Recent Results from the V3VEE Project

Peter Dinda

V3VEE.org
Department of EECS
Northwestern University

Lei Xia
Kyle Hale
Maciej Swiech
Chang Bae
Outline

• V3VEE Project and Palacios VMM

• High performance overlay networking
• Guarded modules for specialized guest execution modes
• VMM-based transactional memory emulation
• Memory content tracking at scale
  – (if time)
V3VEE Project

• Large multi-site effort to create an open source virtual machine monitor framework for modern architectures
  – Started at Northwestern in 2007
  – HPC and architectural research in addition to systems
• Currently involves faculty, staff, and students at Northwestern, University of New Mexico, University of Pittsburgh, Sandia National Labs and Oak Ridge National Lab
• Supported by NSF and DOE (X-Stack)
• See v3vee.org for all collaborators
Palacios
An OS Independent Embeddable VMM

• BSD-licensed virtual machine monitor
• Integrates into Linux, Kitten LWK, Minix 3, etc
• Currently in Release 1.3
  – Next release planned for summer
• Git repository, including full development branch, accessible from v3vee.org

Is it Feasible to Run a Supercomputer Virtualized?

Sandia Red Storm  
Cray XT3  
38208 cores  
~3500 sq ft  
2.5 MegaWatts  
$90 million

Virtualization within 3% of native…   Scalable to 6000+ nodes

Scaling Study (Weak Scaling of Application)

CTH: multi-material, large deformation, strong shockwave simulation

Within 3% of Native  
Scalable  
Palacios
VNET/P: A High Performance Overlay

• Give a collection of VMs, no matter where they run, the same network abstraction
  – No special hardware requirements
  – No boundaries
  – Migration tolerance
  – Workload-adaptive topology and routing

• VNET/P: a pure software overlay in the VMM
  – VMs see what looks like one, simple Ethernet

• How can such a system perform at the limits of the networking hardware on the fastest networks?
  – 10+ Gbit/s

Collaborators: Xia, Cui, Tang, Bridges, Lange, Dinda
VNET Generations

  - Adaptive overlay topology and forwarding
  - Supported workload-adaptive distributed virtualization (Virtuoso)
- **VNET/P [HPDC 2012] (available in Palacios now)**
  - VNET in the VMM
  - Parallelized, overlapped packet processing
  - Adaptive packet extraction / injection techniques
- **VNET/P+ [SC 2013]**
  - Optimistic interrupts
  - Cut-through forwarding (encapsulation within guest)
  - Noise-free kernel for minimal latency variance
- **VNET/P++ [HPDC 2013]**
  - Virtual TCP offload on advanced interconnects

Currently: 13 Gbps (app-to-app, Cray Gemini), latency hit limited by interrupt virtualization
Guarded Modules

• How can we provide a controlled, specialized execution environment for part of the guest?
  – Elevated privilege for guest components
    • GEARS code injection of VMM services
    • Example: VMM-injected driver has direct hardware access
  – Compartmentalized trusted computation
    • Application code need not trust the guest OS, only VMM
  – Proactive resilient computation
    • VMM-based redundant computation of critical elements

Implementation available in Palacios development branch

Collaborators: Hale, Dinda  [Also see poster]
Guarded Linux Kernel Modules

• Compile-time transformation of module sources
  – Developer determines valid entry and exit points from the module
  – Toolchain “wraps” module so these “border crossings” invoke the VMM

• Run-time state machine in VMM validates each border crossing, including stack sanity checking
  – Switches on specialized execution mode on valid “border in”, off on “border out”
  – Interleaving allowed

Invariants: Specialized execution mode active only between a programmer-specified valid “border in” and a “border out”. Guest cannot modify the module or module’s hidden state (if any).
Guarded Linux Kernel Modules Detail

Border crossing cost

With no stack check: 4000-5000 cycles
With typical stack check for NIC driver: 4000-9000 cycles
Hardware Transactional Memory

