Microvisor: A Scalable Hypervisor Architecture for Microservers

Research · March 2016
DOI: 10.13140/RG.2.1.1496.5525

Michail Alvanos
The Cyprus Institute
29 PUBLICATIONS **64** CITATIONS
SEE PROFILE

Michail D. Flouris
OnApp Ltd., UK
37 PUBLICATIONS **341** CITATIONS
SEE PROFILE

Microvisor: A Scalable Hypervisor Architecture for Microservers

Xenia Ragiadakou, Michail Alvanos, Julian Chesterfield, John Thomson, Michail Flouris
OnApp Ltd, UK
{firstname.lastname}@onapp.com

1 Introduction

Virtualization of server hardware is a commonly used practice to provide scalable resource management. There are many benefits to virtualizing hardware resources, primarily the ability to efficiently share resources (CPU, memory, NICs) across a multi-tenant platform hosting many Virtual Machines (VMs). Each VM presents a closed environment for booting a standard OS kernel and running applications within that OS container. In addition to commonly deployed commercial hypervisors (HVs), there are two dominant open-source virtualization platforms: Kernel-based Virtual Machine (KVM) [1] and the Xen Hypervisor [2]. The platforms have mature support for established Intel x86 architecture chipsets and are both making fast advances towards mature support for ARM architecture hardware.

Traditional x86 server architectures, having reached frequency scaling limits, have improved performance through increasing the number of cache-coherent logical cores and available caches. To improve scalability in a power-efficient manner, recent server architectures implement non-uniform memory access (NUMA), where each CPU core is directly attached to a portion of memory, but is also able to address all system memory, albeit at different speeds. Due to the power and performance limitations, alternative architectures have emerged with non-cache-coherent memories between groups of processor cores [3] commonly referred to as Microservers. New OS system architectures are required to enable applications to execute across multiple, low-power, independent, non-cache-coherent islands (denoted as chiplets) with access to shared hardware I/O device resources. These architectures have better scaling properties that allow increasing numbers of CPU cores, while maintaining a high performance-to-power ratio, which is the key metric if hardware is to continue to scale to meet the expected demand of ExaScale computing and Cloud growth.

In contrast to the KVM Hypervisor platform, the Xen Hypervisor provides a true Type I Hypervisor. That is to say that the Hypervisor layer runs directly on the baremetal hardware, managing guest OS instances directly above it. There is no Host Operating system required and in this way the Type I architecture is considered to be a minimal, high performance shim. In parallel to the virtualized guest systems running on the Type I Hypervisor, traditional Xen systems utilize a control domain, known as Dom0, which has privileged access to the hypervisor and is responsible for various tasks including; administering guest virtual machines, managing resource allocation for guests, providing drivers for directly attached hardware, and offering network communication support. Guest Virtual Machines (DomUs) do not typically have direct access to real hardware, and thus all guest domain network and storage communication is managed through paravirtual device drivers hosted in Dom0 which in turn handles safe resource access multiplexing of the physical hardware.

In order to meet a variety of emerging technology trends, a novel, super-lightweight Type I Hypervisor architecture is essential. The proposed '*Microvisor*' architecture is significant for a number of reasons.

1. Resource utilization efficiency. Microservers are servers with limited resources and high performance-to-power ratios, which are built and managed as a cluster of cache-coherent CPU islands.

2. Performance and scalability. To meet the performance and scale demands of emerging Network Function Virtualization (NFV) platforms, a significantly re-architected paravirtual network IO path is required.

Network performance between virtualized functions executing on commodity hardware platforms today is limited primarily due to context switches between the VM and the control domain that handles the packet switching. In order, therefore to meet the growing demand of Network Function Virtualization on commodity hardware, using individual worker VMs to provide enhanced network packet filtering, middlebox functionality and packet forwarding, requires significant changes to the architecture of commodity virtualization platforms. It is thus key for performance that the packet forwarding layer is as fast as possible, with slow path network functions offloaded to separate processing engines (SDN Network Function Virtualization model).

We propose a new hypervisor design called *Microvisor*, which is under development. The platform is based on the

Open Source Xen Hypervisor but presents significant architectural changes. The core idea behind the *Microvisor* is to **remove any dependency on a local Dom0 domain** for inter-domain paravirtual device communication but support unmodified virtualized guest OS and kernels. Additional features include the support of VM setup, bootstrap and resource allocation without any control domain dependency. We move this generic functionality into the Xen Hypervisor layer directly whilst maintaining Hardware driver support through a super-lightweight driver domain interface. This results in an extremely efficient, minimal virtualization shim, that provides all the generic inter-domain virtualization functions.

