;login:

THE MAGAZINE OF USENIX & SAGE

August 2002 volume 27 • number 5

inside:

CONFERENCE REPORTS

Java™ VM 2002

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

## 2nd Java™ Virtual Machine Research and Technology Symposium

### San Francisco, CA August 1–2, 2002

*Summarized by Jose F. Osorio*

#### OPENING REMARKS

JVM '02 Program Chair Sam Midkiff of IBM's T.J. Watson Research Center welcomed symposium attendees and provided an overview of the topics that were going to be presented in the intensive two-day event. These included research work in the areas of JVM memory management, advanced JVM architectures, just-in-time compilers, method inlining, realtime JVMs, embedded JVMs for portable, personal, mobile devices, and hardware-based Java machines for stack-based microprocessors.

The Best Student Paper Award went to "Supporting Binary Compatibility with Static Compilation," by Dachuan Yu, Zhong Shao, and Valery Trifonov, Yale University. The project's Web site is *http://flint.cs.yale.edu/flint/publications/bincomp.html*. The Best Paper Award was given to "An Empirical Study of Method In-lining for a Java Just-in-Time Compiler," by Toshio Suganama, Toshinki Yasue, and Toshio Nakatani, IBM Tokyo Research Laboratory.

#### Keynote Address: Stop Thinking Outside the Box. There Is No Box.

Robert Berry, Distinguished Engineer, IBM Centre for Java Technology

"Virtual Machine technology is here to stay as a fundamental technology supporting real business computing," said Robert Berry during his remarks.

As an absolute core of much of IBM's software and hardware business, JVMs are a proven technology for real computing: they form an essential component of WebSphere's Application Server runtime environment, are integrated with the DB2 database management system, and are a key enabler for Web services infrastructure. Proof of this is also Microsoft's .NET framework and its common language runtime (CLR) for managed code, which is a VM-based technology. Furthermore, JVMs are at the core of business computing. Java researchers are also evaluating CLR and looking into performance compared to JVMs.

Customer requirements such as viability and legacy expectation, competitive pressure such as functionality and performance, hardware evolution, technology changes, and new ideas are some of the principal motivating drivers for JVM innovation, explained Berry.

IBM's focus on innovation reflects the growing maturity of JVMs, evolving from an inward client-first, server-next focus to virtual machines in the context of middleware such as database, transaction, and applications servers.

Berry presented a history trail of IBM's innovations in such key areas as JVM architecture, just-in-time compilation, middleware, and autonomic systems.

Innovations in JVM memory allocation and garbage collection were explained. Different garbage collection strategies were compared, including mark-sweep-compact, compaction avoidance, parallel marking, parallel marking plus sweeping, and concurrent marking. The impact of full vs. incremental compaction was discussed.

In the area of JIT compilation, Berry explained method inlining and escape analysis strategies (where the detection of objects that do not survive a particular method invocation scope is done to allocate them on the stack so as to reduce heap and memory synchronization overhead).

The high-performing JVMs' need for execution of short and repetitive transactions and strict isolation requirements between transactions, plus the constraints of a 31-bit addressing mode, were key factors that led to the invention of IBM's Persistent Reusable JVMs, which reduce initializing and startup JVM overhead, are equipped with efficient garbage collectors, and support transaction isolation among multiple JVMs.

In autonomic computing, the drive to increase reliability, availability, serviceability (RAS) resulted in the creation of two well-known Java specification requests (JSRs): JSR 174 – jvmmi for monitoring, and JSR 163 – jvmsi/ti for performance.

IBM has innovated successfully with JVM for four reasons: (1) they listen to their customers (whether external or internal) and to their users; (2) they provide significant investment in research and development for key JVM infrastructures; (3) they deploy novel technology in the field and in the community early enough for experimentation; and (4) they do not give up on any ideas.

Berry explained that work and innovation is needed in various areas, including JVM footprint, very large heaps (VLH) on the order of 500GB and beyond, object pooling, object sharing, decimal arithmetic, RAS, Web services, JVM execution performance, simplification and integration of resources monitoring, control and management, and the impact on JVM technology from aspect-oriented software development (the next generation of modularity beyond the object-oriented paradigm). Visit *http://www.aosd.org* for more information on aspect-oriented software development.

#### Adaptive Garbage Collection for Battery-Operated Environments

G. Chen, M. Kandemir, N. Vijaykrishnan and M.J. Irwin, The Pennsylvania State University; M. Wolczko, Sun Microsystems

Narayanan presented an adaptive garbage collection strategy for embedded devices in which object creation and memory allocation history on a time interval is used to tune the frequency at which the garbage collector is invoked to

reduce energy consumption from Java applications. He also showed how energy leakage is reduced by exploiting supply-gated power management mechanisms which shut down energy supply to memory banks that do not hold useful data. Using KVM – Sun Microsystems' JVM designed for embedded and battery-operated environments – and a set of nine applications typical for handheld devices, the adaptive garbage collection strategy incurred fewer performance penalties than other approaches.

