Advanced virtualization capabilities of POWER5 systems

W. J. Armstrong
R. L. Arndt
D. C. Boutcher
R. G. Kovacs
D. Larson
K. A. Lucke
N. Nayar
R. C. Swanberg

IBM POWER5™ systems combine enhancements in the IBM PowerPC™ processor architecture with greatly enhanced firmware to significantly increase the virtualization capabilities of IBM POWER™ servers. The POWER hypervisor, the basis of the IBM Virtualization Engine™ technologies on POWER5 systems, delivers leading-edge mainframe virtualization technologies to the UNIX® marketplace. In addition to being able to create computing-intensive partitions with dedicated resources (processors, memory, and I/O adapters), customers can harness idle processor capacity to configure micropartitions with virtualized resources in order to consolidate many AIX™, i5/OS™, and Linux® servers onto a single platform. The POWER hypervisor provides support for virtualized processors, an IEEE virtual local area network (VLAN)-compatible virtual Ethernet switch, virtual small computer system interface (VSCSI) adapters, and virtual consoles. Many of these features are dependent upon, or take advantage of, the new facilities provided in the POWER5 processor, including the hypervisor decrementer, a fast page mover, and simultaneous multithreading support. The technology behind the virtualization capabilities that are available on the POWER5 servers, enabling customers to better utilize the industry-leading computing capacity of the POWER5 processor, is discussed in this paper.

Introduction

IBM zSeries* servers pioneered the logical partition (LPAR). Its PR/SM* and z/VM* hypervisors[1] retain leadership in transparent server virtualization technology. The zSeries hypervisors use sophisticated processor architecture extensions to completely and efficiently virtualize the hardware to each logical partition so that an operating system that runs natively on the hardware can also run in a logical partition without any required changes. The zSeries also introduced the concept of optional hypervisor calls that enable a hypervisor-aware version of an operating system (OS) to improve the utilization of the system resources by interacting directly with the hypervisor. An *hcall* instruction is a special program-context-switching instruction, similar to a system call, which gives control to the hypervisor. As is standard documentation practice with a system call, a function invocation made with the *hcall* instruction is generically termed an *hcall* in this paper.

Most virtualization products for current Intel platforms use a trap-and-emulate approach for privileged instructions to provide full virtualization of the processor and I/O so that no changes are required in the OS in order to run in a partition. The POWER implementation takes the approach that is sometimes referred to in the literature as *paravirtualization* [1, 2]. Paravirtualization requires a hypervisor-aware version of the operating system that must utilize *hcalls* in order to run in a logical partition. Typically, these hypervisor calls are confined to a relatively small number of the lowest-level routines in

[1]The hypervisor is a platform firmware component that controls the allocation and isolation of platform resources among the various logical partitions of the platform.

0018-8646/05/$5.00 © 2005 IBM

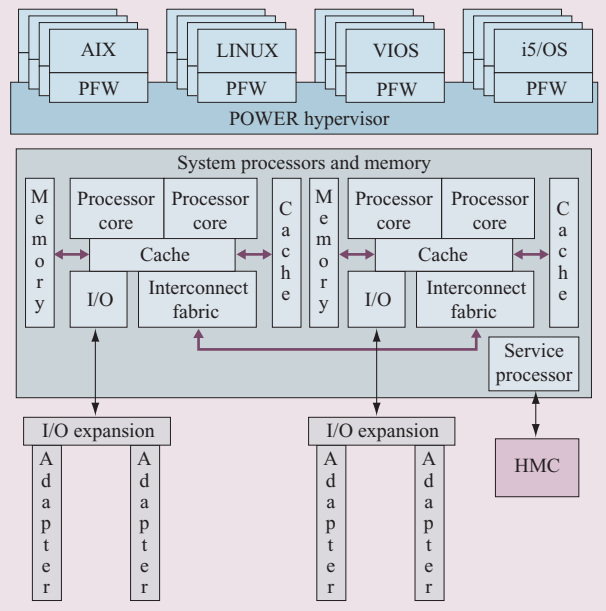

Figure 1

POWER5 platform structure.

the OS hardware adaptation layer. Instead of trapping, verifying, and emulating a number of privileged instructions required to perform a logical operation, such as updating the partition's virtual address translation table, the hypervisor in iSeries* and pSeries* platforms provides an *hcall* that performs the entire logical operation of updating the virtual address translation table. This approach is typically much more efficient when compared with the trap-and-emulate method because of the reduction it provides in context-switching and parameter-checking overhead.

Any attempt by the OS to perform operations that would result in access to resources of another partition or the hypervisor is prevented through a combination of hardware and firmware design. Paravirtualization represents a performance middle ground, at the cost of a relatively few OS changes, between the performance loss that is typical of the pure trap-and-emulate method and the complexity of sophisticated processor virtualization extensions.

