conference reports

3rd Virtual Machine Research and Technology Symposium (VM '04)

San Jose, California
May 6–7, 2004

This issue's reports focus on the 3rd Virtual Machine Research & Technology Symposium, held in San Jose, California, May 6–7, 2004.

OUR THANKS TO THE SUMMARIZERS:
Maria Cutumisu
Vivek Haldar
Yahya H. Mirza
Feng Qian
Ananth I. Sundararaj

TECHNICAL SESSIONS

Photo: VM '04 Program Chair Tarek Abdelrahman, with Best Paper winners Vivek Haldar and Deepak Chandra (not shown: co-author Michael Franz)

(The first Keynote Address, on "Virtual Machines: Past, Present, and Future," was presented by Mendel Rosenblum of Stanford University.)

2D KEYNOTE ADDRESS

The Mono VM

Miguel de Icaza, Co-Founder and CTO, Ximian

Summarized by Maria Cutumisu

Miguel de Icaza discussed the implementation of Mono, an open source execution engine for the ECMA CLI specification. The Mono VM was implemented by an enthusiastic group of people who were newcomers to the virtual machine domain but were attracted by the social, technical, and personal aspects of this project. They were interfacing with a large community of developers around the world and were observing a growing user community. The speaker started with a brief description of various systems, interesting for their capabilities with respect to the Mono project: UNIX, the Gnome Project, and Latte 2000. He continued with historical information about Ximian which was focused on making Linux succeed on the desktop.

At the time Mono was launched, the intent of the authors was to bring .NET features to Linux (C# compiler, virtual machine, core class libraries) and to provide open source features well suited to distribute the work. The team did not have any experience in compilers or virtual machines at the time. Currently, Mono has become an open source implementation of .NET that is based on the ECMA/ISO standards, includes C# and VB compilers, and works with third-party compilers, such as Delphi, Eiffel, COBOL, FORTRAN, Mercury, Python.NET, PerlSharp, and Nemerle.

The speaker described in detail several Mono features, including multi-language support, two stacks, C# compiler, virtual execution system, runtime, JIT environment, and support for optimized code compilation. Today Mono benefits from extensive inlining of intrinsic operations and an SSA-based representation. The talk concluded with an interesting discussion about research in virtual machines and compilers, as well as a brief outline of .NET limitations. Mono URL: *http://www.go-mono.com*.

A Virtual Machine Generator for Heterogeneous Smart Spaces

Doug Palmer, CSIRO ICT Centre

Summarized by Maria Cutumisu

Doug Palmer presented a virtual machine generator that provides "numerous virtual machines, each tailored to the capabilities of a class of resources." The speaker started by defining heterogeneous smart spaces as "networks of communicating, embedded resources and general-purpose computers that have a wide spread of power and capabilities." These spaces are typical of commercial, agricultural, or other outdoor environments.

The author then stated the central programming problem: Each heterogeneous smart space is unique; therefore a programming model that allows domain knowledge to be reused across smart spaces is necessary. The virtual machine generator constitutes a solution to the problem of providing a single virtual machine implementation that operates in heterogeneous smart spaces.

The speaker illustrated the virtual machine generation process and talked about the subset declaration for a virtual machine. The virtual machine is specified in an XML document, and this specification allows a stack-based virtual machine to be generated. A virtual machine specification and a subset declaration together constitute the input for the generator. The generator analyzes the virtual machine and generates source code files for Java and C that implement the subset virtual machine. These source files are compiled and linked in the presence of a standard library of support functions and classes; an assembler is generated at this point. The complete virtual machine is analyzed, and instruction codes, event codes, and stores are allocated before subsetting.

In conclusion, the advantages of the compact generator were outlined, including the fact that any optimizations that are made will propagate to future generated virtual machines. Moreover, "using a generator allows virtual machines to be quickly generated for new resources and to try new instruction sets."