### Concurrent Remembered Set Refinement in Generational Garbage Collection

David Detlefs and Ross Knippel, Sun Microsystems; William D. Clinger, Northeastern University; Matthias Jacob, Princeton University

With generational garbage collection, garbage collection latency can be decreased, increasing throughput and reducing execution pauses. It divides the heap memory into multiple layers of allocation generations. In this strategy the youngest generation is the one most frequently collected, and a complex data structure is used to represent the generation layers as remembered sets in order to identify links from older generations to younger ones.

The refined strategy focuses on the challenge of efficiently maintaining the accuracy of remembered sets in a concurrent environment in a way that reduces the cost penalties from write/update barriers involved in maintenance of the complex data structure and that scales well as the size of the old generation increases. The strategy uses a write barrier implementation whose cost is not significantly greater than card marking techniques.

Two remembered-set organizations were considered in the implementation of the proposed strategy: a card table augmented with a summary table, and a refined version of the first as a two-level card table (a coarse-grained summary table, and a fine-grained detail table with a 2N ratio between each table for some large N). Two write-barrier refinement techniques were considered as well: direct refinement on one-level and two-level card tables, and log-base refinement with mutator and refinement threads.

An attendee asked why hashtables were not used in the strategy implementation. Detlefs responded that separate benchmarks had shown that their two-level card table approach is as good as hashtable direct access during scanning of the coarse-grained table.

### To Collect or Not to Collect? Machine Learning for Memory Management

Eva Andreasson, BEA/Appeal Virtual Machines; Frank Hoffmann, Royal Institute of Technology; Olof Lindholm, BEA/Appeal Virtual Machines

Andreasson explained how reinforcement learning contributes to JVM autonomous decision-making to perform garbage collection. Given the elements of system environment – State, Action, Reward, new State, new Action (SARSA) – Eva explained how after an iterative and systematic process of decision-making exploration and prediction, the SARSA approach yields an optimal learning scheme for adaptive garbage collection. Benchmarks demonstrated how the RLS (reinforcement learning system)-based JVM outperformed JRockit, a conventional JVM, in a dynamic environment in which memory allocation behavior changes more rapidly.

### Optimizing Precision Overhead for x86 Processors

Takeshi Ogasawara, Hideaki Komatsu, and Toshio Nakatani, IBM Japan

A novel approach to optimize floating-point operations in JVMs for x86 target processors was presented by Takeshi Ogasawara. The strategy involves tracking floating-point precision-type code blocks in a Java program, performing region analysis, and also tracking precision-aware method invocations. By transforming the original bytecode and generating code blocks with the appropriate floating-point precision, the default precision mode can be ignored. Region analysis investigates code blocks to find strategic points where single- and double-precision mode-switch instructions can be inlined to reduce rounding and store-reloads overhead. After transformation, the just-in-time compiled code calls the native target code with the same floating-point precision type, therefore eliminating any redundant mode switches across method boundaries of precision-aware invocations.

Ogasawara emphasized that their strategy does not sacrifice strictness of floating-point precision; it actually reduces the x86-specific overhead of preserving strictness.

### A Modular and Extensible JVM Infrastructure

Patrick Doyle and Tarek S. Abdelrahman, University of Toronto

The design of Jupiter, a modular scalable JVM framework, was outlined by Patrick Doyle. The Jupiter project investigates JVM architectures for high performance on large-scale parallel systems with 128+ microprocessor clusters and facilitates research and future enhancements to their JVM. Jupiter has been constructed out of a multitude of discrete units with small and simple interfaces in a manner similar to how shells in UNIX build complex command pipelines out of discrete programs. Numerous aspects of JVM system architecture (memory allocation, metadata, method dispatching, object manipulation, call stack, bytecode interpretation, just-in-time compilation multithreading, synchronization, and many others) are covered in this paper in a clear and simple manner.

### A Lightweight Java Virtual Machine for a Stack-Based Microprocessor

Mirko Raner, PTSC

The Symposium ended with a presentation of a hybrid implementation of a JVM. The Ignite microprocessor is a stack-based bytecode processor with a

32-bit dual-stack architecture, an 18-word operand stack, and a separate 16-word stack for call-stack frames. It was originally designed as an embedded computing platform for efficient execution of C and Fortran. Ignite is not a Java technology-enabled microprocessor, but its architecture is very similar to the JVM, and Java bytecode translation is easy and efficient. Ported from the original LVM (Lightweight Virtual Machine), the main challenges were the separation from the underlying operating system by means of JELLO, an abstraction layer; optimization of ahead-of-time (ATO) compilation; lazy class resolution; and optimization of method invocation.

Raner gave a detailed explanation of the operand stack, local stack, and global/general register-set structure, the differences between the stack model of a typical JVM and Ignite's stack model, and Ignite's instruction-set format.

Raner also explained that the differences between the argument-passing mechanisms of a standard JVM and Ignite's made it necessary to move method arguments from the Ignite's operand stack to Ignite's local variable stack.

The LVM is still a work in progress. Plans for its future include placement in very small devices, cellular phones, and mobile devices.

To view the presentation slides, visit *http://java.sh/lvm.html*.