Previous iSeries and pSeries systems [3], based upon POWER4* technology [4], provided the capability of dividing the platform's hardware resources into disjoint subsets. Each independent subset is controlled by its own copy of an OS, which runs its own application programs. Each of these divisions of the system is called a logical partition (LPAR). In response to commands, an LPAR may give up some of its resources and another system LPAR may acquire free resources, thus allowing the system administrator[2] to balance the use of the platform's resources among its workloads over time. However, the processing capacity of each LPAR is still usually oversized to accommodate moment-to-moment variations in workload in order to be responsive to instantaneous workload peaks. As the number of independent workloads increases, the probability that each workload will experience an instantaneous peak at the same time decreases. The POWER5* processor [5] provides mechanisms to allow the platform firmware[3] to instantaneously reassign an idle processor to another LPAR, such that the platform appears to have more processors than are physically present. These *extra processors* are a virtualization of the physical resources of the platform. Systems based upon the POWER5 processor provide significantly improved resource utilization and partitioning capabilities when compared with their predecessors because of the resource virtualization ability of the POWER5 processor. This paper provides an overview of these improvements.

Partitions on POWER5 systems can simultaneously run multiple copies, in any combination, of the AIX*, i5/OS*, and Linux** [6] operating systems and VIOS. VIOS is the virtual I/O server platform firmware component that runs in a logical partition. VIOS provides virtual I/O services to one or more partitions (see the section on POWER5 I/O virtualization). A partition is assigned a non-overlapping subset of the platform's resources, including one or more virtual processors, regions of system memory, and I/O adapter bus slots. Platform firmware communicates the resource configuration of the partition to the OS image. Allocatable resources are directly controlled by an OS. Other platform resources, such as memory controllers, interrupt controllers, and major portions of the I/O infrastructure, are controlled by the hypervisor. The OS makes requests to use these resources via the *hcall* instruction.

The POWER hypervisor is a common design for both iSeries and pSeries servers. It builds upon many thousands of machine years of combined field experience accumulated by the firmware in both series. The virtualization capabilities of the POWER5 hypervisor are further explained in the sections that follow.

Figure 1 shows a simplified diagram of a POWER5 system. The hypervisor layer is responsible for validating OS requests to use shared platform resources, thus ensuring the integrity and isolation of these partitioned systems. Operational management for the platform is performed via the Hardware Management Console (HMC). The HMC is a set of firmware tools, optionally

[2]The system administrator has the authority to dynamically change the amount of resources assigned to each of the partitions on a system.
[3]Platform firmware is microcode embedded in the platform hardware.

running on replicated hardware, that manages platform resources via messages to the hypervisor and the operating systems of the partitions. The HMC creates the initial configuration definition, controls boot and termination of the various partitions, and provides virtual console support. It is the control point for dynamic reconfiguration of the resources of the partition, platform hardware hot-plug operations (I/O adapters and I/O drawers[4]), and deferred and concurrent maintenance of both hardware and firmware.

Loaded within the partition memory space is partition firmware (PFW). This code is platform-architecture-dependent. For i5/OS partitions, the partition firmware is System Licensed Internal Code (SLIC). For AIX and Linux partitions, the partition firmware is IEEE 1275 Boot (Initialization Configuration) Firmware [7] and PowerPC* Microprocessor Common Hardware Reference Platform–Run Time Abstraction Services [8].

A POWER5 platform is assembled by interconnecting basic computation blocks, each of which has its own processing cores, cache, memory, and I/O capability. The interconnect fabric provides virtually uniform memory access time, but the bandwidth between the blocks can vary. Therefore, assigning the virtual processors of the partition to physical processors that have the greatest affinity with the partition memory locations optimizes system performance. The POWER hypervisor considers computation resource affinity in several ways in its allocation algorithms.

POWER partitioning and virtualization support

To support robust partitioning and virtualization on POWER processors, extensions to the PowerPC Architecture* were required. These extensions were created in a staged approach, with the base support for robust partitioning provided in the POWER4 family of processors [3, 4] and the advanced support for virtualization being delivered by the POWER5 [5] family of processors. The base support in POWER4 processors for robust partitioning included the introduction of a new privileged state of the processor, called hypervisor mode. The processor must be in hypervisor mode in order to have write access to some of the processor system registers, such as the register that defines the location and size of the hardware page table associated with the partition. This relatively simple extension to the PowerPC architecture provides the mechanism required for complete partition isolation [9].

Also introduced in POWER4 processors was support for a virtual address zero in each partition when address translation is disabled. When the OS kernel is running with address translation disabled, it can access only a small portion of the real memory owned by the partition via a zero-based address; the remainder of the real memory owned by the partition must be accessed by the partition using a virtual address. If the partition attempts to reference beyond the boundary of the accessible portion of real memory when address translation is disabled, the processor generates an exception. To accomplish this, the new *real mode region* (RMR) register was defined for the partition. This register can be accessed only when running in hypervisor mode; it defines the offset that is added to the zero-based address specified by the partition when address translation is disabled to access the physically contiguous region of real memory that has been allocated as the RMR area for the partition. All of the interrupt vectors for the operating system will get control, with address translation disabled at the architecturally defined offset into the partition's RMR.

