Virtualization: State of the Art

Version 1.0, April 3, 2008

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

SCOPE Alliance,
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Chairman

Or

For questions or feedback, use the web-based forms found under the Contacts tab on www.scope-alliance.org

1. PURPOSE

This purpose of this document is to describe the state-of-the-art of virtualization with a focus on system virtualization, rather than application virtualization. This document aims to classify the various virtualization approaches, along with their goals, advantages and drawbacks. Such a classification is expected to help identify virtualization technologies that might be applied within the network/telecommunications equipment space. In addition to describing the state-of-the-art, the document provides a glossary of terms and definitions related to virtualization.

This document is part of a set of documents being delivered by the SCOPE Alliance Virtualization Working Group and is expected to provide a common knowledge and understanding of virtualization. Given the goals of the SCOPE Alliance, this document focuses on properties and attributes of virtualization solutions that fit within the SCOPE Alliance architecture and, in particular, those useable in embedded and real-time systems.

This document serves as a basis for subsequent documents on virtualization use cases and requirements that the SCOPE Alliance Virtualization Working Group is delivering.

2. AUDIENCE

This document is intended for the following audiences:

- Virtualization solutions providers
- SCOPE Alliance members planning to use, or currently applying, virtualization technologies to network/telecommunications equipment
- Other specification bodies.

3. REFERENCES

1. K. Adams and O. Agesen, A Comparison of Software and Hardware Techniques for x86 Virtualization, http://www.vmware.com/pdf/asplos235_adams.pdf

2. DMTF, DSP 1042, System Virtualization Profile, http://www.dmtf.org/standards/published_documents/DSP1042.pdf

3. DMTF, DSP 1057, Virtual System Profile, http://www.dmtf.org/standards/published_documents/DSP1057.pdf

4. DMTF, DSP 1059, Generic Device Resource Virtualization Profile, http://www.dmtf.org/standards/published_documents/DSP1059.pdf

5. R. Goldberg, Survey of Virtual Machine Research, Computer, 1974.

6. Linux Foundation, Mobile Platform Virtualization, http://www.linux-foundation.org/en/Mobile_Platform_Virtualization_Guidelines

7. Linux Foundation, Virtualization White Paper (not yet a public document).

8. Open Virtual Machine Format, Whitepaper on OVF Specifications, v. 0.9, http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf , http://www.citrixxenserver.com/Documents/OVF-whitepaper-0.9-open_authors_lettersize.pdf

9. Open Virtual Machine Format Specification, v. 0.9, http://www.vmware.com/pdf/ovf_spec_draft.pdf , http://www.citrixxenserver.com/Documents/OVF-spec-0.9-open_authors_lettersize_noDMTF.pdf

10. G. Popek and R. Goldberg, Formal Requirements for Virtualizable Third Generation Architectures, Communications of the ACM, vol. 17, no. 7, 1974.

11. J. E. Smith and R. Nair, Virtual Machines, Versatile Platforms for Systems and Processes, Morgan Kaufman, 2005.

12. Standard Performance Evaluation Corporation (SPEC) Virtualization Committee, http://www.spec.org/specvirtualization/

13. Wikipedia, Virtualization, http://en.wikipedia.org/wiki/Virtualization

4. INTRODUCTION

This document describes the current landscape of virtualization and provides a glossary of acronyms, terms and definitions, so that further documents can use and refer to a common terminology. By describing the current state-of-the-art, the document aims to enable the identification of gaps that should be filled to satisfy the goals of the SCOPE Alliance.

5. ACRONYMS

ABI	Application Binary Interface
CPU	Central Processing Unit
EDA	Electronic Design Automation
FRU	Field Replaceable Unit
GPOS	General Purpose Operating System
IDE	Integrated Drive Electronics
ISA	Instruction Set Architecture
JVM	Java Virtual Machine

MMU	Memory Mapping Unit
NAS	Network Attached Storage
NEP	Network Equipment Provider
NUMA	Non-Uniform Memory Architecture
OS	Operating System
QoS	Quality of Service
RAS	Reliability Availability Serviceability
RT	Real Time
RTOS	Real-Time Operating System
SAN	Storage Area Network
SATA	Serial ATA (Advanced Technology Attachment)
TEM	Telecommunications Equipment Manufacturer
USB	Universal Serial Bus
VF	Virtualization Facilities
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network