MCI-Java: A Modified Java Virtual Machine Approach to Multiple Code Inheritance

Maria Cutumisu, Calvin Chan, Paul Lu, and Duane Szafron, University of Alberta

Summarized by Yahya H. Mirza

Duane Szafron presented an attempt to decouple the various roles a class plays—concept, interface, implementation, representation, factory, and extent—by separating them. The paper makes the case that most object-oriented languages do not separate these notions. The authors state that Java loses an opportunity for code reuse. This problem is illustrated by showcasing a concrete example from the Java I/O libraries. In Java a class can't inherit code from two parents, since it does not support multiple code inheritance.

A new language construct, "implementation" is presented as a solution. An implementation is essentially an interface with pure behavior code, but does not include data. With this approach, one can inherit code, but not data, from two parents, thus relaxing Java's inheritance semantics. The authors claim that they achieved significant code reuse by adding this feature to Java. The paper also adds a new "multi-super" mechanism which can be used to define an inheritance path to a particular super-implementation.

The "implementation" language feature is applied by making a minimal number of changes to the Jikes Java compiler and the Sun JVM. The compiler code generation process includes generating an invokeinterface for calls for which the static receiver type is an implementation. An invokespecial is generated for multi-super calls but with a reference to an interface instead of a class; the virtual machine can recognize these calls since all invokespecial bytecodes refer to classes. Finally, when the receiver is "this," an invokeinterface is generated instead of the usual invokespecial.

The changes to the Sun JVM include changes to the class loader (interface method table construction algorithms). These changes included detecting and reporting potential ambiguities, copying code pointers from interfaces to classes, and, finally, creating new method blocks on the JVM C-heap in two rare scenarios. The presenter stated that the resolution and dispatch of invokevirtual and invokeinterface bytecodes and the quickening of these bytecodes did not change. A key lesson learned from this project was that to make an efficient change to the VM, one must make changes when the class is loaded, but never during dispatch.

Semantic Remote Attestation—A Virtual-Machine-Directed Approach to Trusted Computing

Vivek Haldar, Deepak Chandra, and Michael Franz, University of California, Irvine

Summarized by Maria Cutumisu

This paper won the Best Paper award. Vivek Haldar presented a framework for semantic remote attestation, as well as two example applications built within this framework: a distributed computing client-server application (Mersenne Primes) and a Gnutella-like peer-to-peer network protocol. In contrast with current static techniques for remote applications, his team's approach uses language-based virtual machines to enable the remote attestation of dynamic program properties independently of the underlying platform. Their two examples illustrate applications that distribute trust dynamically.

One of the key questions addressed in the talk was how to transcend the notion of trust from closed systems to open systems. Trusted computing constitutes the effort of adding components and mechanisms in open systems with the goal of providing trust. As a result, the integrity of the system is checked and enforced, and the system is allowed to authenticate itself to remote systems.

The speaker talked about critical issues in trusted computing and remote attestation, with a focus on integrity (ensuring a secure boot process), authenticity, and trust vs. security. Moreover, he stressed how virtual machines can make trusted computing more secure, flexible, and effective. In particular, problems with remote attestation were discussed, including issues such as the lack of program behavior attestation, the nature of the remote attestation (static, inexpressive, and inflexible), the heterogeneity of devices and platforms, and the revocation problem inherited from public-key cryptography.

The solution proposed by the authors is the implementation of a prototype framework for semantic remote attestation, i.e., the use of a trusted virtual machine (TrustedVM) for remote attestation. Virtual machines execute platform-independent code with rich meta-information. In addition, the code runs under the control of a virtual machine. A trusted VM can attest to properties of classes, as well as dynamic and system properties.

Several advantages of semantic remote attestation were outlined during the presentation, such as certified program behavior, the capability of allowing various implementations of the same program respecting certain security requirements, dynamicism and flexibility, explicit trust relationships (checked and enforced) between nodes, and a mechanism for finer-grained trust using degrees of trustworthiness. In conclusion, the speaker pointed out that currently proposed mechanisms for trusted computing are severely limited and that leveraging VM technologies can make trusted computing more flexible and effective.