In POWER5 processors, several very important extensions were made to the base partitioning support in order to efficiently support the virtualization of processors and sharing of a physical processor by multiple partitions. The most important extension was the introduction of the hypervisor decrementer (HDECR) facility, which is used for fine-grained dispatching of multiple partitions on shared processors. The HDECR provides the hypervisor with a guaranteed timer interrupt regardless of the partition execution state. Unlike the regular decrementer used by the partition for timer interrupts, the HDECR interrupt is routed directly to the hypervisor and uses only hypervisor resources to capture the state of the partition. An additional assist provided by POWER5 processors for virtualization of processors is the capability to route external interrupts to the hypervisor (instead of the partition).

POWER5 processors also introduced simultaneous multithreading (SMT) support. Between SMT and shared processors, support for a new register to track the cycles consumed by a partition on each thread was required. The processor utilization of resources register (PURR) was added in POWER5 processors so that a partition receives a very accurate accounting of the number of cycles executed by processor threads when it executes on that physical processor.

The SMT support on POWER5 processors also introduced support for a *dormant thread*. When the partition does not have work to dispatch on one of the two threads of the physical processor, it can invoke an *hcall* to make the thread dormant. If the other thread is active, the hypervisor makes the invoking thread dormant so that all of the register resources and cycles available on the physical processor are utilized by the remaining active thread. When an interrupt or time-out occurs for the dormant thread, the hardware revives the thread, its state is fully restored, and control is returned to the partition.

[4]An I/O drawer is a package of hardware that may be attached to the basic platform to extend the I/O capacity of the platform.

Several other smaller processor extensions were made for partitioning and virtualization, but this section has touched on the most significant ones. The remainder of the paper discusses the advanced virtualization capabilities that were built upon these processor extensions.

Virtualization of processors

A partition, which can be either a dedicated or a shared processor partition, views its processors as virtual processors. The virtual processor of a dedicated processor partition has a physical processor allocated to it, while the virtual processor of a shared processor partition shares the physical processors of the shared processor pool with virtual processors of other shared processor partitions.

The configuration of a shared processor partition requires that the system administrator specify the entitled capacity for the partition, in addition to the number of virtual processors that are configured for the partition. The entitled capacity (in hundredths of a processor) is the allocation of physical processor resources to the shared processor partition. The physical processor allocation for a virtual processor depends on the entitled capacity and the number of virtual processors that are online for that partition. The system administrator must also specify whether the shared processor partition is *capped* or *uncapped*. A capped partition cannot receive more cycles than its entitled capacity. An uncapped partition will receive processor cycles beyond its entitled capacity if excess processor cycles are available in the shared processor pool. The HMC allows the system administrator to configure a partition's uncapped weight parameter, which represents a priority share of unused processing capacity relative to the uncapped weight parameters of other uncapped partitions sharing the same processor pool. When there is contention for the excess cycles, this parameter is used by the hypervisor to allocate the excess cycles in the shared processor pool. Dynamic logical partition (DLPAR) operations can dynamically change the entitled capacity, the number of virtual processors, and the uncapped weight of a partition.

Shared processor dispatching

The hypervisor attempts to dispatch a virtual processor on the same physical processor where it last ran. If that processor is not available, it attempts to find one on the last POWER5 chip used, and, failing that, one on the last basic computation block. If the virtual processor has not run for a long time and has therefore lost all affinity, the virtual processor is dispatched on its *home* node. The home node for a partition is the basic physical computation block from which the majority of its memory is allocated. Similarly, when an idle processor looks for work and cannot find a virtual processor with affinity for the current physical processor, it attempts to dispatch a virtual processor that has the current basic computation block as its home node or a virtual processor that has no affinity for any other basic computation block.

Operating system optimizations

Several optimizations can be implemented by an operating system to improve not only the performance of its own partition, but also the overall performance of the partitions running within the shared pool of physical processors. The operating system first registers a virtual processor area (VPA) for each of its assigned virtual processors with the hypervisor. The VPA serves as a two-way communication area between the operating system and the hypervisor regarding details of the virtual processor. One of the fields in the VPA is an idle flag. Whenever the operating system enters its idle process, the OS sets the VPA idle flag. When work becomes available for the virtual processor, or when an interrupt is received on the virtual processor, the OS clears the VPA idle flag.

When the OS is idle, it can cede the physical processor back to the hypervisor by invoking the *h_cede hcall*, essentially yielding the remainder of its entitlement on that physical processor. The hypervisor can then dispatch another virtual processor on that physical processor or utilize the physical processor for its own purposes. Once the virtual processor has ceded, it remains *blocked* until its next interrupt occurs or until it is explicitly *prodded* by another virtual processor within the same partition. If the OS, running on another virtual processor within its partition, determines that new work is available for one of its idle virtual processors, the OS can prod the ceded virtual processor by invoking the *h_prod hcall*. This causes the hypervisor to *unblock* the prodded virtual processor, which then becomes available for dispatch on a physical processor. When dispatched, the virtual processor resumes execution at the instruction following the *h_cede hcall*.