6. GLOSSARY

- **Administrative Domain:** A group of elements that are administered by the same user or group of users.
- **Application Binary Interface (ABI):** A low-level interface between an application program and the operating system, between an application and its libraries, or between component parts of the application. An ABI differs from an application programming interface (API) in that an API defines the interface between source code and libraries, so that the same source code will compile on any system supporting that API, whereas an ABI allows compiled object code to function without any changes to a system using a compatible ABI.
- **Application Compatible VMs:** The term application compatible refers to the ability to move applications from one physical compute environment to another and to run them unmodified (i.e., without the need for recompilation) on compatible hardware in a non-virtualized environment. The term application compatible VMs captures the same ability in a virtualized environment (i.e., the application can be moved from one VM to another VM and can be run without change). Here, an application is the entire software stack constituting the VM workload.
- **Application Virtualization:** A kind of virtualization that isolates the application from the underlying OS to increase portability and ease management.
- **Bare Metal Computer System (Physical Compute Element):** A physical computer system with no installed control software.
- **Boot Device:** A physical or virtual entity that is used to bootstrap the software. Such a physical entity might be a flash partition, hard disk, Ethernet interface, etc.

- **Chassis:** A physical enclosure for containing system elements.
- **Cluster:** A computer system composed of two or more compute elements that operate together as a functional whole, to provide higher levels of performance, resource utilization, and/or RAS characteristics than those provided by the individual component compute elements. A cluster requires control software (cluster controller) that functionally complements the operating systems installed on its compute elements and that implements cluster-wide resource management functionality and policies.
- **Compute Element (Compute Resource):** A physical or a virtual compute device capable of booting and executing software such as an operating system image, a standalone application, or a hypervisor. Examples of compute elements are standalone physical computer systems and physical or logical domains (see also Bare Metal Computer System).
- **Domain:** In the Xen environment, a domain is one of the VMs in the system. Domain 0 is the first domain started by the hypervisor on system boot. It is a privileged domain in that it can directly access the hardware, unlike the other (guest) domains associated with the virtual devices.
- **Element (Resource):** A distinguishable component of a system, including the system itself. Examples of elements include hardware components (e.g., processors, disk drives, etc.) and software components (e.g., operating systems, applications, etc.).
- **Field Replaceable Unit (FRU):** A hardware element supported, maintained and/or upgraded at the customer's location.
- **Full Virtualization (Native Virtualization):** A virtualization technique used to implement a certain kind of VM environment, namely, one that provides complete simulation of the underlying hardware. The result is a system in which all software (including OSes) capable of execution on the raw hardware can be run in the virtual machine.
- **Guest OS:** An operating system running on top of a hypervisor (virtual machine monitor) within a virtual machine. Different virtual machines may run different guest OSes simultaneously on top of the same hypervisor.
- **Hardware Emulation:** A hardware emulator is usually application software reproducing the behavior of a complete hardware platform (processor and devices) that allows an unmodified guest OS and applications for a different processor to be run. This approach has been used for a long time to enable the creation of software for new processors before they become physically available.
- **Hardware Virtualization:** A specific form of virtualization that uses the same ISA.
- **Host OS:** When virtualization is provided by a Type 2 hypervisor (defined below), an operating system must be started first on the physical platform. Such an OS is named a host OS.