Towards Scalable Multiprocessor Virtual Machines

Volkmarr Uhlig, Joshua LeVasseur, Espen Skoglund, and Uwe Dannowski, University of Karlsruhe

Summarized by Feng Qian

The paper presented a new algorithm, time ballooning, for better scheduling of virtual machines in a multiprocessor environment. The combination of techniques enables scalable multiprocessor performance with flexible virtual processor scheduling. Experimental results demonstrate that the new approach is effective.

Using Hardware Performance Monitors to Understand the Behavior of Java Applications

Peter F. Sweeney, Brendon Cahoon, Perry Chen, David Grove, and Michael Hind, IBM T.J. Watson Research Center; Mathias Hauswirth and Amer Diwan, University of Colorado at Boulder

Summarized by Feng Qian

Large Java applications have many complex components. The paper introduces the design of an extension of a Java Virtual Machine (JikesRVM) for helping programmers to understand the application behaviors.

The new extension generates traces of hardware performance monitor counters. The mechanism can generate separate traces for each thread in a multithreaded and multiprocessor environment. The events, such as instruction per circle (IPC), cache misses, etc., expose the behavior of a Java application at the architecture level. These traces are useful for JVM developers to improve JIT compilers and garbage collectors. Authors also reported the design of a tool, Performance Explorer, for visualizing trace data. The tool can extract metrics from a trace file. Using SPECjbb2000 as an example, the paper shows anomalies observed by Performance Explorer.

vBlades: Optimized Paravirtualization for the Itanium Processor Family

Daniel J. Magenheimer and Thomas W. Christian, Hewlett-Packard Laboratories

Summarized by Vivek Haldar

Daniel Magenheimer specifies that, because of their design, some processors are more "virtualizable" than others. The Intel x86 and Itanium architectures are hard to virtualize, while the PowerPC and future Intel architectures (Vandebuilt) are easier to virtualize. When an architecture cannot be fully virtualized, this has adverse impacts on both complexity and performance of a virtual machine. Parts of guest operating systems have to be dynamically rewritten, page tables need to be explicitly managed, and privilege-level leakage must be guarded against. Performance suffers due to additional ring crossings and an increased number of context switches between the virtual machine monitor and the guest OS.

The alternative to this is paravirtualization. The virtual machine monitor provides an interface similar but not identical to the physical machine. The guest OS in turn needs to be modified to accommodate this differing abstraction. The advantage of this approach is that full multi-application commercial OSes can be supported, application-level modification is not needed, and there is near-native performance. The disadvantage, of course, is that the guest OS needs to be modified. The author described vBlades, an HP Labs research prototype. It is an Itanium-based hostless virtual machine monitor that runs on bare metal. It provides the capability for full virtualization. A few sensitive instructions are statically translated. An API for paravirtualization is provided. It achieves within 2% of native performance.

Kernel Plugins: When a VM Is Too Much

Ivan Ganev, Greg Eisenhauer, Karsten Schwan, Georgia Institute of Technology

Summarized by Vivek Haldar

Ivan Ganev describes an extension mechanism for operating system kernels that provides safety, extensibility, and low performance overhead. The claim is that full virtualization is not necessary for providing strong isolation to kernel plugins—using virtual machines to solve this is overkill. Virtual machines are not lightweight and have to deal with a whole array of low-level machine issues, such as the BIOS, I/O, and other legacy hardware.

The alternative is to use kernel plugins that employ other mechanisms for safety and isolation. This is done with a combination of hardware and software techniques. The hardware memory man-agement unit is used to enforce segmentation and memory isolation. Dynamic code generation enables arbitrary and heterogeneous adaptation on the fly. Dynamic linking maintains a clean interface between the kernel and plugins and manages namespaces.

This architecture was evaluated on a client-server benchmark. An in-kernel Web server (khttpd) was used on the server. The client was set up to be much faster than the server so that the server could be saturated. The cost of running a null plugin was negligible. The throughput of the server with and without the plugin was almost the same. Future avenues of work include fault recovery and isolation, and an IA64 port.

