Virtualization as the Catalyst for Cloud Computing

Natarajan Meghanathan
Department of Computer Science, Jackson State University, USA

INTRODUCTION

In a conventional system architecture, even though the three major components (hardware, operating systems and application programs) are largely decoupled from a development point of view, they still work together only in the proper combinations. Executables of application software compiled for a particular ISA (Instruction Set Architecture) will not run on hardware that implements a different ISA (Smith & Nair, 2005). For example, Windows application binaries will not run directly on a Power PC (Mac) processor. If software is restricted to run on only certain nodes (operating systems/ISAs) in the network, then a great deal of flexibility and interoperability is lost; especially with the significant growth of the Internet and the heterogeneity of the nodes that are part of it (Chowdhury & Boutaba, 2010).

Virtualization is a technology that facilitates running more than one operating system side-by-side just on the same processing hardware. Virtualization allows processing that would have been achieved on multiple computers to run on just one powerful multi-core processor. As multi-core processors with 4, 8 and 16 cores on a chip are becoming more common, many processor cores are likely to be underutilized in conventional system architecture. Even with parallelization of applications, most applications will have only a finite amount of parallel tasks that can be executed at a given time, leaving many processor cores to remain idle. Virtualization lets to allocate groups of processor cores to individual operating systems.

Each instance of an operating system (OS) is called a virtual machine (VM). Each VM can run on its own operating system (called guest operating system), applications, etc. The objective of virtualization is to make each VM act like a standalone machine would and at the same time free developers and users from traditional interface and resource constraints thereby enhancing software interoperability, system impregnability and platform versatility. VMs encapsulate entire systems (hardware configuration, operating system, applications) in files and these files can be moved from one host to run on a different host as long as the latter has the supporting hypervisor that can load these files. This is similar to creating a file with slides using Microsoft Powerpoint on one host machine, then moving the file to another host machine and loading the slides on this host machine as long as it has Microsoft Powerpoint installed on it.

To virtualize a given computer, a piece of software called the Virtual Machine Monitor – VMM (also commonly referred to as a hypervisor) must be installed on the host hardware. The VMM enables multiple operating systems to run in parallel on the same hardware (Adams & Agesen, 2006). The VMM is responsible for emulating the hardware ISA so that the guest software can potentially execute a different ISA from the one implemented on the host (Sud et. al., 2012). In addition, the VMM is also responsible for providing virtualized hardware resources like I/O devices, hard disks, etc (Li et. al., 2013). The objective of this article is to provide an informative description of the different virtualization architectures and techniques to implement these architectures on x86 hardware (as it is the most commonly used hardware for computer systems); virtualization of memory address spaces and virtual address translation using shadow page table and nested page table. Most of the literature and software available for virtualization are also based on the x86 hardware.

BACKGROUND

There are two prominently used virtualization architectures: Hosted architecture and Bare-metal architecture. The two architectures differ in the amount of control the VMM has on the underlying hardware

DOI: 10.4018/978-1-4666-5888-2.ch105

and the host machine, which in turn has an impact on the performance of the applications running on the virtual machines. The Hosted architecture is suitable for virtualizing PC platforms which cannot be virtualized using the traditional mainframe approach of the Bare-metal architecture.

Hosted Architecture

With the Hosted architecture (see Figure 1), the VMM runs on the top of the host operating system and interfaces with the guest operating systems that run on the top of the VMM (Dowty & Sugerman, 2009). When a guest operating system performs a privileged instruction or operation that requires direct interaction with the shared hardware resources, the VMM intercepts the operation, checks it for correctness, and performs it on behalf of the guest by executing it using the ISA of the host operating system and the underlying hardware (Chen et. al., 2013). The common products that use the Hosted architecture are VMWare Workstation and Parallels Desktop for Mac. In the hosted virtualization architecture, the guest operating system running on the VM has access only to a limited subset of the I/O devices (Sugerman et. al., 2001). The ownership of the physical I/O devices connected to the computer is retained by the host operating system; whenever an application running on a VM attempts to access the underlying hardware of the host, the VMM provides an emulated view of the actual hardware to the VM (see Figure 2 as an example of how a VMM emulates the underlying hardware). However, the VMM can often emulate only generic I/O devices such as network interface cards and CD-ROM drives (Bourguiba et. al., 2012). Most of the time, the VMM does not have access or knowledge about the non-generic I/O devices such as PCI data acquisition cards and the VMM cannot present an emulated view of the non-generic I/O devices to the VMs. In addition to the I/O devices, many hosted virtualization architecture solutions provide pass through functionality support (see Figure 3) for the USB ports. With this feature, a user working on a VM can directly access the USB devices connected to the host. Pass through I/O minimizes CPU utilization.

The Hosted architecture is typically used during the software development process. For example, a hosted VM architecture could be used for testing alpha and beta software as each individual VM is isolated

Figure 1. Hosted virtualization architecture