Cases exist in which the operating system execution on a virtual processor encounters a dependency on one of its other virtual processors. Two examples of this are spinlocks and synchronous interprocessor interrupt communication. A spinlock is a loop executing a nonblocking atomic serialization primitive used to acquire a software lock, or reservation, in a symmetric multiprocessing (SMP) system. If the lock is currently held, the spinlock function spins in place waiting for the lock to become available. Synchronous interprocessor interrupt communication occurs when one processor of an SMP system posts an interrupt for another processor within the system and spins in place waiting for acknowledgment of the interrupt. In these cases, the

operating system can further optimize the utilization of the underlying physical processor by conferring its remaining entitlement on the virtual processor on which it is dependent by invoking the *h_confer hcall*.

When the hypervisor is performing a context switch of a virtual processor, it must save all of the architecturally program-visible processor state. In many instances, not all of the processor state is being used by the operating system, so the complete state save is unnecessary. To minimize the cost of a virtual processor context switch, the OS can indicate to the hypervisor whether some resources are in use. Some examples of this on the POWER5 processor are the floating-point registers, performance monitor registers, and segment lookaside buffer (SLB) registers. If the operating system is currently using one of these resources, it sets the corresponding VPA in-use field. In the case of SLB registers, the OS can maintain a shadow copy in the VPA of all SLBs currently in use. This allows the hypervisor to avoid having to save the SLB registers on a virtual processor switch-out, and allows the hypervisor to restore only the SLBs currently in use on a virtual processor switch-in. One other aspect of running in a shared processor partition that must be handled by the operating system is the concept of processor time. This is especially important when the operating system is performing fine-grain process-level accounting. In a dedicated processor environment, the amount of processor time an operating system is using must correspond directly to the elapsed time. On PowerPC Architecture processors, the operating system typically uses the *timebase* facility of the processor for this purpose. In a shared processor environment, the elapsed *timebase* ticks no longer correspond to execution time received by the partition on a virtual processor. The PURR (see the previous section on POWER partitioning and virtualization support) is used to accumulate ticks only when the virtual processor is dispatched on a physical processor. The operating system can then tailor its accounting and utilization algorithms to be based on accumulated PURR cycles rather than elapsed *timebase* cycles.

POWER5 I/O virtualization

I/O partitioning provides multiple operating systems running in a single system with their own set of I/O devices, such as storage adapters, network adapters, and console devices. IBM POWER5 systems support partitions with either physical or virtual I/O devices as well as partitions with a mixture of both types.

As in POWER4 systems, POWER5 systems support slot-level partitioning of physical devices. This means that each peripheral component interconnect (PCI) slot in the system can be individually assigned to a logical partition. The hypervisor ensures that each logical partition can access only the PCI slots assigned to it; it cannot access other PCI devices, even if they are on the same bus.

In addition, POWER5 systems support the definition of virtual adapters. Virtual adapters provide capabilities similar to those of physical adapters, but they are implemented entirely in software and do not require dedicated physical PCI slots. A key component of I/O virtualization on POWER5 servers is the IBM Virtual I/O Server (VIOS), which executes in a logical partition explicitly created by the system administrator and provides VSCSI and shared Ethernet adapter virtual I/O capabilities to client logical partitions within the POWER5 system. The VIOS requires physical resources (CPU, memory, I/O adapters) and special software that is distributed on VIOS installation media. It takes advantage of new POWER5 platform features including Logical Remote Direct Memory Access (LRDMA). When this is coupled with powerful redundancy solutions such as High Availability Cluster Multiprocessing (HACMP), as well as multipathing options at both the VIOS and client logical partitions, it provides the building blocks necessary to create an enterprise-level virtualized I/O environment.

Virtual adapters

The virtualization of processors allows IBM POWER5 systems to support many more active partitions than there are physical processors in the server. The ability to support hundreds of partitions is less useful, however, if each partition requires dedicated I/O adapter cards, such as network and storage adapters. A key virtualization feature of the new IBM POWER5 systems is the ability to create virtual adapter cards for an operating system. In its most extreme example, a system could support hundreds of operating system images with only a single real Ethernet adapter and a single real storage adapter.

One area of operating systems that leads to complexity is the support of I/O devices and adapters. To reduce the complexity of the hypervisor, a design decision was made to limit support for I/O adapters in the hypervisor itself, delegating that function to the operating systems running in logical partitions. This approach is designed to simplify the hypervisor tremendously, eliminate a source of errors that could have a potential impact on the whole system, and eliminate the need for updates to the hypervisor to support new I/O devices.

The decision to keep awareness of I/O adapters out of the hypervisor significantly affects the design of I/O virtualization. The hypervisor itself does not own real I/O devices or provide virtualized interfaces to those devices. Instead, all I/O devices are owned by logical partitions. These logical partitions provide access to real hardware to client partitions through virtual devices. The logical partitions that own the physical resources are referred

