Denali: Lightweight Virtual Machines for Distributed and Networked Systems

Andrew Whitaker, Marianne Shaw, and Steven D. Gribble
(andrew, mar, gribble)@cs.washington.edu --- **http://denali.cs.washington.edu**
Department of Computer Science and Engineering, University of Washington

The Case for Denali

A set of new application domains is emerging…

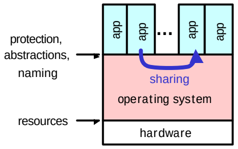
- pushing dynamic content generation code into caches and CDNs
- pushing "grassroots" or third-party Internet services into hosting infrastructure
- deploying new routing and content location services into peer-to-peer systems or application-layer routing infrastructure (e.g., FreeNet, RON)
- deploying measurement experiments into existing infrastructure (e.g., NIMI)
- … and the list goes on.

These applications all share several properties:

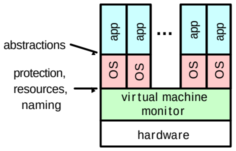
- they require the ability to safely execute untrusted code
- they benefit from the ability to scale to a large number of concurrent protection domains per physical host (hundreds, or perhaps even thousands)
- they necessitate a large degree of resource multiplexing, and correspondingly, they require performance isolation
- the degree of information sharing between applications is small

Virtual machine monitors (VMMs)

- we argue that virtual machine monitors (in the style of Vmware, Disco, and VM/370) provide a strong foundation for these new application domains
  - in a VMM, the monitor is responsible for exporting a virtualized HW/SW boundary to each virtual machine
  - each virtual machine is completely isolated from others, and can run its own "guest" OS and set of applications



Conventional OS view of the world



Denali (VMM) view of the world

Why VMMs are well-suited to these applications

1. ***VMMs don't export any shared, high-level abstractions***
   - conventional operating systems provide shared, high-level abstractions
     - e.g., file systems, network stacks
   - in practice, these abstractions are often subverted by tunneling below the abstraction layer ("layer below" problems)
     - e.g., running a packet sniffer, inducing coredumps, etc.
2. ***VMMs impose a simple sharing policy***
   - virtual machines are completely isolated from each other
     - the only sharing allowed is through the (virtual) network
   - the simplicity of this policy means we are likely to get it right!
     - it also obviates many administration issues
3. ***VMMs enforce private namespaces***
   - all names in a VM are virtual
     - one VM cannot even *name* resources in another VM
     - one VM can't gain access to resources in another due to misconfigured ACLs

Key Idea

The dangers of virtualization…

VMMs can be inefficient because existing physical architectures and OS's were not designed to support virtual machines. Potential pitfalls include:

- *High virtualization overhead*
  - virtualization requires trapping and emulating "privileged" instructions, such as I/O and the modification of page tables
- *Information loss*
  - the VMM can make poor decisions because it does not have higher-level OS knowledge; for example by scheduling a VM that is executing an idle loop
- *Implementation complexity*
  - faithfully emulating a physical architecture is made difficult by idiosyncrasies such as segmentation hardware, the BIOS, and unneeded I/O devices

Ideally, we would like to be able to codesign the machine architecture, the VMM, and the VMs' operating systems to overcome these issues.

… and our solution: "Para-virtualization"

Our approach is to make the virtual architecture exposed by the VMM similar to, but not precisely identical to, the underlying physical architecture.

"Para-virtualization" is the act of slightly and strategically modifying the underlying physical architecture while virtualizing it.

Pros:

- can drastically reduce implementation complexity
- can improve performance: opportunity to codesign virtual architecture and OS
- can improve scalability (larger # of concurrent VMs supported)

Cons:

- cannot run unmodified legacy OS's
  - but: this frees us from backwards compatibility constraints!
- can no longer run guest OS on raw physical hardware

Key research question: How should we design a virtual architecture to support a large number of concurrent VMs (rather than the other way around)?

Elements of Denali's para-virtual machine architecture

- instruction set: based on x86 instruction set
  - most instructions execute natively without requiring monitor intervention
- radically simplified x86 architecture
  - no virtual memory supported in our VMs
  - no user/kernel boundary in our VMs (implies each VM is a single protection domain)
  - limited spectrum of simple I/O devices: ethernet, disk, console
- new "purely virtual" instructions
  - idle loop instruction yields control of CPU, with a timeout allowed
  - I/O instructions for minimizing VMM/VM crossings
- virtualization-aware emulated hardware
  - batched interrupt delivery to avoid "bursts" of virtual interrupts
- no paging in either the VMM or the VMs
  - virtual machines are swapped in and out in their entirety
- redefine the semantics of non-virtualizable x86 instructions as "undefined"

Isolation and virtual machines

A second key research challenge: How do we enforce isolation (in both the security and the performance sense) across VMs?