The Virtual Processor: Fast, Architecture-Neutral Dynamic Code Generation

Ian Piumarta, Université Pierre et Marie Curie

Summarized by Yahya H. Mirza

Ian Piumarta presented VPU, a reusable dynamic code generation infrastructure that can be used as a back end for dynamically compiled languages. Piumarta emphasized that a key element of VPU's design was to make adding dynamic code generation capabilities to an existing application essentially "plug-and-play." Today the vast majority of compiler infrastructures are either designed for static compilation, focus on low-level code generation, or are tightly coupled to their underlying source languages. These issues make it difficult to retarget current compiler infrastructures to other applications or language implementations. Additionally, Piumarta illustrated how a client interacts with the VPU's stack-based, processor-independent computational model to generate efficient native code.

The presentation also described the phases of the VPU's compilation process, including conversion to an internal abstract representation, application of several optimizations, instruction selection, register allocation, and native-code generation. Since the VPU tries to generate code as fast as possible, it only implements a small number of processor-independent optimizations. These optimizations are designed to occur in parallel with other traversals of the VPU's abstract representation, such as type or control flow analyses. Instruction selection is implemented through a table-driven approach using a small number of heuristics. The tables themselves are generated by feeding a processor-description file to a program called cheeseburg, which shares similarities with existing instruction selection generators such as iburg and lburg.

Systems using VPU are insulated from the underlying processor architecture and are supported on all VPU platforms, including the Pentium, SPARC, and PowerPC architectures. The VPU currently serves as the execution engine for the YNVM dynamic interactive incremental compiler and as the code generator for the JNJVM.

LIL: An Architecture-Neutral Language for Virtual-Machine Stubs

Neal Glew, Spyridon Triantafyllis, Michal Cierniak, Marsha Eng, Brian Lewis, and James Stichnoth, Intel

Summarized by Feng Qian

Machine code stubs are often used in implementing high-performance runtime systems for languages such as Java and CLI. To ease the task of coding, the authors presented a domain-specific language, LIL, for describing the functionality of such code stubs in a high-level, architecture-neutral manner. A special compiler transfers the description in LIL to architecture-dependent native instructions. LIL also has engineering benefits, such as improved readability and validity checks of stubs. The LIL compiler is faster and produces efficient machine code for stubs.

Detecting Data Races Using Dynamic Escape Analysis Based on Read Barrier

Hiroyasu Nishiyama, Hitachi, Ltd.

Summarized by Feng Qian

Data race can result in unexpected behaviors, and data race detection is an important method for locating potential bugs in concurrent programs. This paper proposed a new dynamic data race detection algorithm for Java. Based on the observation that only objects truly accessed by multiple threads require data-race monitoring, the new approach uses read-barrier to build the set of objects potentially subjected to data race. The number of monitored objects was reduced when compared with a write-barrier-based approach, which assumes all objects reachable from global objects are escaping. Furthermore, the author improves the dynamic escape analysis of arrays by dividing an array object into sub-blocks. The smaller number of monitored objects at runtime reduces the cost of dynamic data race detection and also improves the precision (reducing false alarms). The implementation of the proposed method and evaluation on a set of standard Java benchmarks shows the new approach is superior, both in accuracy and efficiency, to existing write-barrier approaches.

Towards Dynamic Interprocedural Analysis in JVMs

Feng Qian and Laurie Hendren, McGill University

Summarized by Vivek Haldar

The goal of this paper, presented by Feng Qian, was to perform interprocedural analysis in order to support speculative optimizations in a JIT compiler. This is a challenging problem because: (1) it is hard to construct a high-quality call graph efficiently; (2) dynamic class loading must be handled; and (3) the analysis must accommodate unresolved symbolic references. The problem attacked in this paper was the first one: to construct a call graph dynamically.

The call graph is constructed incrementally, under conservative assumptions. Profiling stubs are inserted into methods to accomplish this. Rapid type analysis and class hierarchy analysis is used to resolve non-virtual and interface method calls. The runtime overhead for this is 2–3%. These results are optimistic, and future work hopes to undertake and make use of more advanced interprocedural analysis.

Java Just-in-Time Compiler and Virtual Machine Improvements for Server and Middleware Applications

Nikola Grcevski, Allan Kielstra, Kevin Stoodley, Mark Stoodley, and Vijay Sundaresan, IBM Canada Ltd.