- **Hypervisor (Virtual Machine Monitor):** A hypervisor is (usually small) runtime executive software in charge of logically replicating the physical platform. Within each logical replica virtual machine, a guest OS may run independently of other guest OSes running in other virtual machines. Such replication is achieved by partitioning and/or virtualizing platform resources. A distinction is made between two types of hypervisors:
    - Type 1 hypervisors (bare-metal hypervisors) run directly on the physical hardware platform.
    - Type 2 hypervisors require an operating system to run first on the physical hardware platform. See also host OS.
- **Instruction Set Architecture (ISA):** The part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handlers, and external I/O. An ISA includes a specification of the set of op-codes (machine language), and native commands, implemented by a particular CPU design.
- **Isolation (Complete and Partial):** Complete isolation is the realization of VMs with no interaction or dependency among the VMs. Thus, if one VM fails, it does not impact the other VMs. Partial isolation is sometimes employed for performance reasons, where one of the VMs (referred to as the primary VM) has direct access to the hardware. In partial isolation, the primary VM might need to act as the back-end driver for the VMs that use the same physical device (to which the primary VM has direct access). Note that, with partial isolation, if the primary VM fails, the other VMs cannot access the hardware it was managing.
- **Live Migration:** The movement of a VM from one hardware machine to another during normal operation. This movement is typically achieved by using shared storage and a common network and by replicating memory from one machine to another, followed by a brief quiescent transition period. This period should be minimized to reduce the observability of the migration.
- **Management Controller (Management Access Point):** A computer system (e.g., service processor, system controller) specialized to provide system management capabilities.
- **Multi-Tier Application:** An application composed of two more components in different tiers communicating over a network. The application is architected to run across multiple domains or VMs. Each tier typically employs its own set of dedicated physical or virtual servers.
- **Multi-Tenant Hosting**: Hosting where a service provider provides the same set of services to multiple customers on the same infrastructure via logical partitioning. For example, a “wholesale” service provider might build an infrastructure of servers and routers, and host multiple “retail” service providers, isolating the retail tenants from one another. In such a system, a tenant is unaware of the existence or operation of the other tenants, except perhaps through performance.

- **OS Virtualization:** Permits the multiplexing of a single running OS instance so that it is logically replicated and seen as distinct guest OS environments (sometimes named containers, virtual servers or zones). The guest OS environments share the same OS as the host system (i.e., the same OS kernel is used to implement the guest OS environments). Applications running in a given guest OS environment view it as a stand-alone system, and may be managed independently of the others.
- **Para-Virtualization:** When para-virtualization is used, the logically replicated hardware differs slightly from the underlying physical hardware. Instead of fully supporting the physical hardware ISA, para-virtualization offers a special API that can be used only by modifying the guest OS. This special API is implemented by system calls to the hypervisor (referred to as hypercalls).
- **Partition:** See Virtual Machine.
- **Physical and Virtual Devices:** In a virtualized environment, a physical device may be dedicated to a VM or shared with multiple VMs. In the dedicated case, there is a one-to-one mapping between the device seen by the VM and the actual physical device. In the shared case, the physical device is virtualized so that each VM's access to the virtual device is channeled to the physical device in a managed fashion. Each VM assumes that it is accessing its own physical device without any knowledge of the virtualization. A virtual device may be realized by Virtualization Facilities (VF) defined below or, alternately, by one of the VMs.
- **Real-Time VM and Non-Real-Time VM:** A real-time VM is one in which commitments or contracts are made, ensuring a certain performance (throughput, latency, etc.) or other QoS requirements. A non-real-time VM is one created without a specific contract or guarantee.
- **Scheduling Attributes (per VM):** Attributes, typically set statically, that are used by the VF to determine the VMs and their position in the scheduling queue. For example, a VM that has certain QoS requirements might have its attributes set so that it is treated as a high-priority VM for priority-based scheduling.
- **Scheduling Policy:** A scheme used to schedule the individual VMs managed by a VF. For example, priority-based scheduling might be used in situations where some VMs need the VF to provide them a higher share of the CPU.
- **Simplex (Non-HA Configuration):** A configuration in which there is no concept of a backup for any node or service.
- **Software Element (Software Resource):** A piece of software capable of running on a compute element. Examples of software elements are OSes, firmware, patches and application packages/images.
- **System Virtualization:** A kind of virtualization that enables several OSes or executives, known as guests, to run over a software virtualization layer. System virtualization can involve the same ISA (hardware virtualization) or a different ISA (hardware emulation). System virtualization has the goal of logically replicating physical resources. Each logical replica is named a virtual machine, a partition, or

a domain. The virtualization layer can be implemented in different ways. It might run on the bare hardware (type 1 hypervisor), or it might require a host OS (type 2 hypervisor).