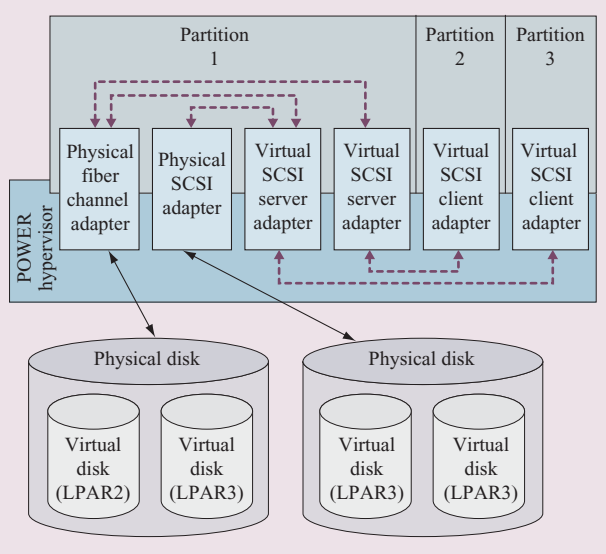

Figure 2

Virtual SCSI topology.

to as virtual I/O server partitions. The VIOS, an i5/OS partition, and a Linux partition can all provide virtual I/O services to other partitions.

The approach taken was to implement two sets of operations in the hypervisor. The first set provides support for the instantiation of virtual adapters in a partition. This includes definitions of direct memory access (DMA) windows, interrupts, and adapter information similar to that required to identify a physical adapter to a partition. The second set of operations allow partitions to interact, in a very controlled and secure way, such that one partition can gain access to physical I/O resources in a different partition.

Three kinds of virtual adapters are supported—SCSI, Ethernet, and serial/console. While the primitive operations are slightly different for each, the concept of allowing one partition to provide access to physical I/O devices for one or more other partitions is similar.

Virtual adapters are created by the system administrator as part of logical partition definition. Virtual adapters exist in virtual I/O slots, and the administrator can configure the number of I/O slots and the types of virtual adapters in those slots.

Information about virtual adapters, such as interrupt numbers and DMA windows, is presented to the operating system through the device tree, similarly to the way in which information about the physical adapters owned by the partition is presented. Specific device drivers were written within the operating systems for each virtual adapter type, just as would be required for a new type of physical adapter.

Virtualization of I/O interrupts

In order to maintain partition isolation, the hypervisor controls the hardware interrupt management facilities. The partition is provided controlled access to the interrupt management facilities through a set of interrupt management *hcalls*. Furthermore, the interrupt management facilities are virtualized such that the facilities and the semantics of the *hcalls* are preserved for shared processor partitions. The same set of *hcalls* is used for interrupts from real adapters as well as interrupts from virtual sources such as VLAN, VSCSI, and virtual console adapters. The hypervisor uses a combination of software queues and the PowerPC interprocessor interrupt (IPI) facilities for virtualization of the interrupt management facilities.

Virtual SCSI support

The small computer system interface (SCSI) protocol was chosen as the mechanism for virtualizing storage devices, including disk, CD, and tape. The VSCSI support is implemented as two paired virtual adapters, the virtual SCSI client adapter and the virtual SCSI server adapter, as shown in **Figure 2**. The client adapter is the SCSI initiator and follows all of the semantics of any SCSI host bus adapter; to the operating system that implements it, it is no different from any other SCSI adapter. The adapters are used only to transfer SCSI commands between partitions; the SCSI commands themselves are generated by the client operating system, as with any storage operation.

The server adapter, known in SCSI terminology as a target adapter, is responsible for executing any SCSI command received. However, it is entirely up to the operating system owning the SCSI server adapter how the command is executed. For instance, for some devices, the SCSI server may simply pass all commands directly to the real physical device. Alternatively, the SCSI server may emulate the SCSI target device completely. A combination of the two is also possible; this flexibility helps ensure that operating systems using a virtual SCSI client adapter are completely isolated from the implementation on the server side.

The virtual SCSI function is based on two primitive operations implemented by the hypervisor. First, there is a message-queuing function that allows one partition to send small messages to another partition, with an interrupt mechanism allowing the receiving partition to be notified when a new message arrives. All I/O requests and responses are passed through this message-passing facility. SCSI Remote DMA Protocol (SRP) [10] is used to govern the rules for exchanging information between the SCSI initiator (VSCSI client) and the SCSI target (VSCSI server).

Second, there is the ability to issue direct memory access (DMA) operations between partitions. While

preserving the security and integrity of the system requires that one partition must not be able to directly read or write the memory of another partition, the hypervisor provides a virtual DMA capability between explicitly authorized partitions. These DMA operations are designed to preserve the integrity of both sides of the operation, since both the client and the server explicitly inform the hypervisor of the DMA regions before the operation takes place.

The virtual DMA capability is used to move SCSI commands and data between partitions. When redirecting SCSI commands to a real device, the server partition can cause the device to transfer data (via DMA) directly to and from the memory of the client partition, eliminating the need to copy data between partitions.

Virtual IEEE VLAN switch

The POWER hypervisor provides a virtual Ethernet switch which is designed to enable partitions on the same system to quickly and securely communicate with one another without requiring physical Ethernet adapters. For systems that do not require a dedicated connection to networks outside the system, the VLAN switch provides an ideal solution without requiring the dedicated use of the finite number of physical I/O slots available. The switch is based on the IEEE 802.1q VLAN standard [11], allowing for isolation of the partitions connected to the switch on the basis of their VLAN membership.