Summarized by Yahya H. Mirza

IBM Canada's JVM product team presented a series of optimizations to enhance server and middleware performance. These optimizations are shipped as a part of the IBM Developer Kit for Java and the J9 Java Virtual Machine products. As a result of going over large amounts of customer-specific Java code, IBM identified three issues that significantly impacted performance: bytecode generation, finally blocks, and large usage of exceptions. To remedy these and other performance issues, IBM introduced 12 separate enhancements, including optimizations to synchronization and Java class libraries.

Server performance optimizations include both JIT and VM improvements. Many of the server enhancements target, in particular, the SPECjbb2000 benchmark. These optimizations include object allocation inlining, lock coarsening, thread-local heap batch clearing, and the utilization of the Intel SSE instructions. The performance of middleware applications have been improved through the SPEC-jAppServer2002 benchmark. Start-up time is improved through multiple recompilation strategies by the JIT. Interface dispatch is optimized by polymorphic inline caches. In addition, 64-bit variables, themselves used to perform unsigned 32-bit calculations, are recognized and dealt with. Finally, code reordering is utilized to minimize instruction cache misses and branch mispredictions.

The results from this project indicate that such performance improvements are not necessarily additive, and some are platform specific. Thus the performance improvements achieved are not indicative of potential improvements for future platforms. The SPECjAppServer-2002 benchmark shows the potential of these optimizations: Polymorphic inline caches and code reordering give a combined improvement of 14% for the IBM Developer Kit for Java.

Java, Peer-to-Peer, and Accountability: Building Blocks for Distributed Cycle Sharing

Ali Raza Butt, Xing Fang, Y. Charlie Hu, and Samuel Midkiff, Purdue University

Summarized by Ananth I. Sundararaj

This paper, presented by Ali Raza Butt, is based on the increased popularity of grid systems and cycle sharing across organizations. The authors attempt to build one such system that would be scalable and provide means to locate resources and further ensure that these resources are used fairly. The main goal is that all the participants should be able to utilize the system. The problem of resource discovery and management is solved using existing P2P networks. Portability is provided by leveraging the Java Virtual Machine. The ability to remotely monitor Java programs' progress provides for some security. The authors have developed a distributed credit-based system of accountability to ensure fairness.

Because cheaters can be effectively and easily removed from the system, the overhead for monitoring jobs is virtually eliminated. So the main building blocks for providing fair cycle sharing are peer-to-peer networks, Java-based progress monitoring and security, and credit-based accountability mechanisms. More information on this project can be accessed at *http://expert.ics.purdue.edu/~butta.*

Towards Virtual Networks for Virtual Machine Grid Computing

Ananth I. Sundararaj and Peter A. Dinda, Northwestern University

Summarized by Ananth I. Sundararaj

The work has been done in the context of Virtuoso. The high-level aim of the Virtuoso project is to provide arbitrary amounts of computational power to ordinary people to perform distributed and parallel computations. The traditional methodology of doing grid computing, which involves resource multiplexing using OS level mechanisms, addresses this aim. A problem with this approachis that it presents too much complexity both from the perspective of the resource provider and that of the resource user. Virtuoso proposes to do grid computing in the context of OS-level virtual machines, where the abstraction is that of a raw machine.

A very interesting networking problem shows up in this new context. A particular user's virtual ma-chines are spread over a number of foreign networks. These machines are at the mercy of the foreign network administrator for their network connectivity. The authors wish to move this network management problem back to the home network of the user. VNET is the virtual network tool that accomplishes this. The authors provided performance results for VNET and showed that its performance is quite close to that attained in the underlying network. VNET is an overlay network of VNET daemons and has a lot of potential to improve performance through, for example, network reservation. The first iteration of VNET is publicly available from the Virtuoso Web site. More information on this project can be accessed at *http://virtuoso.cs.northwestern.edu*.

WORK-IN-PROGRESS REPORTS

Efficient Code Caching for an Embedded Dynamic Adaptive Compiler

Oleg Pliss and Bernd Mathiske, Sun Microsystems, Inc.