- **Virtual Machine (VM):** Software emulation of a physical computing environment. A virtual machine may be used to run an OS, or stand-alone applications when provided by a hypervisor. A virtual machine may also be used to run applications on top of an OS, as a JVM does.
- **Virtual Machine Monitor (VMM):** See hypervisor.
- **Virtualization:** The process of presenting computing resources in ways that users and applications can easily get value from them, rather than in ways dictated by their implementation, geographic location, or physical packaging. Virtualization provides a logical rather than a physical view of data, computing power, storage capacity, and other resources. Two ways in which resources can be virtualized are aggregation and replication. Aggregation presents multiple physical resources as a single virtual resource. Replication (also referred to as partitioning) multiplexes multiple virtual resources on a single physical resource. Aside from storage virtualization and network virtualization, the main kinds of virtualization are system virtualization and application virtualization.
- **Virtualization Facilities (VF):** The entire set of software that provides virtualization services to virtual machines on a physical compute element. This term carries fewer implementation connotations than the term Virtual Machine Monitor.
- **VM Boot:** The sequence of steps involved in the instantiation of a VM until the workload inside the VM starts executing its first instruction.
- **VM Payload (VM Workload):** The software running in a VM. A VM payload may involve an OS and applications, or it may be real-time code without any OS, etc.

7. VIRTUALIZATION TAXONOMY

This section presents a virtualization taxonomy with examples of contexts in which virtualization is used. It distinguishes between partitioning and aggregation and focuses, in particular, on partitioning, where it distinguishes between system VMs and process VMs. It also discusses other configurations, including hardware support for virtualization.

7.1 Partitioning vs. Aggregation

Virtualization can be characterized in terms of:

- Partitioning: Makes a (usually) physical resource appear as multiple different virtual resources
- Aggregation: Makes multiple (physical) resources appear as a single resource.

Note: SAN and NAS use both partitioning and aggregation. Network virtualization is bound to partitioning except when interface bonding is used.

7.1.1 Partitioning

Partitioning involves ways to:

- Partition a cluster of machines into sub-clusters, typically on the basis of physical boundaries, with a machine belonging to a sub-cluster
- Partition a machine into multiple subsets on physical boundaries (boards or processors), usually applied to NUMA architectures
- Partition a (set of) processor(s) and related resources (memory, I/O devices) into multiple VMs, each supporting a different OS
- Partition an OS into different containers
- Partition the resources managed by an OS between different applications
- Partition the resources of a JVM between different Java applications.

7.2 Virtualization Taxonomy and Examples

Virtualization is used in many different contexts. Examples include:

- EDA environments that provide virtualization to emulate chips being designed (usually various kinds of hardware emulators)
- Data center storage virtualized through logical volumes, NAS or SAN capabilities
- VPNs in communications applications
- Java execution environments based on JVMs
- Virtual machine products for desktop and data centers (e.g., Parallels, Svista, VMware, Xen, Virtual Iron, Microsoft Application Virtualization and Hyper-V)
- Virtualization solutions for the embedded and real-time systems (e.g., VirtualLogix, L4)
- Solutions to virtualize OSes (e.g., OpenVZ, Linux-Vserver, Solaris Containers)
- Virtualized and streamed applications (e.g., Microsoft Application Virtualization).

As a result of these many kinds and uses of virtualization, there is a need to sort out the landscape of virtualization for the SCOPE Alliance. A taxonomy derived from [13] is shown in the diagram below.