The VLAN switch is configured by assigning virtual Ethernet adapters to each of the partitions that are connected to the switch. When a virtual Ethernet adapter is assigned to a partition, the user configures the virtual switch port to which the adapter is conceptually connected. The ports can be configured to be IEEE 802.1q-aware or not, have a default port VLAN ID (PVID) assigned, and be assigned membership in multiple VLAN IDs if the port is 802.1q-aware. A simple virtual Ethernet topology is shown in **Figure 3**.

Functionally, the switch performs the functions of a typical midrange-managed Ethernet switch. Media access control (MAC) address tables are maintained for filtering unicast[5] traffic. VLAN membership tables are maintained to provide isolation of adapters based on VLAN IDs. IEEE 802.1q headers and VLAN tags are transparently inserted and removed as traffic flows between 802.1q-aware and non-aware ports. MAC address takeover is detected and logged. Trunk adapter[6] facilities are provided to allow transparent layer-2 bridging.

From the operating system perspective, its connection to the VLAN switch is seen as a virtual Ethernet adapter. This adapter operates similarly to a physical Ethernet adapter, but without many of the complexities found in physical adapters. Virtual Ethernet adapters are not limited to an arbitrary maximum packet size of 1,514 bytes (or 9,014 bytes for jumbo-packet-aware adapters) as are physical Ethernet adapters. The virtual Ethernet adapters need not be concerned with physical link configuration, asynchronous transmit operations, or hardware errors. Transmit operations are performed synchronously and are complete as soon as the *hcall* to send a frame returns. Receive operations are performed by providing a pool of buffers to the hypervisor for receiving frames. As incoming frames are received by the adapter, the hypervisor places the new frames on a receive queue and informs the device driver with a virtual interrupt. Interfaces are also provided for the device driver to dynamically change the MAC address of the adapter and manage the multicast[7] filtering behavior and tables.

Each virtual Ethernet adapter can also be configured to be a trunk adapter. Virtual Ethernet adapters, like most other physical Ethernet adapters, receive only frames with a destination MAC address that matches their specific MAC address. If a partition sends an Ethernet frame to a destination MAC address that is unknown to the hypervisor virtual switch, the hypervisor will deliver the unknown packet to any trunk adapter defined for that VLAN ID. This allows layer-2 bridging to a physical adapter to be performed, extending the virtual Ethernet network outside the system onto a physical Ethernet network. Without trunk adapter support, bridging the virtual Ethernet networks outside the system would require a higher-level, more CPU-intensive bridging method such as IP routing, network address translation, or transparent subnets.

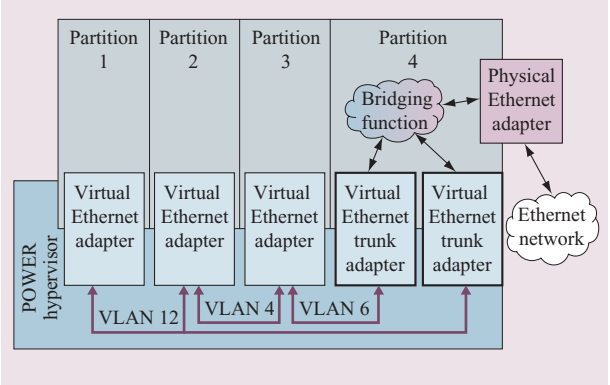

Figure 3

Virtual Ethernet topology.

[5]A unicast frame is an Ethernet frame targeted at one specific MAC address.

[6]A trunk adapter is a special virtual adapter that receives virtual network traffic so that it can be bridged to a physical network.

[7]A multicast frame is an Ethernet frame which is delivered to multiple Ethernet adapters using a reserved portion of the MAC address space.

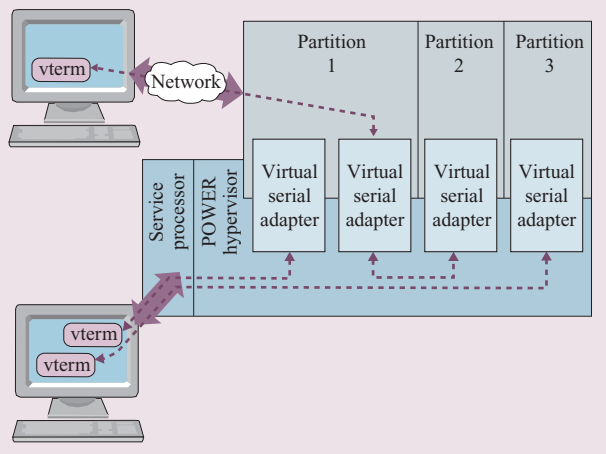

Figure 4

Virtual console topology.

The virtual Ethernet adapters also provide a form of TCP/IP checksum[8] offload. The transmission control protocol (TCP) requires the TCP/IP stack to calculate a 16-bit checksum over the entire user data payload of a TCP packet, a TCP header, and a TCP pseudo-header, which includes portions of the Internet protocol (IP) header. The TCP checksum helps to guarantee the integrity of the packet as it travels through the various routers, switches, and bridges making up the physical Ethernet. Performing the TCP checksum on POWER5 systems can take an estimated 10% of the total CPU processing required to send the packet, especially for large packets. Many physical adapters provide facilities to perform the TCP checksum while sending the packets, offloading the host CPU.