Summarized by Maria Cutumisu

Bernd Mathiske and Oleg Pliss reported on various code caching techniques in an embedded Java Virtual Machine (JVM) for memory-constrained devices, such as mobile phones. In such environments, the presence of a dynamic adaptive compiler is salutary, as the compiled code cache management becomes performance critical.

The compiled code cache can be dynamically adjusted in size due to combined results of profiling and garbage collection feedback. Recently and frequently executed methods are profiled using a combination of sampling and instrumentation techniques with the goal of constructing a cache eviction policy based on method weight and decay.

The talk highlighted the process of code cache eviction from the perspective of the results collected from the garbage collector. Methods that are identified as nonrelevant are selected as victims, while currently executed methods are retained in the cache by setting the high bit of their weight.

The authors presented results on all EEMBC benchmarks showing large performance increases, due to the improvements in the working method set detection and cache size management.

Solaris Zones: Operating Systems Support for Server Consolidation

Andrew Tucker and David Comay, Sun Microsystems, Inc.

Summarized by Maria Cutumisu

Andrew Tucker and David Comay introduced Zones, a new operating system abstraction for partitioning systems such that multiple applications run in isolation from each other on the same hardware, within a single operating system instance. Zones has an abstraction layer that separates applications from the physical attributes of the designated machine.

Different zones can be administered in a similar manner on separate machines. A zone can have access to dedicated resources or can share resources with other zones. Each zone has its own name service identity, password file, and root user. With Zones, there are multiple virtualized views of the process table corresponding to processes running within individual zones, as reflected in the /proc file system. Moreover, each zone has a virtualized /etc/mnttab file that shows only file system mounts in that zone. Even if a zone is compromised by an intruder, the system and other zones are not affected.

The isolation provided by Zones prevents processes running in different zones from monitoring or altering each other, seeing each other's data, or manipulating the underlying hardware. The cost of running multiple workloads on the same system is reduced through a better hardware utilization, reduced infrastructure overhead, and lower administration costs. The authors presented results showing that the performance impact from using zones is negligible.

During the talk, the authors indicated several resources supporting their system, including http://www.sun.com/bigadmin/content/zones. Zones are developed as part of the N1 Grid Containers feature in Solaris 10. A version of Solaris 10 that includes an initial implementation of zones is available for download from http://wwws.sun.com/software/solaris/10.

Opcode Level Energy Consumption Model for a JVM

Sebastian Lafond and Johan Lilius, Turku Centre for Computer Science, Finland

Summarized by Vivek Haldar

Sebastian Lafond presented a simulation to measure the energy consumption of Java programs on mobile devices. Java bytecode is executed on the KVM (an implementation of the J2ME standard, which interprets Java bytecode) running on an ARM processor, and the resulting instruction trace is passed through an instruction-level energy profiler. The authors found that some KVM stages consume a constant amount of energy independently of the Java application being run. The most energy-expensive operation was the dup2_x2 instruction.

Real-Time Garbage Collector for Embedded Applications in CLI

Okehee Goh and Yann-Hang Lee, Arizona State University; Ziad Kaakani and Elliot Rachlin, Honeywell International Inc.

Summarized by Vivek Haldar

In the .NET Common Language Infrastructure (CLI), determinism is an issue for time-constrained applications. However, garbage collection is non-deterministic. The goal is to schedule garbage collection by applying real-time scheduling algorithms. This can be used to control the garbage collector's pause time and do incremental garbage collection. So far, the authors have modified the Mono runtime (which uses the mark-sweep Boehm collector) to generate write barriers. This can help to make garbage collection incremental at a fine granularity.

One-Click Distribution of Preconfigured Linux Runtime State

Richard Potter, Japan Science and Technology Corporation

Summarized by Feng Qian

Richard Potter's work-in-progress reports the idea and applications of ScrapBook for User-Mode Linux (SBUML). SBUML can take a snapshot of the transient runtime state of the Linux OS, and the state can rapidly be restored in another computer. SBUML could be used to distribute preconfigured Linux runtime state for demos or debugging. More details can be found at http://sbuml.sourceforge.net.