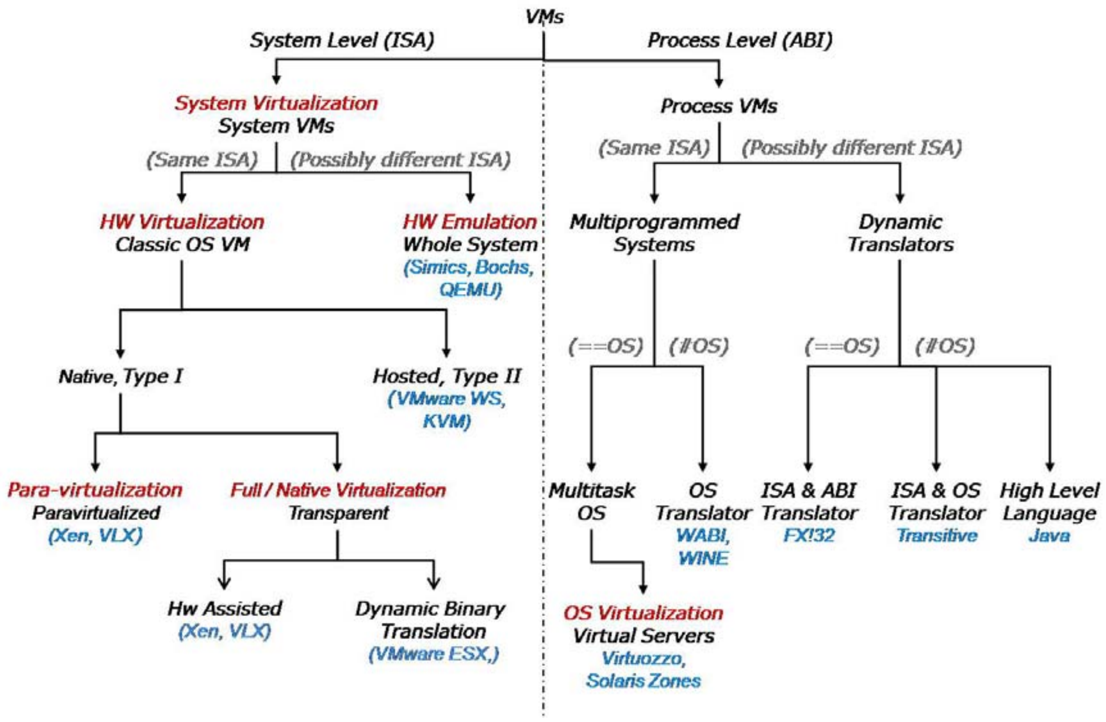

Figure 1. Virtualization taxonomy.

The above diagram covers only the partitioning side of virtualization. Partitioning is based on software that provides either an ISA, or an ISA and an ABI. When offering an ISA, partitioning supports system VMs. When offering an ABI and ISA, partitioning supports process VMs. The left side of the diagram corresponds to system VMs, and the right side corresponds to process VMs. System VMs are used to run full OSes with their own applications or standalone applications that require no OS, whereas process VMs support only applications.

The following list provides a brief description of each of the examples from left to right in the above diagram.

- Xen: Free Open Source hypervisor for x86, Itanium and some PPC platforms. The hypervisor runs on bare metal and supports guest OSes in VMs named domains. Domain 0 is used for physical I/Os. It relies on para-virtualization or on hardware-assisted virtualization when available. (http://www.cl.cam.ac.uk/research/srg/netos/xen/)

- VLX: A hypervisor for embedded and real-time systems developed by VirtualLogix that runs on bare metal and supports real-time OSes. It relies on para-virtualization or on hardware-assisted virtualization when available. (http://www.virtuallogix.com)

- VMware ESX: A Virtual Machine Monitor for x86 and Itanium platforms running on bare metal. It assumes that there is a service console system for the management of the platform. It supports binary guest OSes and relies on dynamic binary translation or on hardware-assisted virtualization for 64 bit platforms when available. (http://www.vmware.com)

- VMware WS: Hosted virtualization solution for x86 and Itanium platforms. It requires a Linux or Windows host operating system to be started prior to launching the virtual machines. It relies on dynamic binary translation. (http://www.vmware.com)

- KVM (Kernel Virtual Machine): A recent extension of the Linux kernel for platforms providing hardware-assisted virtualization that allows the user to create virtual machines running unmodified guest OSes. It requires a Linux host to boot and relies on a subset of QEMU for I/O accesses made by guest OSes. (http://kvm.qumranet.com/kvmwiki)

- Simics: Provides simulators for different processors and platforms allowing execution of software without access to a real platform in a so-called virtual environment. (http://www.virtutech.com/)

- Bochs: An open source IA-32 (x86) PC emulator that runs on most popular platforms. It includes emulation of the Intel x86 CPU, common I/O devices, and a custom BIOS. (http://bochs.sourceforge.net/)

- QEMU: A generic and open source machine emulator and virtualizer. As an emulator, it can support OSes and applications built for one machine (e.g., ARM) on a different machine (e.g., x86). It uses dynamic binary translation and supports binary guest OSes. When both the emulated machine and the host machine use x86 processors, QEMU enables running the guest code directly on the x86 processor. (http://www.qemu.org)

- Virtuozzo, OpenVZ: OpenVZ is a free open source version of the Linux kernel enabling the creation of many independent virtual environments that can be managed independently as if they were separate full-blown Linux instances, although they actually share the same Linux kernel code. Virtuozzo also has a Windows-based solution. (http://openvz.org/) (http://www.virtuozzo.com/)

