ISSN:2320-0790

An International Journal of Advanced Computer Technology

Study of Virtual Machine and its application

Rohaan Chandra[1], Ricky Mudaliar[2], Dashrath Mane[3]

[1,2,3]Department of MCA
Vivekananda Educational Society's Institute Of Technology, Chembur
Mumbai, Maharashtra

rohaanagarwal@gmail.com[1], ricky.mudaliar69@gmail.com[2], dashumane@gmail.com[3]

Abstract: A virtual machine is software that's capable of executing programs as if it were a physical machine—it's a computer within a computer. A virtual machine (VM) is a software implemented abstraction of the underlying hardware, which is presented to the application layer of the system. Virtual machines may be based on specifications of a hypothetical computer or emulate the computer architecture and functions of a real world computer.

Keywords: **Virtual Machine; VM; Hip Hop; PHP; HHVM**

I. INTRODUCTION

A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Virtual machines are separated into two major classifications, based on their use and degree of correspondence to any real machine [1].

II. TECHNIQUES OF VIRTUALIZATION

A. Emulation of the underlying raw hardware (native execution)

This approach is described as full virtualization of the hardware, and can be implemented using a Type 1 or Type 2 hypervisor. (A Type 1 hypervisor runs directly on the hardware; a Type 2 hypervisor runs on another operating system, such as Linux). Each virtual machine can run any operating system supported by the underlying hardware. Users can thus run two or more different "guest" operating systems simultaneously, in separate "private" virtual computers [2].

The pioneer system using this concept was IBM's CP-40, the first (1967) version of IBM's CP/CMS

(1967–1972) and the precursor to IBM's VM family (1972–present). With the VM architecture, most users run a relatively simple interactive computing single-user operating system, CMS, as a "guest" on top of the VM control program (VM-CP). This approach kept the CMS design simple, as if it were running alone; the control program quietly provides multitasking and resource management services "behind the scenes"[4].

Full virtualization is particularly helpful in operating system development, when experimental new code can be run at the same time as older, more stable, versions, each in a separate virtual machine. The process can even be recursive: IBM debugged new versions of its virtual machine operating system, VM, in a virtual machine running under an older version of VM, and even used this technique to simulate new hardware [5].

B. Emulation of a non-native system

Virtual machines can also perform the role of an emulator, allowing software applications and operating systems written for computer processor architecture to be run. Some virtual machines emulate hardware that only exists as a detailed specification. This technique allows diverse computers to run any software written to that specification; only the virtual machine software itself must be written separately for each type of computer on which it runs.

C. Operating system-level virtualization

It is a server virtualization technology which virtualizes servers on an operating system (kernel) layer. It can be thought of as partitioning: a single physical server is sliced into multiple small partitions (otherwise called virtual environments (VE), virtual private servers (VPS), guests,

zones, etc.); each such partition looks and feels like a real server, from the point of view of its users.
For example, Solaris Zones supports multiple guest OSes running under the same OS (such as Solaris 10). All guest OSes have to use the same kernel level and cannot run as different OS versions. Solaris native Zones also requires that the host OS be a version of Solaris; other OSes from other manufacturers are not supported.

The operating system level architecture has low overhead that helps to maximize efficient use of server resources. The virtualization introduces only a negligible overhead and allows running hundreds of virtual private servers on a single physical server. In contrast, approaches such as full virtualization (like VMware) and paravirtualization (like Xen or UML) cannot achieve such level of density, due to overhead of running multiple kernels. From the other side, operating system-level virtualization does not allow running different operating systems (i.e. different kernels), although different libraries, distributions, etc. are possible [3].

III. TYPES OF VIRTUAL MACHINE

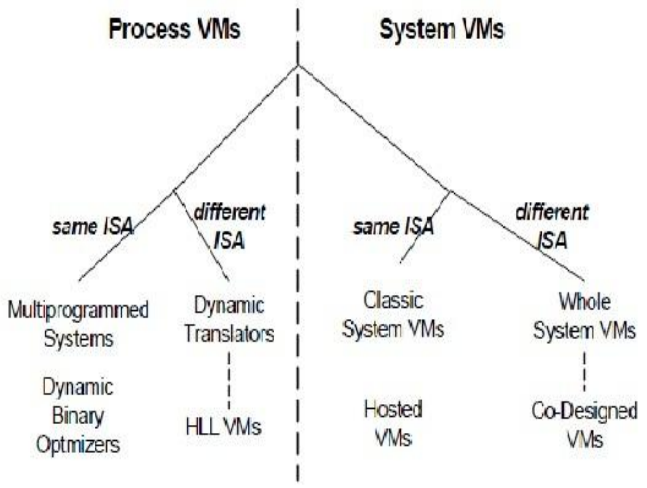
A. System virtual machine

It provides a complete system platform which supports the execution of a complete operating system. These usually emulate an existing architecture, and are built with the purpose of either providing a platform to run programs where the real hardware is not available for use (for example, executing software on otherwise obsolete platforms), or of having multiple instances of virtual machines lead to more efficient use of computing resources, both in terms of energy consumption and cost effectiveness (known as hardware virtualization, the key to a cloud computing environment), or both.

A system virtual machine consists entirely of software, but an operating system and the applications running on that OS see a CPU, memory, storage, a network interface card, and all the other components that would exist in a physical computer.

