVM Strategy
- Product Update
  - HotSpot
  - JRockit
- Converged JVM Roadmap
  - Convergence Projects
  - Futures

Convergence Projects: JRockit Value-Add (1)

- Over roughly the next 2 years, incrementally deliver
- Mission Control and Flight Recorder Support
  - MC console UI and enhanced JMX agent support
  - Support for FR´s unified event and logging mechanisms
- No artificial class metadata size limit
  - App servers load or reload an unpredictable number of classes, often exceeding fixed size bounds
  - Eliminate the Permanent Generation, move metadata to native memory
  - Metadata size limited by virtual address space size
- Zero-copy I/O
  - JRockit can pin objects in place
  - Delaying GC until I/O is done may work too

Convergence Projects: JRockit Value-Add (2)

- Native memory tracking and its MBean
  - As JRockit R28
- GC and compiler control APIs
  - As JRockit R28
- Non-contiguous Java heap for 32-bit Windows
- Virtual Edition support
  - True Java virtual appliances!
- Soft real-time GC
  - Adopt the best from JRockit Deterministic GC
- 8-byte object headers for 64-bit JVMs
  - Adopt JRockit lock reservation and thin lock scheme

Futures: What's Next (1)

- Internal restructuring as necessary to improve reliability and maintainability
- The Practically Pauseless (and Scalable) VM
  - Garbage First as a large-heap CMS replacement
  - Garbage First concurrent evacuation
  - Plus super secret hush-hush project
- (Much) improved NUMA GC support
  - Vertical (multi-socket) scalability
- Pervasive ergonomics
  - Just say "java"
  - Internal and external ergonomics: "good citizen"
  - Per-app/user run history feedback, minimize time to optimized performance

Futures: What's Next (2)

- Improved support for dynamic languages
  - Particularly JavaScript
  - Hidden classes and fixnums
- Much faster startup, smaller footprint
  - MVM: multiple applications in a single JVM process
    - Share metadata and compiled code while preserving isolation
    - Per-task Java heaps and GC policies
  - Application class data sharing (saved class metadata)
  - Ahead-of-Time compilation
    - Of static initializers!
  - Tie-in to pervasive ergo and Java modules
- Faster native code interfaces: C <-> Java
  - JNI designed to be difficult to use: invent something else

Futures: What's Next (3)

- Improved inter-procedural analysis, more aggressive inlining, dependence-based vectorizer
  - Expands scope for escape analysis
  - Intel AVS SIMD and GPU support
  - JOCL (Java OpenCL wrapper) and a Java equivalent of Apple's Grand Central Dispatch
  - Resurrect Java Grande: HPC in Java
- G1-lite
  - Incremental GC for single-hardware-thread machines integrated into the Garbage First framework
- Platform support
  - ARM and PPC server compiler ports
  - MIPS support

Futures: What's Next (4)

- Resource Monitoring and Management APIs
  - Great for virtualization and cloud-awareness
  - Enables detailed chargeback
- Next-gen Java (?!) Mission Control
  - Continued enhancements in diagnostics, profiling, monitoring and management

Summary

- JRockit and HotSpot continue
  - Incremental feature enhancements as needed
  - Will be maintained for years to come
- *Current* R&D projects
  - Cross-pollination of QA and performance
  - Convergence
  - JDK 7 related – invokedynamic, etc.
  - Startup time – aiming to solve this once and for all!
  - Low pause GC
  - Performance – a Sisyphean task…
- First converged JVM based on HotSpot in CY2011
  - With JDK 7?
  - First batch of servicability and performance features
    - JRockit JMX agent, JRMC support and more!

Q & A

Related Sessions

- Monday
  - Oracle JRockit: Advances in Java Virtual Machine Technology (4 pm)
- Tuesday
  - How to Tune and Write Low-Latency Applications on the Java Virtual Machine (11:30 am)
  - JVM Analysis: Oracle JRockit Mission Control and Oracle JRockit Flight Recorder (1 pm)
  - Performance and Debugging Advancements in OpenJDK (6 pm)
- Wednesday
  - Experience Talk: Understanding Adaptive Runtimes (10 am)
  - Showdown at the Java Virtual Machine Corral (4:45 pm)


ORACLE®