• Long-standing idea
• Will finally appear in commodity hardware later this year
  – Intel Transactional Synchronization Extensions (TSX)
    • RTM: “Restricted” Transactional Memory
    • HLE: Hardware Lock Elision
  – Intel Haswell will include this feature
• Many implementation-dependent aspects to TSX
  – Initial implementations appear to leverage existing write buffers and cache invalidate / write update mechanisms
  – Short transactions, probably best for lock-free data structures
Restricted Transactional Memory

start_label:
   XBEGIN abort_label

   body of transaction
   may use XABORT and XTEST
   may nest
   other cores may cause abort
due to real memory conflict
may also abort for numerous
implementation-specific reasons

   XEND
   transaction’s writes committed

success_label:
   handle transaction committed

abort_label:
   transaction’s writes discarded
   handle transaction aborted

• Like a database transaction, but over main memory
  – Gives you the “A” and “I” of ACID

• TM is really about inherent conflicts, but in RTM spec, numerous implementation-specific aborts “may” happen
  – A wide range of Instructions, interrupts, particular register accesses, I/O port accesses, etc.
  – Or resource limits (cache size, associativity, etc)
VMM-based Emulation of Intel RTM

• Try RTM now on existing hardware with full guest OS
  – including AMD
• Test RTM-using code for implementation-dependencies
• Evaluate code that uses future features
  – Such as unlimited size transactions
• Consider ties with software transactional memory
• Avoid decoding or emulating complex x86 instructions
• Be much faster than existing tool (Intel SDE)
• Be open

Collaborators: Swiech, Hale, Dinda  [Also see poster]
M. Swiech, K. Hale, P. Dinda, *VMM-based Emulation of Intel Hardware Transactional Memory*, NWU-EECS-13-03
Implementation soon to be available in Palacios development branch
Memory and Instruction Meta-Engine (MIME)

• VMM-based technique that allows us to *incrementally execute* a single instruction on the hardware
  – Leverages virtualization of virtual memory
  – COW-style page-flipping allows staging of memory reads/writes through VMM
  – Only instruction size needed, no decoding or emulation

• Captures memory aspects of instruction execution
  – Instruction fetch (addresses and data)
  – Memory operand reads (addresses and data)
  – Memory operand writes (addresses and data)
Key Idea 1: Separates detection of **inherent conflicts** (memory conflicts) from **implementation-specific conflicts**

Key Idea 2: MIME-based execution **only** when a transaction is running
Performance

• No transaction running
  – Full speed of execution on VMM
  – Essentially native for compute-intensive code

• Transaction running
  – 1500x slowdown
  – Yet 60x faster than full emulation (Intel SDE)

• Implementation size
  – ~1300 lines of C
  – Instruction emulation completely avoided
Memory Content Tracking At Scale

• Best-effort tracking of shared memory content across a cluster or supercomputer
  – For example, page content across all active VMs
• Service simplifies construction of other tools
  – Collective checkpointing of applications and VMs
  – VM co-migration and co-reconstruction
  – Content duplication control for resilience
• There is considerable content sharing within many parallel applications

Lei Xia’s Thesis Work
ConCORD Architecture

- Memory update monitors push content changes into DHT
- DHT contains best effort content_hash->node_list mapping
- Services query DHT or parameterize collective command it runs
Queries and Collective Commands

• Queries
  – ”How many copies are there of this content?”
  – “What nodes hold this content?”
  – “What is the degree of sharing among these nodes?”
  – “How many distinct content blocks have at least k replicas?”

• Collective Commands
  – Similar to a map-reduce, parameterized by the service
  – Underlying query is a for the content_hash->{node_list} info limited to a set of nodes of interest
  – Query result is optimistically partitioned across nodes (ideally one node handles each content hash) and delivered via callbacks to service
  – Two phase protocol exposes outdated DHT info or new updates such that end result reflects all current content on the nodes
Collective Co-checkpoint Preliminary Result

Complete operation, including data save

Two-phase protocol, finds all data to save, excludes save

First phase only, finds all ConCORD-visible data

Number of Nodes (1VM/Node, 1GB/VM)
Conclusions

• Palacios VMM is open source and freely available from us (v3vee.org)
• Most of the research I have described (and a lot I did not mention) is implemented in the codebase
• We are always looking for collaborators!
• V3VEE Project
  – http://v3vee.org

• Prescience Lab
  – http://presciencelab.org

• Peter Dinda
  – http://pdinda.org