Scale 14x:
The Latest From the Xen Project

Lars Kurth
Cat Herder, Xen Project
Chairman, Xen Project Advisory Board
Xen 4.x Hypervisor Release History

Xen 4.x Number of New Major Features
• Developed per month (y axis via ◆)
• Absolute number (number beside ◆)

- 2013: 12
- 2014: 20
- 2015: 36
- 2016: 38
Hypervisor Git Commits

Year | Commits
--- | ---
2011 | 2000
2012 | 2200
2013 | 2400
2014 | 2600
2015 | 2800
Rate of Innovation is Accelerating

While quality and security requirements are increasing simultaneously
A type-1 Hypervisor

with a twist
Architectural Advantages

Density: It's thin
Excellent for supporting many very small workloads (e.g. unikernels)

Scalability: It can support huge numbers of VMs
Terrific for highly dense workloads (e.g. unikernels, disaggregation, …)

Security: Host OS isolated within a VM
This makes it harder to attack the Host OS

Scheduling: Can use dedicated scheduler
Enables specialized workload profiles (mix and match schedulers on one host)

Paravirtualization: Simplified interface
Easy to implement a unikernel base
Enables fast boot times necessary for unikernels
The Interface Between Xen and Unikernels

Virtualisation Modes
The Future of Virtualization Modes
Implications for Unikernel bases
Evolution of Virtualization Modes (x86)

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Xen</th>
<th>Mode</th>
<th>With</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>3.0</td>
<td>HVM</td>
<td>PV Drivers</td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>4.0</td>
<td>HVM</td>
<td>PVHVM Drivers</td>
</tr>
<tr>
<td>PVH</td>
<td>4.4/4.5</td>
<td>PV</td>
<td>pvh=1</td>
</tr>
<tr>
<td>PV</td>
<td></td>
<td>PV</td>
<td></td>
</tr>
</tbody>
</table>

**Unikernel Bases:**
Primarily depend on PV
E.g. rumprun and Mini-OS
Will work on Xen based clouds and hosting services
<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Disk and Network</th>
<th>Interrupts &amp; Legacy Boot</th>
<th>Privileged Instructions, Page Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td></td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>PV Drivers</td>
<td>PV</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>PVHVM</td>
<td>HVM</td>
<td>PVHVM Drivers</td>
<td>PV</td>
<td>PV</td>
<td>VS</td>
</tr>
<tr>
<td>PVH</td>
<td>PV</td>
<td>pvh=1</td>
<td>PV</td>
<td>PV</td>
<td>PV</td>
</tr>
<tr>
<td>PV</td>
<td>PV</td>
<td></td>
<td>PV</td>
<td>PV</td>
<td>P</td>
</tr>
</tbody>
</table>

PV = Paravirtualized
VS = Software Virtualized
VH = Hardware Virtualized
Making PVH better

The **motivation** behind PVH
- HVM (like) Dom0: performance & Dom0 modification
- PVH as fast or faster than HVM
- PVH runs a PV guest within a HVM container (essentially a mix of PV & HVM)

**BUT:** PVH inherits all the **PV limitations**, e.g.
- Paging restrictions, lack of access to emulated devices (if needed), …
- Concept designed prior to “additional quality and security requirements”

**Solution:** HVMLite to eventually replace PVH
- A lot simpler to implement: less code to maintain and thus to keep secure
- Behaves exactly like PVH (but internal implementation different)
- HVM without QEMU
<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>Without PVHVM</th>
<th>Disk and Networking</th>
<th>Interrupts &amp; Timers</th>
<th>Emulated Motherboard</th>
<th>Page Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td></td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>VH</td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>PV</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>VH</td>
</tr>
<tr>
<td>PVHVM</td>
<td>HVM</td>
<td>PVHVM Drivers</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>VH</td>
</tr>
<tr>
<td>PVH / HVMLite</td>
<td>HVM</td>
<td>pvh=1</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>VH</td>
</tr>
<tr>
<td>PV</td>
<td>PV</td>
<td></td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>P</td>
</tr>
</tbody>
</table>

Currently:

builder = "hvm"
device_model_version="none"

There will need to be some sort of migration strategy from pvh=1
Status

Xen 4.7 (June 2016)
  – HVMlite DomU support in xen.git
  – Agree on config file changes and naming (PVH or HVMlite)

Ongoing
  – HVMlite Dom0 prototype for FreeBSD
  – No Linux implementation yet
  – Some clean-up required
  – Interfaces not yet declared stable (but almost there)
  – Benchmarks already very impressive
PVH / HVMLite not currently used as unikernel base
- Unikernel developers make sure the architecture works for you (before APIs are declared stable)
- MiniOS / Rumprun not yet ported to HVMLite (some different approaches needed compared to pure PV)

Opportunity: Avoid Duplication
- There was a bit of duplication of unikernel bases in the early days of unikernel development (MiniOS clones)
Performance

Focus on significant performance and scalability improvements since Xen 4.5+
Performance Improvements

Examples

HPET: Better and faster resolution values
Parallel memory scrubbing on boot (large machines)
Lower interrupt latency for PCI passthrough (machines > 2 sockets)
Soft affinity for non-NUMA machines
Multiple IO-REQ services for guests
(remove bottlenecks for HVM guests by allowing multiple QEMU back-ends)
SandyBridge: VT-d posted interrupts for HVM
(I/O intensive workloads)

Grant table scalability by using finer grained locks
Ticket locks for improved fairness and scalability

...
Benchmarks: Xen & KVM
(Kernel 4.2 with Xen 4.5 & QEMU 2.3)
Xen 4.6 should be even better

Schedulers

Overview
Possibilities

Resources:
Docs: bit.do/xen-schedulers
The Xen Project Hypervisor supports several different schedulers with different properties.