- Solaris Containers: A recent extension to Sun's Solaris OS that uses an approach similar to that of OpenVZ. The features and services offered by Solaris containers are greater than those provided by OpenVZ. (http://www.sun.com)

- WABI: An ABI conversion that allows applications developed for Windows (x86) to run on Solaris/Linux (x86) without recompilation. With the same underlying ISA, WABI converts calls to Windows into calls to Solaris/Linux. (http://en.wikipedia.org/wiki/Wabi_%28software%29)

- WINE: Free GPL software that delivers Windows emulation on Linux systems for x86 platforms and that enables Windows applications to be run on Linux systems. (http://www.winehq.org/)
- FX32: A DEC translator that allows unmodified Windows x86 applications to run on Windows on DEC alpha platforms. (http://www.usenix.org/publications/library/proceedings/usenix-nt97/full_papers/chernoff/chernoff.pdf)
- Transitive: Allows unmodified applications generated for OS1 on platform P1 to be run on OS2 on platform P2. Not all combinations are supported; the most well-known product emulates Windows/x86 on MAC OS/ppc. (www.transitive.com).

7.2.1 System VMs

For system VMs, one can distinguish classic OS VMs that partition or replicate the underlying hardware from virtual environments (whole system, emulators, simulators) that provide an ISA typically different from, but possibly identical to, the underlying physical hardware.

Within classic OS VMs, a distinction can be made between:

- Solutions that run natively on the bare metal (Native VMs or Type I VMs)
- Solutions that require a host OS to run first (Hosted VMs or Type II VMs).

Depending on the hardware properties and on implementation choices, Native VMs or Type I VMs might require modifications to the guest OSes before running them (para-virtualization) or not (transparent virtualization).

Transparent virtualization can be provided in two different ways, either through dynamic binary translation (as done by VMware ESX) or through hardware-assisted virtualization (as used by XEN or VLX on Intel VT or AMD SVM processors).

Hosted VMs or Type II VMs use either dynamic binary translation or hardware-assisted virtualization.

7.2.1.1 States of Virtual Machines

The DMTF Virtual System Profile [3, § 6.4, p. 17] defines various states for Virtual Systems. In addition to the usual Initial and Final States, it describes the following states:

- Defined: VM is defined from a Virtualization Facilities standpoint, but is not instantiated; hence, it cannot perform any task, and does not consume any resources (except persistent resources such as disk allocations).
- Active: VM is instantiated; it use resources and can perform tasks.

- Paused: Identical to the Active state (VM is instantiated and resources are allocated) but no task can be performed by the VM.
- Suspended: The state of the virtual system and its virtual resources are stored on non-volatile storage. The system and its resources are not enabled to perform tasks. Whether or not resources are freed is implementation-dependent.

In addition to the above states, there might also be vendor-defined states.

The DMTF document also defines transitions between states, including:

- Define
- Activate
- De-activate
- Pause
- Suspend
- Shutdown
- Reboot
- Reset.

The transition diagram can be found in the DTMF document [3].

7.2.2 Process VMs

Process VMs are distinguished by whether the underlying hardware supports the ISA for which the application was generated (multi-programmed systems) or not (dynamic translators).

Multi-programmed systems are further classified depending on whether or not the ABI provided by the underlying system is the same as the ABI used by the application. If the ABIs are the same, the system is an OS, which might in turn be multiplexed (Virtual Servers). If the ABIs are different, an OS translator is employed to run an application using an ABI X on a system providing a different ABI Y.

Dynamic translators are outside the scope of this discussion.

7.3 Other Configurations

The above view is necessarily simplified. For example, as stated above, there are configurations of QEMU that run only applications and, thus, act as process VMs. Also, some native transparent and hosted VM solutions based on hardware assistance make use of a subset of QEMU for I/Os. Other configurations might employ virtualization-aware device drivers (often called para-virtualized device drivers), rather than emulated I/Os of native device drivers.

Classic OS VMs rely on one of the following mechanisms:

- Hardware-assisted virtualization
- Para-virtualization
- Dynamic binary translation.

A hypervisor must run in the most privileged hardware mode of the processor and cannot let a guest OS manage the hardware, as it normally does when running natively and alone on the hardware. In particular:

- Guest OSes cannot mask hardware interrupts but, instead, must mask virtualized interrupts.
- Guest OSes cannot set the MMU at will but must set a virtualized MMU.

