Xen is a VMM or hypervisor developed by the Computing Laboratory at the University of Cambridge,

United Kingdom, in 2003. Since 2010 Xen has been free software, developed by the community of users

and licensed under the GNU General Public License (GPLv2). Several operating systems, including

Linux, Minix, NetBSD, FreeBSD, NetWare, and OZONE, can operate as paravirtualized Xen guest

operating systems running on x86, x86-64, Itanium, and ARM architectures.

The goal of the Cambridge group, led by Ian Pratt, was to design a VMM capable of scaling to about

100 VMs running standard applications and services without anymodifications to the application binary

interface (ABI). Fully aware that the x86 architecture does not support efficiently full virtualization, the

designers of Xen opted for paravirtualization.

Next we analyze the original implementation of Xen for the x86 architecture discussed in [41]. The

creators of Xen used the concept of domain (Dom) to refer to the ensemble of address spaces hosting a guest OS and address spaces for applications running under this guest OS. Each domain runs on a virtual

x86 CPU. Dom0 is dedicated to the execution of Xen control functions and privileged instructions, and

DomU is a user domain (see Figure 5.6) .

The most important aspects of the Xen paravirtualization for virtual memory management, CPU

multiplexing, and I/O device management are summarized in Table 5.2 [41]. Efficient management

of the translation look-aside buffer (TLB), a cache for page table entries, requires either the ability to

identify the OS and the address space of every entry or to allow software management of the TLB.

Unfortunately, the x86 architecture does not support either the tagging of TLB entries or the software

management of the TLB. As a result, address space switching, when the VMMactivates a different OS,

requires a complete TLB flush. This has a negative impact on performance.

The solution that was adopted was to load Xen in a 64MB segment at the top of each address space

and delegate the management of hardware page tables to the guest OS with minimal intervention from

Xen. The 64 MB region occupied by Xen at the top of every address space is not accessible or not

remappable by the guest OS. When a new address space is created, the guest OS allocates and initializes

a page from its own memory, registers it with Xen, and relinquishes control of the write operations to

the VMM. Thus, a guest OS could only map pages it owns. On the other hand, it has the ability to batch

multiple page-update requests to improve performance. A similar strategy is used for segmentation.

The x86 Intel architecture supports four protection rings or privilege levels; virtually all OS kernels

run at Level 0, the most privileged one, and applications at Level 3. In Xen the VMM runs at Level 0,

the guest OS at Level 1, and applications at Level 3.

Applications make system calls using the so-called hypercalls processed by Xen. Privileged instructions

issued by a guest OS are paravirtualized and must be validated by Xen.When a guest OS attempts

to execute a privileged instruction directly, the instruction fails silently.

Memory is statically partitioned between domains to provide strong isolation. To adjust domain

memory, XenoLinux implements a balloon driver, which passes pages between Xen and its own page

allocator. For the sake of efficiency, page faults are handled directly by the guest OS.

Xen schedules individual domains using the borrowed virtual time (BVT) scheduling algorithm

discussed in Section 6.11. BVT is a work conserving7 and low-latency wake-up scheduling algorithm.

BVT uses a virtual-time warping mechanism to support low-latency dispatch to ensure timely execution

when this is needed – for example, for timely delivery of TCP acknowledgments.

A guest OS must register with Xen a description table with the addresses of exception handlers for

validation. Exception handlers are identical to the native x86 handlers. The only one that does not follow

this rule is the page fault handler, which uses an extended stack frame to retrieve the faulty address

because the privileged register CR2, where this address is found, is not available to a guest OS. Each guest OS can validate and then register a “fast” exception handler executed directly by the processor

without the interference of Xen. A lightweight event system replaces hardware interrupts. Notifications

are delivered using this asynchronous event system. Each guest OS has a timer interface and is aware

of “real” and “virtual” time.

XenStore is a Dom0 process that supports a system-wide registry and naming service. It is implemented

as a hierarchical key-value storage; a watch function of the process informs listeners of changes

to the key in storage to which they have subscribed. XenStore communicates with guest VMs via shared

memory using Dom0 privileges rather than grant tables.

The Toolstack is another Dom0 component responsible for creating, destroying, and managing the

resources and privileges of VMs. To create a new VM a user provides a configuration file describing

memory and CPU allocations as well as device configuration. Then the Toolstack parses this file and

writes this information in the XenStore. Toolstack takes advantage of Dom0 privileges to map guest

memory, to load a kernel and virtual BIOS, and to set up initial communication channels with the

XenStore and with the virtual console when a new VM is created.