Since the transmission of a packet on the virtual Ethernet network does not involve any physical networks, eliminating the TCP checksum from the process required to transmit packets can provide huge savings in CPU cycles consumed with no risk. Each TCP/IP stack and device driver for the three different operating systems providing virtual Ethernet support has the ability to bypass the checksum calculations. The hypervisor provides several *hcalls* enabling the virtual Ethernet device driver to notify the hypervisor that the operating system is aware that checksums need not be calculated on the virtual Ethernet. If one partition sends an Ethernet packet with no TCP checksum to another that has not performed this *hcall*, the hypervisor will calculate the correct TCP checksum and place it in the packet before finishing the virtual DMA.

[8]A checksum is a mathematical operation performed across a set of data that is used to detect changes in the data and ensure data integrity across unreliable transports.

Virtual console support

The system console is the last piece of I/O infrastructure that is required for a full partition. Most operating systems require a dedicated console for processes such as installing the operating system, setting up the network, or performing problem analysis of early boot failures. In order to virtualize the console support, the hypervisor provides a virtual teletype (TTY)/serial adapter with a suite of *hcalls* that operate on this virtual adapter.

Depending on the specific system configuration, the operating system console may be provided by an HMC virtual terminal (vterm), through another partition connected via its own virtual serial adapter, or via a terminal emulator connected to physical serial ports on the system service processor. **Figure 4** shows the multiple ways in which a virtual console may be provided.

Dynamic reconfiguration

Many modern systems support dynamic reconfiguration of system hardware. This reconfiguration includes adding or removing processors, adding or removing memory, and adding or removing I/O devices. A common example is the ability to replace a failing I/O adapter while the system continues to run.

In a virtualized environment, the requirements for dynamically reconfiguring the hardware used by an operating system increase dramatically. Workload changes may require that processors, memory, and I/O adapters be moved between logical partitions. Customers using these systems in high-availability environments cannot tolerate rebooting operating system images to accommodate such changes.

The POWER5 hardware and firmware supports adding and removing virtual processors, entitled processor capacity, uncapped processor weight, memory, and I/O devices for logical partitions while the operating systems are running. Such resource moves involve cooperation between the HMC, where the moves are initiated, and the operating systems in question. A component in the operating system known as the dynamic reconfiguration manager (DRMGR) communicates over a network connection to the HMC to coordinate, execute, and acknowledge such changes.

Cross-logical-partition workload managers can also initiate dynamic resource changes between partitions utilizing these same interfaces. Support of dynamic reconfiguration required changes to the firmware as well as to the operating systems involved. A cross-partition workload manager manages resources in a set of partitions on a system. The system administrator can use the HMC to define the set of partitions that can be managed as a group. The OS provides a set of application programming interfaces (APIs) and the hypervisor provides a set of hypervisor calls so that the workload

manager running in each partition of the group can move resources between partitions to achieve its goals. The hypervisor is designed to ensure that the set of partitions defined to be in a group can use only the resources that the system administrator has assigned to the partitions in the group.

Concluding remarks

The combination of the POWER5 processor with the POWER hypervisor provides IBM with leading-edge virtualization and partitioning support in the latest POWER servers. The paravirtualization, or cooperative partitioning provided by POWER5 systems in conjunction with the AIX, i5/OS, and Linux operating systems, results in an excellent combination of virtualization capabilities with minimal overhead. These features include micro-partitioning of servers through the virtualization of processors and I/O, efficient resource utilization through recovery of idle processing cycles, dynamic reconfiguration of partition resources, consolidation of multiple operating systems on a single platform, and platform-enforced security and isolation between partitions.

Acknowledgments

A great many individuals have been involved in the development of the POWER hypervisor from its earliest conceptual stages, through design, implementation, performance analysis, and system test. This project included engineers and programmers from several IBM research and development laboratories. The focal point for the project was in the IBM Systems and Technology Group Development Laboratory in Rochester, Minnesota, and Austin, Texas. Unfortunately, space does not permit us to name all of the individuals, as the list would be too long, but their efforts and contributions are very significant and are much appreciated.

References

1. A. Whitaker, M. Shaw, and S. D. Gribble, "Denali: Lightweight Virtual Machines for Distributed and Networked Applications," *Technical Report 02-02-01*, University of Washington, Seattle, 2002.
2. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugeberger, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *Proceedings of the 2003 Symposium on Operating Systems Principles*, October 2003, pp. 164–177.
3. "Partitioning for the IBM eServer pSeries 690 System," IBM white paper; see *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar.html*.
4. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM J. Res. & Dev.* **46**, No. 1, 5–26 (November 2002).
5. R. Kalla, B. Sinharoy, and J. M. Tendler, "IBM POWER5 Chip: A Dual-Core Multithreaded Processor," *IEEE Micro* **24**, No. 2, 40–47 (March–April 2004).
6. D. Boutcher and D. Engebretsen, "Linux Virtualization on IBM POWER5 Systems," *Proceedings of the Linux Symposium*, Vol. 1, July 2004, pp. 113–120.
7. IEEE 1275 Boot (Initialization Configuration) Firmware documentation; see *http://playground.sun.com/pub/p1275/*.
8. *Microprocessor Common Hardware Reference Platform: A System Architecture*, Morgan Kaufmann, San Francisco, August 1997; ISBN 1-55860-394-8.
9. "Logical Partition Security in the IBM eServer pSeries 690," IBM white paper; see *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar_security.html*.
10. "SCSI RDMA Protocol (SRP)," *INCITS Technical Report for Information Technology, INCITS:365:2002*, December 2002; see *http://webstore.ansi.org/ansidocstore/product.asp?sku=ANSI+INCITS+365-2002*.
11. *IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks 2003, 802.1q*, 2003 edition; ISBN0-7381-3663-8.