Consequently, instructions that deal with the physical resources (or at least their behavior) must be changed or affected when the OS is running in a VM. Usually the guest OS is run at a lower-privilege mode to facilitate interception of physical device interactions.

Hardware-assisted virtualization implies that sensitive instructions when issued from a lower-privilege mode will generate an exception that will be caught by the hypervisor.

Para-virtualization replaces sensitive instructions by calls to the hypervisor (hypercalls) at system build time.

Dynamic binary translation requires the hypervisor to inspect the code of the guest OS and convert it appropriately if a sensitive instruction is found. Typically, a caching mechanism is used to avoid translating the same part of the guest OS code each time it is executed, and to avoid translating the entire guest OS.

These mechanisms deliver different properties from a system build and testing point of view and also from a timing and real-time standpoint.

7.3.1 Virtualization Support by Hardware

Support of virtualization by processors differs from one processor to another. In the past, most of the processors had no hardware support for virtualization. This situation is evolving quite rapidly with continual improvements in not only the basic operations of the processor but also MMU and I/O support.

Historically, X86-based processors had no support for virtualization. Introduction of the Intel VT and AMD-V has changed the landscape, permitting easier implementation of native virtualization. The Itanium processor now also has support for virtualization. To get a better understanding about performance impacts, see [1].

Power processors had no support for virtualization. The Power 4 architecture has introduced support for transparent virtualization, while the Power 5 architecture has introduced support for para-virtualization.

The SPARC architecture has also recently evolved from no support to support for para-virtualization.

At the time of writing of this document, the members of the SCOPE Alliance Virtualization Working Group know of no other processors that assist virtualization.

7.4 On-Going Work

Apart from the SCOPE Alliance, the following groups are also involved in virtualization activities.

- The Linux Foundation (formerly OSDL) has published guidelines for mobile platform virtualization [6].
- The DMTF has a working group on this topic, and has started delivering specifications [2, 3, 4]. They are also working on an Open Virtual Machine Format (OVF) specification [8, 9].
- The SPEC Virtualization Committee has started defining benchmarks related to virtualization [12].

8. VIRTUALIZATION EXAMPLES AND TOPICS OF INTEREST TO THE SCOPE ALLIANCE

This section provides some examples of virtualization aggregation and partitioning in the IT data center and in embedded/real-time systems. It also discusses the characteristics of virtualization solutions for embedded and real-time systems, which are the most relevant to the members of the SCOPE Alliance.

8.1 For Data Centers

This section provides some examples of aggregation and partitioning solutions for IT data center scenarios.

8.1.1 Aggregation Solutions

- Altiris
- Citrix
- Microsoft
- HP
- IBM
- Sun.

8.1.2 Partitioning Solutions

- VMware
- Xen
- Virtual Iron

- Svista
- Parallels
- Virtuozzo
- Sun
- HP
- IBM.

8.2 For Embedded and Real-Time Systems

Virtualization solutions for the embedded/real-time space are relatively new compared to those for the IT data center. Consequently, this section addresses characteristics/requirements for such virtualization solutions, rather than specific examples.

8.2.1 Existing Aggregation Solutions

At the time of writing of this document, there are no commercially available aggregation solutions known to the SCOPE Alliance Virtualization Working Group.

8.2.2 Characteristics of Partitioning Solutions

Real-time and embedded systems exhibit certain characteristics and require specific virtualization capabilities and features, as described below. A particular implementation might not satisfy all of these properties.

- **Ability to Run Real-Time Guest OSes:** It should be possible to run real-time OSes and their applications, whether commercial or home-made legacy OSes. The native properties and determinism of the RTOS scheduler must be preserved when running as a guest OS.
- **Schedulers:** Hypervisors need to support scheduling policies that enable a real-time constrained VM to run without problems. Latency, especially in OS switching operations, must be bounded and as low as possible. The virtualization layer must be deterministic. In addition, advanced scheduling policies that enable the scheduling of threads of different OSes is desirable.
- **Ability to Mix and Match OSes:** A GPOS (such as Linux or Windows) is usually run aside a RTOS in these virtualization scenarios.
- **Hardware Support:** Many embedded systems use hardware (processors, boards and devices) that are quite different from the usual platforms used for IT data centers. Hence, virtualization should be supported on a variety of processors such as ARM, MIPS, DSP and PPC. Homogeneous and heterogeneous multi-core processors are also relevant to embedded and real-time systems.
- **Devices:** Depending on the end-system, the set of devices and their use can differ substantially. Some devices might be dedicated to the exclusive use of a single guest OS, while others might be used (shared) by more than one guest OS.