Xen defines abstractions for networking and I/O devices. Split drivers have a front-end in the DomU

and a back-end in Dom0; the two communicate via a ring in shared memory. Xen enforces access control

for the shared memory and passes synchronization signals. Access control lists (ACLs) are stored in

the form of grant tables, with permissions set by the owner of the memory.

Data for I/O and network operations move vertically through the system very efficiently using a

set of I/O rings (see Figure 5.7). A ring is a circular queue of descriptors allocated by a domain and

accessible within Xen. Descriptors do not contain data; the data buffers are allocated off-band by the

guest OS. Memory committed for I/O and network operations is supplied in a manner designed to

avoid “cross-talk,” and the I/O buffers holding the data are protected by preventing page faults of the

corresponding page frames.

Each domain has one or more virtual network interfaces (VIFs) that support the functionality of

a network interface card. A VIF is attached to a virtual firewall-router (VFR). Two rings of buffer

descriptors, one for packet sending and one for packet receiving, are supported. To transmit a packet,

a guest OS enqueues a buffer descriptor to the send ring, then Xen copies the descriptor and checks

safety and finally copies only the packet header, not the payload, and executes the matching

rules.

The rules of the form (<pattern>,<action>) require the action to be executed if the pattern is

matched by the information in the packet header. The rules can be added or removed by Dom0; they

ensure the demultiplexing of packets based on the destination IP address and port and, at the same time,

prevent spoofing of the source IP address. Dom0 is the only one allowed to directly access the physical

IDE (Integrated Drive Electronics) or SCSI (Small Computer System Interface) disks. All domains

other than Dom0 access persistent storage through a virtual block device (VBD) abstraction created and

managed under the control of Dom0.

Xen includes a device emulator, Qemu, to support unmodified commodity operating systems. Qemu

emulates a DMA8 and can map any page of the memory in a DomU. Each VM has its own instance of

Qemu that can run either as a Dom0 process or as a process of the VM.

Xen, initially released in 2003, underwent significant changes in 2005, when Intel released the VT-x

processors. In 2006 Xen was adopted by Amazon for its EC2 service, and in 2008 Xen running on Intel’s

VT-d passed the ACPI S3 9 test. Xen support for Dom0 and DomU was added to the Linux kernel in 2011.

In 2008 the PCI pass-through was incorporated for Xen running on VT-d architectures. The PCI10

pass-through allows a PCI device, whether a disk controller, network interface card (NIC), graphic card,

or Universal Serial Bus (USB), to be assigned to a VM. This avoids the overhead of copying and allows

setting up of a driver domain to increase security and system reliability. A guest OS can exploit this

facility to access the 3D acceleration capability of a graphics card. To prepare a device for pass-through,

one must know its BDF.11

An analysis of VM performance for I/O-bound applications under Xen is reported in [298]. Two

Apache Web servers, each under a different VM, share the same server running Xen. The workload

generator sends requests for files of fixed size ranging from 1KB to 100 KB.When the file size increases

from 1 KB to 10 KB and to 100 KB, the CPU utilization, throughput, data rate, and response time are,

respectively: (97.5%; 70.44%; 44.4%), (1,900; 1,104; 112) requests/s, (2,018; 11,048; 11,208) KBps,

and (1.52; 2.36; 2.08) msec. From the first group of results we see that for files 10 KB or larger the

system is I/O bound; the second set of results shows that the throughput measured in requests/s decreases

by less than 50% when the system becomes I/O bound, but the data rate increases by a factor of five

over the same range. The variation of the response time is quite small; it increases about 10% when the

file size increases by two orders of magnitude.

The paravirtualization strategy in Xen is different from the one adopted by a group at the University

of Washington, the creators of the Denali system [372]. Denali was designed to support a number

of virtual machines running network services one or more orders of magnitude larger than Xen. The

design of the Denali system did not target existing ABI. It does not support some features of potential

guest operating systems – for example, it does not support segmentation. Denali does not support

application multiplexing, running multiple applications under a guest OS, whereas Xen does.

Finally, a few words regarding the complexity of porting commodity operating systems to Xen. It is

reported that a total of about 3,000 lines of Linux code, or 1.36%, had to be modified; for Windows XP

this figure is 4,620, or about 0.04% [41].