Security isolation:

- there is **no** sharing policy across VMs; complete isolation is the default
- sharing is allowed only through a network; each VM can define its own security policy for deciding what to expose to the network
  - can apply firewalling, IDS, and switching techniques to isolate networks

Performance isolation:

- fairness mechanisms must be implemented in VMM
- fair queueing across virtual ethernet cards (both Rx and Tx path)
  - also lazy receiver processing to avoid DOS inside VMM
- static physical memory allocation across VMs that doesn't change in response to individual VMs physical memory demands
  - either entire VM is (allocated) in physical memory or entire VM is swapped out

Current status

- implemented the "Yakima" VMM for our para-virtualized x86
  - type I VMM (runs on bare HW), built using Flux OSKit, a non-preemptive kernel, includes memory management, virtual ethernets, timers, serial ports, keyboards
  - in process of adding support for disk and fair queueing of I/O
- implemented a basic library OS
  - support for preemptive multithreading, has a port of BSD networking stack, includes subset of libC and the BSD sockets API
  - have written a web server using this libOS
  - support for privileged "supervisor" VM, which can create/destroy/manage other VMs

Example performance numbers, on 930MHz PIII w/ 100Mb/s 3c90x ethernet:

- VMM physical interrupt NULL handler: 3545 cycles
- VMM physical interrupt enable/disable: 17 cycles
- creation of 12MB VM: 38,831,728 cycles (bzero dominated)
- destruction of 12MB VM: 11,184 cycles
- ethernet frame Rx, phys NIC to virt NIC: 12,200 cycles (linux driver dominated)
- ethernet frame Rx, virt NIC to VM driver: 1,581 cycles + 3.23 cycles/payload-byte
- NULL supervisor VM "syscall" into VMM: 427 cycles

Future work

Near term:

- understand the strengths/limitations of a single processor VMM/VM system
- attempt to scale the Yakima VMM up to 500 VMs, and see what breaks
  - we already know there are implications in the OS's, such as TCP stacks and timers, having "unpopular" VMs not consume undue resources, and issues of how interrupts are dispatched and vectored

Longer term:

- explore the relationship between clusters and VMMs
  - use the fine-grained performance and resource mgmt. control that VMMs offer to implement virtual clusters inside physical clusters, à la "cluster reserves"
  - use VM checkpoint/migrate as a load-balancing or virtual cluster growth mechanism
- explore the relationship between VMMs and the wide-area
  - transparent replication and migration of services across the WAN?
- tackle the data issue
  - many services are data-driven: how do you "push" a large database along with a VM?

Denali: Lightweight virtual machines for distributed and networked systems

Andrew Whitaker, Marianne Shaw, and Steven D. Gribble
Department of Computer Science and Engineering, The University of Washington

SOSP 2001 Poster session submission

Advances in networking and computing technologies have accelerated the proliferation of infrastructure such as content distribution, caching, middleware services, and network measurement testbeds. In conjunction with this, several application domains have begun to emerge that are not well-supported by existing technologies, such as distributing dynamically generated web content, rapidly deploying untrusted Internet services into hosting infrastructure, and injecting network measurement code into network experimentation infrastructure.

Although several sandboxing technologies have been proposed, none have the combination of water-tight isolation and the ability to scale to a large number of protection domains required by these new applications. The Denali project seeks to remedy this situation by constructing a software platform that provides both performance and security isolation to hundreds (or perhaps thousands!) of untrusted applications/OSs executing concurrently on a single physical host.

To achieve this high degree of scalability and isolation, we are exploring the use of virtual machine monitors (VMMs). A VMM is a software layer that runs immediately on top of the hardware/software boundary, virtualizing all resources to give higher-level virtual machines (VMs) the illusion of their own dedicated physical machine. While VMs are known to have strong isolation properties, they have traditionally been regarded as a heavyweight mechanism, permitting only a small number of virtual machines to execute concurrently. As a step towards building lightweight VMs, we are exploring the notion of "para-virtualization", in which the virtual architecture that we expose to VMs is similar to but slightly divergent from the underlying physical architecture. Allowing this divergence permits optimizations that improve performance and scalability, and reduce implementation complexity.

Using the Flux OSKit, we have implemented an x86 VMM, called Yakima, which directly runs on the bare hardware. Yakima uses para-virtualization in several ways, including the delivery of batched interrupts upon context-switching into a VM, providing a reduced set of simplified I/O devices, and the elimination of virtual memory and user/kernel boundaries in a VM. We have developed an OS library (similar to an Exokernel libOS) tuned to our para-virtual architecture. This library, which contains preemptive threads and a networking stack, has been used to implement a Yakima-supported supervisor VM (able to create, destroy, and allocate system resources to individual VMs) and also a web server application.

We are currently in the process of quantifying Yakima's performance and addressing the challenge of resource management across virtual machines. To fully isolate one VM from another, each VM's resource usage (e.g., CPU consumption, I/O rates, memory footprint) must be bounded by the VMM. Once we have completely isolated lightweight VMs, we intend to leverage this new mechanism to explore several research topics: (1) VMs as sandboxes to enable web servers to dynamically inject new content-generation code into CDNs or web caching systems; (2) using VMs to enable untrusted code authors to upload new Internet services into a virtual hosting platform; and (3) the role of VMs as a resource container in a cluster-of-workstations to create the effect of isolated, dynamically resizable ``virtual clusters" within a physical cluster.