1.1 Current Xen Architectural Design

The Control domain (Dom0) typically boots immediately after the Xen Hypervisor image. It provides hardware driver support for key shared IO resources (network and storage) and management tools to control VM lifecycle operations (start/stop/suspend/resume) for a guest domain as well as the ability to migrate the live state of a running guest to another physical hypervisor with only minor service interruption. The Control domain is also responsible for managing the Xenbus communication channel which is primarily used to signal virtual device information to a guest domain. It also manages the physical memory pool which is shared across VMs, and the virtual CPU to VM mapping by communicating a CPU scheduling algorithm to the HV.

For every active paravirtualized device of a guest VM, there is a corresponding driver in the control domain that allocates resources and handles the communication ring over which virtual I/O requests are transmitted. The control domain is then responsible for mapping those I/O requests onto the underlying physical hardware devices, such as Ethernet frames over a NIC, or block I/O requests to an iSCSI block device. The drivers, known as frontend (guest-side) and backend (Dom0-side), are implemented using ring buffers between Dom0 and the associated DomU PV device instance.

1.2 Design of the Microvisor HV architecture

The new hypervisor implementation boots as a standalone image without loading any control domain. Various major changes are required to complete the implementation of the proposed vision. The first step is the migration of the Xenbus functionality to the Xen Hypervisor layer by implementing a minimal set of Xenbus functionality inside the core Xen HV. Next, we implemented a simple key/value store (Xenstore) interface to communicate values to guest domains via Xenstore. Third, and most importantly the *netback* handling was integrated directly into a fast packet soft switching layer integrated directly in the Xen Hypervisor.

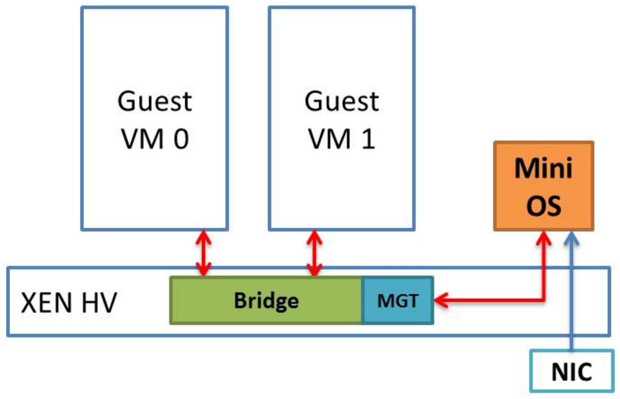

Figure 1. The new Hypervisor Architecture.

Remote management of a thin *Microvisor* is provided over a network interface. The long-term vision is that Microvisors will form a cluster of cooperative hypervisor shims that execute directly on baremetal hardware and can be remotely managed as simple commodity raw Ethernet appliances, or via custom hardware interfaces for more complex Microserver hardware. In the former case, secure access is physically enforced using Layer 2 Ethernet mechanisms to ensure that only trusted elements can communicate directly with the *Microvisor*, as shown in Figure 1.1.

A simple packet forwarding soft switch is embedded directly in the Xen Hypervisor layer which provides simple and fast packet forwarding functionality between frontend guest *netfront* instances using a zero-copy mechanism. Packet forwarding off the host is provided via a hardware driver domain, plugged directly into the logical soft switch.

VM Block storage is provided either by access to persistent direct attached storage drives, or via network-attached storage protocols (e.g. iSCSI, ATAoE). In both cases, a virtual soft block switch integrated into the Xen HV layer maps virtual IO requests to either physical block requests serviced by the hardware driver domain, or to Ethernet frames generated by the network block protocol logic.

References

[1] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The Linux Virtual Machine Monitor," in *Proceedings of the Linux Symposium*.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, and R. Neugebauer, "Xen and the art of virtualization," in *Proc. of the nineteenth ACM symposium on Operating systems principles (SOSP19)*.

[3] Y. Durand, P. M. Carpenter, S. Adami, A. Bilas, D. Dutoit, A. Farcy, G. Gaydadjiev, J. Goodacre, M. Katevenis, M. Marazakis, *et al.*, "Euroserver: Energy efficient node for european micro-servers," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*.