B. A process virtual machine

It is designed to run a single program, which means that it supports a single process. Such virtual machines are usually closely suited to one or more programming languages and built with the purpose of providing program portability and flexibility amongst other things. An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine it cannot break out of its virtual environment.
A process virtual machine is limited to running a single program. A system virtual machine, on the other hand, enables one computer to behave like two or more computers by sharing the host hardware's resources.



A taxonomy of virtual machines.

IV. APPLICATION OF VIRTUAL MACHINE BY FACEBOOK

HipHop for PHP (shortened as HipHop) describes a series of PHP execution engines and improvements created by Facebook. The original motivation of HipHop was to save resources on Facebook servers, given the large PHP codebase of facebook.com. As development of HipHop progressed, it was realized that HipHop can substantially increase the speed of PHP applications in general. Web page generation throughput by factors of up to 6 have been observed over Zend PHP. A stated goal of HipHop is to provide a high level of compatibility for Zend PHP, where most Zend-based PHP programs run unmodified on HipHop.

The current version of HipHop, known as HHVM (or the HipHop Virtual Machine) was open-sourced in late 2011. HipHop is currently around 1.9 million lines of mostly C++, C and PHP source code and is distributed as open source and free software on GitHub (under the terms of version 3.01 of the PHP License).

HipHop VM for PHP helps in reducing the CPU usage on Facebook Web servers on average by about fifty percent, depending on the page. Less CPU means fewer servers, which means less overhead. This project has had a tremendous impact on Facebook.

Prior to HHVM, Facebook development environments (they call them "sandboxes") used a custom-built PHP interpreter called HPHPi to shortcut the long and slow HPHPc compilation cycle and provide a rapid "edit, save, run" development workflow. HPHPi was flexible but slow (slower than the Zend engine that HPHPc replaced). Developers like

to move fast, and HPHPi started to strain under the load of more complex product features like Timeline and Ticker.
Development on the HipHop virtual machine (known as the HHVM) began in early 2010. HHVM builds on top of HPHPc, using the same runtime and extension function implementations. HHVM converts PHP code into a high-level bytecode (commonly known as an intermediate language). This bytecode is then translated into x64 machine code dynamically at runtime by a just-in-time (JIT) compiler. In these respects, HHVM has similarities to virtual machines for other languages including C#/CLR and Java/JVM.

HHVM brings with it many benefits over HPHPc. There is near full support for the entire 5.4 PHP language (including the create_function() and eval() functions). There is one execution engine for both production and development (i.e., no need to maintain HPHPi any longer). There is both production and development integration with HPHPd. And the push process becomes much simpler; no more lengthy binary build time that existed with HPHPc. Debugging with HPHPd is also supported.

As an adjunct to HPHPc, Facebook engineers also created a "developer mode" of HipHop (known as HPHPi) and the HipHop debugger (known as HPHPd). This allowed developers to run PHP code through the same logic provided by HPHPc while, at the same time, allowing them to interactively debug PHP code. Developers could set watches, breakpoints, etc. Of course, the code run through HPHPi was not as performant as the code run through HPHPc, but the developer benefits were, at the time, worth having to maintain these two execution engines for production and development. HPHPc, HPHPi and HPHPd were all open-sourced in 2010.

By many accounts, HPHPc was a huge success, especially within Facebook as it allowed facebook.com to run much faster, using less resources. However, in early 2013, Facebook deprecated HPHPc.There were many reasons for this. For all the performance gains that HPHPc provided, the curve for further performance improvements had flattened. HPHPc did not fully support the PHP language, including the create_function() and eval() constructs. HPHPc required a very different push process, requiring an over 1GB binary to be compiled and distributed to many machines in short order. HPHPc did not support HPHPd, and, given the amount of lines of code that made up facebook.com, HPHPi was becoming slow for development. Plus, maintaining HPHPc and HPHPi in parallel (as they needed to be for production and development consistency) was becoming cumbersome. Finally, it was not a drop in replacement for Zend as external customers would have to change their whole development and build process to use HPHPc. When HHVM(Hip Hop Virtual Machine) deployed to development team sandboxes in 2011, reduced page load times by over 3x compared to HPHPi, all while keeping the rapid workflow that HPHPi provided.

HHVM brings with it many benefits over HPHPc. There is near full support for the entire 5.4 PHP language (including the create_function() and eval() functions). There is one execution engine for both production and development (i.e., no need to maintain HPHPi any longer). There is both production and development integration with HPHPd. And the push process becomes much simpler; no more lengthy binary build time that existed with HPHPc. Debugging with HPHPd is also supported.

However, the key question is around performance. As a virtual machine, HHVM has the ability to use live type information to produce more efficient native code, leading to higher web server throughput and lower latency. In Q4 2012, the performance of facebook.com running on HHVM achieved parity with HPHPc. In Q1 2013, the production version of facebook.com started running on HHVM, replacing HPHPc.

V. CONCLUSION

A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface identical to the underlying bare hardware. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

VI. REFERENCES

[1] Virtual Machines: Versatile Platforms for Systems and Processes. Author Jim Smith, Ravi Nair.
[2] Virtual Machines. Author Iain D. Craig.
[3] http://en.wikipedia.org/wiki/Virtual_machine
[4] http://en.wikipedia.org/wiki/HipHop_for_PHP
[5] http://www.gitam.edu/eresource/comp/gvr(os)/3.4.htm