Different schedulers can be assigned to...

... an entire host

e.g. Credit2 Scheduler
The Xen Project Hypervisor supports several different schedulers with different properties.

Different schedulers can be assigned to...

... an entire host

... a pool of physical CPU’s (=CPU Pool) on a host (VMs need to be assigned to a pool or pinned to a CPU)

e.g. RTDS Scheduler

e.g. Credit Scheduler
Scheduler parameters can be modified per …
# Schedulers Overview

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Use-cases</th>
<th>Xen 4.6</th>
<th>Plans for 4.7+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit</td>
<td>General Purpose</td>
<td>Supported Default</td>
<td>Supported Default</td>
</tr>
<tr>
<td>Credit 2</td>
<td>General Purpose, Optimized for lower latency, higher VM density</td>
<td>Experimental</td>
<td>Supported</td>
</tr>
<tr>
<td>RTDS</td>
<td>Soft &amp; Firm Real-time Multicore, Embedded, Automotive, Graphics &amp; Gaming in the Cloud, Low Latency Workloads</td>
<td>Hardening Optimizations Better XL support</td>
<td>Adaptive granularity</td>
</tr>
<tr>
<td>ARINC 653</td>
<td>Hard Real-time Single core, Avionics, Drones, Medical</td>
<td>Supported Compile time</td>
<td>No change</td>
</tr>
</tbody>
</table>

**Legend:**
- likely in 4.7
- possible in 4.7
RTDS Scheduler: Use-cases

Embedded & automotive
Latency sensitive workloads
Guaranteed QoS

Cloud based gaming, video, TV delivery, …
Guaranteed QoS
(Price ➔ SLAs ➔ QoS)
Other major and ongoing Innovations in Xen

Hardware Support
Graphics
Security: VMI
Security: QEMU and Isolation
Security: xSplice
Security: Configurability
…
X86 Hardware Support

Intel Platform QoS technologies (CMT, CAT, MBM, …)
Virtual Performance Monitoring Unit vTPM v2.0

Code/Data Prioritization
Memory Protection Keys
VMX TSC scaling
Intel PState Drivers
Posted Interrupts
…
ARM Hardware Support

Tracking ARM/ARM-partner server roadmap
Hardening (more 64 bit servers in Test Lab)
Live Migration
What is Intel GVT-g (XenGT)?

Demo Scenarios:

1. Ubuntu Dom0: 2 Windows VMs
2. VM1: World Rally Championship 3
3. VM2: Google Earth
4. VM1/VM2/Dom0 switch

Watch the demo at
https://www.youtube.com/watch?v=V2i8HCcAnY8

Virtual GPU per VM
Performance critical resources directly assigned to VM
Intel GVT-g (XenGT) – What’s next?

GVT-g support is partly out-of-tree

In use by XenClient 5.5 and XenServer Dundee

Most Xen patches are part of xen.git

BUT: some Linux and QEMU patches that are still in progress

Motivation: create a common code base for Xen & KVM

Similar approach for embedded developed by GlobalLogic
(for ARM based architectures)
Demo: Virtual Machine Introspection

Watch the demo at https://www.youtube.com/watch?v=ZJPHfpDiN4o

Credit: Tamas K Lengyel
Installed in-guest agents, e.g. anti-virus software, VM disk & memory scanner, network monitor, etc.

Anti virus storm, deployment/maintenance, …
A new model for Cloud Security?

**VMI Approach**

- Dom0
  - Dom0 Kernel
    - Drivers
- Security Appliance VM\(_1\)
  - Introspection Engine
- Several VMs (VM\(_2\), VM\(_3\), ..., VM\(_n\))
  - App
  - Guest OS
  - Agent

Hybrid approach: no need to move everything outside (chose best trade-off)

Protected area

XSM/Flask
Other Security Features being developed

QEMU and Emulation for Xen secure by default (4.7)
  – Response to Venom and other QEMU bugs as an alternative to Stub Domains
  – Defense in depth mechanisms to secure the execution of QEMU + Inbuilt Emulation

Hot Patching or x-splice (4.7+)
  – Response to “Cloud Reboots” of 2014 and 2015
  – Hypervisor and Workload generation Tooling
  – Start with some use-cases and successively add less common ones

Better Configurability (4.7+)
  – Response to criticism from Invisible Things Labs
  – Use KCONFIG to disable Hypervisor functionality
  – A more wholesome approach to disable and remove undesired functionality
The project has a history of proactively innovating
The rate of innovation is increasing
(e.g. more features, more quickly)
The demands on the project are shifting
(e.g. quality and security, conflicting requirements)
The project has a track record of adapting
(e.g. to criticism, challenges, …)
Best Platform for Unikernels in the Cloud
(e.g. reach, innovation, unique features)