Received July 6, 2004; accepted for publication April 8, 2005; Internet publication September 6, 2005

William J. Armstrong *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota (billarm@us.ibm.com).* Mr. Armstrong is an IBM Distinguished Engineer; he is the lead architect for the POWER hypervisor firmware. He has worked for IBM since 1988 on a variety of projects, focusing on kernel development, systems performance, and logical partitioning and virtualization of POWER platforms for the IBM pSeries and iSeries systems. Mr. Armstrong holds numerous patents in the areas of partitioning, virtualization, and kernel design. He received a bachelor's degree in computer engineering from the University of Notre Dame and an M.S. degree in computer engineering from Iowa State University.

Richard L. Arndt *IBM Systems and Technology Group, 11501 Burnet Road, Austin, Texas 78758 (rlarndt@us.ibm.com).* Mr. Arndt is a Senior Technical Staff Member and a RISC platform architect. He started his career on active duty with the U.S. Army developing instrumentation at the White Sands Missile Range in New Mexico. Since joining IBM in 1970, he has had many hardware and software architecture and development assignments, including data collection terminals, communications subsystem adapter logic and firmware design, architecture for several operating systems including AIX, and server system firmware. Mr. Arndt has worked with several industry groups to develop standards for I/O and system firmware and architecture. He is responsible for defining the architecture for IBM pSeries logical partitioning (LPAR) and holds numerous patents in the area of LPAR and platform resource virtualization. He received bachelor's and M.S. degrees in electrical engineering from the University of Wisconsin at Madison.

David C. Boutcher *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (boutcher@us.ibm.com).* Mr. Boutcher is a Senior Technical Staff Member in the IBM Linux Technology Center. He has worked at IBM development laboratories in Canada and the U.S. since 1986 on a variety of operating systems, focusing primarily in the areas of communications and distributed computing. He received a bachelor's degree in computer science from the University of Manitoba and is currently a graduate student at the University of Minnesota.

Robert G. Kovacs *IBM Systems and Technology Group, 11501 Burnet Road, Austin, Texas 78758 (bkovacs@us.ibm.com).* Mr. Kovacs is a Senior Software Engineer with the IBM Systems and Technology Group in Austin, Texas. He joined IBM in 1988 and has worked in AIX kernel development and in storage software development for AIX and pSeries. He received a bachelor's degree in computer science from Old Dominion University.

David Larson *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (larson1@us.ibm.com).* Mr. Larson is an Advisory Engineer with the IBM Systems and Technology Group in Rochester, Minnesota. He has worked for IBM since 1999 on a variety of projects including i5/OS PASE, the AIX runtime environment for i5/OS, and the POWER hypervisor. He received a bachelor's degree in computer science from North Dakota State University.

Kyle A. Lucke *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (klucke@us.ibm.com).* Mr. Lucke is an Advisory Software Engineer with the IBM Systems and Technology Group in Rochester, Minnesota. He has worked for IBM since 1996 on a variety of projects, focusing on communication device drivers and virtualization of hardware for hypervisors, Linux, OS/400, and i5/OS. He received a bachelor's degree in computer science from the University of North Dakota.

Naresh Nayar *IBM Systems and Technology Group, 3605 Highway 52 N., Rochester, Minnesota 55901 (nayar@us.ibm.com).* Dr. Nayar is a Senior Technical Staff Member with the IBM Systems and Technology Group in Rochester, Minnesota. He joined IBM in 1992 and has worked on many i5/OS kernel projects, with a focus on synchronization primitives and task dispatching. His most recent work has been in the area of LPAR for iSeries and pSeries systems, and he holds numerous patents in the area of partitioning and kernel design. Dr. Nayar received a Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, and M.S. and Ph.D. degrees in computer science from Iowa State University.

Randal C. Swanberg *IBM Systems and Technology Group, 11501 Burnet Road, Austin, Texas 78758 (rswanber@us.ibm.com).* Mr. Swanberg is a Senior Technical Staff Member with the IBM Systems and Technology Group in Austin, Texas. After beginning his career working on defense navigation systems for the U.S. Army at Fort Hood, Texas, he joined IBM in 1989 and has worked on several operating system projects including AIX, OSF, Monterey, and Linux. He received a bachelor's degree in computer science from Baylor University.