Devices might be shared between different kinds of guest OSes, some being RTOSes and others being GPOSes. The typical devices used in embedded systems include audio devices, frame buffers, keyboards, keypads, power management, serial lines, modems, USB, flash disks, disks, IDE, SATA, Ethernet and more. Guest OSes might use these devices in different ways:

  - **Dedicated Devices:** Some devices might be dedicated to the exclusive use of a given partition. In such a case, the guest OS will be the only one to access such a device.
  - **Shared Devices:** Some devices might be accessed by more than one guest OS at the same time. For example, a single network interface or a single disk drive might be accessed concurrently by different guest OSes. The virtualization mechanisms should support such sharing. When sharing is used, access of a guest OS to a device might be subject to controls and limits to guarantee a particular QoS.
  - **Virtual Devices:** For communication between guest OSes, or rather between applications running on different guest OSes, it might be necessary to create and manage virtual devices such as virtual Ethernet devices or virtual modems. Such virtual devices do not have any physical counterpart but are seen as devices by the guest OSes.

- **Native Device Drivers:** Due to the complexity of writing and debugging device drivers, re-use of native device drivers must be straightforward in order to reduce risks, shorten the time to market, and minimize lengthy revalidation steps.

- **Memory Management and Isolation:** Some of the processors on which virtualization must be supported might not have hardware MMU support. Hence, complete memory isolation between guest OSes might not always be provided. Complete memory isolation necessarily has a run-time cost when MMU support is involved. It is desirable to leave the trade-off choices to the system builder so that he/she can tune the system according to actual needs.

- **Security:** Virtualization is often considered in conjunction with security. External threats against the system do not change due to the use of virtualization. However, because more than one guest OS runs on the platform, it might be necessary to protect each guest OS from other misbehaving, malicious or infected guest OSes. (This consideration also relates to the memory management and isolation property discussed above.) By allowing multiple guest OSes to run on a single platform, virtualization permits the splitting of a given workload into pieces, minimizing the risks. Finally, some partitions might be used for specific security purposes such as filtering network packets, logging data, controlling access permissions to specific resources, securing access to devices, etc.

- **Memory Footprint:** Embedded systems can have very stringent memory constraints. Often, embedded systems do not have swap capabilities. The virtualization layer must accommodate these limitations via a small memory footprint.

Physical memory is typically statically partitioned between the various VMs and must not be overcommitted.

- **Inter-VM Communication:** In server configurations, guest OSes are typically loosely coupled. In contrast, in embedded systems, guest OSes and their applications must cooperate to deliver their services and, consequently, there are specific needs for inter-VM communication. Depending on the system being built, such communication might be based on a virtual Ethernet or a virtual serial line, or whatever best fits the needs of the applications. As a result, inter-VM communication must be selectable and not imposed by the virtualization layer.
- **Static Configuration:** Embedded systems might not be able to rely on a Domain 0 or service console to start and manage the various guest OSes, memory footprint being one reason. Moreover, embedded systems are often headless; hence, configuration of the guest OSes is often best described at system build or configuration time, rather than at run time.
- **Management:** Virtualization impacts the way a platform is managed. Because several guest OSes share the same hardware, there is an additional configuration level to take into account: an OS is not necessarily granted access to all of the underlying hardware at the site where it runs, but only to a given subset. New possibilities are offered by virtualization: an OS may be updated without rebooting the physical machine but simply by stopping and restarting its virtual machine. Introduction of a GPOS aside a RTOS allows augmentation of the features of the platform with regular management facilities from the GPOS.

9. CONCLUSION

This document has presented the state-of-the-art of virtualization as a foundation for further work of the SCOPE Alliance Virtualization Working Group. It has also provided a glossary of virtualization terminology that will be used in further work of the SCOPE Alliance Virtualization Working Group.